Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

IL LAMBDA-CALCOLO CON LA STRATEGIA CALL-BY-VALUE

Relatore:
Chiar.mo Prof.
Claudio Sacerdoti Coen

Presentata da:
Marco Russo

Quarta Sessione
Anno Accademico 2023-2024

"So long and thanks for all the fish."

Douglas Adams

Introduzione

La tesi vuole rappresentare un punto di partenza per chi si approcci al lambda-calcolo, introducendo il tema e mostrando in particolare alcuni dei più recenti sviluppi del formalismo.

All'interno del primo capitolo viene presentato il lambda-calcolo: vengono presentate le ragioni che ne portarono all'invenzione da parte di Alonzo Church, ne viene presentata la sintassi e le operazioni di riduzione che lo caratterizzano.

Nel secondo capitolo, vengono presentate due diverse forme di lambda-calcolo che implementano la strategia Call-by-Value: il λ_v-calcolo introdotto da Plotkin [1] e il λ_{fire}-calcolo introdotto da Paolini e Della Rocca [2] e viene mostrato come entrambi i calcoli presentino dei problemi semantici.

Nel terzo capitolo viene presentato un nuovo modello di lambda-calcolo, il Value Substitution Calculus, che ambisce a risolvere i problemi semantici che caratterizzano le due presentazioni precedenti.

Indice

Capitolo 1

Il λ-calcolo

Il lambda-calcolo è un **linguaggio formale**, che fornisce delle regole per definire e manipolare le funzioni sintatticamente. Esprime una nozione di computazione basata sui concetti di astrazione e applicazione di funzione, utilizzando il legame delle variabili e le sostituzioni. Le espressioni del linguaggio sono dette *lambda-termini* e vengono definite delle regole per manipolare le espressioni.

1.1 La nascita del lambda-calcolo

Nel decennio 1930-1940, molti matematici erano interessati alla seguente domanda: *cosa significa per una funzione* $f : \mathbb{N} \to \mathbb{N}$ *essere calcolabile?* Una definizione informale di calcolabilità è che ci dovrebbe essere un *metodo applicabile con carta e penna* che permetta ad una persona istruita di calcolare $f(n)$, per ogni n. Ma come formalizziamo il concetto di *metodo con carta e penna?* E come definiamo una *persona istruita?* Tre diversi ricercatori cercarono di dare una risposta al problema:

1. **Turing** definì una *nozione astratta di computer, detta macchina di Turing*, postulando che una funzione è intuitivamente calcolabile se e solo se può essere calcolata da una *macchina di Turing*

2. **Gödel** definì la classe di *funzioni generali ricorsive*, come il più piccolo insieme di funzioni che contenga tutte le funzioni costanti, la funzione successore e chiusa sotto certe operazioni, come composizione e ricorsione. Postulò che una funzione è intuitivamente calcolabile se e solo se è generale ricorsiva.

3. **Church** definì un *linguaggio di programmazione idealizzato, il λ-calcolo*, e postulò che una funzione è intuitivamente calcolabile se e solo se può essere scritta come un lambda-termine.

Sebbene sia impossibile definire formalmente cosa significhi *intuitivamente calcolabile*, negli anni seguenti fu dimostrato che tutti e tre i modelli proposti erano *equivalenti tra loro*, ovvero ogni modello definisce la stessa classe di funzioni computabili. Questo risultato è oggi noto come **Tesi di Church-Turing**.

Il significato di un programma può essere considerato come una funzione che mappa degli input a degli output. Le funzioni a questo punto giocano un ruolo fondamentale nella definizione della semantica di un linguaggio di programma.

Il lambda-calcolo rappresenta il più semplice linguaggio di programmazione, in quanto consiste di una singola regola di trasformazione, la β-riduzione, ed è alla base dei linguaggi di programmazione funzionale. Inoltre, la maggior parte dei linguaggi di programmazione moderni implementano oggi il lambda-calcolo, fornendo dei costrutti per costruire *funzioni anonime* - si pensi alle arrow-function di Javascript oppure alle lambda-espressioni di Java.

1.2 I sistemi di riscrittura dei termini

Il lambda-calcolo può essere studiato come un sistema di riscrittura dei termini.In questa sezione vengono introdotte alcune delle principali proprietà dei sistemi di riscrittura dei termini.

Un sistema di riscrittura astratta è un *formalismo* che cattura la nozione e le proprietà dei sistemi di riscrittura. Viene rappresentato come una coppia

$(A, \to)$, dove la *riduzione* $\to$ è una relazione binaria su un insieme A di oggetti, ovvero $\to \subseteq A \times A$. É possibile avere una o più relazioni di riduzione, nel caso di più relazioni queste vengono descritte come $\to_1$, $\to_2$, ... Con un piccolo abuso di notazione, scriviamo $a \to b$ invece di $(a, b) \in \to$.

La restrizione dei sistemi di riscrittura astratta al caso in cui i suoi oggetti siano *termini*, ovvero espressioni con nidificate delle sotto-espressioni, viene detto *sistema di riscrittura dei termini*. Una regola viene quindi descritta come una coppia di termini, indicata $l \to r$, ad esprimere che la parte sinistra l viene riscritta nella parte destra r attraverso la regola di riduzione $\to$.

Data una relazione $\to$ su un insieme di termini, si definisce $\to^+$ la chiusura transitiva e $\to^*$ la chiusura riflessiva e transitiva.

Data una relazione $\to_r$, una r-*valutazione* d è una sequenza finita di termini $(t_i)_{0 \leq i \leq n}$, per qualche $n \geq 0$ tale che $t_i \to t_{i+1}$ per ogni $1 \leq i < n$, e scriviamo $d : t \to_r^* u$ se $t_0 = t$ and $t_n = u$.

La *lunghezza* n di d viene descritta con $|d|$, e $|d|_a$ rappresenta di numero di *a-passi*, ovvero il numero di $t_i \to_a t_{i+1}$ per qualche $1 \leq i \leq n$ in d per una data sottorelazione $\to_a$ di $\to_r$.

Un termine t viene detto *r-normale* se non esiste alcun u tale che $t \to_r u$.

Una valutazione $d : t \to_r^* u$ viene detta *r-normalizzante* se u è r-normale.

Un termine t è *debolmente r-normalizzante* se esiste esiste una valutazione r-normalizzante $d : t \to_r^* u$.

Un termine t è *fortemente r-normalizzante* se non esistono sequenza infinite $(t_i)_{i \in \mathcal{N}}$ tali che $t_0 = t$ e $t_i \to_r t_{i+1}$ per ogni $i \in \mathcal{N}$.

Una relazione $\to_r$ è *confluente* se $t \to_r^* u_1$ e $t \to_r^* u_2$ implica che esiste un termine w tale che $u_1 \to_r^* w$ e $u_2 \to_r^* w$. Si può scrivere, in forma più compatta come:

$$u_2 \,{}_r^*\!\leftarrow t \to_r^* u_1 \text{ implica che esista } u_2 \to_r^* w \,{}_r^*\!\leftarrow u_1.$$

La confluenza è una delle proprietà più importanti dei sistemi di riscrittura dei termini: afferma che se un termine raggiunge una forma normale, allora ogni sua possibile riduzione raggiunge quella forma normale.

La variante *di un passo* della confluenza viene detta *proprietà diamante* e ha la seguente forma: se $u_2\ {}_r\!\leftarrow t \rightarrow_r u_1$ e $u_1 \neq u_2$ allora esiste un r tale che $u_2 \rightarrow_r r\ {}_r\!\leftarrow u_1$. La proprietà diamante induce delle proprietà estremamente importanti.

Se una relazione $\rightarrow_r$ gode della proprietà diamante, allora:

1. $\rightarrow_r$ è confluente, ovvero $u_1\ {}^*_r\!\leftarrow t \rightarrow^*_r u_2$ implica $u_1 \rightarrow^*_r p\ {}^*_r\!\leftarrow u_2$ per qualche p;

2. Ogni termine t ha al massimo una forma normale, ovvero se $t \rightarrow^*_r u$ e $t \rightarrow^*_r p$, con u e p entrambi r-normali, allora $u = p$

3. Tutte le r-valutazioni con gli stessi termini iniziali e finali hanno la stessa lunghezza, ovvero se $d : t \rightarrow^*_r u$ e $d' : t \rightarrow^*_r u$ allora $|d| = |d'|$

4. t è debolmente r-normalizzante se e solo se è fortemente r-normalizzante.

Queste proprietà garantiscono che se esiste una r-valutazione normalizzante allora non esistono r-valutazioni divergenti, e che ogni possibile sequenza di riduzione ad una forma normale ha la stessa lunghezza. La proprietà diamante svolge un ruolo fondamentale nei capitoli successivi, in quanto consente, anche in presenza di strategie non-deterministiche, degli studi computazionali.

Due relazioni $\rightarrow_{r_1}$ e $\rightarrow_{r_2}$ *commutano fortemente* se $u_1\ {}_{r_2}\!\leftarrow t \rightarrow_{r_2} u_2$ implica $u_1 \rightarrow_{r_2} p\ {}_{r_2}\!\leftarrow u_2$ per qualche p. Se $\rightarrow_{r_1}$ e $\rightarrow_{r_2}$ commutano fortemente e godono della proprietà diamante, allora

1. $\rightarrow_r = \rightarrow_{r_1} \cup \rightarrow_{r_2}$ gode della proprietà diamante.

2. Tutte le r-valutazioni con gli stessi termini iniziali e finali hanno lo stesso numero di passi di ogni tipo, ovvero $d : t \rightarrow^*_r u$ e $d' : t \rightarrow^*_r u$ allora $|d|_{r_1} = |d'|_{r_1}$ e $|d|_{r_2} = |d'|_{r_2}$

La *commutazione forte* è una *forma forte di confluenza* ed implica *normalizzazione uniforme*, ovvero se esiste una sequenza normalizzante da t allora

non esistono sequenze divergenti da t, e la *proprietà di discesa casuale*, ovvero tutte le sequenze normalizzanti da t hanno la stessa lunghezza.

Lemma di Hindley-Rosen: Siano $\to_1$ e $\to_2$ due relazioni di riscrittura su un insieme $\mathcal{X}$. Se sono entrambe confluenti e commutano, ovvero se $t_1 \to_1^* u_1$ e $t \to_2^* u_2$ allora esiste un s tale che $u_1 \to_2^* s$ e $u_2 \to_1^* s$, allora $\to_1 \cup \to_2$ è confluente.

1.3 Sintassi del lambda-calcolo

La definizione dei lambda-termini in Backus-Naur Form è la seguente:

$$\text{Lambda-termini} \quad M, N ::= x \mid (\lambda x.M) \mid (MN)$$

Le seguenti regole forniscono una *definizione induttiva* che può essere applicata per costruire tutti i **lambda-termini** validi sintatticamente:

- La **variabile** x è un lambda-termine.
- Se t è un lambda-termine ed x è una variabile, allora $(\lambda x.t)$ è un lambda-termine - detto **astrazione**.
- Se t ed s sono lambda-termini, allora (ts) è un lambda-termine - detto **applicazione**.

Nel dettaglio:

1. x : **Variabile**, ovvero un carattere o una stringa, rappresenta un parametro;
2. $(\lambda x.M)$: **Lambda-astrazione**, ovvero la definizione di una funzione, che prende la *variabile legata* x come input e restituisce in output il corpo M;
3. (MN) : **Applicazione**, ovvero l'applicazione della funzione M all'argomento N

Alcune parentesi possono essere omesse in accordo alle seguenti regole:

- Le parentesi più esterne possono essere omesse, scriviamo $M\ N$ invece che $(M\ N)$.
- Le applicazioni vengono assunte associative a sinistra, scriviamo $M\ N\ P$ invece che $((M\ N)\ P)$.
- Il corpo di una astrazione si estende *quanto più a destra possibile*: $\lambda x.MN$ si legge come $\lambda x.(MN)$ e non $(\lambda x.M)N$.

1.4 Variabili libere e legate

L'occorrenza di una variabile x all'interno di un termine della forma $\lambda x.M$ viene detta **legata**. Il corrispondente operatore λx viene detto *binder*, mentre il sotto-termine M viene detto *scope* del binder. Una variabile che non è legata viene detta **libera**.

Inoltre, si noti come le nozioni di variabili libere e legate e di α-equivalenza sono notazioni matematica usuali. La variabile x risulta legata nei seguenti esempi:

$\lim_{x\to\infty} 2^x$
$\sum_{x=1}^{5} \frac{1}{x}$
int succ(int x) { return x+1;}

1.4.1 Definizione dell'insieme di variabili libere

L'insieme delle *variabili libere* di un lambda-termine M viene descritto con $FV(M)$ ed è definito ricorsivamente sulla struttura dei termini, come segue:

$$\begin{aligned} FV(x) &= \{x\} \\ FV(\lambda x.M) &= FV(M) \setminus \{x\} \\ FV(MN) &= FV(M) \cup FV(N) \end{aligned}$$

Ad esempio, $FV(\lambda x.xy) = \{y\}$, $FV(\lambda x.x) = \emptyset$ e $FV(\lambda x.y) = \{y\}$.

Un lambda-termine che non contiene variabili libere viene detto *chiuso*.

1.4.2 Definizione dell'insieme delle variabili legate

L'insieme delle *variabili legate* viene rappresentato come $BV(M)$ e viene definito ricorsivamente sulla struttura dei termini come:

$$\begin{aligned} BV(x) &= \emptyset \\ BV(\lambda x.M) &= BV(M) \cup \{x\} \\ BV(MN) &= BV(M) \cup BV(N) \end{aligned}$$

Ad esempio, $BV(\lambda x.xy) = \{x\}$, $BV(\lambda x.x) = \{x\}$ e $BV(\lambda x.y) = \emptyset$

1.5 α-conversione

L'α-conversione permette di rinominare le variabili legate. Ad esempio, possiamo α-convertire il termine $\lambda x.x$ in $\lambda y.y$.

I lambda-termini $\lambda x.x$ e $\lambda y.y$ rappresentano entrambi la stessa funzione, la *funzione identità*, in quanto differiscono soltanto per il nome delle loro variabili legate. Si utilizza la notazione $\lambda x.x =_\alpha \lambda y.y$ per indicare che questi due termini sono α-equivalenti.

Per dare una precisa definizione di α-equivalenza, bisogna innanzitutto definire cosa significhi *rinominare* una variabile in un termine. Siano x, y due variabili distinte e sia M un termine. La notazione $M\,[y/x]$ indica il risultato della rinominazione di x con y in M. La definizione formale della *rinominazione* è data da:

$$\begin{aligned} x\,[y/x] &\equiv y && \\ z\,[y/x] &\equiv z && \text{se } x \neq z \\ (MN)\,[y/x] &\equiv (M\,[y/x])(N\,[y/x]) && \\ (\lambda x.M)\,[y/x] &\equiv \lambda x.M && \\ (\lambda z.M)\,[y/x] &\equiv \lambda z.(M\,[y/x]) && \text{se } x \neq z \text{ e } y \notin FV(M) \end{aligned}$$

Il processo di rinominazione sostituisce ogni occorrenza di x con y. La variabile y utilizzata nel processo di rinominazione è detta **fresca**, ad indicare il fatto che questa non sia già in uso in M.

Si definisce **α-equivalenza** la relazione sui lambda-termini tale che, per ogni termine M e per ogni variabile y che non occorre libera in M,

$$\lambda x.M =_\alpha \lambda y.(M[y/x])$$

In particolare, la relazione deve soddisfare le seguenti regole:

$$\frac{}{M = M}\,(refl) \qquad \frac{M = M' \qquad N = N'}{MN = M'N'}\,(cong)$$

$$\frac{M = N}{N = M}\,(symm) \qquad \frac{M = M'}{\lambda x.M = \lambda x.M'}\,(\xi)$$

$$\frac{M = N \qquad N = P}{M = P}\,(trans) \qquad \frac{y \notin M}{\lambda x.M = \lambda y.(M\,[y/x])}\,(\alpha)$$

E' comunque possibile lavorare *a meno di α-sostituzione*, assumendo che le variabili legate vengano sempre rinominate, se necessario, per essere sempre ben distinte.

1.6 Sostituzione

Il cuore del lambda-calcolo sono le **sostituzioni**, che permettono di sostituire una variabile con un lambda-termine. Utilizziamo la notazione $M\{x \leftarrow N\}$ per indicare la sostituzione della variabile x con il lambda-termine N all'interno del lambda-termine M.

E' necessario notare che:

- E' necessario sostituire soltanto *variabili libere*. Ad esempio, la sostituzione $x(\lambda x.\lambda y.x)\{x \leftarrow N\}$ restituisce $N(\lambda x.\lambda y.x)$ e non $N(\lambda x.\lambda y.N)$.

- E' necessario evitare la cattura "involontaria" di variabili libere. Prendiamo in considerazione i termini $M \coloneqq \lambda x.yx$ ed $N \coloneqq \lambda z.xz$. La variabile x risulta essere libera in N e legata in M. Una sostituzione errata di N con y in M restituisce $M\{y \leftarrow N\} = (\lambda x.yx)\{y \leftarrow N\} = \lambda x.Nx = \lambda x.(\lambda z.xz)x$ Ma questo non è il risultato voluto, in quanto la variabile x era libera nel termine N, ma dopo il processo di sostituzione risulta legata. Questo perchè la variabile x che era legata in M non è la stessa variabile x che era libera in N. E' allora necessario *rinominare* la variabile legata *prima* di eseguire la sostituzione:

$$M\{y \leftarrow N\} = (\lambda x'.yx')\{y \leftarrow N\} = \lambda x'.Nx' = \lambda x'.(\lambda z.xz)x'$$

L'operazione di sostituzione obbliga talvolta a rinominare una variabile legata. In questo caso, si sceglie una variabile non attualmente in uso, definita *fresca*, dall'insieme infinito delle possibili variabili $\mathcal{V}$ e si rinomina la variabile legata.

Definizione. La *sostituzione-senza-cattura* del termine N con ogni occorrenza libera della variabile x nel termine M, in simboli $M\{x \leftarrow N\}$ viene definita ricorsivamente sulla struttura dei termini, come segue:

$$\begin{array}{lcll}
x\{x \leftarrow N\} & = & N & \\
y\{x \leftarrow N\} & = & y & \text{se } x \neq y \\
(MP)\{x \leftarrow N\} & = & (M\{x \leftarrow N\})(P\{x \leftarrow N\}) & \\
(\lambda x.M)\{x \leftarrow N\} & = & \lambda x.M & \\
(\lambda y.M)\{x \leftarrow N\} & = & \lambda y.(M\{x \leftarrow N\}) & \text{se } x \neq y \text{ e } y \notin FV(N) \\
(\lambda y.M)\{x \leftarrow N\} & = & \lambda y'.(M\,[y'/y]\,\{x \leftarrow N\}) & \text{se } x \neq y,\ y \in FV(N) \\
 & & & \text{e } y' \text{ fresca}
\end{array}$$

Da questo punto in poi i lambda-termini vengono sempre identificati *a meno di α-equivalenza.*

1.7 β-riduzione

Il processo con cui vengono valutati i lambda-termini, passando argomenti alle funzioni, viene chiamato **β-riduzione**. Un termine della for-

ma $(\lambda x.M)N$, che consiste dell'applicazione di un termine ad una lambda-astrazione, viene chiamato β-redesso. Il β-redesso **riduce** a $M\{x \leftarrow N\}$, dove quest'ultimo viene chiamato *ridotto*.

Il processo di valutazione del λ-calcolo si svolge trovando un lambda-termine che sia un redesso e sostituendolo con il suo ridotto. L'operazione di sostituzione va avanti quante volte necessario, potenzialmente infinite, o finchè non ci sono più redessi da ridurre. Un lambda-termine senza alcun β-redesso viene detto in *β-forma normale*.

Si osservi la riduzione del lambda-termine $(\lambda x.y)((\lambda z.zz)(\lambda w.w))$.
Il termine sottolineato è quello soggetto alla riduzione.

$$\begin{aligned}(\lambda x.y)\underline{((\lambda z.zz)(\lambda w.w))} &\rightarrow_\beta (\lambda x.y)\underline{((\lambda w.w)(\lambda w.w))} \\ &\rightarrow_\beta \underline{(\lambda x.y)(\lambda w.w)} \\ &\rightarrow_\beta y\end{aligned}$$

L'ultimo termine, y, non ha alcun redesso, e quindi è in forma normale. Si osservi come sia possibile ridurre il termine anche in maniera differente, scegliendo diversamente i redessi da ridurre:

$$\underline{(\lambda x.y)((\lambda z.zz)(\lambda w.w))} \rightarrow_\beta y$$

Si noti dagli esempi come il lambda-calcolo sia **non-deterministico**. All'interno di applicazioni reali, è necessario definire una strategia di riduzione, ovvero una relazione che stabilisca quale, tra più β-redessi, debba essere ridotto prima.

Non tutti i termini raggiungono una forma normale: alcuni vanno avanti nella valutazione all'infinito. Ad esempio:

$$\begin{aligned}(\lambda x.xx)(\lambda y.yy) &\rightarrow_\beta (\lambda y.yy)(\lambda y.yy) \\ &\rightarrow_\beta (\lambda y.yy)(\lambda y.yy) \\ &\rightarrow_\beta \ldots\end{aligned}$$

In particolare, il termine $(\lambda x.xx)(\lambda x.xx)$ viene detto il termine di loop, e denotato come Ω.

La definizione formale della β-riduzione è la seguente:

Definizione. Definiamo la *β-riduzione* come la più piccola relazione $\rightarrow_\beta$ sui termini che soddisfi:

$$(left)\ \frac{M \rightarrow M'}{MN \rightarrow_\beta M'N} \qquad (right)\ \frac{M \rightarrow_\beta M'}{NM \rightarrow_\beta NM'}$$

$$(\beta)\ \frac{}{(\lambda x.M)N \rightarrow_\beta M\{x \leftarrow N\}} \qquad (\xi)\ \frac{M \rightarrow_\beta M'}{\lambda x.M \rightarrow \lambda x.M'}$$

1.8 Strategie di riduzione

Nella sezione precedente si è mostrato come il lambda-calcolo sia non-deterministico, in quanto in presenza di più β-redessi possiamo scegliere liberamente quale ridurre.

Una **strategia di riduzione** specifica quale, tra tutti i redessi, debba essere ridotto.

Una strategia è **strong** se permette di ridurre dentro il corpo di una astrazione.

Una strategia è **debole** se non permette di ridurre dentro il corpo di una astrazione.

Le due principali strategie di riduzione sono la normal order e la applicative order.In particolare:

- **Normal order**: detta anche *leftmost-outermost*, procede applicando prima la funzione e valutando in seguito le sotto-espressioni. La strategia riduce sempre il β-redesso più a sinistra prima di ridurre le sotto-espressioni interne e quelle che seguono.

- **Applicative order**: detta anche *leftmost-innermost*, che prima valuta le sotto-espressioni e poi applica le funzioni. La strategia valuta da sinistra la sotto-espressione più interna, ed esegue una applicazione soltanto quando ogni sotto-espressione è stata ridotta, e non esistono più redessi tolta l'applicazione più in alto.

Un esempio delle due strategie in azione:

$$\begin{array}{lll} \text{Normal order} & \underline{(\lambda x.\lambda y.y)((\lambda z.zz)(\lambda z.zz))} & \to_\beta (\lambda y.y) \\ \text{Applicative order} & (\lambda x.\lambda y.y)\underline{((\lambda z.zz)(\lambda z.zz))} & \to_\beta (\lambda x.\lambda y.y)\underline{((\lambda z.zz)(\lambda z.zz))} \\ & & \to_\beta \ldots \end{array}$$

Si noti come le due diverse strategie di riduzione si comportino diversamente: mentre la *strategia normal order* termina sulla forma normale $(\lambda y.y)$, la *strategia applicative order* diverge, in quanto tenta di ridurre prima il β-redesso $((\lambda z.zz)(\lambda z.zz))$, ovvero il termine di loop Ω, che diverge.

La restrizione delle due strategie analizzate sopra al caso *debole*, in cui la riduzione dentro il corpo di una astrazione non è permessa, forniscono rispettivamente le strategie **Call-By-Name (CbN)** e **Call-By-Value (CbV)**.

Sia $I \coloneqq \lambda x.x$ il lambda-termine identità e sia $t \coloneqq (\lambda y.yy)(Iz)$ e si eseguano le relative riduzioni con Call-by-Name e Call-by-Value:
$(\lambda y.yy)(Iz) \to_{\beta_{CbN}} (Iz)(Iz) \to_{\beta_{CbN}} z(Iz) \to_{\beta_{CbN}} zz$
$(\lambda y.yy)(Iz) \to_{\beta_{CbV}} (\lambda y.yy)(z) \to_{\beta_{CbV}} zz.$

Sia poi $u \coloneqq (\lambda x.y)(II)$ e si considerino le relative riduzioni tramite Call-by-Name e Call-by-Value:
$(\lambda x.y)(II) \to_{CbN} y,$
$(\lambda x.y)(II) \to_{CbV} (\lambda x.y)I \to_{CbV} y.$

Se si considera la β-riduzione come unità di misura della complessità, ovvero assumendo che una singola β-riduzione rappresenti un passo computazionale, si osservi dagli esempi l'impossibilità di comparare le strategie e trovarne una più efficiente a priori.

1.9 Contesti

I *contesti* sono una nozione usata con costanza nella teoria del λ-calcolo.

Intuitivamente, un contesto è un termine nel quale un sotto-termine è stato rimosso e rimpiazzato da un *buco*, che viene indicato con il simbolo speciale $\langle\cdot\rangle$. Definiamone la grammatica:

$$\text{CONTESTI} \quad C, D \quad ::= \langle\cdot\rangle \mid \lambda x.C \mid Ct \mid tC$$

L'operazione base sui contesti è il *plugging*, che consiste nel *riempire il buco* $\langle\cdot\rangle$ in un contesto C con un termine u, producendo il termine $C\langle u\rangle$. Viene definito per induzione sui contesti, come segue:

$$\begin{array}{rclcrcl} \langle\cdot\rangle\langle u\rangle & := & u & & (\lambda x.C)\langle u\rangle & := & \lambda x.C\langle u\rangle \\ (Ct)\langle u\rangle & := & C\langle u\rangle t & & (tC)\langle u\rangle & := & tC\langle u\rangle \end{array}$$

L'operazione di plugging in un contesto è differente rispetto alla sostituzione. In particolare, nella definizione di plugging per una astrazione nulla viene detto rispetto potenziale occorrenze di x in u, che può quindi essere catturata. Si noti la differenza tra $(\lambda z.y\langle\cdot\rangle)\langle zz\rangle = \lambda z.y(zz)$, mentre $(\lambda z.yx)\{x \leftarrow zz\} = \lambda z'.y(zz)$, in quanto la sostituzione esegue un'operazione di α-rinominazione delle variabili legate per evitare la cattura delle variabili libere, mentre il plugging no.

A questo punto la β-riduzione viene definita con le seguenti regole deduttive:

$$(\beta)\ \frac{}{(\lambda x.M)N \to_\beta M\{x \leftarrow N\}} \qquad (\gamma)\ \frac{M \to_\beta N}{C\,[M] \to_\beta C\,[N]}$$

$$(C_l)\ \frac{M \to_\beta N}{MP \to_\beta NP} \qquad (C_r)\ \frac{M \to_\beta N}{PM \to_\beta PN}$$

$$(C_\lambda)\ \frac{M \to_\beta N}{\lambda x.M \to_\beta \lambda x.N}$$

I contesti sono molto utili per specificare le strategie di valutazione. Ad esempio, il seguente contesto con le relative regole deduttive:

$$D ::= \langle\cdot\rangle \mid x \mid tD \mid Dt$$

$$(\beta_w)\ \frac{}{(\lambda x.M)N \to_{\beta_w} M\{x \leftarrow N\}} \qquad (\gamma_w)\ \frac{M \to_{\beta_w} N}{D\,[M] \to_{\beta_w} D\,[N]}$$

$$(D_l)\ \frac{}{MP \to_{\beta_w} NP} \qquad (D_r)\ \frac{M \to_{\beta_w}}{PM \to_{\beta_w} PN}$$

definisce una strategia di valutazione *debole*, dove non è permesso ridurre sotto astrazione. In questo caso si parla di contesto di valutazione.

Lo studio sull'equivalenza dei programmi viene analizzato con i contesti.

Dato un linguaggio di termini t con associata una nozione di contesti C e una semantica operazionale $\to$, definiamo il *preordine contestuale* $\precsim_C$ e la *stabilità contestuale* $\backsimeq_C$ come segue:

- $t \precsim_C t'$ se $C\,\langle t\rangle$ debolmente $\to$-normalizzante implica $C\,\langle t'\rangle$ debolmente $\to$-normalizzante per ogni contesto C tale che $C\,\langle t\rangle$ e $C\,\langle t'\rangle$ siano termini chiusi.

- $\backsimeq_C$ è la relazione di equivalenza indotta da $\precsim_C$: $t' \backsimeq_C t'$ se e solo se $t \precsim_C t'$ e $t' \precsim_C t$.

1.10 Semantiche del lambda-calcolo

Il lambda-calcolo può essere analizzato da diversi punti di vista[3]:

- **Operazionale:** Definisce il processo di valutazione di un programma e ne studia le *proprietà*, come confluenza, normalizzazione e strategie di riduzione.

- **Denotazionale:** un modello denotazionale è formato da una famiglia di oggetti matematici, come relazioni e domini, nei quali i lambda-termini vengono mappati, in modo che il processo di valutazione sui termini corrisponda ad uguaglianza sugli oggetti.

- **Equazionale:** è lo studio delle tante diverse possibili nozioni di *equivalenza dei programmi* tra termini, detto anche *teoria equazionale*, come ad esempio l'equivalenza contestuale. Le semantiche equazionali possono essere studiate sia in maniera equazionale che denotazionale.
- **Logico:** grazie alla *corrispondenza di Curry-Howard* tra λ-calcolo e i sistemi di dimostrazione, i termini del λ-calcolo possono essere viste come frammenti di dimostrazione della *logica intuizionista*, in cui le forme logiche hanno il ruolo di *tipi* dei programmi [4].

1.11 Codificare dati nel lambda-calcolo

E' possibile codificare dati all'interno del lambda-calcolo, ad esempio booleani o numeri naturali, e definire poi programmi che svolgono operazioni sui dati.

1.11.1 Booleani

I cosidetti *booleani di Church*, che codificano i valori "vero" e "falso", sono definiti come:

$$\text{TRUE} \coloneqq \lambda x.\lambda y.x$$
$$\text{FALSE} \coloneqq \lambda x.\lambda y.y$$

Con queste due definizioni, si può codificare l'operatore "NOT":

$$\text{NOT} \coloneqq \lambda z.z\ \text{FALSE}\ \text{TRUE}$$

Un esempio di esecuzione della funzione NOT:

$$\begin{aligned} \underline{\text{NOT}}\ \text{TRUE} &= \underline{(\lambda z.z\ \text{FALSE}\ \text{TRUE})}\mathit{TRUE} \\ &\to_\beta \underline{\ \text{TRUE}}\ \text{FALSE}\ \text{TRUE} \\ &= \underline{(\lambda x.\lambda y.x)}\ \text{FALSE}\ \text{TRUE} \\ &\to_\beta \text{FALSE} \end{aligned}$$

Possiamo poi definire gli operatori AND e NOT:

$$\text{AND} \coloneqq \lambda w.\lambda z.wzw$$
$$\text{OR} \coloneqq \lambda w.\lambda z.wwz$$

1.11.2 Numeri naturali

Definiamo i numerali di Church

$$\begin{aligned} 0 &\coloneqq \lambda f.\lambda x.x \\ 1 &\coloneqq \lambda f.\lambda x.fx \\ 2 &\coloneqq \lambda f.\lambda x.f(fx) \\ 3 &\coloneqq \lambda f.\lambda x.f(f(fx)) \\ &\dots \end{aligned}$$

Capitolo 2

Il lambda-calcolo con Call-by-Value

Sebbene la strategia Call-by-Name sia quella universalmente più utilizzata, e quando si nomina in maniera generale il lambda-calcolo ci si riferisce allo Strong Call-by-Name, questo non viene quasi mai utilizzato nell'ambito di applicazioni reali.

All'interno del capitolo vengono introdotte due diverse presentazioni del lambda-calcolo che implementano una strategia Call-by-Value.

Il λ-calcolo con la strategia Call-by-Value costituisce il cuore dei linguaggi di programmazione funzionale come OCaml e viene utilizzato, assieme ad altre strategie di valutazione, anche all'interno dei dimostratori di teoremi come Coq.

2.1 Il λ_v-calcolo di Plotkin

Uno dei primi a studiare la strategia fu Plotkin[1], che introdusse il **Call-by-Value λ-calcolo**, qui definito λ_v, come restrizione del λ-calcolo in cui i β-redessi possono essere **ridotti** soltanto quando il loro argomento è un valore, e i valori vengono definiti come variabili e astrazioni.

Presentiamo adesso una versione leggermente differente, per dettagli non-essenziali, rispetto al λ_v-calcolo fornito da Plotkin.

TERMINI	t, u, s, q	$::= x \mid \lambda x.t \mid tu$
VALORI	v, v'	$::= \lambda x.t$
CONTESTI DI VALUTAZIONE	E	$::= \langle\cdot\rangle \mid tE \mid Et$
REGOLA TOP-LEVEL		$(\lambda x.t)v \mapsto_{\beta_v} t\{x \leftarrow v\}$
CHIUSURA CONTESTUALE		$E\langle t\rangle \to_{\beta_v} E\langle u\rangle$ SE $t \mapsto_{\beta_v} u$

Tuttavia è ben noto [5] come λ_v funzioni correttamente soltanto qualora i termini siano *chiusi*, ovvero non contengano variabili libere, e la valutazione sia *debole*, ovvero non permetta di ridurre nel corpo di una astrazione.

Infatti, non appena vengano permessi dei termini *aperti*, ci si ritrova con delle **false forme normali** [6], ovvero termini *contestualmente equivalenti* al termine di loop $\Omega := (\lambda x.xx)(\lambda x.xx)$, ma che risultano normali. In particolare, esistono delle β-forme normali aperte che non sono valori, ad esempio xx. Come conseguenza, esistono dei β-redessi bloccati, ad esempio $(\lambda y.t)(xx)$, ovvero β-redessi che non verranno mai ridotti in quanto l'argomento *non è un valore*, e mai lo diventerà. Il problema legato alle variabili aperte diventa ancora più importante quando si considera una valutazione *forte*, ovvero dove è permesso valutare i corpi delle astrazioni, in quanto si è costretti a dover lavorare con *termini aperti localmente*. Consideriamo ad esempio il termine $s := \lambda x.((\lambda y.t)(xx))$, e notiamo come la variabile x risulti localmente aperta rispetto al sotto-termine $(\lambda y.t)(xx)$.
I β-redessi bloccati impediscono la creazione di altri redessi, e rappresentano delle *β-forme normali premature*.

Le considerazioni fatte sopra sono molto importanti, in quanto il problema arrivare ad influire sulla terminazione. Sia $\delta := \lambda x.xx$ la funzione duplicatore, la cui auto-applicazione diverge. Consideriamo poi i termini $t := (\lambda y.\delta)(zz)\delta$ e $u := \delta((\lambda y.\delta)(zz))$.

All'interno del λ_v, sia t che u sono delle false forme normali, in quanto hanno un β-redesso bloccato che impedisce alla valutazione di andare avanti,

mentre ci si aspetterebbe che i termini si comportino come il termine divergente Ω. Risulta impossibile ridurre il termine t attraverso il λ_v-calcolo, in quanto il primo argomento zz di $(\lambda y.\delta)$ non è un valore. Quindi il termine è una forma normale nel λ_v, mentre da un punto di vista semantico il termine è equivalente ad Ω, e dovrebbe quindi divergere. Invece, nel termine u, è la valutazione dell'argomento che risulta bloccata, sempre perchè zz non è un valore, e questo impedisce la creazione di nuovi redessi.

2.2 Il fireball-calcolo λ_{fire}

Uno dei primi tentativi di rimozione dei β-redessi bloccati è il *fireball-calcolo* λ_{fire}, introdotto da Paolini e Della Ronca in BIB

L'idea è quella di generalizzare i valori del CBV λ-calcolo ai cosidetti *fireball*, da *fire-able*, con l'aggiunta dei *termini inerti*, che in particolare contengono le variabili. Fireballs e termini inerti vengono definiti per mutua induzione:

TERMINI	t, u, s, q	$::= x \mid \lambda x.t \mid tu$
VALORI	v, v'	$::= \lambda x.t$
CONTESTI DI VALUTAZIONE	E	$::= \langle\cdot\rangle \mid tE \mid Et$
FIREBALLS	f, f', f''	$::= v \mid i$
TERMINI INERTI	i, i', i''	$::= xf_1 \dots f_n$,con $n \geq 0$
REGOLA VALORI TOP LEVEL		$(\lambda x.t)v \mapsto_{\beta_v} t\{x \leftarrow v\}$
REGOLA INERTI TOP LEVEL		$(\lambda x.t)i \mapsto_{\beta_i} t\{x \leftarrow i\}$
CHIUSURA CONTESTUALE		$E\langle t\rangle \to_{\beta_i} E\langle u\rangle$ SE $t \mapsto_{\beta_i} u$

Ad esempio, $\lambda x.t$ è un fireball come valore, mentre x, xy, $y(\lambda x.x)$ sono fireball come termini inerti.

Il vantaggio principale dei termini inerti è che sono aperti, normali e quando vengono pluggati in un contesto non possono creare un redesso.

Dinamicamente, i β-redessi possono essere ridotti anche se l'argomento è un termine inerte, attraverso la regola $\to_{\beta_i}$. La valutazione è debole, in quanto i contesti di valutazione non riducono sotto astrazione.

Osserviamo di nuovo i termini t ed u e osserviamo come divergano correttamente:

$$t \coloneqq \underline{(\lambda y.\delta)(zz)}\delta \to_{\beta_i} \delta\{y \leftarrow zz\}\delta = \underline{\delta\delta} \to_{\beta_v} \underline{\delta\delta} \to_{\beta_v} \dots$$
$$u \coloneqq \delta(\underline{(\lambda y.\delta)(zz)}) \to_{\beta_i} \delta(\delta\{y \leftarrow zz\}) = \underline{\delta\delta} \to_{\beta_v} \underline{\delta\delta} \to_{\beta_v} \dots$$

Proprietà del fireball-calcolo

1. La riduzione è *non-deterministica* ma gode della proprietà diamante.
2. Un termine è una forma normale se e solo se è un fireball.

2.2.1 Problemi del fireball-calcolo

Anche il λ_{fire}-calcolo presenta dei problemi semantici, in quanto permette di cancellare e duplicare termini inerti[7], in particolare la cancellazione è particolarmente problematica.

Si consideri il termine $t \coloneqq (\lambda x.I)(yy)$, dove $I \coloneqq \lambda z.z$ è la funzione identità. Notiamo che $t \to_{\beta_i} I$, in quanto yy è un termine inerte.

Si consideri poi il contesto $C \coloneqq (\lambda y.\, \langle\cdot\rangle)\delta$ e si verifichi che

$$C\langle I\rangle = (\lambda y.I)\delta \to_{\beta_v} I, \text{ mentre}$$
$$C\langle t\rangle = (\lambda y.((\lambda x.I)(yy)))\delta \to_{\beta_v} (\lambda x.I)(\delta\delta) \to_{\beta_v} (\lambda x.I)(\delta\delta) \to_{\beta_v} \dots$$

Ovvero, $t \to_{\beta_i} I$ ma $t \not\simeq_C I$, in quanto $C\langle t\rangle$ diverge mentre $C\langle I\rangle$ termina. Il fireball-calcolo viola quindi la *stabilità contestuale*.

2.2.2 Problemi del fireball-calcolo forte

In [8], viene presentata una versione deterministica del fireball-calcolo, che procede da destra-verso-sinistra, estesa per effettuare valutazioni sotto astrazione.

A livello aperto, il problema è soltanto semantico, ma questo induce un problema di terminazione a livello forte. Il fireball-calcolo forte infatti soffre di un fenomeno simile a quello delle false forme normali, sebbene non ci

siano dei redessi bloccati. Si consideri il termine $w \coloneqq (\lambda x.I)(y(\lambda z.\Omega))$. Nel fireball-calcolo si ha $w \to_{\beta_i} I$, in quanto $y(\lambda z.\Omega)$ è un termine inerte, ovvero u termina su una forma normale forte, I. Invece, u è *semanticamente divergente* a livello forte. Intuitivamente, il sottotermine inerte $y(\lambda z.\Omega)$ andrebbe in qualche modo mantenuto, invece di essere cancellato, e valutato in maniera forte, che porterebbe a divergenza vista la valutazione di Ω sotto astrazione.

Si noti che il fireball-calcolo forte violi nuovamente la stabilità contestuale. Si è osservato come $w \to_{\beta_i} I$ in quanto $y(\lambda z.\Omega)$ è un termine inerte. Si consideri adesso il contesto $C \coloneqq (\lambda y.\langle\cdot\rangle)(\lambda u.uI)$:

$$\begin{array}{lllll} C\langle u\rangle & = & (\lambda y.((\lambda x.I)(y(\lambda z.\Omega))))(\lambda u.uI) & \to_{\beta_v} & (\lambda x.I)((\lambda u.uI)(\lambda z.\Omega)) \\ & \to_{\beta_v} & (\lambda x.I)((\lambda z.\Omega)I) & \to_{\beta_v} & (\lambda x.I)\Omega \to_{\beta_v} \dots \\ C\langle I\rangle & = & (\lambda y.I)(\lambda u.uI) & \to_{\beta_v} & I \end{array}$$

Ovvero, $w \to_{\beta_i} I$ ma $w \not\leftarrow_C I$ in quanto $C\langle w\rangle$ diverge mentre $C\langle I\rangle$ termina.

Capitolo 3

Il Value Substitution Calculus

In questo capitolo viene definito il Value Substitution Calculus, abbreviato come VSC, introdotto in [5] e che ambisce a risolvere i problemi semantici introdotti nella sezione precedente. Il calcolo viene presentato assieme ad una strategia, detta esterna, che è stata provata essere *normalizzante* e per cui vale la *proprietà diamante*. Questo ci permette, nonostante la strategia sia non-deterministica, di poterne effettuare studi semantici, viste le proprietà indotte dalla proprietà diamante.

Prima di definire formalmente il Value Substitution Calculus [5], che viene abbreviato VSC, introduciamo formalmente le *sostituzioni esplicite*.

3.1 Sostituzioni esplicite

Si introduce una let-espressione **let** $x = u$ **in** t come una più compatta *sostituzione esplicita* $t\,[x \leftarrow u]$, abbreviata ES, dall'inglese *explicit substitution*. La notazione utilizzata non fissa un ordine di valutazione tra t e u. La definizione *lega* la variabile x in t. Come conseguenza, gli insiemi delle variabili libere e legate vengono espansi con le seguenti regole:

$$FV(t\,[x \leftarrow u]) = FV(t) \setminus \{x\} \cup FV(u)$$
$$BV(t\,[x \leftarrow u]) = BV(t) \cup \{x\} \cup BV(u)$$

L'introduzione delle sostituzioni esplicite rende necessario modificare il meccanismo di valutazione, e questo viene fatto in quattro passi [9]:

1. **Sintassi:** si espande la sintassi dei termini con il costruttore delle sostituzioni esplicite $t\,[x \leftarrow u]$.

2. **Decomporre β:** si modifica la β-riduzione, per trasformare il redesso $(\lambda x.t)u$ in $t\,[x \leftarrow u]$, che è una annotazione per *ritardare* la sostituzione $t\{x \leftarrow u\}$ e quindi mantiene u *in condivisione.*

3. **Riscrivere le ES:** si introduce una regola di riscrittura per $t\,[x \leftarrow u]$.

4. **Fissare i contesti di valutazione:** infine si specifica in quale contesti di valutazione è possibile applicare le regole. La scelta delle regole e dei contesti di valutazione definisce la *strategia di valutazione.*

Uno dei modi più semplici di decomporre la β-riduzione è come segue:

$$(\lambda x.t)s \to_\beta t\,[x \leftarrow s] \to_{ES} t\{x \leftarrow s\}$$

Si consideri il termine $w := (\lambda x.t)\,[y \leftarrow s]\,u$ e si noti come la sostituzione esplicita $[y \leftarrow s]$ stia bloccando la possibilità di ridurre il β-redesso $(\lambda x.t)u$. In genere, il calcolo viene esteso con una regola come $t\,[y \leftarrow s]\,u \to_@ (tu)\,[y \leftarrow s]$.

Una alternativa più semplice consiste nel generalizzare la nozione di β-redesso, permettendo alle astrazione e ai loro argomenti di interagire *a distanza*, ovvero anche se ci sono ES nel mezzo.

$$(\lambda x.t)\,[y_1 \leftarrow s_1] \ldots [y_k \leftarrow s_k]\,u \to_{dB} t\,[x \leftarrow u]\,[y_1 \leftarrow s_1] \ldots [y_k \leftarrow s_k]$$

La regola può poi essere compattata utilizzando i *contesti.*

Definiamo una lista di sostituzione e la relativa regola di riduzione come:

$$\begin{array}{lll} L & ::= & \langle\cdot\rangle \mid L\,[x \leftarrow t] \\ L\,\langle\lambda x.t\rangle\,u & \to_{dB} & L\,\langle t\,[x \leftarrow u]\rangle \end{array}$$

3.2 Definizione del Value Substitution Calcolo

Il VSC è un λ-calcolo con Call-By-Value esteso con *let-espressioni*.

La grammatica segue:

VALORI	$v ::=$	$\lambda x.t$
TERMINI	$t, u, s ::=$	$x \mid v \mid tu \mid t\,[x \leftarrow u]$
CONTESTI	$C ::=$	$\langle\cdot\rangle \mid Ct \mid tC \mid \lambda x.C \mid C\,[x \leftarrow t] \mid t\,[x \leftarrow C]$
LISTE DI SOSTITUZIONE	$L ::=$	$\langle\cdot\rangle \mid L\,[x \leftarrow t]$

Le regole di riduzione del VSC sono leggermente inusuali, in quanto utilizzano i contesti sia per ridurre i redessi all'interno dei sotto-termini, che è una procedura standard, sia per definire i redessi stessi, che è meno standard. Questo tipo di regola viene detto *a distanza*.

Esistono due regole di riscrittura, i cui nomi derivano dalla corrispondenza con le *proof-net* [4]. I loro casi radice, prima della chiusura contestuale, sono:

REGOLA RADICE MOLTIPLICATIVA	$L\,\langle\lambda x.t\rangle\,u \mapsto_m L\,\langle t\,[x \leftarrow u]\rangle$
REGOLA RADICE ESPONENZIALE	$t\,[x \leftarrow L\,\langle v\rangle] \mapsto_e L\,\langle t\{x \leftarrow v\}\rangle$
CHIUSURA CONTESTUALE	$\dfrac{t \mapsto_a t'}{C\,\langle t\rangle \to_a C\,\langle t'\rangle}\,(a \in \{m, e\})$
NOTAZIONE	$\to_{vsc} := \to_m \cup \to_e$

Entrambe le regole lavorano a distanza, in quanto utilizzano i contesti anche nella definizione delle regole radice, prima della chiusura contestuale. In particolare, si basano sulle liste di sostituzione esplicite L. Le regole radice vengono $\mapsto_m$ e $\mapsto_e$ vengono assunte senza cattura rispettivamente dei termini u e t, quindi le variabili libere di questi termini vengono eventualmente α-rinominati.

Esempi: $(\lambda x.t)\,[y \leftarrow u]\,p \to_m t\,[x \leftarrow p]\,[x \leftarrow u]$, mentre $t\,[x \leftarrow v\,[y \leftarrow u]] \to_e t\{x \leftarrow v\}\,[y \leftarrow u]$, Nel primo esempio, la distanza sulla regola $\to_m$ ci consente di ridurre il β-redesso $(\lambda x.t)p$ nonostante la presenza della ES presente tra i due termini $(\lambda x.t)$ e p. Nel secondo esempio invece la distanza su $\to_e$ permette la riduzione nonostante le ES nidificate.

Un esempio con α-rinominazione è $(\lambda x.y)\,[y \leftarrow t]\,y \mapsto_m z\,[x \leftarrow y]\,[z \leftarrow t]$.

La risoluzione al problema dei β-redessi bloccati consiste nel permettere di ridurre un β-redesso sempre, anche qualora l'argomento non sia un valore, attraverso la regola moltiplicativa $\mapsto_m$, che non sostituisce direttamente l'argomento ma lo mette in una ES, in attesa di essere definitivamente sostituito. La sostituzione vera e propria, e quindi la limitazione propria della strategia CbV, viene effettuata dalla regola esponenziale $\mapsto_e$, che richiede che l'argomento della ES sia un valore.

3.3 VSC aperto

Si definisce prima il segmento aperto del VSC, dove la riduzione sotto astrazione non è permessa, e i termini sono *possibilmente* aperti. Questo frammento ha una particolare descrizione induttiva delle forme normali, detti *fireball* e *termini inerti*, in quanto sono una versione modificata delle rispettive nozioni nel fireball-calcolo.

Contesti aperti, fireball, termini aperti e regole di riduzione aperta sono definite come segue:

CONTESTI APERTI	O	$::= \langle\cdot\rangle \mid Ot \mid tO \mid O\,[x \leftarrow t] \mid t\,[x \leftarrow O]$
TERMINI INERTI	i, i'	$::= x \mid if \mid i\,[x \leftarrow i']$
FIREBALL	f	$::= v \mid i \mid f\,[x \leftarrow i]$
CHIUSURA CONTESTUALE :		$\dfrac{t \mapsto_a t'}{O\langle t\rangle \leftarrow_{oa} O\langle t'\rangle}\,(a \in \{m, e\})$

La grammatica permette di avere ES che contengono termini inerti attorno a astrazioni e applicazioni: $(\lambda x.y)\,[y \leftarrow zz]$ è un fireball e $x\,[x \leftarrow y(\lambda x.x)]\,y$ è un termine inerte.

Si noti come il λ_v-calcolo di Plotkin sia correttamente simulato all'interno del VSC, in quanto $(\lambda x.t)v \mapsto_m t\,[x \leftarrow v] \mapsto_e t\{x \leftarrow v\}$, e $(\lambda x.t)v \mapsto_{\beta_v} t\{x \leftarrow v\}$

Non è possibile simulare il VSC nel λ_v, in quanto VSC è una *estensione* del λ_v di Plotkin. Si considerino di nuovo i termini $t := (\lambda y.\delta)(zz)\delta$ e

$u \coloneqq \delta(\lambda y.\delta)(zz)$, che risultavano essere false forme normali all'interno di λ_v, ma divergono correttamente nel VSC:

$$t \to_{om} \delta\,[y \leftarrow zz]\,\delta \to_{om} (yy)\,[y \leftarrow \delta]\,[y \leftarrow zz] \to_{oe} (\delta\delta)\,[y \leftarrow zz] \to_{om} \dots$$
$$u \to_{om} \delta\delta\,[y \leftarrow zz] \to_{om} (yy)\,[y \leftarrow \delta\,[y \leftarrow zz]] \to_{oe} (\delta\delta)\,[y \leftarrow zz] \to_{om} \dots$$

Si noti come la divergenza di t utilizzi crucialmente la distanza su $\to_m$, nel secondo passo, mentre la divergenza di u utilizza crucialmente la distanza su $\to_e$.

Proprietà della riduzione aperta

1. La riduzione aperta $\to_o$ gode della proprietà diamante.
2. Un termine è o-normale se e solo se è una fireball.

3.4 VSC Forte

La sezione forte del VSC viene ottenuta permettendo le regole di riscrittura ovunque, ovvero includendo anche la possibilità di ridurre nei corpi delle astrazioni, attraverso una chiusura tramite contesti generali.

TERMINI INERTI FORTI	i_s	$::= x \mid i_s f_s \mid i_s\,[x \leftarrow i'_s]$
FIREBALL FORTI	f_s	$::= i_s \mid v_s \mid f_s\,[x \leftarrow i_s]$
VALORI FORTI	v_s	$::= \lambda x.f_s$
REGOLE DI RISCRITTURA FORTE		$\dfrac{t \mapsto_a t'}{C\langle t\rangle \to_a C\langle t'\rangle}\,(a \in \{m, e\})$
RIDUZIONE FORTE		$\to_{vsc} := \to_m \cup \to_e$

A differenza della sezione aperta, $\to_{vsc}$ non gode della proprietà diamante:

Si valutino tutte le possibili vsc-valutazioni del termine $(xx)\,[x \leftarrow \lambda y.II]$, dove $I \coloneqq \lambda z.z$ e si osservi come la riduzione forte non goda della proprietà diamante: $(xx)\,[x \leftarrow \lambda y.II] \to_e (\lambda y.II)(\lambda y.II) \to_m II\,[y \leftarrow \lambda y.II] \to_e II \to_m I\,[z \leftarrow I] \to_e I$, mentre $(xx)\,[x \leftarrow \lambda y.II] \to_m (xx)\,[x \leftarrow \lambda y.(z\,[z \leftarrow I])] \to_e (xx)\,[x \leftarrow \lambda y.I] \to_e (\lambda y.I)(\lambda y.I) \to_m I\,[y \leftarrow \lambda y.I] \to_e I$.

Proprietà della riduzione forte

1. La riduzione $\to_{vsc}$ è confluente.

2. Un termine è *vsc*-normale se e solo se è un fireball forte.

La nozione di termine inerte forte e di fireball forti sono una generalizzazione dei termini inerti e dei fireball, rispettivamente, iterando le costruzioni sotto astrazioni.

3.5 La strategia esterna

Si definisce adesso la *strategia esterna* [10], che ha un ruolo analogo alla strategia *leftmost-outermost* del λ-calcolo. Mentre quest'ultima fissa un preciso ordine di valutazione, da sinistra verso destra, nella strategia esterna non viene imposto un ordine di valutazione. Si mantiene però l'aspetto *outermost* della strategia, in quanto la strategia esterna riduce soltanto redessi che non possono essere duplicati o cancellati da altri redessi.

Si rende necessario definire prima i *termini rigidi*, variazione dei termini inerti dove l'argomento della variabile di testa può essere un qualsiasi termine.

$$\text{TERMINI RIGIDI} \quad r, r' \quad ::= x \mid rt \mid r\,[x \leftarrow r']$$

Poi bisogna definire i contesti di valutazione per la strategia esterna $\to_x$, che viene definita al di sopra della valutazione aperta. Il caso base è dato dalle regole di riscrittura aperta, che vengono poi chiuse tramite i *contesti esterni*, definiti per mutua induzione con i *contesti rigidi*:

$$\begin{array}{lll}
\text{CONTESTI ESTERNI} & X ::= & \langle\cdot\rangle \mid \lambda x.X \mid t\,[x \leftarrow R] \mid X\,[x \leftarrow r] \mid R \\
\text{CONTESTI RIGIDI} & R ::= & rX \mid Rt \mid R\,[x \leftarrow r] \mid r\,[x \leftarrow R] \\
\text{REGOLA ESTERNA} & & \dfrac{t \to_{oa} t'}{X\langle t\rangle \to_{xa} X\langle t'\rangle}\,(a \in \{m, e\}) \\
\text{RIDUZIONE ESTERNA} & & \to_x := \to_{xm} \cup \to_{xe}
\end{array}$$

Chiaramente, $\to_x \subsetneq \to_{vsc}$. La strategia diverge sul termine $y(\lambda z.\Omega)$, in quanto $y\lambda z.\langle\cdot\rangle$ è un contesto rigido, quindi esterno, e normalizza il termine

potenzialmente divergente $(\lambda x.y)(\lambda z.\Omega) \to_x^* y$, in quanto i valori possono essere cancellati anche se divergono sotto astrazione.

Inoltre, la strategia diverge sul termine $w \coloneqq (\lambda x.I)(y(\lambda z.\Omega))$, che invece converge nel λ_{fire}-calcolo forte, mostrando che *i due calcoli forti hanno differenti nozioni di terminazione*. Con la strategia esterna, $w \to_{xm} I\,[x \leftarrow y(\lambda z.\Omega)]$ e in seguito la strategia esterna diverge in quanto $y(\lambda z.\Omega)$ non può essere cancellato, e Ω occorre in un contesto di valutazione esterna $I\,[x \leftarrow y(\lambda z.\,\langle\cdot\rangle)]$.

Le grammatiche dei contesti esterni e dei contesti rigidi permettono soltanto la valutazione di astrazioni non applicate, ovvero $(\lambda x.(II))v \not\to_x (\lambda x.(y\,[y \leftarrow I]))v$. Non viene fissato un ordine di valutazione in quanto i contesti aperti non fissano un ordine di valutazione, abbiamo infatti:

$$(II)(y\,[y \leftarrow I])_{xm} \leftarrow (II)(II) \to_{xm} (y\,[y \leftarrow I])(II)$$

La strategia è quindi *non-deterministica*, ma questo è irrilevante in quanto gode della proprietà diamante.

Proprietà della riduzione esterna

1. La riduzione esterna $\to_x$ gode della proprietà diamante.

2. Completezza: sia t un *vsc*-termine. t è x-normale se e solo se t è *vsc*-normale.

Bibliografia

[1] G.D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, 1975.

[2] Luca Paolini and Simona Ronchi Della Rocca. Call-by-value solvability. *RAIRO - Theoretical Informatics and Applications*, 1999.

[3] Beniamino Accattoli. A fresh look at the lambda-calculus (invited talk). In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, 2019.

[4] Beniamino Accattoli. Proof nets and the call-by-value λ-calculus. *Theor. Comput. Sci.*, 2015.

[5] Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. In *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, 2012.

[6] Beniamino Accattoli and Giulio Guerrieri. Open call-by-value. In *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, 2016.

[7] Beniamino Accattoli and Giulio Guerrieri. Types of fireballs. In *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, 2018.

[8] Małgorzata Biernacka, Dariusz Biernacki, Witold Charatonik, and Tomasz Drab. An abstract machine for strong call by value, 2020.

[9] Beniamino Accattoli. Sharing a perspective on the λ-calculus. In *Proceedings of the 2023 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2023, Cascais, Portugal, October 25-27, 2023*, 2023.

[10] Beniamino Accattoli, Andrea Condoluci, and Claudio Sacerdoti Coen. Strong call-by-value is reasonable, implosively. *CoRR*, 2021.