UNIVERSITÀ DEGLI STUDI ROMA TRE
FACOLTÀ DI SCIENZE M.F.N.

Sintesi della Tesi di Laurea in Matematica
presentata da

Maria Doris Di Marco

Un'applicazione della Geometria dell'Interazione al calcolo distribuito

Relatore

Prof. Marco Pedicini

Il Candidato

Il Relatore

ANNO ACCADEMICO 2002 - 2003
ottobre 2003

Classificazione AMS : 03B70, 03F05, 03B40, 68N18, 03B60.
Parole Chiave: logica lineare, programmazione funzionale, lambda-calcolo.

Ogni problema matematico di natura effettiva, ovvero che richiede nella sua soluzione la stesura di un algoritmo, presuppone una *rappresentazione* degli oggetti da studiare. Ad esempio quando scriviamo un intero nella sua rappresentazione decimale utilizziamo un insieme finito di cifre per rappresentare tutti i possibili elementi di $\mathbb{N}$. Generalizzando l'esempio dato si osserva che il metodo è quello di restringersi ad una rappresentazione degli oggetti come parole (successioni finite di simboli) su un alfabeto fissato, di utilizzare cioè un *linguaggio formale*. Per descrivere un sistema formale distinguiamo in esso diversi aspetti:

1. la *sintassi*, che è l'insieme delle regole di costruzione degli oggetti del sistema;

2. il *calcolo*, ovvero un algoritmo caratterizzato dall'applicazione successiva di regole che trasformano gli oggetti e, sotto determinate condizioni, conducono ad un risultato finale;

3. la *semantica*, che è una rappresentazione del sistema formale in un modello matematico concreto; essa può essere una teoria equazionale o una struttura matematica particolare. Il tipo di semantica che consideriamo nel seguito è una semantica operazionale, che rappresenta la computazione associata agli oggetti del sistema mediante l'evoluzione dello stato di una macchina astratta. Il passaggio da uno stato all'altro durante la computazione è espresso mediante una funzione definita sull'insieme degli stati.

Il sistema formale inteso come al punto 2 è un candidato a diventare *modello di calcolo*. I modelli di calcolo classici sono le funzioni ricorsive, le Macchine di Turing e il λ-calcolo puro. Essi hanno le stesse potenzialità espressive: un problema che può essere rappresentato e calcolato in uno di essi potrà essere rappresentato e calcolato negli altri due ottenendo lo stesso esito della computazione. Abbiamo concentrato l'attenzione sul λ-calcolo puro, che è stato introdotto da Church negli anni '30, perché viene utilizzato nella pra-

tica informatica per lo sviluppo dei linguaggi di programmazione funzionale. Il sistema formale inteso come al punto 3 ci permette di utilizzare strumenti derivati da teorie matematiche esterne al sistema formale. Per questo scopo abbiamo scelto di studiare la *logica lineare* LL, che è un sistema formale derivato da altre logiche precedenti, ma la cui nascita si può datare al 1987, con la pubblicazione dei lavori di Girard in [Gir87]. Infatti LL possiede una semantica operazionale molto significativa che è la Geometria dell'Interazione. Il sistema di base su cui è definita la Geometria dell'Interazione è una teoria equazionale che ammette modelli non banali. In generale in logica i modelli sono studiati perchè permettono di raffinare la sintassi; una volta trovato un modello, le proprietà valide in esso possono essere internalizzate. La semantica operazionale da noi usata è costruita su trasformazioni parziali biettive su $\mathbb{N}$; grazie alle proprietà di tali trasformazioni è stato possibile estenderla per dare un'implementazione distribuita del calcolo.
Abbiamo allora cominciato il nostro lavoro con la presentazione separata dei due sistemi formali detti, la logica lineare e il λ-calcolo puro. Un elemento del λ-calcolo puro è detto λ-termine ed è definito come segue. Consideriamo $V = \{x_1, x_2, ...\}$ un insieme numerabile i cui elementi sono detti *variabili*. Consideriamo i simboli (,) e λ; definiamo un alfabeto $A = \{(,), \lambda\} \cup V$ e A^* l'insieme di tutte le parole su A.

Definizione. (λ-termine) *Un* λ-termine t *è una parola di* A^* *costruita per induzione secondo le seguenti regole:*

(i) una variabile $x \in V$ *è un* λ*-termine;*

(ii) se u *e* v *sono* λ*-termini, allora* $t = (u)v$ *è un* λ*-termine;*

(iii) se $x \in V$ *e* u *è un* λ*-termine, allora* $t = \lambda xu$ *è un* λ*-termine.*

Il calcolo sui λ-termini è una trasformazione sintattica detta *β-riduzione*, definita da

$$(\lambda xu)t \to_\beta u[t/x].$$

Il significato dell'espressione di destra è: consideriamo il λ-termine u in cui sostituiamo tutte le occorrenze della variabile x con altrettante occorrenze del termine u.
Un importante risultato della teoria della dimostrazione fornisce una corrispondenza tra i due sistemi formali, che garantisce la possibilità di avvicinare i due sistemi. Originariamente questa corrispondenza è stata data tra il λ-calcolo semplicemente tipato e le deduzioni naturali della logica intuizionista e prende il nome di *Corrispondenza di Curry-Howard*. E' dimostrato che la nozione di calcolo e forma normale (caso in cui non sono più applicabili le regole di riduzione) tra le deduzioni naturali e il λ-calcolo coincidono, pertanto i due sistemi hanno le stesse proprietà computazionali, quindi la corrispondenza di Curry-Howard è un *isomorfismo* tra computazioni.
Inoltre, le deduzioni naturali della logica intuizionista sono in corrispondenza con dimostrazioni della logica lineare (frammento moltiplicativo ed esponenziale, MELL). Questo permette di avere l'isomorfismo di Curry-Howard tra logica lineare e λ-calcolo. In MELL l'analogo delle deduzioni naturali sono i proof net, grafi orientati etichettati con formule che rappresentano classi di equivalenza di dimostrazioni del calcolo dei sequenti. Gli elementi di base che costituiscono un proof net si dicono *link* e corrispondono alle regole del calcolo dei sequenti. Gli archi entranti in un link si dicono *premesse*. Il calcolo

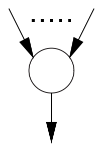

Figura 1: Generico link.

sui proof net è una trasformazione sintattica del grafo detta *cut-elimination*. Questa trasformazione è ottenuta componendo vari *passi di riduzione elementari*. Ne riportiamo uno per dare un esempio in Figura 2.
Abbiamo costruito la versione *pura* dei proof net di MELL, privando i proof net dei tipi, considerando due formule atomiche O e I una duale dell'altra e quozientando le formule con l'equazione $O = !O \multimap O$, che è l'equazione che devono verificare i termini del λ-calcolo puro. Il sistema formale di cui for-

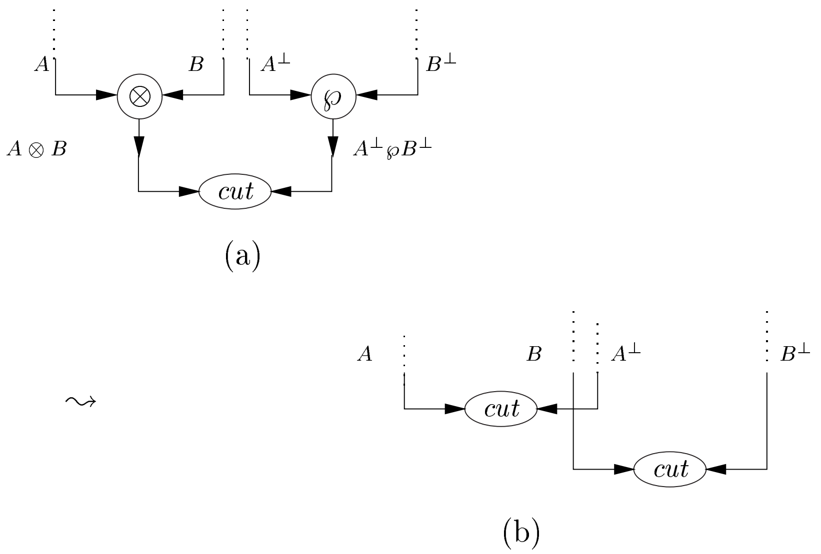

Figura 2: Passo di riduzione.

niamo la semantica operazionale è proprio questo: proof net puri di MELL. La β-riduzione del λ-calcolo a questo punto può essere simulata utilizzando la logica lineare, grazie alla corrispondenza di Curry-Howard, mediante una traduzione del λ-calcolo puro nei *proof net puri* (pure net). Osserviamo che la normalizzazione dei pure net è più dettagliata della β-riduzione e quindi la corrispondenza tra i due sistemi definisce una decomposizione del calcolo per tenere conto della quale è stata necessaria l'introduzione della *op*-equivalenza, che storicamente non fa parte del calcolo dei λ-termini.
Si conclude così la scelta del nostro modello di calcolo, i *pure net*: essi hanno le potenzialità espressive della Macchina di Turing ed inoltre ci danno la possibilità di utilizzare la Geometria dell'Interazione.
Il passaggio dalla sintassi alla semantica consiste nell'assegnazione dei pesi (che sono elementi del modello) ai cammini su un pure net. Il pure net è un grafo orientato i cui gli archi sono occorrenze delle formule $I, O, ?I, !O$. Abbiamo definito un insieme di $\mathcal{O}$-*formule*, cioè formule orientate indicate con $X \uparrow$ oppure $X \downarrow$ in cui X è una delle formule sopra descritte e la freccia è un'etichetta che indica il verso in cui esse sono attraversate. Abbiamo definito un cammino in un pure net come una successione di $\mathcal{O}$-formule. In

particolare abbiamo posto l'attenzione sui cammini straight e massimali; un cammino *straight* è un cammino tale che, se Y, Y_1 e Y_2 sono $\mathcal{O}$-formule con Y_1 e Y_2 premesse dello stesso link, allora esso non contiene sottocammini di tipo Y^*Y e $Y_1{}^*Y_2$. La conclusione di un pure net è un arco di un link del pure net che non è premessa di nessun altro link. Un cammino è massimale se si estende da una conclusione del pure net a una conclusione del pure net. La riduzione di un pure net induce una nozione di calcolo anche sull'insieme dei cammini, che abbiamo visto subiscono delle *deformazioni*.
La *geometria dell'interazione* è una semantica definita su una teoria equazionale chiamata L^*. L'algebra dinamica L^* è una teoria equazionale o *algebra di simboli*.
Il *linguaggio* di L^* è costituito da

- due costanti p e q dette *moltiplicative*; un insieme di costanti x dette *esponenziali*; associamo ad ogni costante esponenziale x un intero c detto *lift* di x e un intero n detto *nome* di x;
- due costanti 0 e 1;
- un simbolo per funzione binaria $\cdot$ che chiameremo *prodotto*;
- un simbolo per funzione unaria $(\cdot)^*$;
- un simbolo per funzione unaria $!(\cdot)$.

Nel seguito usiamo la notazione $x \cdot y = yx$.
Gli *assiomi* di L^* sono

(A1) il prodotto è associativo: $\forall x \forall y \forall z, (xy)z = x(yz) = xyz$;

(A2) l'elemento neutro del prodotto è 1: $\forall x, x1 = 1x = 1$;

(A3) $\forall x, x0 = 0x = 0$

(A4) $\forall u \forall v (uv)^* = v^*u^*$

(A5) $1^* = 1$ e $0^* = 0$

(A6) $(\cdot)^*$ è involutiva: $\forall x, (x^*)^* = x$;

(A7) $!(\cdot)$ è un endomorfismo tale che $\forall x, (!(x))^* =!(x^*)$;

(A8) $!1 = 1$ e $!0 = 0$;

(A9) ! è distributiva rispetto al prodotto: $\forall x, \forall y !(x)!(y) =!(xy)$.

Indicheremo con $!^n(\cdot)$ la composizione di $!(\cdot)$ n volte.
Le *equazioni* di L^* si dividono in due gruppi:

Equazioni di annichilazione

$$p^*p = q^*q = 1$$

$$p^*q = q^*p = 0$$

$$x_{ic}{}^*x_{jd} = 0 \text{ se } i \neq j$$

$$x_{ic}{}^*x_{ic} = 1$$

Equazioni di commutazione

$$!(u)x_{ic} = x_{ic}!^c(u)$$

I termini di L^* saranno chiamati *monomi*.

I pesi sono elementi di L^* e l'assegnazione dei pesi è una applicazione dall'insieme dei cammini sul pure net a L^*. Per la definizione di riduzione che abbiamo dato, un cammino su un pure net può avere come immagine un multinsieme di cammini. Allora si estende la teoria equazionale L^* con un operatore di *somma formale*. Una somma di elementi di L^* è il peso di un multinsieme di cammini.
In una prima fase abbiamo presentato L^* come sistema di riscrittura (ottenuto orientando le equazioni); in seguito abbiamo definito la *formula dell'esecuzione*, che è lo strumento essenziale per fare della geometria dell'interazione un modello di calcolo. Abbiamo dimostrato due teoremi relativi all'algebra dinamica e ai pesi dei cammini su un pure net. In effetti essi risultano essere corollari di una dimostrazione che abbiamo trovato in [Reg92], ma per il ruolo che avranno in seguito li enunciamo come teoremi.
Un monomio $a \in L^*$ non nullo è detto *piatto* se non è della forma $!(a')$. Il

monomio a è detto *positivo* se non contiene applicazioni di $(\cdot)^*$. Una *forma ab^** o *forma stabile* è un monomio $u \in L^*$ della forma $u = a!(m)b^*$ oppure $u = 1$ dove m è una forma stabile e a e b sono piatti e positivi o parole vuote sui generatori.

Teorema. (Proprietà AB^*) *Per ogni cammino ϕ tale che $w(\phi) \neq 0$ esiste una unica forma stabile in cui si riscrive $w(\phi)$ per mezzo delle equazioni orientate.*

Questo teorema fornisce una prova della decidibilità di $u = 0$ in L^*.
I cammini *persistenti* sono i cammini che sopravvivono ad ogni deformazione, ovvero quelli che non vengono cancellati da un passo di riduzione elementare, invece un cammino è *regolare* se ha peso diverso da zero. I due concetti sono legati al seguente teorema.

Teorema. (Teorema Adequacy) *Sia R un pure net e $\phi \in \mathcal{P}(R)$ un cammino straight. Allora ϕ è persistente se e soltanto se ϕ è regolare.*

I cammini straight e massimali sono detti *cammini dell'esecuzione.* La *formula dell'esecuzione* è lo strumento necessario a calcolare la forma normale del pure net attraverso l'algebra dinamica. E' definita come segue.

Definizione. (Matrice dei cammini elementari) *Sia R un pure net. Definiamo la matrice Π_R, detta* matrice dei cammini elementari di R *che ha come indici le conclusioni di R; per ogni coppia di indici (X, Y) l'elemento $(\Pi_R)_{XY}$ è la somma dei pesi di tutti i cammini $\phi : X \to Y$ elementari, cioè*

$$(\Pi_R)_{XY} = \sum_{\phi:X \to Y, \phi \ elem.} w(\phi).$$

Definizione. (Matrice dei cut) *Sia R un pure net. Definiamo la matrice σ_R, detta* matrice dei cut di R, *che ha come indici le conclusioni di R; per ogni coppia di indici (X, Y) l'elemento $(\sigma_R)_{XY}$ è $1 \in {L_R}^*$ se X e Y sono le premesse dello stesso cut-link, altrimenti è $0 \in {L_R}^*$*

La formula di esecuzione usa i pesi dei cammini elementari di R.
Osserviamo che σ è una matrice simmetrica. Definiamo σ^2 come *matrice proiezione*: essa contiene 0 ovunque tranne che nella diagonale principale, dove $\sigma^2_{XX} = 1$ se e soltanto se la conclusione X è la premessa di un *cut*-link. La matrice complementare di σ^2 è $1 - \sigma^2$, dove per 1 si intende la matrice identità. Essa è ancora una matrice proiezione poiché $(1 - \sigma^2)_{XX} = 1$ se e soltanto se X non è premessa di cut.

Definizione. (Formula di esecuzione)

$$EX(\Pi, \sigma) = (1 - \sigma^2)(\Pi \sum_{k=0}^{\infty} (\sigma\Pi)^k)(1 - \sigma^2).$$

La matrice $\sigma\Pi$ contiene i pesi dei cammini che prima attraversano un *cut*-link e poi sono cammini elementari. Più esplicitamente abbiamo che $(\sigma\Pi)^k$ attraversa k cut e k cammini elementari. La matrice $(\sigma\Pi)^{k+1}$ è ottenuta estendendo i cammini in $(\sigma\Pi)^k$ con l'attraversamento di un cut e di un cammino elementare. L'espressione $\sum_{k=0}^{\infty}(\sigma\Pi)^k$ contiene i pesi dei cammini costruiti componendo alternativamente cammini elementari e attraversamenti di *cut*-link. Una conseguenza di questa costruzione è che i cammini che non sono straight non vengono creati.
La somma $\sum_{k=0}^{\infty}(\sigma\Pi)^k$ è moltiplicata a sinistra per la matrice Π. Ciò assicura che la parte centrale della formula dell'esecuzione inizia con un cammino elementare, contiene tutti i cammini conclusione-conclusione che attraversano cammini elementari e attraversa un numero qualunque di *cut*-link.
Infine la moltiplicazione per la matrice proiezione a destra e a sinistra proietta la matrice descritta dalla parte centrale della formula lungo gli indici corrispondenti alle conclusioni del pure net che non sono premesse di cut, e di conseguenza EX contiene solo i pesi dei cammini massimali. Tutti gli altri pesi sono esclusi.
La formula di esecuzione permette di computare la cut elimination calcolando pesi perchè calcola i pesi dei cammini che sopravvivono ad ogni deformazione.

Teorema. *Sia R un pure net e supponiamo che esista R_0 la sua forma normale per riduzioni chiuse. Siano $\Pi = \Pi_R$, $\sigma = \sigma_R$ e $\Pi_0 = \Pi_{R_0}$. Allora*

$$EX(\Pi, \sigma) = \Pi_0.$$

Poiché calcola la forma normale di un pure net, la formula di esecuzione fornisce informazioni sulla normalizzazione forte di R. Osserviamo che i termini della somma in EX sono definitivamente nulli se la matrice $\sigma\Pi$ è nilpotente, ovvero se esiste k_0 tale che i pesi dei cammini ottenuti per potenze $(\sigma\Pi)^k$ con $k > k_0$ sono nulli. In particolare si ha che il pure net è fortemente normalizzabile se e soltanto se la formula di esecuzione è una somma finita. Una dimostrazione è data da Danos in [Dan90] ed è quella che noi abbiamo riportato.
Per rendere effettivo il calcolo della formula di esecuzione e darne un'implementazione è stato necessario studiare ed estendere le proprietà algebriche di L^*. Per prima cosa emerge che L^* è un *monoide inverso.*

Definizione. *Un* monoide *è una coppia $(M, \cdot)$ dove M è un insieme non vuoto e $\cdot$ è un'operazione binaria con un elemento neutro che gode della proprietà associativa.*

Definizione. *Un* monoide inverso *M^* è un monoide con una funzione unaria, lo* star *$(\cdot)^*$, tale che per ogni $u, v \in M^*$*

$$(uv)^* = v^*u^* \tag{1}$$
$$(u^*)^* = u \tag{2}$$
$$uu^*u = u \tag{3}$$
$$uu^*vv^* = vv^*uu^* \tag{4}$$

Abbiamo poi esteso le operazioni su L^* ampliando la sua struttura a quella di *monoide inverso barra.*

Definizione. *Un* monoide inverso barra *è un monoide con uno zero e una* *funzione unaria, la* barra, $[\cdot]$ *tale che per ogni* $u, v \in S$

$$[1] = 0 \ e \ [0] = 1 \tag{5}$$

$$u[v] = [uv]u \tag{6}$$

Il monoide inverso barra che abbiamo costruito a partire da L^* è detto *chiusura barra di* L^* ed è indicato con $[L^*]_\omega$.
A questo punto abbiamo tutti gli strumenti necessari per descrivere la struttura algebrica utilizzata nella implementazione del calcolo di EX. Le proprietà di questa struttura permettono di dimenticare la natura sintattica dei proof net e di costruire delle nuove strutture di grafi, che abbiamo chiamato *reti virtuali*. Una rete virtuale è una coppia $\mathcal{V} = (G, w)$ dove G è un grafo non orientato e w è un peso su G a elementi in L^*. Una rete virtuale può essere dedotta da un pure net e dunque i pesi su $\mathcal{V}$ si definiscono come il peso sui cammini di un pure net. Osserviamo che la formula di esecuzione è un calcolo tra matrici a coefficienti in L^*. Pertanto le reti virtuali possiedono tutti gli strumenti necessari a calcolare la formula dell'esecuzione.
Definiamo il calcolo su una rete virtuale $\mathcal{V}$. Si tratta della *riduzione virtuale*. Applichiamo le trasformazioni di fusione e di scissione ad una classe di archi di G, gli archi *w-componibili*.

Definizione. (Composizione) *Sia* $\mathcal{V} = (G, w)$ *una struttura virtuale. Siano* α *e* β *due archi di* $\mathcal{V}$ *w-componibili. La* composizione di β e α *consiste in tre passi:*

1. *Si effettua la scissione di* β *per mezzo di* $\langle a^* \rangle$ *dove* $a = w(\alpha)$ *rimpiazzando* β *con* β_1 *e* β_2.

2. *Si effettua la scissione di* α^* *per mezzo di* $\langle b \rangle$ *dove* $b = w(\beta)$ *rimpiazzando* α^* *con* α_1^* *e* α_2^*.

3. *Si conclude con la fusione di* β_1 *e* α_1^* *ottenendo l'arco* $\alpha_1 :: \beta_1$ *di peso*

$$w(\alpha_1 :: \beta_1) = ab.$$

Infatti

$$\begin{aligned} w(\alpha_1 :: \beta_1) &= w(\alpha_1)w(\beta_1) \\ &= a\langle b\rangle\langle a^*\rangle b = a\langle a^*\rangle\langle b\rangle b \\ &= aa^*abb^*b \\ &= ab = w(\alpha\beta) \end{aligned} \tag{7}$$

Chiamiamo i termini $\langle b\rangle$ e $\langle a^*\rangle$, che distinguono il peso dei residui dal peso degli archi iniziali, *filtri*.

Definizione. (Riduzione Virtuale) *Una* riduzione virtuale (VR) *di una struttura virtuale $\mathcal{V}$ è una successione di composizioni.*

La riduzione virtuale ha proprietà di confluenza, ovvero il risultato finale, se esiste, è indipendente dall'ordine di applicazione delle composizioni. Inoltre la riduzione di una rete virtuale restituisce una rete virtuale. Su di essa è definita la formula dell'esecuzione in una versione diversa dall'originale, ma indicando con $C_e(G)$ i cammini straight e massimali di una rete virtuale la formulazioni sono equivalenti.

Definizione. (Formula di esecuzione (terza versione)) *La* formula di esecuzione *di $\mathcal{V}$ è un elemento di ${L^*}_+(G)$ definito da*

$$EX(\mathcal{V}) = \sum_{\phi \in C_e(G)} w(\phi).$$

La formula di esecuzione è un invariante per riduzione virtuale. Questa proprietà ci permette di vedere come la riduzione virtuale calcola la forma normale. Considerando una rete virtuale $\mathcal{V}$ in cui non ci sono più composizioni da effettuare, cioè una riduzione virtuale in *forma normale*, si ha che ogni arco in $\mathcal{V}$ è tale che il suo peso compare come termine della somma nella formula dell'esecuzione. Inoltre i termini di $EX(\mathcal{V})$ sono soltanto questi. Allora possiamo dire che effettuare la riduzione virtuale vuol dire calcolare la formula dell'esecuzione. La riduzione virtuale presenta però degli svantaggi.

Iterando il passo di composizione si generano applicazioni successive di barra che si accumulano una dentro l'altra. Una tale rappresentazione dei filtri risulta difficile da gestire.
Per ottenere una versione effettiva della formula di esecuzione viene introdotta la *riduzione virtuale orientata*. Nella riduzione virtuale orientata si considerano soltanto reti virtuali i cui archi sono orientati e non sarà possibile effettuare una composizione tra archi che incidono su nodi diversi.

Definizione. (Rete Virtuale Orientata) *Una* rete virtuale orientata $\mathcal{D}$ *è una rete virtuale aciclica tale che per ogni arco* α

$$w(\alpha) = [b_1, \dots, b_n]a \qquad \text{con } b_1, \dots, b_n, a \text{ monomi positivi di } L^*. \tag{8}$$

e per ogni $i \neq j$ *e per ogni coppia di controarchi* β_1 *e* β_2 *di* α *lungo* τ_1 *e* τ_2 *sono verificate le* condizioni di ortogonalità*:*

$$\langle b_i \rangle \langle b_j \rangle = 0 \tag{9}$$

$$\langle b_i \rangle \langle \tau^* \beta^+ \rangle \tag{10}$$

$$\langle \tau_1^* \beta_1^+ \rangle \langle \tau_2^* \beta_2^+ \rangle \tag{11}$$

in cui abbiamo indicato con γ *il peso dell'arco* γ *e con* γ^+ *il peso di* γ *senza filtro.*

Definizione. (Composizione Orientata) *Sia* $\mathcal{D}$ *una rete virtuale orientata; siano* α *e* β *due archi componibili in* $\mathcal{D}$ *con* $w(\alpha) = [b_1, \dots, b_n]a$ *e* $w(\beta) = [a_1, \dots, a_m]b$*; siano* a' *e* b' *gli unici monomi positivi tali che* $b^* a = a' b'^*$*. La* composizione orientata *di* β *e* α *consiste nei due passi:*

(1) ridurre α *al peso* $w(\alpha) = [b_1, \dots, b_n, b]a$ *e* β *al peso* $w(\beta) = [a_1, \dots, a_m, a]b$*;*

(2) aggiungere a $\mathcal{D}$ *due nuovi archi* α' *e* β' *di peso* $w(\alpha') = a'$ *e* $w(\beta') = b'$ *e un nuovo nodo in modo tale che* α' *e* β' *abbiano come nodo sorgente il nuovo nodo e come nodo di arrivo rispettivamente il nodo sorgente di* β *e di* α*.*

Definizione. (Riduzione Virtuale Orientata) *La* riduzione virtuale orientata (DVR) *di una rete virtuale orientata $\mathcal{D}$ è una successione di composizioni orientate.*

Essa è un caso particolare della riduzione virtuale e per essa valgono tutte le proprietà che abbiamo dimostrato nel caso della riduzione non orientata. In particolare la formula di esecuzione è un invariante per riduzione virtuale orientata.
Descriviamo ora l'implementazione. Una caratteristica fondamentale della DVR è che è un calcolo *locale ed asincrono*. C'è località perchè ogni passo di composizione coinvolge soltanto un nodo e gli archi che incidono su esso. E' asincrono perchè ogni composizione può essere effettuata indipendentemente dalle altre. Sono le caratteristiche adatte ad una implementazione su architettura parallela.
Per implementare la DVR è stata scelta un'architettura multiprocessore e l'implementazione è stata realizzata secondo il modello di comunicazione a scambio di messaggi attraverso lo standard *MPI, Message Passing Interface*. Si procede secondo una particolare strategia, detta *combustion*; diciamo che un arco è ghost se non ha controarchi e la *valenza* di un nodo è il numero di archi non ghost incidenti su esso. La strategia combustion sceglie un nodo di valenza zero ed esegue tutte le possibili composizioni secondo la DVR su di esso. La strategia combustion ha dunque come operazione atomica il calcolo di tutte le composizioni possibili su un nodo. In effetti per come agisce la DVR un nodo di valenza maggiore di zero è in attesa di archi incidenti, per cui il calcolo su tale nodo non verrebbe completato. Da un punto di vista del calcolo parallelo, la ripetizione del calcolo su un arco comporta l'invio di un nuovo messaggio.
Un altro livello di ottimizzazione è una variante della strategia combustion, la strategia *half combustion*. Questa viene introdotta per ovviare allo svantaggio della strategia combustion dato dal fatto che la composizione di due archi coincidenti può iniziare soltanto se il nodo ha valenza zero. Una proprietà importante della HC è che essa può separare l'insieme degli archi incidenti in

un nodo in due insiemi distinti. In altre parole HC associa una etichetta ad ogni arco: *incoming* o *combusto*. Gli archi appena creati sono incoming. Un passo di riduzione consiste nello scegliere un arco incoming e eseguire tutte le possibili composizioni con gli archi combusti coincidenti. Dopo la composizione α è etichettato come combusto. Questa osservazione ci fa capire che possiamo organizzare il calcolo in modo tale che il solo ruolo dei filtri è quello di stabilire l'appartenenza all'insieme degli archi incoming o a quello dei combusti. Pertanto i filtri possono essere scartati portando ad una notevole semplificazione della implementazione.
Per concludere riportiamo il grafico dello speedup ottenuto nel caso dell'esecuzione del λ-termine $(\delta)(\delta)\bar{4}$ dove $\delta = \lambda x(x)x$ e $\bar{4}$ è l'intero di Church.

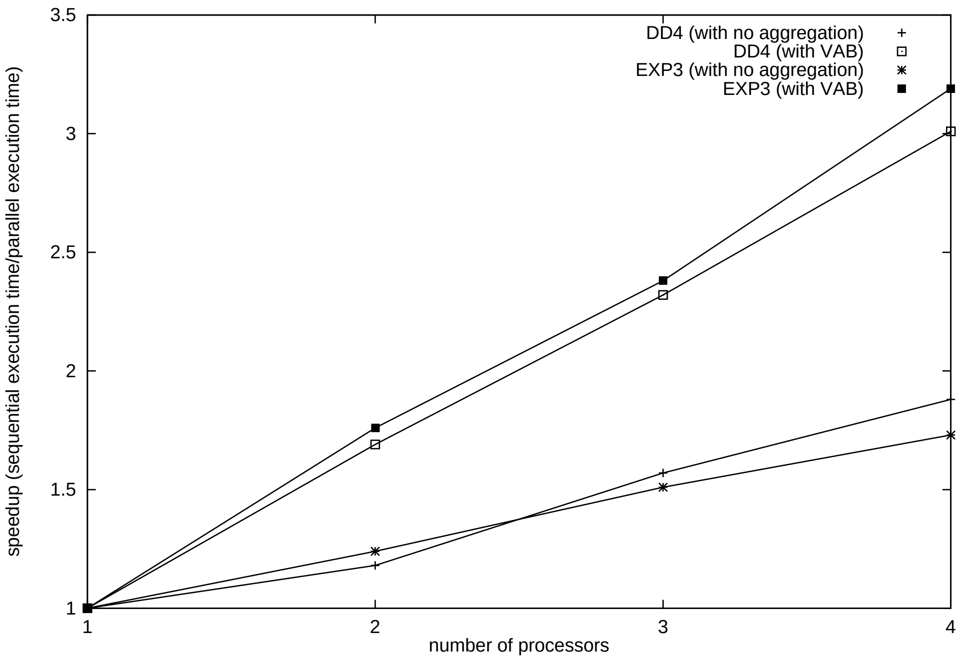

Figura 3: Speedup ottenuto.

L'implementazione che abbiamo realizzato contiene il tentativo originale di includere nella DVR un nuovo costrutto, la *ricorsione*, che permette di estendere il λ-calcolo con una costruzione classica delle funzioni ricorsive.

Definizione. *Siano* $g : \mathbb{N}^r \to \mathbb{N}$ *e* $h : \mathbb{N}^{r+2} \to \mathbb{N}$. *La* funzione definita per ricorrenza di base g e passo h *è la funzione* $f : \mathbb{N}^{r+1} \to$ *definita da*

$$f(\overrightarrow{n}, m) = \begin{cases} g(\overrightarrow{n}) & se\ m = 0 \\ h(\overrightarrow{n}, m, f(\overrightarrow{n}, m-1)) & se\ m > 0 \end{cases}$$

Si estendono i λ-termini con il *l'operatore di ricorsione rec* che segue la regola di riduzione:

$$rec\ x\ in\ M \to_\beta M[rec\ x\ in\ M].$$

Per implementare una funzione definita per ricorsione il termine M deve contenere un termine che effettua il test della definizione per casi (iszero, isnil o istrue).

In questo caso purtroppo la compatibilità tra la DVR e l'operatore di ricorsione che abbiamo definito non è quella aspettata. In alcuni casi la computazione non si arresta e non è neanche possibile fornire un test che il calcolo di EX è completato. Tale comportamento potrebbe essere dovuto

- alla duplicazione delle contrazioni applicate sulle variabili libere come si vede in Figura 4

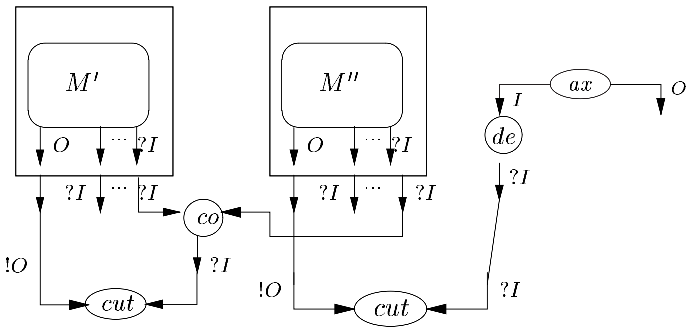

Figura 4: Pure net che rappresenta il λ-termine *rec x in M* dopo un passo elementare di riduzione.

- oppure non sono rispettate le condizioni necessarie alla correttezza del grafo soggetto a DVR, descritto in Figura 5, quindi la riduzione applicata non rispecchia la riduzione attesa conoscendo la forma normale del pure net.

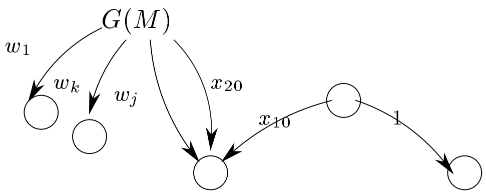

Figura 5: Grafo che rappresenta il pure net *rec x in M*, dove $G(M)$ è il grafo che rappresenta la !-box contenente M.

Per correggere il cattivo funzionamento del costrutto ricorsivo bisognerà pensare ad una analisi della sua correttezza. Tuttavia la soluzione proposta per evitare che configurazioni soggette a duplicazione non siano più controllabili è di implementare un algoritmo di *garbage collection* che elimini la parte che si duplica inutilmente. Un tale algoritmo sarebbe utile a ridurre il costo computazionale anche nel caso di riduzione di cui otteniamo comunque la forma normale.
Siamo partiti dunque da un modello di calcolo astratto come il λ-calcolo e siamo finalmente riusciti realizzare una macchina reale. L'apporto della logica lineare e della geometria dell'interazione è stato fondamentale.

Bibliografia

[Dan89] Vincent Danos. Logique linéaire. Une représentation algébrique du calcul. *Gaz. Math.*, (41):55–64, 1989.

[Dan90] V. Danos. *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul).* PhD thesis, Université Paris VII, Juin 1990.

[DPR97] Vincent Danos, Marco Pedicini, and Laurent Regnier. Directed virtual reductions. In *Computer science logic (Utrecht, 1996)*, volume 1258 of *Lecture Notes in Comput. Sci.*, pages 76–88. Springer, Berlin, 1997.

[DR95] V. Danos and L. Regnier. Proof nets and the Hilbert space. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 307–328, 1995. London Mathematical Society Lecture Note Series 222, Proceedings of the 1993 Workshop on Linear Logic, Cornell University, Ithaca.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir89] J.-Y. Girard. Geometry of interaction 1: Interpretation of system F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic Colloquium '88*, pages 221–260. North-Holland, 1989.

[Gir95a] Jean-Yves Girard. Geometry of interaction. III. Accommodating the additives. In *Advances in linear logic (Ithaca, NY, 1993)*, volume 222 of *London Math. Soc. Lecture Note Ser.*, pages 329–389. Cambridge Univ. Press, Cambridge, 1995.

[Gir95b] Jean-Yves Girard. Linear logic: its syntax and semantics. In *Advances in linear logic (Ithaca, NY, 1993)*, volume 222 of *London Math. Soc. Lecture Note Ser.*, pages 1–42. Cambridge Univ. Press, Cambridge, 1995.

[GLT88] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1988.

[Kri93] J.-L. Krivine. *Lambda-calculus, types and models*. Ellis Horwood Series in Computers and Their Applications. Ellis Horwood, New York, 1993. Translated from the 1990 French original by René Cori.

[Mac71] Saunders MacLane. *Categories for the working mathematician*. Springer-Verlag, New York, 1971. Graduate Texts in Mathematics, Vol. 5.

[MR91] P. Malacaria and L. Regnier. Some results on the interpretation of λ-calculus in operator algebras. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science*, pages 63–72. IEEE Computer Society Press, 1991.

[PQ99] M. Pedicini and F. Quaglia. A parallel implementation for optimal lambda-calculus reduction. Technical Report 16-99, Quaderni dell'Istituto per le Applicazioni del Calcolo, 1999. submitted.

[Reg92] Laurent Reginer. *Lambda-calcul et réseaux*. PhD thesis, Université de Paris 7, 1992.