Sulla struttura logica del calcolo

Lorenzo Tortora de Falco
Dipartimento di Filosofia
Università Roma Tre
e-mail:tortora@uniroma3.it

1 Introduzione

Vorremmo proporre, in questa nota, una breve "passeggiata logica", ripercorrendo le grandi linee di una (piccola) parte della ricerca in *teoria della dimostrazione* (quella branca della logica il cui oggetto di studio sono le dimostrazioni), da Gentzen ad oggi. Abbiamo scelto di farlo prendendo un punto di vista preciso, indicato dal titolo: vogliamo occuparci del calcolo, con strumenti logici, per farne apparire la struttura.
Va da sé che questa "passeggiata logica" non potrà che essere parziale ed incompleta, anche all'interno del microcosmo nel quale intendiamo muoverci. Si pone inoltre una questione delicata, riguardo al tipo di esplorazione che intendiamo fare. Vada per la passeggiata, ma vorremmo che fosse proficua anche a chi voglia tentare l'ascesa di pendii più impegnativi. Abbiamo pertanto cercato un difficile equilibrio tra gli aspetti tecnici e le idee che a volte essi nascondono. I primi sono stati ridotti all'osso (parte di essi è in appendice), ed abbiamo tentato di sfruttare al massimo quelli presenti nel testo per illustrare le idee principali. Alcune definizioni sono assenti, alcune altre volutamente inesatte (in questi casi la cosa è sempre menzionata), ma la speranza è che chi abbia dimestichezza con la pratica matematica riesca a seguire il filo del discorso senza troppe difficoltà. Abbiamo dato ampio spazio a discussioni metodologiche (in particolare quella del capitolo 9) ed alcuni capitoli sono totalmente informali (come il capitolo 7). Se ovviamente i capitoli vanno letti in ordine crescente di numerazione (!), saltare alcuni aspetti tecnici non dovrebbe impedire la comprensione dei capitoli che seguono.
Concludiamo questa prima introduzione con una nota bibliografica: abbiamo fatto il possibile per attribuire con chiarezza i risultati enunciati ai loro autori, in generale all'inizio di ogni capitolo. Spesso però, specie nelle parti più specifiche, abbiamo preferito citare i riferimenti più facilmente accessibili (ad esempio libri invece di articoli) o i più recenti (in cui le dimostrazioni sono state semplificate) senza ossessivamente ripetere quanto detto all'inizio del capitolo circa la paternità di un dato risultato.

2 Contenuto della nota

Riassumiamo in modo molto generale quanto verrà sviluppato nei capitoli successivi, perché serva da filo conduttore nella lettura del testo che segue.
Nel capitolo 3 presentiamo (qualitativamente) tre ben noti modelli della calcolabilità: le funzioni ricorsive, la macchina di Turing ed il λ-calcolo, soffermandoci su quest'ultimo che sarà importante nel seguito.
Il λ-calcolo gioca un ruolo fondamentale, sia in logica che in informatica teorica. Da un lato, esso costituisce un modello semplice ed elegante della nozione di calcolabilità (teorema 1), dall'altro

esso rende anche conto di *come* vengono calcolati i valori di una funzione, senza portare con sé la pesantezza del modello della macchina di Turing. Proprio quest'ultima caratteristica lo rende interessante anche dal punto di vista informatico: tutti i linguaggi di programmazione funzionale hanno un "nucleo" costituito dal λ-calcolo (o meglio da una qualche versione del λ-calcolo). Un λ-termine può essere considerato come un programma, e la sua riduzione (il calcolo effettuabile a partire da esso) è l'esecuzione del programma.

Nel capitolo 4, ci occupiamo dei risultati ottenuti da Gerhard Gentzen nella prima metà degli anni '30 del secolo scorso.
Se i teoremi di Gödel del 1931 permisero di rispondere negativamente al secondo problema di Hilbert (la coerenza dell'aritmetica), questi non escludevano a priori la possibilità di dimostrare la coerenza dell'aritmetica con metodi finitisti *che non fossero aritmetici*. La questione della coerenza delle teorie matematiche continuò dunque ad occupare i logici. Alcuni di essi produssero effettivamente delle "dimostrazioni" di coerenza dell'aritmetica, il cui valore fondazionale è però dubbio, poiché utilizzano principi più forti di quelli dell'aritmetica. Una di queste "dimostrazioni" di coerenza segnò però una rivoluzione all'interno della teoria della dimostrazione, ad opera di Gentzen e del suo teorema di "eliminazione del taglio".
Molto più del risultato ottenuto, fu la tecnica dimostrativa usata da Gentzen ad attirare l'attenzione della comunità scientifica. Egli mostra come sia possibile trasformare una dimostrazione corta intelligente e comprensibile in una dimostrazione lunga stupida ed incomprensibile. Straordinario. Sì, straordinario davvero, perché tale trasformazione corrisponde (in un senso molto preciso) ad un *processo di calcolo*.
La dimostrazione di Gentzen è oggi riconosciuta dagli esperti del settore come un contributo di grande profondità, che è alla base dell'intensa interazione tra logici ed informatici teorici, sviluppatasi nella seconda metà del XX^o secolo.

Nel capitolo 5, cominciamo a rendere più esplicito l'aspetto computazionale della dimostrazione di Gentzen, introducendo la corrispondenza di Curry-Howard tra dimostrazioni e programmi: una dimostrazione (in un certo sistema deduttivo) è un programma, e l'esecuzione di questo programma corrisponde all' applicazione della procedura di eliminazione del taglio definita da Gentzen alla dimostrazione di partenza.
Quest'idea ci aiuta a capire meglio la portata della dimostrazione di Gentzen: l'aspetto "intelligente", "sintetico", "creativo" presente in una dimostrazione si riflette nel *processo* che conduce dalla dimostrazione alla dimostrazione senza tagli ad essa associata. Quest'aspetto non va dunque ricercato nella successione delle regole logiche presenti in una dimostrazione, ma piuttosto nella dinamica interna alla dimostrazione stessa o ancora nella sua capacità di interagire (tramite l'eliminazione del taglio) con le altre dimostrazioni. In una frase, si potrebbe dire che *l'eliminazione del taglio restituisce sotto forma di algoritmo l'aspetto creativo delle dimostrazioni.*
Più concretamente, la corrispondenza di Curry-Howard conduce ad un nuovo approccio, sia alla nozione di costruttività in logica che alla programmazione.
Dal punto di vista logico, non ci si chiede più solamente, ad esempio, se è possibile da una dimostrazione di $\exists x A$ estrarre un testimone che realizzi A (cioè un termine t e una dimostrazione di $A[t/x]$), ma soprattutto, se, come, e cosa è possibile *calcolare* con la dimostrazione di $\exists x A$. Se (ad esempio) la dimostrazione di $\exists x A$ permette di affermare l'esistenza di un intero che soddisfa una certa proprietà, allora l'eliminazione del taglio calcola il valore dell'intero.

Nel capitolo 6 portiamo al massimo grado di precisione l'idea di algoritmo presente nella dimostrazione di Gentzen. Mostriamo come, dal punto di vista informatico, la corrispondenza di

Curry-Howard sia il punto di partenza del paradigma "proofs as programs" (proposto da Leivant e sviluppato negli anni '80 a Parigi da Krivine e Parigot) che suggerisce un nuovo approccio alla relazione tra un programma e la sua specifica. Con una frase, questo si può riassumere dicendo che *è possibile calcolare i valori di una funzione ricorsiva (totale) eliminando i tagli dalla dimostrazione che afferma la totalità della funzione* (nel paragrafo 6.2 diamo un'idea di cosa questo significhi nel caso dell'addizione).
La potenza di questo approccio alla programmazione risiede nella posibilità di programmare manipolando solo oggetti logici (le dimostrazioni), e quindi di applicare ai programmi tutti i risultati matematici sulle dimostrazioni.
Si può dire che lo stretto legame tra dimostrazioni e programmi messo in evidenza nei capitoli 5 e 6 abbia cambiato in modo significativo tanto la teoria della dimostrazione quanto l'informatica teorica.
Da un lato, gli informatici hanno contribuito (e tuttora contribuiscono) a fornire nuove intuizioni ai logici, ed a suggerire loro nuove domande. Le teorie più recenti (come la logica lineare di Girard di cui parleremo nei capitoli 8 e 9, e ancora più recentemente la ludica, vedi [Gir99] e [Gir01]) si occupano esclusivamente del contenuto computazionale delle dimostrazioni, e devono proprio all'incontro tra logica ed informatica la loro nascita ed il loro successo.
D'altro canto, la teoria della dimostrazione fornisce delle basi teoriche solide all'informatica. L'esempio del teorema di correttezza (discusso al capitolo 6) è particolarmente eclatante da questo punto di vista. E non è da meno il teorema 8 del capitolo 9, che fornisce un approccio logico al tempo polinomiale.
L'interazione tra logici ed informatici negli ultimi 30 anni è diventata talmente intensa che ne è nata quella che si può ormai definire una nuova e vasta area di ricerca, chiamata "Logic in Computer Science", come lo testimoniano le numerosissime conferenze internazionali del settore che si svolgono annualmente (tra le più famose possiamo citare ad esempio "Logic in Computer Science", che raduna annualmente i ricercatori di tutto il mondo -oltre 200 partecipanti ogni anno- e la conferenza europea "Computer Science Logic").

Nel capitolo 7 affrontiamo (in modo puramente qualitativo) la questione della rappresentazione del calcolo all'interno di strutture matematiche.
La ricerca di invarianti algebrici del calcolo (l'eliminazione del taglio in logica, l'esecuzione dei programmi in informatica) ha portato allo sviluppo di un campo di ricerca con una ormai lunga tradizione: la semantica denotazionale. Ad ogni formula A viene associata una strutura matematica $\mathcal{A}$ e ad ogni dimostrazione π di A un elemento di $\mathcal{A}$, in modo tale che se π' è una dimostrazione di A ottenuta a partire da π eliminando un certo numero di tagli (cioè effettuando un certo numero di passi di calcolo), allora alle dimostrazioni π e π' è associato lo stesso elemento di $\mathcal{A}$.
Discutiamo nel capitolo 7 di come la semantica denotazionale abbia portato Girard ad introdurre la logica lineare, e di come questa possa essere vista come un raffinamento della logica intuizionista e della logica classica, ottenuto mediante la decomposizione dei connettivi logici usuali e mediante l'introduzione di nuovi connettivi che conferiscono uno status *logico* alle operazioni di duplicazione e di cancellazione (che corrispondono alle "regole strutturali" della logica classica e della logica intuizionista).

Nel capitolo 8, presentiamo una tra le più notevoli novità inrodotte dalla logica lineare: le reti di prova, che introduciamo per un frammento della logica lineare (il caso generale è sommariamente discusso nell'appendice B). Queste portano ad una visione più "geometrica" del processo di eliminazione del taglio, e sono dunque (dal punto di vista da noi adottato) un oggetto fondamentale: suggeriscono che la struttura del calcolo abbia qualcosa di geometrico.

La logica lineare *non è* dunque una logica "esotica", bensì un oggetto matematico con una teoria della dimostrazione ben strutturata, che introduce nuovi concetti.
Dalla sua apparizione nel 1986, la logica lineare è stata usata come strumento in vari settori di ricerca, dalla teoria della dimostrazione al lambda-calcolo, dalle macchine astratte al calcolo distribuito, dalla teoria della complessità alla programmazione logica.

Nell'ultimo capitolo discutiamo l'uso della logica lineare, e più precisamente delle reti di prova, per fornire una rappresentazione puramente logica del tempo polinomiale.
È ben noto come la congettura $P \neq NP$ sia al cuore della ricerca matematica contemporanea, ed appaia nelle forme più varie in moltissimi contesti (in apparenza sconnessi tra loro). La ricerca della natura logica del tempo polinomiale, oltre ad essere affascinante in sé, fornisce strumenti nuovi per affrontare la difficile congettura.
Mostreremo come una bella intuizione metodologica di Girard (di cui illustreremo i passaggi essenziali nel paragrafo 9.4) abbia permesso la caratterizzazione del tempo polinomiale tramite l'operazione principe della moderna teoria della dimostrazione (dal punto di vista da noi preso inizialmente): l'eliminazione del taglio.

Concludiamo questo riassunto introducendo una convenzione che abbiamo cercato di seguire scrupolosamente per maggiore chiarezza: abbiamo scelto di usare il termine "prova" per indicare le dimostrazioni all'interno di un certo sistema deduttivo, riservando il termine "dimostrazione" al consueto uso che se ne fa in matematica.

3 Modelli di calcolo

Se il calcolo è da sempre onnipresente in matematica, fu solo negli anni '30 del secolo scorso che i matematici fornirono rappresentazioni astratte della computazione, o *modelli di calcolo*. Le motivazioni per la ricerca di "rappresentazioni matematiche" della nozione intuitiva di calcolabilità si possono far risalire ai tentativi di rispondere al secondo dei problemi posti da Hilbert (la coerenza dell'aritmetica) al congresso internazionale di matematica di Parigi nel 1900. Infatti, per fare ciò risultò opportuno introdurre la nozione di "sistema deduttivo", e di "regola di inferenza". Con le parole di Gödel ([Sie97] p. 162), ad un sistema deduttivo si chiedeva:
"We require that the rules of inference [...] be constructive; that is, for each rule of inference there shall be a finite procedure for determining whether a given formula B is an immediate consequence (by that rule) of given formulas $A_1, \ldots, A_n$"
Divenne pertanto di capitale importanza produrre una definizione precisa della nozione di *calcolabilità* (la procedura finita di cui parla Gödel). Accettando, ad esempio, come regola di inferenza il modus ponens (da A e $A \to B$, segue B), era necessario convincersi del fatto che esistesse una funzione *effettivamente calcolabile* che prendendo A e $A \to B$ in input producesse B come output. Anche se nel caso del modus ponens questo è evidente, si poneva la questione generale di trovare una definizione matematica del termine "effettivo" (o "calcolabile").
La risposta più convincente fu data da Alan Turing nel 1936 ([Tur37]), che introdusse una macchina astratta con una nozione estremamente precisa e semplice di *passo di calcolo*, evidentemente meccanizzabile. Il contributo di Turing fu tanto significativo che ancora oggi la macchina di Turing rimane il modello di riferimento per la misura della complessità computazionale di un problema (il famoso problema P versus NP fa riferimento al modello di calcolo introdotto da Turing, come precisato all'inizio del capitolo 9).

Nello stesso periodo vennero introdotti altri due modelli di calcolo: le funzioni ricorsive (introdotte da vari autori, ma la cui definizione in uso oggi è dovuta a Kleene) ed il λ-calcolo di Church. Questi tre modelli di calcolo definiscono la stessa classe di "funzioni calcolabili": una funzione tra interi è Turing-calcolabile sse è ricorsiva sse è λ-calcolabile. Essi hanno però caratteristiche diverse:

1. Funzioni ricorsive (parziali, Kleene 37-38): questo modello può essere visto come un approccio "assiomatico" alla calcolabilità. La definizione di funzione ricorsiva è semplice ed elegante matematicamente (La classe delle funzioni ricorsive è "la più piccola classe contenente alcune funzioni di base e *chiusa* rispetto a certe operazioni") ed è pertanto facile stabilire proprietà matematiche di questa classe, usando questa definizione.

   Il notevole inconveniente di tale approccio è che non tiene in alcun conto la *complessità* dei calcoli (il *modo*, anche astratto, con il quale i calcoli vengono effettuati).

2. Macchina di Turing (1937): questo modello può essere visto come un approccio "concreto" alla calcolabilità. È molto facile definire la complessità temporale e spaziale di un calcolo effettuato dalla macchina di Turing (semplicemente come numero dei passi di calcolo o come spazio di calcolo, anch'esso facilmente definibile).

   L'inconveniente principale del modello è quello di fornire una descrizione "troppo" esplicita della computazione, molto poco sintetica e quindi matematicamente molto inelegante. Di conseguenza è estremamente difficile dimostrare proprietà non banali usando tale modello, e di questo ha molto sofferto (e soffre ancora) la teoria della complessità computazionale, almeno fino ai suoi recentissimi sviluppi (vedi anche capitolo 9).

3. Del λ-calcolo (Church 1930), si potrebbe dire che è "un piccolo miracolo". Riesce ad abbinare le virtù matematiche delle funzioni ricorsive con (parte del)l'aspetto "effettivo" del modello di Turing. Esiste infatti una nozione esplicita di "passo di calcolo" (la β-riduzione), che pur non essendo semplice ed elementare quanto quella analoga della Macchina di Turing, permette comunque di rappresentare i vari momenti del processo di calcolo del valore di una funzione. Tutti i linguaggi di programmazione funzionale hanno un "nucleo" costituito da una qualche versione del λ-calcolo.

   L'inconveniente è che un passo di (λ-)calcolo è in generale molto più complesso del passo di calcolo di una Macchina di Turing.

Diamo ora una definizione precisa della nozione di λ-calcolabilità, che è al tempo stesso quella che utilizzeremo nel seguito e forse anche la meno nota tra le tre precedentemente citate.

Definizione 1 (λ-termini). *Sia $\mathcal{V}$ un insieme infinito (numerabile) di "variabili". Un λ-termine è un elemento di $\mathcal{V}$, oppure un'espressione della forma λxt o $(t)u$, dove t ed u sono termini ed $x \in \mathcal{V}$ è una variabile. Più sinteticamente, si scrive:*

$$t ::= x|\lambda xt|(t)t.$$

La definizione che segue è incompleta, in quanto non fornisce tutti gli strumenti per definire correttamente la sostituzione di un termine ad una variabile in un altro termine. Per una definizione precisa, si rimanda ad un qualsiasi manuale di λ-calcolo (ad esempio [Kri90] o [Bar81]).

Definizione 2 (β-riduzione). *Si definisce sull'insieme dei λ-termini una relazione binaria $\rightarrow$ (quando $t \rightarrow t'$ si dice che t' è ottenuto a partire da t con un passo di β-riduzione). La definizione è per induzione su t:*

- *se t è una variabile, $t \to t'$ è falso per ogni λ-termine t'.*
- *se $t = \lambda x\ u$, allora $t \to t'$ sse $t' = \lambda x\ u'$ con $u \to u'$*
- *se $t = (u)v$, allora $t \to t'$ sse una delle seguenti condizioni è soddisfatta:*
  - $t' = (u)v'$ *con* $v \to v'$
  - $t' = (u')v$ *con* $u \to u'$
  - $u = \lambda x\ w$ *e* $t' = w[v/x]$[1].

Si denota con $\twoheadrightarrow$ la chiusura riflessiva e transitiva di $\to$.
Un termine t si dice normale *quando non esiste alcun termine t' tale che $t \to t'$. Un termine t si dice* normalizzabile *quando esiste un termine normale t' tale che $t \twoheadrightarrow t'$.*

Definizione 3 (funzioni λ-calcolabili). *Il seguente λ-termine* rappresenta *l'intero n e si denota con $\underline{n}$: $\lambda f \lambda x (f)^n x (= \lambda f \lambda x\ (f) \ldots (f)(f)\ x)$.*
Una funzione (parziale) $\phi : \mathbb{N}^p \to \mathbb{N}$ è λ-calcolabile *se esiste un termine t_ϕ tale che per ogni $(n_1, \ldots, n_p) \in \mathbb{N}^p$:*

- *se $\phi(n_1, \ldots, n_p)$ è definita, allora $(\ldots((t_\phi)\underline{n_1})\ldots)\underline{n_p} \twoheadrightarrow \underline{\phi(n_1, \ldots, n_p)}$*
- *se $\phi(n_1, \ldots, n_p)$ non è definita, allora $(\ldots((t_\phi)\underline{n_1})\ldots)\underline{n_p}$ è un termine non normalizzabile.*

Teorema 1 (Church,Kleene,Rosser, vedi [Sie97]). *Una funzione (parziale) tra interi è λ-calcolabile sse è ricorsiva.*

I due capitoli seguenti mostreranno come le dimostrazioni all'interno di un opportuno sistema deduttivo (che seguendo la convenzione adottata nell'introduzione chiameremo prove) costituiscano anche esse un modello di calcolo, meno potente dei tre finora presentati (la classe delle funzioni in esso rappresentabili è più piccola), ma non per questo meno interessante...

4 Gentzen e l'eliminazione del taglio (1934)

Presenteremo brevemente, in questo capitolo, (un frammento del)la deduzione naturale ND di Gentzen, e cercheremo di dare un'idea della dimostrazione del teorema di eliminazione del taglio in ND. Questa si presenta sotto forma di "algoritmo", e precisamente di procedura effettiva che trasforma qualsiasi prova di ND in una prova di ND senza tagli. Nei successivi capitoli 5 e 6 quest'idea di algoritmo presente nella dimostrazione di Gentzen verrà resa etremamente esplicita. Quanto segue è materia di un corso di base di teoria della dimostrazione e si può trovare in vari manuali di logica, ad esempio si veda [GLT89] o per una trattazione più dettagliata [Gir87b].

I due principali contributi di Gentzen furono i seguenti (vedi [Sza69]):

1. L'introduzione di sistemi logici (il *calcolo dei sequenti LK* dell'appendice A e la *deduzione naturale ND* che presenteremo in questo capitolo) nei quali le "dimostrazioni" (le prove) corrispondono al naturale procedere del ragionamento logico-matematico (contrariamente ai sistemi introdotti ad esempio da Hilbert). In questi sistemi, inoltre, le regole logiche applicate assumono un ruolo di prima importanza nella rappresentazione delle prove: queste sono alberi (grafi aciclici e connessi), aventi come nodi le formule e come diramazioni le regole logiche.

[1] Se t ed u sono termini e se x è una variabile, si indica con $t[u/x]$ il termine ottenuto sostituendo il termine u ad ogni occorrenza della variabile x "nel termine t". È precisamente questo il punto che richiede una certa attenzione nella definizione rigorosa di β-riduzione.

2. Il teorema di eliminazione del taglio, che si può dire abbia segnato l'inizio della *moderna* teoria della dimostrazione.

Soffermiamoci su quest'ultimo punto. Un enunciato preciso del teorema di Gentzen è il seguente (si veda l'appendice A per la definizione di formula e di sequente):

Teorema 2 (Gentzen 1934). *Esiste una trasformazione $\mathcal{T}$ che associa ad ogni prova π del sequente $\vdash \Gamma$ nel calcolo dei sequenti per la logica classica LK, una prova $\mathcal{T}(\pi)$ dello stesso sequente $\vdash \Gamma$ che non contiene alcuna occorrenza della regola di taglio.*

La dimostrazione è per induzione su varie grandezze, ed in particolare sulla complessità delle formule di taglio (intesa come numero di simboli che occorrono nella formula), e vedremo in seguito (capitolo 9) che questo fa "esplodere" la complessità computazionale del processo di eliminazione del taglio.

Una conseguenza notevole del teorema di eliminazione del taglio di Gentzen è la *proprietà della sottoformula*: se $\vdash \Gamma$ è dimostrabile in LK, allora esiste una prova di $\vdash \Gamma$ che fa intervenire esculsivamente sottoformule delle formule di Γ[2]. (Basta per questo osservare che la proprietà della sottoformula è soddisfatta dal sequente conclusione di una qualsiasi regola di LK diversa dalla regola di taglio).
Un sistema che goda della proprietà della sottoformula è dunque un sistema nel quale qualsiasi teorema è dimostrabile facendo esclusivamente ricorso a metodi e costruzioni direttamente connessi con il risultato da dimostrare.

4.1 La deduzione naturale

Il teorema di completezza di Gödel (dimostrato nel 1930) permette di affermare che le verità logiche sono tutte e sole le formule dimostrabili in LK. Il sistema logico della deduzione naturale ND, o meglio la versione che stiamo per introdurre, non permette di dimostrare tutte le verità logiche (bisogna per questo aggiungere alcune regole, come precisato nell'osservazione 1) ma permette di esprimere con chiarezza la corrispondenza tra le prove ed i termini del λ-calcolo. Vedremo nei capitoli 5 e 6 come questa scoperta, semplice e fondamentale, abbia permesso di stabilire un nesso profondo tra la logica e l'informatica teorica.

Useremo un frammento molto piccolo della deduzione naturale, detto "frammento minimale", in cui l'unico connettivo logico è l'implicazione $\rightarrow$.

Definizione 4 (Formule della logica minimale). *Dato un insieme numerabile di atomi At, l'insieme delle formule della logica minimale è il più piccolo insieme tale che:*

- *un atomo $X \in At$ è una formula*
- *se A e B sono formule, allora $A \rightarrow B$ è una formula.*

Un sequente *minimale è un'espressione della forma $\Gamma \vdash A$ dove A è una formula e Γ è un multinsieme di formule (cioè un insieme di formule in cui viene precisato il numero di occorrenze di una stessa formula).*

[2] Usiamo qui -e nel seguito- la nozione "estesa" di sottoformula: una sottoformula della formula A nella quale si sostituisce una variabile con un termine è ancora una sottoformula di A.

Il significato intuitivo del sequente $A_1, \ldots, A_n \vdash B$ è che dalle ipotesi $A_1, \ldots, A_n$ segue la conclusione B, come appare anche dalle regole del (frammento minimale del) sistema ND:

Assioma:

$$\frac{}{A \vdash A}\, ax$$

Implicazione:

$$\frac{\Gamma \vdash A \to B \qquad \Gamma' \vdash A}{\Gamma, \Gamma' \vdash B} \to_e \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to_i$$

Regole strutturali (solo a sinistra !):

$$\frac{\Gamma \vdash C}{\Gamma, B \vdash C}\, \mathrm{W} \qquad \frac{\Gamma, B, B \vdash C}{\Gamma, B \vdash C}\, \mathrm{Cont}$$

La regola di indebolimento (W) corrisponde alla possibilità di indebolire un enunciato aumentando il numero delle ipotesi[3], mentre la regola di contrazione ($Cont$) corrisponde alla possibilità di usare un'ipotesi un numero arbitrario di volte in una dimostrazione. Queste due regole (dette "strutturali"), che vengono utilizzate costantemente nella pratica matematica, sono cruciali dal punto di vista computazionale: parte dei risultati recenti in teoria della dimostrazione deriva da una gestione logica molto scrupolosa di queste regole (vedi capitoli 8 e 9).

Definizione 5. *Una* prova *in deduzione naturale è una successione finita di regole.*

Osservazione 1. *Aggiungendo ad ND le regole per gli altri connettivi logici ($\vee$, $\wedge$, $\neg$) e per i quantificatori, e le due regole seguenti:*

Assurdità intuizionista e classica:

$$\frac{\Gamma \vdash}{\Gamma \vdash A} \perp_i \qquad \frac{\Gamma, \neg A \vdash}{\Gamma \vdash A} \perp_c$$

si ottiene l'equivalenza del sistema con LK: per ogni formula A, esiste una prova di A in ND (con $\perp_i, \perp_c$) sse esiste una prova di A in LK. Con queste due regole (e le regole relative ai connettivi logici diversi dall'implicazione) si possono dunque dimostrare tutte le verità logiche.

Non vi è una regola di taglio nel sistema ND. Un taglio in deduzione naturale è una qualsiasi obiezione alla proprietà della sottoformula (definita dopo il teorema 2): è facile vedere che la proprietà della sottoformula può essere violata solo se nella prova vi è una regola di introduzione immediatamente[4] seguita da una regola di eliminazione. Nel frammento da noi considerato, esiste un unico tipo di taglio[5]:

[3] Si osservi quanto sia implicitamente usata spessissimo tale regola nella pratica matematica, ad esempio nell'applicare un lemma generale ad una situazione particolare.

[4] Per essere precisi bisogna aggiungere, vista la formulazione che abbiamo scelto per ND, "a meno di regole strutturali": un taglio è un'introduzione seguita eventualmente da un certo numero di indebolimenti e contrazioni (a sinistra) e poi da un'eliminazione.

[5] Vale la stessa osservazione appena fatta sulla presenza di eventuali regole strutturali.

$$\dfrac{\beta \quad \begin{array}{c}\alpha\\ \vdots\\ \Gamma, A \vdash B\end{array} \to_i}{\Gamma' \vdash A \qquad \Gamma \vdash A \to B \quad \to_e}$$
$$\Gamma, \Gamma' \vdash B$$

La formula $A \to B$ occorre nella prova di $\Gamma, \Gamma' \vdash B$, ma non ha nessun motivo di occorrere nel suo sequente conclusione $\Gamma, \Gamma' \vdash B$.
Se per fissare le idee supponiamo che vi sia un unico taglio nella prova precedente, non è difficile immaginare come sia possibile trasformare questa prova in una prova che goda della proprietà della sottoformula (cioè senza tagli). Basta osservare che il sistema ND gode della seguente proprietà: "ogni occorrenza di formula A presente nel sequente $\Gamma, A \vdash B$ di una prova di ND proviene necessariamente da un assioma o da un indebolimento (la regola W)". È dunque possibile tracciare la "storia" di una occorrenza di formula C a sinistra del simbolo "$\vdash$" nel sequente conclusione di una prova: anch'essa si può rappresentare sotto forma di albero, le cui foglie saranno occorrenze di C a sinistra del simbolo "$\vdash$" nei sequenti $C \vdash C$ conclusione di una qualche regola assioma, oppure occorrenze di C a sinistra del simbolo "$\vdash$" nei sequenti (della forma) $\Gamma, C \vdash D$ conclusione di una qualche regola W. In particolare, questo sarà il caso per l'occorrenza di A nella conclusione $\Gamma, A \vdash B$ della sottoprova α nella prova precedente. Pertanto, possiamo sostituire le foglie dell'albero di A che sono assiomi con la prova β di $\Gamma' \vdash A$, e le foglie dell'albero di A che sono regole W con tanti W quante sono le occorrenze delle formule di Γ': otterremo in questo modo una prova di $\Gamma, \Gamma' \vdash B$ *che non farà intervenire la formula* $A \to B$.

Teorema 3 (Gentzen). *Se π è una prova di $\Gamma \vdash A$ in ND, allora si può trasformare π in una prova π' di $\Gamma \vdash A$ nella quale occorrono solo sottoformule di formule di Γ o di A.*

Avendo enunciato con precisione il teorema di Gentzen, possiamo essere più precisi sul suo significato, che può essere espresso dicendo che "è sempre possibile eliminare i lemmi da una prova di logica pura". Infatti, supponiamo di voler dimostrare un certo risultato (non banale) B. Il tipico procedimento del ricercatore sarà quello di semplificare il problema: dimostrerà B sotto una certa ipotesi A (un lemma, appunto), e dovrà poi dimostrare A (utilizzando in generale altri lemmi). In caso di successo avremo in definitiva una prova β di $\vdash A$, ed una prova α di $A \vdash B$: per ottenere una prova di B, applicheremo (come sopra) una regola $\to_i$ immediatamente seguita da una regola $\to_e$, cioè un taglio.
In questo senso il risultato di Gentzen può sorprendere, poiché afferma che l'uso di lemmi (la parte dunque "creativa" del lavoro del ricercatore) è "inutile". Una chiave di lettura della corrispondenza di Curry-Howard che segue può forse essere proprio questa: l'aspetto "creativo" (la scelta dei lemmi giusti) presente in una dimostrazione si riflette nel *processo* che conduce dalla dimostrazione alla dimostrazione senza tagli ad essa associata. In altri termini, si può capire l'aspetto creativo di una dimostrazione π (cioè la dimostrazione stessa !) analizzando il senso computazionale del comportamento di π rispetto alla procedura di eliminazione del taglio (si parla del contenuto computazionale di π).

5 La corrispondenza di Curry-Howard: prove e programmi

Mostreremo in questo capitolo come sia possibile stabilire una corrispondenza molto forte tra il frammento di ND introdotto nel capitolo precedente ed un sottoinsieme dei λ-termini. Questo rende precisa l'intuizione che il processo di eliminazione del taglio sia assimilabile ad un processo

di calcolo, e verrà sfruttato nel prossimo capitolo per estrarre programmi da prove (cioè calcolare i valori di funzioni ricorsive eliminando i tagli dalle prove di un particolare sistema deduttivo). La corrispondenza di Curry-Howard fu presentata per la prima volta in modo esplicito in [How80]; chi volesse approfondire quanto segue in un linguaggio più moderno potrà farlo consultando, ad esempio, [GLT89].

Introduciamo il sottoinsieme dei λ-termini che viene chiamato *λ-calcolo (semplicemente) tipato*. Considereremo in questo capitolo solo le formule introdotte nella definizione 4, che chiameremo anche "tipi". I termini del λ-calcolo semplicemente tipato sono termini del λ-calcolo (vedi definizione 1) ai quali viene associata una formula (un tipo), secondo certe regole, partendo da una quantità numerabile di variabili di tipo A, per ogni formula A. Un termine t di tipo B si denota spesso con t^B.

Definizione 6 (λ-termini semplicemente tipati). *L'insieme dei λ-termini semplicemente tipati è il più piccolo insieme tale che:*

- *tutte le variabili (di qualsiasi tipo) sono λ-termini semplicemente tipati*
- *se x^A è una variabile di tipo A, e se t è un termine tipato di tipo B, allora $\lambda x^A t$ è un termine tipato di tipo $A \to B$*
- *se t è un un termine tipato di tipo $A \to B$, e se u è un termine tipato di tipo A, allora $(t)u$ è un termine tipato di tipo B.*

L'insieme dei termini così definito è dotato della stessa regola di calcolo dell'insieme di tutti i λ-termini (definizione 2), la β-riduzione: $(\lambda x^A t^B)u^A \longrightarrow t^B[u^A/x^A]$, dove $t^B[u^A/x^A]$ indica il termine ottenuto sostituendo ogni occorrenza della variabile x^A (di tipo A) con il termine u^A (anch'esso di tipo A).

Osservazione 2. *Come certo l'attento lettore avrà notato, la β-riduzione preserva il tipo: $(\lambda x^A t^B)u^A$ e $t^B[u^A/x^A]$ hanno entrambi lo stesso tipo B.*

Abbiamo già detto che tutti i linguaggi di programmazione funzionale hanno un "nucleo" costituito dal λ-calcolo (o meglio da una qualche variante del λ-calcolo). Un λ-termine è dunque assimilabile ad un programma, e l'applicazione della β-regola (la relazione $\to$) al λ-termine corrisponde all'esecuzione del programma.
Definiamo ora la corrispondenza tra i λ-termini tipati e le prove di ND. Per quanto appena segnalato, una tale corrispondenza permette di rileggere il processo di eliminazione del taglio definito da Gentzen come l'esecuzione di un programma (come vedremo in modo ancora più preciso nel prossimo capitolo).

Ad ogni prova di $A_1, \ldots, A_n \vdash B$ nel sistema ND è possibile associare n variabili $x_1^{A_1}, \ldots, x_n^{A_n}$ ed un termine t^B (dunque di tipo B), procedendo per induzione sulla lunghezza della prova:

- ad un assioma $\quad A \vdash A \quad$ si associa la variabile x^A sia all'occorrenza sinistra di A che alla prova completa, che in questo caso consta dell'unica regola assioma (una trattazione più formale richiederebbe alcune considerazioni sulla differenza tra x_i^A ed x_j^A, nelle quali non ci addentriamo per alleggerire l'esposizione: si veda ad esempio [GLT89])
- alla prova

$$\dfrac{\begin{array}{c}\alpha\\ \vdots\\ C_1,\ldots,C_n,A\vdash B\end{array}}{C_1,\ldots,C_n\vdash A\to B}\to_i$$

vengono associate le variabili $x_1^{C_1},\ldots,x_n^{C_n}$ ed il termine $\lambda y^A t^B$, se alla prova α premessa della regola $\to_i$ sono associate le variabili $x_1^{C_1},\ldots,x_n^{C_n},y^A$ ed il termine t^B di tipo B.

- alla prova

$$\dfrac{\begin{array}{cc}\beta & \alpha\\ \vdots & \vdots\\ D_1,\ldots,D_m\vdash A & C_1,\ldots,C_n\vdash A\to B\end{array}}{C_1,\ldots,C_n,D_1,\ldots,D_m\vdash B}\to_e$$

vengono associate le variabili $x_1^{C_1},\ldots,x_n^{C_n},y_1^{D_1},\ldots,y_m^{D_m}$ ed il termine $(t^{A\to B})u^A$, se alla prova α premessa della regola $\to_e$ sono associate le variabili $x_1^{C_1},\ldots,x_n^{C_n}$ ed il termine $t^{A\to B}$ di tipo $A\to B$, e se alla prova β premessa della regola $\to_e$ sono associate le variabili $y_1^{D_1},\ldots,y_m^{D_m}$ ed il termine u^A di tipo A.

- alla prova

$$\dfrac{\begin{array}{c}\alpha\\ \vdots\\ C_1,\ldots,C_n\vdash C\end{array}}{C_1,\ldots,C_n,B\vdash C}\,\mathrm{W}$$

vengono associate le variabili $x_1^{C_1},\ldots,x_n^{C_n},y^B$ ed il termine t^C, se alla prova α premessa della regola W sono associate le variabili $x_1^{C_1},\ldots,x_n^{C_n}$ ed il termine t^C.

- alla prova

$$\dfrac{\begin{array}{c}\alpha\\ \vdots\\ C_1,\ldots,C_n,B,B\vdash C\end{array}}{C_1,\ldots,C_n,B\vdash C}\,\mathrm{Cont}$$

vengono associate le variabili $x_1^{C_1},\ldots,x_n^{C_n},y^B$ ed il termine t^C, se alla prova α premessa della regola *Cont* sono associate le variabili $x_1^{C_1},\ldots,x_n^{C_n},z_1^B,z_2^B$ ed il termine t^C.

Associando ad ogni prova il termine come sopra descritto (trascurando dunque le variabili), si può dimostrare che la corrispondenza appena definita tra termini e prove è (in un certo senso) biunivoca. Ma l'aspetto più rilevante sta senz'altro nel fatto che la regola di calcolo definita sui λ-termini semplicemente tipati (la β-riduzione) corrisponde esattamente all'eliminazione del taglio. In altri termini, se chiamiamo F la funzione che ai λ-termini semplicemente tipati associa le prove di

ND, applicando il processo di eliminazione del taglio alla prova $F((\lambda y^A t^B)u^A)$ di ND, otteniamo la prova $F(t^B[u^A/y^A])$, cioè il seguente diagramma commuta (dove le frecce verticali indicano l'azione della funzione F, mentre quelle orizzontali indicano un passo di β-riduzione per i λ-termini ed un passo di eliminazione del taglio come descritto prima dell'enunciato del teorema 3 per le prove di ND):

$$\begin{array}{ccc} (\lambda y^A t^B)u^A & \longrightarrow & t^B[u^A/y^A] \\ \downarrow & & \downarrow \\ F((\lambda y^A t^B)u^A) & \longrightarrow & F(t^B[u^A/y^A]) \end{array}$$

Non è azzardato dire che è proprio questo semplice diagramma il punto di partenza dell'interazione tra logici ed informatici che ha caratterizzato gli ultimi 30 anni, e che ha portato alla nascita dell'area di ricerca chiamata "Logic in Computer Science".

6 Programmare con le prove

Vediamo con maggior precisione, come sia possibile utilizzare la corrispondenza di Curry-Howard per programmare una funzione ricorsiva, manipolando esclusivamente prove. Procederemo prima esponendo schematicamente il procedimento generale di "programmazione con le prove", formulato esplicitamente per la prima volta in [Lei83]. Cercheremo poi di illustrare il metodo con un esempio, e precisamente programmando l'addizione in un sistema logico ben preciso: l'aritmetica funzionale del secondo ordine AF_2. Il sistema AF_2 è un sistema "intuizionista del secondo ordine": è il risultato di un approccio puramente logico alla programmazione, sviluppato negli anni '80 a Parigi da Krivine e Parigot (vedi [KP90] e [Kri90]). In AF_2, da un lato la potenza espressiva dei quantificatori del secondo ordine permette di definire in modo puramente logico i tipi di dato più usati dagli informatici (booleani, interi, liste, alberi...), e dall'altro questo stesso potere espressivo non impedisce al sistema di godere di buone proprietà strutturali (ottenute grazie al notevole contributo di Girard, che in [Gir72] introduce il sistema F).

6.1 Il metodo

Supponiamo ad esempio di voler calcolare i valori della funzione ricorsiva totale f. Si procede nel modo seguente:

1. si definisce f mediante un sistema di equazioni (Eq)
2. si produce una prova π (in un dato sistema deduttivo) di $\forall x(Nx \to Nfx)$ (cioè una dimostrazione di "per ogni x, se x è un intero allora tale è anche $f(x)$"). Una tale prova permette di affermare che la funzione f è *rappresentabile* nel sistema deduttivo considerato. Si dice anche che f è *dimostrabilmente totale* nel sistema deduttivo considerato (il che spiega la restrizione alle funzioni totali), e π viene anche chiamata una *prova di totalità*.
3. si associa (utilizzando l'isomorfismo di Curry-Howard) a π il programma (o meglio il λ-termine) t_f.

Questo approccio permette in particolare di dimostrare *astrattamente* che:

- qualunque sia la strategia utilizzata per eseguire un programma associato ad una prova, l'esecuzione termina (teorema di forte normalizzazione)

- due strategie diverse di esecuzione di un programma associato ad una prova conducono allo stesso risultato (teorema di confluenza)
- un programma associato ad una prova è sempre corretto, nel senso che, con le notazioni precedenti, il programma t_f calcola effettivamente i valori della funzione f definita da (Eq) (teorema di correttezza).

Vale la pena di sottolineare l'importanza "pratica" del teorema di correttezza. Uno dei problemi principali degli informatici contemporanei è proprio quello della correttezza dei programmi. Non appena si tratta di scrivere un programma non banale, le linee di codice necessarie diventano ben più di quelle umanamente controllabili. È quindi facilissimo che nel programma scritto vi sia un errore, che sarà in generale difficile da individuare. E naturalmente, il fatto che un certo numero di test non abbia messo in evidenza alcun errore non garantisce assolutamente la correttezza del programma. È facile immaginare le conseguenze *disastrose* che un programma scorretto potrebbe causare, se usato su larga scala. Come infatti è accaduto a più riprese: ad esempio in occasione dell'esplosione di "Arianne 5" nel 1996 o del blocco completo dell'areoporto di Denver all'inizio degli anni '90.
Se fino ad un decennio fa un programma era considerato corretto fin tanto che qualcuno non vi trovava un errore, attualmente (a seguito di alcuni "disastri informatici") la tendenza è quella opposta: un programma è corretto solo quando è stato *dimostrato* che non contiene errori. Il teorema di correttezza precedentemente enunciato fornisce l'unico metodo *sicuro* che permetta di garantire l'assenza di errori in un programma.

6.2 Programmazione dell'addizione

Diamo ora un esempio di applicazione del metodo appena presentato. Lo faremo programmando l'addizione nel sistema AF_2: assoceremo ad una prova di $\forall x \forall y(Nx \to (Ny \to N(x+y)))$ (cioè ad una dimostrazione di "per ogni x e per ogni y, se x ed y sono interi allora tale è anche $x+y$", ovvero una dimostrazione di totalità della funzione somma in AF_2) un termine t_{add} che *per il teorema di correttezza* sarà necessariamente un programma per l'addizione, nel senso che: se $\underline{n}$ ed $\underline{m}$ sono i termini associati agli interi n ed m, allora $((t_{add})\underline{n})\underline{m}$ si ridurrà nel termine $\underline{n+m}$ associato all'intero $n+m$.
Per fare ciò con tutti i dettagli sarebbe necessario introdurre molte altre nozioni (e non lo faremo): il presente esempio ha dunque come unico scopo quello di illustrare il metodo precedentemente introdotto. Una trattazione rigorosa e completa si trova in [Kri90].

Al sistema ND, bisogna aggiungere le regole di introduzione ed eliminazione dei quantificatori (del primo e del secondo ordine). Per il quantificatore universale del primo ordine, le regole saranno le seguenti:

$$\frac{\Gamma \vdash \forall x A}{\Gamma \vdash A[t/x]}\,\forall^1_e \qquad \frac{\Gamma \vdash A[x]}{\Gamma \vdash \forall x A}\,\forall^1_i$$

dove t è un termine del primo ordine (nel senso della definizione 13 dell'appendice A e non dei termini del λ-calcolo!) e nella premessa della regola $\forall^1_i$ la variabile x non è libera in Γ[6].

Per il quantificatore universale del secondo ordine, le regole saranno le seguenti:

[6]Intuitivamente, un'occorrenza di variabile è libera in una formula quando non è oggetto di alcuna quantificazione (esistenziale o universale). Per la definizione precisa si rimanda a qualunque manuale di logica.

$$\frac{\Gamma \vdash \forall X A}{\Gamma \vdash A[B/X]}\,\forall_e^2 \qquad \frac{\Gamma \vdash A[X]}{\Gamma \vdash \forall X A}\,\forall_i^2$$

dove B è una formula, e nella premessa della regola $\forall_i^2$ la variabile X non è libera in Γ. In questo caso la variabile X è una variabile di predicato, cioè sta ad indicare non più un generico termine del primo ordine, bensì una generica formula (come appare chiaro dalla regola $\forall_e^2$). La sostituzione $A[B/X]$ sta ad indicare che ad ogni occorenza della variabile di predicato X viene sostituita la formula B[7].

I termini associati alle prove nel sistema AF_2 rimangono invece sempre i termini del λ-calcolo semplicemente tipato (definizione 6): seguendo questo approccio le regole dei quantificatori "non hanno significato computazionale".
Applichiamo ora, punto per punto, il metodo precedentemente descritto.

1. Definiamo l'addizione (la funzione $+$) mediante un sistema di equazioni (Eq). Un possibile sistema di equazioni ricorsive che definiscono l'addizione è il seguente:

   $x + 0 = x$

   $x + (y + 1) = (x + y) + 1$

2. Produciamo una prova π (nel sistema deduttivo AF_2) di $\forall x \forall y (Nx \to (Ny \to N(x+y)))$.

   Per fare ciò, supponiamo di avere già una prova di $\forall x (Nx \to N(x+1))$, e quindi (per l'isomorfismo di Curry-Howard) un λ-termine S ad essa associato (che sarà dunque un programma per il successore).

   In AF_2, il predicato "x è un intero" si scrive: $Nx = \forall X((\forall z(Xz \to X(z{+}1)) \to (X0 \to Xx))$[8].

   Una delle possibili dimostrazioni cercate sarà la seguente prova π , dove abbiamo fatto precedere ogni sequente dal termine ad esso associato mediante le regole precedentemente descritte (per non confondere le variabili del linguaggio logico e quelle dei λ-termini, abbiamo usato i consueti simboli x,y,z,u per le prime ed n,m per le seconde):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{S :\vdash \forall u(Nu \to N(u+1))}\,\forall_e^1}{S :\vdash N(x+z) \to N((x+z)+1)}\,(Eq)}{S :\vdash N(x+z) \to N(x+(z+1))}\,\forall_i^1}{S :\vdash \forall z(N(x+z) \to N(x+(z+1)))} \qquad \dfrac{m : Ny \vdash Ny}{m : Ny \vdash (\forall z(N(x+z) \to N(x+(z+1)))) \to (N(x+0) \to N(x+y))}\,\forall_e^2}{(m)S : Ny \vdash N(x+0) \to N(x+y)}\,(\to_e)}{((m)S)n : Nx, Ny \vdash N(x+y)}\,(\to_i)}{\lambda m((m)S)n : Nx \vdash Ny \to N(x+y)}\,(\to_i)}{\lambda n \lambda m((m)S)n :\vdash Nx \to (Ny \to N(x+y))}\,\forall_i^1}{\lambda n \lambda m((m)S)n :\vdash \forall x \forall y(Nx \to (Ny \to N(x+y)))}$$

$$\dfrac{\dfrac{n : Nx \vdash Nx}{n : Nx \vdash N(x+0)}\,(Eq)}{\qquad\qquad}\,(\to_e)$$

3. associamo (utilizzando l'isomorfismo di Curry-Howard) a π il programma (o meglio il λ-termine) t_{add}: per quanto precede, il termine cercato sarà $t_{add} = \lambda n \lambda m((m)S)n$.

   (Osservare che il punto più delicato nella prova precedente è la regola di eliminazione del quantificatore universale del secondo ordine da Ny: abbiamo sostituito Xu con $N(x+u)$).

[7] Anche qui bisognerebbe precisare meglio come vengono gestiti termini e variabili del primo ordine, quando X è di arità diversa da 0.

[8] Questo è un tipico esempio del potere espressivo dei quantificatori del secondo ordine. Si può vedere che prendendo nel linguaggio un simbolo di costante 0 ed un simbolo di funzione unaria s (la cui interpretazione intuitiva è la funzione successore), alla prova "canonica" di $Ns^n(0)$ (cioè alla dimostrazione "canonica" di "l'n-esimo successore di 0 è un intero") viene associato il λ-termine $\underline{n}$ che secondo la definizione 3 è la rappresentazione dell'intero n nel λ-calcolo.

Come detto, per il teorema di correttezza, il termine t_{add} calcola *necessariamente* i valori della funzione ricorsiva "somma", come si può del resto verificare "a mano" su alcuni esempi: $((t_{add})\underline{2})\underline{3} \twoheadrightarrow \underline{5}$. Abbiamo dunque portato ad un grado di precisione considerevole l'idea di algoritmo presente dietro il teorema di eliminazione del taglio di Gentzen: dimostrare la totalità di una funzione ricorsiva significa produrre un vero e proprio *programma* per calcolare i valori della funzione, l'esecuzione del programma essendo esattamente il procedimento di eliminazione del taglio definito da Gentzen.

Osservazione 3. *Concludiamo menzionando un limite (teorico) dell'approccio "proof-teoretico" alla programmazione: esistono funzioni ricorsive totali che* non *sono rappresentabili nel sistema AF_2. Questo si può dimostrare (come molti teoremi dello stesso genere di teoria della ricorsività), applicando opportunamente l'argomento diagonale di Cantor (vedi [Kri90]).*
È però abbastanza chiaro che è possibile in AF_2 programmare qualsiasi funzione "ragionevole"...ed anche molte funzioni "irragionevoli" (come ad esempio la funzione di Ackermann, esempio principe di funzione ricorsiva ma non ricorsiva primitiva: la sua crescita è tanto rapida da dominare tutte le funzioni ricorsive primitive)!
Per sapere come limitare la complessità computazionale delle funzioni rappresentabili in un sistema deduttivo, il lettore dovrà aspettare il capitolo 9.

7 La semantica denotazionale

La semantica denotazionale è una vasta area di ricerca all'interno dell'informatica teorica, e non intendiamo certo presentarla nei dettagli. È però fondamentale che il lettore sappia che è solo grazie alla ricerca in questo settore che sono state introdotte le innovazioni in teoria della dimostrazione che presenteremo nei prossimi capitoli. Un riferimento recente sull'argomento (e perfettamente in linea con quanto segue) è [AC98]. Meno approfondito ma di più facile lettura è [GLT89].

La semantica denotazionale fornisce una visione matematica della programmazione, introducendo degli *invarianti* rispetto al processo di calcolo: ad un programma (o ad una prova[9]) viene associato un oggetto matematico (che viene anche chiamato *interpretazione* del programma), che è appunto lo stesso nei vari stadi di computazione del programma: se $t \to t'$, allora a t ed a t' verrà associato lo stesso oggetto.
Se certo gli scopi di questa associazione sono vari, seguendo [AC98], possiamo citarne almeno due:

1. Dare definizioni *rigorose* degli enti in gioco, che siano indipendenti dagli aspetti legati alle singole implementazioni

2. *Dimostrare* delle proprietà di questi enti, che permettano di *stabilire* nuove relazioni tra di essi.

L'idea fondamentale è che una prova di $A \vdash B$ può essere interpretata come una funzione da $\mathcal{A}$ in $\mathcal{B}$ (essendo $\mathcal{A}$ e $\mathcal{B}$ strutture matematiche di un certo tipo).
Più precisamente, si dice che una certa interpretazione è un *modello* (o una semantica denotazionale) per un dato sistema deduttivo quando ad ogni formula A viene associata una certa struttura matematica $\mathcal{A}$ e ad ogni prova π di A viene associato un elemento $[\pi]$ di $\mathcal{A}$ in modo tale che siano soddisfatte le seguenti condizioni (seguendo [Gir91]):

[9] L'isomorfismo di Curry-Howard ci autorizza a trasferire una considerazione fatta sui programmi alle prove (e viceversa), pertanto tenderemo a confondere le due nozioni nel seguito.

- se π e π' sono due prove (o termini o programmi) tali che $\pi \twoheadrightarrow \pi'$[10], allora $[\pi] = [\pi']$ **(invariante)**
- se $[\pi] = [\pi']$ e λ (risp. λ') è ottenuta a partire da π (risp. π') applicando la stessa regola logica, allora $[\lambda] = [\lambda']$ **(congruenza)**
- esiste una formula A ed esistono due prove π, π' di A t.c. $[\pi] \neq [\pi']$ **(non banale)**.

La questione cruciale che si pone è chiaramente quella di determinare *quali* strutture matematiche utilizzare. La semantica denotazionale fa ampio uso della teoria delle categorie (una formula è un oggetto ed una prova una freccia), ma anche di strutture matematiche "concrete", come i "domini" introdotti da Scott e Plotkin negli anni '70 per ottenere una semantica denotazionale del λ-calcolo puro (quello delle definizioni 1 e 2).
Discutiamo brevemente questo secondo approccio (un approfondimento può essere la prefazione di [AC98]). I domini possono vedersi come spazi topologici soddisfacenti la più debole delle proprietà di separazione (T_0) o equivalentemente come ordini parziali. Gli sviluppi di questo aproccio alla semantica denotazionale sono stati fortemente influenzati dalla ricerca di legami sempre più stretti tra le strutture matematiche in gioco e le formule (e tra gli elementi delle strutture e le prove), in particolare la questione della "full abstraction" per il linguaggio PCF di Scott-Plotkin-Milner ha avuto un ruolo centrale. Si trattava di trovare una nozione di funzione tra strutture matematiche che permettesse di catturare l'aspetto "sequenziale" del linguaggio PCF. Un passo molto importante fu l'introduzione delle funzioni stabili da parte di Gérard Berry ([Ber78]). Una semplificazione delle strutture proposte da Berry condusse Jean-Yves Girard ad introdurre gli spazi coerenti come strutture per interpretare le formule del sistema ND dei capitoli precedenti. Durante questa operazione di semplificazione, Girard si accorge che l'unica operazione tra formule di ND (l'implicazione $\rightarrow$) può essere decomposta negli spazi coerenti (quindi da un punto di vista puramente matematico). Non era a priori chiaro che questa decomposizione potesse essere internalizzata, cioè espressa con delle nuove operazioni logiche. Girard mostra che questo è possibile: l'implicazione $A \rightarrow B$ può decomporsi in $!A \multimap B$, dove "$\multimap$" sta ad indicare l'implicazione lineare che ha il senso di una causalità (algoritmicamente, $\sigma \multimap \tau$ è il tipo degli algoritmi funzionali da σ verso τ che chiamano il loro argomento esattamente una volta); "!" ha il senso di una ripetizione "A quante volte si vuole" e corrisponde algoritmicamente a "mettere A in memoria". È così che appare in [Gir87a] la logica lineare (LL), un raffinamento della logica intuizionista e della logica classica, caratterizzato dalla decomposizione dei connettivi logici usuali ("e" e "o") in due classi (quella dei moltiplicativi e quella degli additivi), e dall'introduzione di nuovi connettivi (gli esponenziali) che conferiscono uno status *logico* alle "regole strutturali" della logica classica e della logica intuizionista. Le regole strutturali (l'indebolimento e la contrazione), avevano (prima dell'avvento della logica lineare) solitamente uno status marginale, quando non erano addirittura lasciate implicite. Eppure, esse parlano della gestione (duplicazione o cancellazione) delle risorse, ovvero di un aspetto centrale di qualsiasi approccio teorico alla nozione di calcolo.

8 La logica lineare e le reti di prova (proof-net)

Si osservi come la nascita della logica lineare sia perfettamente in linea con la seconda delle due motivazioni date all'inizio del capitolo precedente per lo studio della semantica denotazionale: il conferimento di status "logico" alle regole strutturali mediante l'introduzione dei connettivi esponenziali

[10] Qui $\twoheadrightarrow$ indica il processo di calcolo, si pensi ad esempio alla β-riduzione per i λ-termini oppure all'eliminazione del taglio per le prove.

("?" e "!") è un modo di stabilire nuove relazioni tra gli enti in gioco (nel nostro caso le prove). Questo cambiamento di punto di vista (che ha dato luogo al calcolo dei sequenti dell'appendice B) ha avuto conseguenze sorprendenti in teoria della dimostrazione, tra cui una delle più importanti è senz'altro l'introduzione delle reti di prova (i "proof-net"), che forniscono una rappresentazione geometrica del calcolo (l'eliminazione del taglio).
Una prova in LL si può vedere come un grafo, ed il processo di calcolo (l'eliminazione del taglio) viene rappresentato come una deformazione di grafi. Presenteremo in questo capitolo il frammento "moltiplicativo" della LL (MLL), nel quale tutto funziona molto bene. Va detto che il caso generale delle reti di prova per tutta la LL (che vengono presentate sommariamente nell'appendice B) è decisamente più complesso e non completamente convincente. Inoltre, il potere espressivo di MLL è molto limitato (cioè le funzioni in esso rappresentabili, nel senso del capitolo 6, sono pochissime). Questo è essenzialmente dovuto all'assenza, in MLL, della regola di contrazione, cioè della possibilità di duplicare un oggetto durante la computazione. Discuteremo di frammenti di LL con migliori proprietà computazionali nel capitolo 9.

Mostreremo come associare ad ogni prova di MLL un grafo. Ma soprattutto, spiegheremo come sia possibile, tramite una proprietà *puramente geometrica*, caratterizzare, all'interno di una determinata classe di grafi, tutti e soli i grafi che sono prove.
Per la nozione di formula lineare si veda la definizione 14 dell'appendice B. Le formule di MLL sono tutte e sole le formule di LL in cui intervengono solo i connettivi moltiplicativi $\wp$ (la "o" moltilplicativa) e $\otimes$ (la "e" moltiplicativa).

Definizione 7 (Strutture di prova). *Una* struttura di prova *(SP) è un grafo orientato, i cui nodi sono detti legami, ed i cui archi sono etichettati da formule di MLL. Ogni legame ha un certo numero di archi incidenti (premesse del legame) e di archi emergenti (conclusioni del legame).*
I legami sono i seguenti:

- *un legame* assioma *o ax-link non ha alcuna premessa ed ha due conclusioni, etichettate da formule duali (vedi definizione 14),*
- *un legame* taglio *o* cut*-link ha due premesse etichettate da formule duali e nessuna conclusione*
- *un legame $\wp$ (risp. $\otimes$) o $\wp$-link (risp. $\otimes$-link) ha due premesse ed una conclusione. Se la premessa sinistra è etichettata dalla formula A e la premessa destra è etichettata dalla formula B, allora la conclusione è etichettata dalla formula $A \wp B$ (risp. $A \otimes B$).*

Un insieme di legami G t.c. ogni arco è conclusione di esattamente un legame e premessa di al più un legame è una struttura di prova. *Gli archi di G che non sono premesse di alcun legame sono le conclusioni di G.*

Osservazione 4. *Il lettore potrà convincersi facilmente che ad ogni prova del frammento MLL del calcolo dei sequenti lineare (appendice B) si può associare una SP.*
Più interessante è osservare che a diverse prove del calcolo dei sequenti lineare potrà essere associata la stessa SP. Poiché il calcolo (l'eliminazione del taglio) verrà definito direttamente sulle SP (definizione 11), questo vuol dire che le prove del calcolo dei sequenti alle quali è associata la stessa SP sono gli elementi di una classe di equivalenza il cui rappresentante canonico *è la SP. Lungi dall'essere un'osservazione secondaria, dal punto di vista concettuale questo è un passaggio cruciale, come sempre quando in matematica si individua il rappresentante canonico di una classe di equivalenza di oggetti. Si osservi anche che questo è un vero punto di rottura con la tradizione precedente, al quale si è giunti grazie ad una visione più "geometrica" delle prove, ma sarebbe forse*

meglio dire grazie ad una visione geometrica del contenuto computazionale delle prove, mettendo così in relazione i due grandi contributi di Gentzen (capitolo 4).

Definizione 8 (proof-net). *Una rete di prova (diremo più semplicemente una rete) è una SP associata ad una prova del calcolo dei sequenti lineare.*

Se una rete è un grafo associato ad una prova del calcolo dei sequenti lineare, si pone la questione di sapere se esiste una proprietà che permetta di isolare, nell'insieme di tutti i grafi orientati costruiti coi legami della definizione 7 (cioè nell'insieme delle SP) tutti e soli quelli provenienti da prove.

Definizione 9 (Switching, grafo di correttezza). *Diremo che due archi di un grafo orientato sono coincidenti quando sono entrambi incidenti in uno stesso nodo. La coppia $(G, App(G))$ è detta* grafo con coppie *quando G è un grafo orientato e $App(G)$ è un insieme di coppie (non ordinate) di archi coincidenti.*
Se R è una SP, ad essa si può sempre associare un grafo con coppie $R_{ap} = (G_R, App(R))$, ponendo $G_R = R$ e prendendo come $App(R)$ l'insieme delle coppie di archi premesse di uno stesso legame $\wp$.
Uno switching S *(o un "sistema di interruttori") di R è la scelta di un arco in ogni coppia di $App(R)$.*
Ad ogni switching S è associato un grafo non orientato $S(R)$, detto grafo di correttezza*: per ogni coppia di $App(R)$, cancellare gli archi di G_R che non sono stati selezionati da S, e dimenticare l'orientamento e le etichette degli archi del grafo.*

Definizione 10 (criterio di correttezza). *Diremo che una struttura di prova R è* corretta*, quando soddisfa la seguente condizione, nota anche come "criterio di correttezza":*
(ACC) per ogni switching S di R il grafo di correttezza $S(R)$ è aciclico e connesso (è un albero).

Non è difficile dimostrare che ogni rete è una SP corretta, mentre meno evidente è il viceversa:

Teorema 4 (sequenzializzazione). *Ogni SP corretta è una rete.*

Dimostrazione: Vedi [ST00]. ⊠

Osservazione 5. *Nel suo primo lavoro sulla logica lineare ([Gir87a]), Girard introduce le reti di prova e dimostra il teorema di sequenzializzazione usando una proprietà diversa da quella della definizione 10, cioè usando un diverso criterio di correttezza. Da allora ne sono stati introdotti molti altri. Segnaliamo tra questi anche il criterio omologico di [Mét94].*

Passiamo agli aspetti computazionali delle reti, cioè all'eliminazione del taglio, che in MLL sarà una pura deformazione di grafi:

Definizione 11 (Eliminazione del taglio). *Sia R una SP. Nel frammento da noi considerato esistono solo due tipi di taglio, e quindi solo due passi elementari di calcolo (o di riduzione):*

- *il passo assioma si applica quando almeno una delle due premesse del legame cut è conclusione di un legame assioma: la struttura di prova R si riscrive allora nel grafo R' ($R \xrightarrow{[ax]} R'$) come descritto dalla figura 1*
- *il passo moltiplicativo si applica quando una delle due premesse del legame cut è conclusione di un legame $\wp$ mentre l'altra è conclusione di un legame $\otimes$: la struttura di prova R si riscrive allora nel grafo R' ($R \xrightarrow{[\wp/\otimes]} R'$) come descritto dalla figura 2.*

Scriveremo semplicemente $R \longrightarrow R'$ per indicare che R si riscrive in R' applicando uno dei due passi appena definiti. Come al solito, $\twoheadrightarrow$ indicherà la chiusura riflessiva e transitiva di $\longrightarrow$.

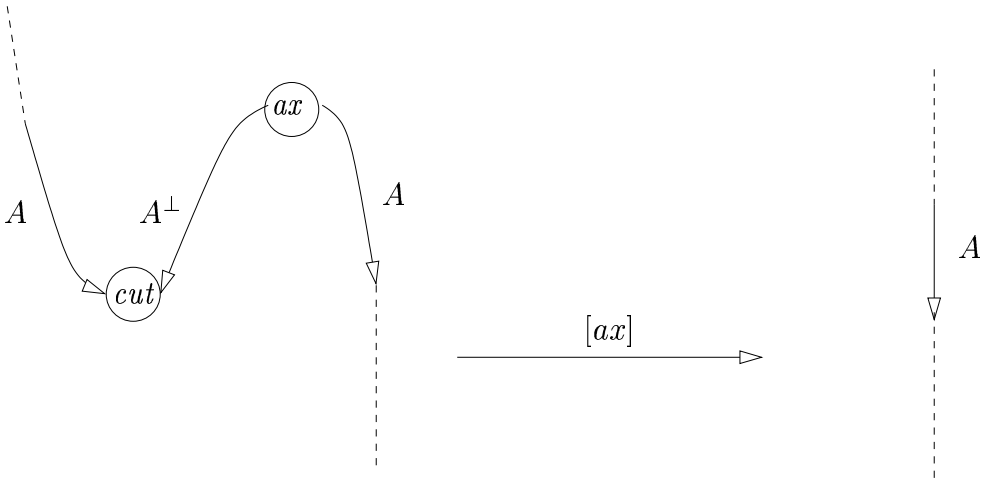

Figura 1. Il passo elementare di calcolo $[ax]$.

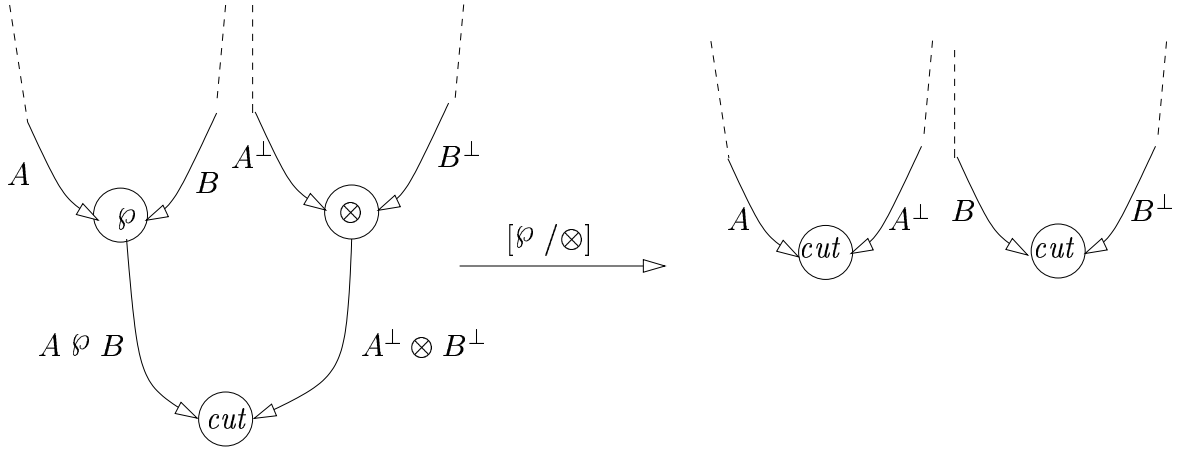

Figura 2. Il passo elementare di calcolo $[\wp\ /\otimes]$.

Appare chiaro che in entrambi i casi $[ax]$ e $[\wp\ /\otimes]$ R' è una SP. Meno chiaro è che se R è una rete tale sarà anche R', ed è l'oggetto del teorema di preservazione della correttezza che segue.

Osservazione 6. *È importante osservare che il processo di eliminazione del taglio è definibile sulle SP (e dunque anche su oggetti non corretti). Questo è uno dei punti che hanno permesso a Girard di sviluppare in [Gir99] e [Gir01] la ludica, un approccio da lui chiamato "interattivo" alla logica, in cui è importante poter calcolare con oggetti logicamente scorretti (che non provengono cioè da prove).*

Teorema 5 (Preservazione della correttezza). *Se R è una rete e se $R \twoheadrightarrow R'$, allora R' è anch'essa una rete.*

Dimostrazione: Vedi ad esempio [Dan90]. ⊠

Teorema 6 (forte normalizzazione). *Se R è una rete, allora qualunque successione di passi di riduzione che parta da R è finita.*

Dimostrazione: Evidente, nel frammento moltiplicativo che stiamo considerando. Ad ogni passo di riduzione (definizione 11) il numero dei nodi del grafo diminuisce strettamente. ⊠

Osservazione 7. *Nel caso generale (per le reti della logica lineare del secondo ordine) il teorema di forte normalizzazione è lungi dall'essere banale: la tecnica dimostrativa fa uso della nozione di "candidato di riducibilità" introdotta da Girard in [Gir72] ed adattata alla logica lineare in [Gir87a].*

Teorema 7 (confluenza). *Se R è una rete, allora esiste un'unica rete R' senza tagli (senza cut-link) tale che $R \twoheadrightarrow R'$. Si dice che la rete R' è* la forma normale di R[11].

Dimostrazione: Vedi ad esempio [Dan90]. ⊠

Osservazione 8. *Nella dimostrazione del teorema 6 di forte normalizzazione (banale nel nostro caso), abbiamo in realtà dimostrato che il numero massimo dei passi di calcolo che portano da una rete R alla sua forma normale è lineare in una misura definibile su R (il numero dei suoi nodi) che potremmo chiamare "strutturale" (dipende dalla struttura del grafo), e che comunque è indipendente dalla complessità delle formule che etichettano gli archi della rete R. Come vedremo nel prossimo capitolo, questo è uno dei punti di partenza di Girard per definire un sistema logico nel quale siano rappresentabili tutte e sole le funzioni calcolabili in tempo polinomiale (da una macchina di Turing deterministica).*

9 Logica lineare e complessità

In questo capitolo discuteremo il contenuto dell'articolo [Gir98], nel quale Girard introduce il sistema "Light Linear Logic" (LLL), e dimostra che una funzione è rappresentabile (nel senso del capitolo 6) in LLL sse è calcolabile in tempo polinomiale da una macchina di Turing deterministica (teorema 8).

9.1 La teoria della complessità

La *teoria della complessità computazionale* si occupa di definire e studiare classi di problemi, classificando questi ultimi in funzione delle risorse (di tempo e di spazio) necessarie alla loro risoluzione, e ciò relativamente ad un certo modello di calcolo. Il modello di riferimento è la macchina di Turing di cui si è discusso nel capitolo 3: l'unità di tempo comunemente accettata per misurare la complessità di un calcolo è il passo elementare di calcolo della macchina di Turing. Il lettore interessato alle definizioni precise potrà consultare, ad esempio, [Pap94] oppure [Sip96].
La questione dell'uguaglianza tra le classi P ed NP si può formulare così: "esiste un problema risolubile in tempo polinomiale da una macchina di Turing non deterministica che non sia risolubile in tempo polinomiale anche da una macchina di Turing deterministica ?"
Questa questione è attualmente al centro della ricerca matematica contemporanea, e ne esistono tantissime formulazioni equivalenti, esprimibili nei linguaggi più diversi.
Il problema fu posto esplicitamente per la prima volta da Cook in [Coo71], ma la questione era già implicita in lavori precedenti di Edmonds, Levin, Yablonski, ed in una lettera di Gödel a Von Neumann degli anni '50. Per una discussione storica sul problema P versus NP, si veda [Sip92].

[11] In riferimento alla terminologia introdotta nella definizione 2 del capitolo 3, sempre a causa della corrispondenza Curry-Howard.

9.2 La teoria della complessità implicita

Una delle difficoltà maggiori nell'approcio al problema risiede nella mancanza di mezzi teorici appropriati per affrontarlo, come già accennato nel capitolo 3 a proposito dei "difetti" della macchina di Turing.
Alcuni progressi recenti sono stati fatti in questo senso, e una parte della ricerca nel settore viene oggi chiamata "teoria della complessità implicita", in quanto cerca di caratterizzare le classi di complessità (ed in particolare P ed NP) senza fare riferimento alla macchina di Turing, ed in contesti più "strutturati", nei quali sia possibile avvalersi di strumenti matematici più elaborati e potenti della mera descrizione del calcolo fornita dalla macchina di Turing. Tali progressi si basano principalmente sullo studio dei fondamenti della computabilità. Le aree dell'informatica teorica coinvolte comprendono: la teoria della ricorsività, la teoria della dimostrazione ed il lambda-calcolo, lo studio di sistemi formali indeboliti e la teoria dei modelli finiti. In tutti questi campi si è assistito ad una esplosione di risultati negli ultimi tempi.

9.3 La natura logica del tempo polinomiale

Prima del sistema LLL di Girard, dei sistemi logici che caratterizzassero il tempo polinomiale erano già stati introdotti (si veda ad esempio [LM93]). Molto grossolanamente, si può dire che in questi sistemi vengono operate delle restrizioni all'applicazione della regola di taglio. Senza entrare nello specifico, la critica che si può muovere a tali sistemi è di non essere "intrinsecamente polinomiali" (di non dire molto sulla "natura logica del tempo polinomiale"): se il taglio è (come è) un modo di comunicare tra prove (o programmi), certo limitando opportunamente la comunicazione si può ottenere un sistema di complessità limitata, ma (per quanto molto interessante e difficile) questo non fornisce una "proprietà astratta del sistema" che caratterizzi il tempo polinomiale. È proprio la ricerca della *natura logica* del tempo polinomiale che motiva il lavoro di Girard. Osserviamo che la speranza di raggiungere questo obiettivo riposa sia sulla possibilità di rappresentare il calcolo in un sistema deduttivo (che ci viene da Gentzen, Curry-Howard, e dalla programmazione per prove del capitolo 6), che dalle reti di prova le quali lasciano intravedere la possibilità di definire sottosistemi di LL senza introdurre limitazioni "ad hoc" sulla regola di taglio ma piuttosto tramite proprietà "geometriche" delle reti stesse.
Dovrebbe essere ormai chiaro al lettore che nel nostro approccio alla teoria della dimostrazione, l'operazione logica di base è l'eliminazione del taglio. Pertanto, il punto di partenza nella ricerca della natura logica del tempo polinomiale non può essere che il seguente: la complessità di un sistema deduttivo è quella della sua procedura di eliminazione del taglio. Questo si spiega anche con la seguente osservazione:

Osservazione 9. *La classe delle funzioni rappresentabili in un sistema deduttivo (nel senso del capitolo 6) è strettamente legata alla complessità del sistema (intesa appunto come complessità della procedura di eliminazione del taglio).*
Questo discende dal fatto che in tutti i sistemi logici comunemente considerati, un passo di eliminazione del taglio è sempre polinomiale nel senso della Turing-calcolabilità: se $t \to t'$[12], allora esiste una macchina di Turing che fissata una qualche codifica di t, se questa ha "taglia" n, allora la macchina è in grado di calcolare la codifica di t' in un numero di passi non superiore a $p(n)$, essendo p un polinomio.

[12] Si intende come al solito che la prova t si trasforma nella prova t' applicando un passo di eliminazione del taglio.

Si tratta in definitiva di trovare un sistema deduttivo per il quale l'eliminazione del taglio sia "polinomiale"[13].
È noto che la complessità della procedura di eliminazione del taglio in logica classica (ed anche in logica intuizionista) è catastrofica: in generale non è neanche elementare (non può essere limitata da una torre di esponenziali di altezza fissata). Si veda in merito [ST00] che discutono il risultato di Orevkov contenuto in [Ore79]. Lo stesso si può dire per la complessità dell'eliminazione del taglio in LL.

9.4 Paradossi alla riscossa

È a seguito di un'intuizione metodologica tipica di un logico che Girard riesce ad individuare la restrizione opportuna da applicare alle reti di prova per ottenere un sistema polinomiale. Cercheremo in questo paragrafo di esporne i passaggi essenziali.
Passo 1: L'osservazione di partenza richiama quanto accennato nell'osservazione 8: i sistemi in cui la complessità dell'eliminazione del taglio è "controllabile" sono quelli (come il sistema MLL del capitolo precedente) in cui essa non dipende dalla complessità delle formule, contrariamente a quanto accade nei sistemi di logica classica. Cerchiamo dunque un sistema in cui la complessità dell'eliminazione del taglio dipenda da parametri legati alla "struttura" della dimostrazione.
Passo 2: Il secondo passo è ricordarsi dell'appendice B di [Pra65], in cui Dag Prawitz aggiunge alla deduzione naturale completa (quella dell'osservazione 1) lo schema di comprensione della teoria *naïve* degli insiemi di Cantor. È ben noto che si ottiene in tal modo una teoria inconsistente (ad esempio col paradosso di Russel). Prawitz osserva però che:

1. per tale sistema è definita una procedura di eliminazione del taglio
2. non esistono prove *senza tagli* dei paradossi.

Poiché la teoria è inconsistente, esistono prove dei paradossi, ma *nessuna di esse è senza tagli*. Questo significa ovviamente (poiché la procedura di eliminazione del taglio è sempre definita) che la procedura di eliminazione del taglio applicata alle prove dei paradossi non termina (risulta essere ciclica).
La stessa procedura si può definire per la logica lineare a cui si è aggiunto l'assioma di comprensione (esattamente come per la deduzione naturale): anche in LL la procedura di eliminazione del taglio applicata alle prove dei paradossi è ciclica.
Passo 3: Supponiamo che (come la deduzione naturale e come LL) il sistema che cerchiamo possa estendersi tramite lo schema di comprensione in un sistema con sempre definita una procedura di eliminazione del taglio e supponiamo che anche in tale sistema esteso non esistano prove *senza tagli* dei paradossi.
Se (come richiesto nel passo 1) nel sistema che stiamo cercando la complessità della procedura di eliminazione del taglio non deve dipendere dalle formule, allora esiste una funzione f (di complessità opportuna) tale che *qualunque* prova π di taglia n si potrà trasformare in una prova senza tagli in $f(n)$ passi (dove n non dipende dalle formule di π): in particolare questo sarà il caso per le prove dei paradossi (dopo aver esteso il nostro sistema con l'assioma di comprensione). Di conseguenza la procedura di eliminazione del taglio applicata a tali prove dovrà terminare, e siccome non esistono prove senza tagli dei paradossi, vuol dire che i paradossi non potranno essere dimostrabili nel sistema !

[13] Vedremo negli enunciati dei teoremi cosa si debba intendere esattamente con questa parola.

Se ne deduce che, se esiste, tale sistema deve essere compatibile con la teoria naïve degli insiemi: ad esso si dovrà poter aggiungere l'assioma di comprensione senza produrre paradossi.
Passo 4: A questo punto, per capire come modificare le reti di prova di LL, non resta che da analizzare con la lente di ingrandimento i paradossi della teoria degli insiemi (espressi in LL) e capire cosa impedisce alla procedura di eliminazione del taglio di terminare quando viene applicata alle prove dei paradossi.
Girard si rende conto durante quest'analisi che la profondità esponenziale dei vari legami della rete e quella della rete stessa (vedi definizione 15 dell'appendice B) aumentano e diminuiscono applicando la procedura di eliminazione del taglio alla prova del paradosso di Russel in LL. È inibendo questa possibilità che si ottengono i sistemi cercati: la profondità esponenziale dei legami rimane (sostanzialmente) costante in questi sistemi durante l'eliminazione del taglio (come espresso dal lemma 1).

9.5 La logica lineare elementare ELL e la logica lineare leggera LLL

Introdurremo con precisione il sistema ELL (o meglio la variante introdotta da Danos e Joinet in [DJ03]) nel quale sono rappresentabili tutte e sole le funzioni elementari, cioè le funzioni computabili da una macchina di Turing (deterministica) in un numero di passi limitato da una torre di esponenziali di altezza fissata: questa classe di funzioni fu introdotta da Kalmar negli anni '40 del secolo scorso. Un'ulteriore restrizione sulle reti (non preciseremo però quale) permette di ottenere il sistema LLL.

Nella definizione che segue useremo la terminologia introdotta nell'appendice B (definizione 15). Abbiamo anche bisogno della nozione di *discendente* di un arco α etichettato da una formula $?A$ in una rete. Ad ogni tale arco α si può associare il cammino massimale avente α come primo arco, orientato secondo l'orientamento "naturale" della rete (dall'alto verso il basso) e che attraversa solo pax-link o $?co$-link. I *discendenti* di α sono tutti e soli i nodi attraversati da tale cammino.

Definizione 12. *Una rete R di LL è una rete di ELL sse:*

1. *la conclusione di ogni legame dereliction ha tra i suoi discendenti esattamente un pax-link*
2. *ogni conclusione etichettata da una formula di tipo $?A$ di un legame assioma non ha alcun pax-link tra i suoi discendenti.*

Esiste una procedura di eliminazione del taglio per le reti di ELL (che non definiremo), per la quale valgono i risultati che seguono:

Proposizione 1 (stabilità di ELL). *Se R è una rete di ELL e se $R \to R'$, allora R' è anch'essa una rete di ELL.*

Lemma 1. *Sia R una rete di ELL e sia $d(R)$ la sua profondità esponenziale.*
Se $R \to R'$, allora $d(R') \leq d(R)$[14].

Sia $\theta(x,y)$ la torre di esponenziali di altezza y applicata ad x, cioè $\theta(x,y) = \underbrace{2^{2^{\cdot^{\cdot^{2^x}}}}}_{y \text{ volte } 2}$. Si noti che per ogni fissato y la funzione $\theta(x,y)$ (della sola variabile x) è elementare.

[14] "Moralmente" si ha $d(R') = d(R)$, anche se tecnicamente è la disuguaglianza a valere.

Proposizione 2. *Sia R una rete di ELL, s il numero di nodi*[15] *di R, d la profondità esponenziale di R.*
La funzione $\theta(s, d)$ limita superiormente il numero dei passi di eliminazione del taglio necessari per raggiungere la forma normale di R.

Il risultato analogo per LLL è il seguente:

Proposizione 3. *Sia R una rete di LLL, s il numero di nodi di R, d la profondità esponenziale di R.*
La funzione s^{2^d} limita superiormente il numero dei passi di eliminazione del taglio necessari per raggiungere la forma normale di R.

Osservazione 10. *I due teoremi precedenti sono sufficienti per poter affermare che in ELL (risp. LLL) si potranno rappresentare* solo *le funzioni computabili da una macchina di Turing (deterministica) in un numero di passi limitato da una torre di esponenziali di altezza fissata (risp. da un polinomio).*
Questo discende dall'osservazione 9 e dal fatto che le reti di ELL e di LLL che rappresentano gli interi hanno tutte la stessa profondità esponenziale (pari ad 1). Si veda ad esempio [DJ03] per i dettagli.

Le due proposizioni 3 e 2 permettono di affermare che i nostri sistemi sono "corretti", cioè che potremo programmare in essi *solo* le funzioni della complessità voluta. Rimane da dimostrare che tali sistemi sono "completi", cioè che è possibile programmare al loro interno *tutte* le funzioni della complessità voluta.

Proposizione 4. *Se f è una funzione calcolabile in tempo polinomiale (risp. elementare), allora esiste una rete R_f di LLL (risp. ELL), che rappresenta la funzione f.*

In definitiva, otteniamo il risultato cercato

Teorema 8. *Le funzioni rappresentabili in LLL (risp. ELL) sono tutte e sole le funzioni polinomiali (risp. elementari).*

Concludiamo menzionando il fatto che dopo il contributo di Girard sono stati numerosissimi i lavori su LLL e le sue varianti. Questo è certo uno dei temi più "caldi" della ricerca in logica lineare. Il sistema LLL è un prezioso strumento che permette di riformulare in termini di eliminazione del taglio (e quindi in termini logici) alcuni problemi di complessità computazionale: in particolare, è ora possibile anche un approccio proof-teoretico alla congettura-chiave della teoria della complessità ($P \neq NP$).

References

[AC98] Roberto Amadio and Pierre-Louis Curien. *Domains and Lambda-calculi.* Cambridge University Press, 1998.

[Bar81] Henk Barendregt. *The Lambda Calculus its Syntax and Semantics.* Studies in Logic and the Foundations of Mathematics. Elsevier, 1981.

[15] In realtà in questo enunciato come nei seguenti s non è esattamente il numero dei nodi ma una taglia molto simile. Si veda [Gir98] e [DJ03] per la definizione esatta.

[Ber78] Gérard Berry. Stable models of typed lambda-calculi. In *Proc. 5th Int. Coll. on Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 72–89. Springer, 1978.

[Coo71] Stephen Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd ACM Symp. on Theory of Computing*, pages 151–158. ACM Press, 1971.

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

[DJ03] Vincent Danos and Jean-Baptiste Joinet. Linear logic and elementary time. *Information and Computation*, 183(1):123–137, 2003.

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* Thèse de doctorat d'Etat, Université Paris VII, 1972.

[Gir87a] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir87b] Jean-Yves Girard. *Proof-Theory and Logical Complexity.* Bibliopolis, 1987.

[Gir91] Jean-Yves Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

[Gir96] Jean-Yves Girard. Proof-nets: the parallel syntax for proof-theory. In Ursini and Agliano, editors, *Logic and Algebra*, New York, 1996. Marcel Dekker.

[Gir98] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.

[Gir99] Jean-Yves Girard. On the meaning of logical rules I: syntax vs. semantics. In U. Berger and H. Schwichtenberg, editors, *Computational Logic.* Springer, 1999. NATO series F 165.

[Gir01] Jean-Yves Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3):301–506, June 2001.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[How80] W A Howard. The formulae-as-types notion of construction. In *In J. P. Seldin and J. R. Hindley, editors, To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980. Hitherto unpublished note of 1969, rearranged, corrected, and annotated by Howard.

[HvG03] Dominic Hughes and Rob van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. In *Proceedings of the eighteenth annual symposium on Logic In Computer Science*, pages 1–10. IEEE, IEEE Computer Society Press, June 2003.

[KP90] Jean-Louis Krivine and Michel Parigot. Programming with proofs. *Journal of Information Processing and Cybernetics EIK (formerly Elektronische Informationsverarbeitung und Kybernetik)*, 26(3):149–167, 1990.

[Kri90] Jean-Louis Krivine. *Lambda-Calcul : Types et Modèles.* Etudes et Recherches en Informatique. Masson, 1990.

[Lau99] Olivier Laurent. Polarized proof-nets: proof-nets for LC (extended abstract). In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications '99*, volume 1581 of *Lecture Notes in Computer Science*, pages 213–227. Springer, April 1999.

[Lei83] Daniel Leivant. Reasoning about functional programs and complexity classes associated with type disciplines. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science FOCS*, pages 460–469, Tucson, Arizona, November 1983. IEEE Computer Society Press.

[LM93] D. Leivant and J-Y Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19(1,2):167–184, September 1993.

[LTdF01] Olivier Laurent and Lorenzo Tortora de Falco. Slicing polarized additive normalization. Quaderno 12, Istituto per le Applicazioni del Calcolo, Roma, Italy, June 2001. To appear in *"Linear Logic in Computer Science", Thomas Ehrhard, Jean-Yves Girard, Paul Ruet and Phil Scott editors*, London Mathematical Society Lecture Notes Series, Cambridge University Press.

[Mét94] François Métayer. Homology of proofnets. *Archive for Mathematical Logic*, 33:169–188, 1994.

[Ore79] V. P. Orevkov. Lower bounds for the lengthening of proofs after cut-elimination. *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Ordena Lenina Matematicheskogo Instituta imeni V. A. Steklova Akademii Nauk SSSR (LOMI)*, 88:137–162,242–243, 1979. Translation *Journal of Soviet Mathematics*, 20 (1982), 2337–2350.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Pra65] Dag Prawitz. *Natural Deduction A Proof-Theoretical Study.* Almquist and Wiksell, 1965.

[Sie97] Wilfried Sieg. Step by recursive step. *The Bulletin of Symbolic Logic*, 3(2):154–180, June 1997.

[Sip92] Michael Sipser. The history and status of the P versus NP question. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 603–618. ACM Press, 1992.

[Sip96] Michael Sipser. *Introduction to the Theory of Computation.* ACM Press, 1996.

[ST00] Helmut Schwichtenberg and Anne Troelstra. *Basic Proof Theory.* Cambridge University Press, 2000.

[Sza69] M.E. Szabo. *The Collected Works of Gerhard Gentzen.* North Holland, 1969.

[TdF00] Lorenzo Tortora de Falco. *Réseaux, cohérence et expériences obsessionnelles.* Thèse de doctorat, Université Paris VII, January 2000. Available at: http://www.logique.jussieu.fr/www.tortora/index.html.

[TdF03] Lorenzo Tortora de Falco. The additive multiboxes. *Annals of Pure and Applied Logic*, 120(1–3):65–102, January 2003.

[Tur37] Alan Turing. On computable numbers with an application to the entscheindungsproblem. *Proceedings of the London Mathematical Society*, 120(2):230–265, 1937.

A Il calcolo dei sequenti LK (Gentzen)

Definizione 13 (Formule). *Darsi un* linguaggio del primo ordine *significa darsi un insieme*

$$\mathcal{L} = \mathcal{V} \cup \{), (, \wedge, \vee, \rightarrow, \forall, \exists\} \cup \bigcup_{n\geq 0} \mathcal{R}_n \cup \bigcup_{n\geq 0} \mathcal{F}_n$$

dove $\mathcal{V}$ è un insieme infinito numerabile di variabili, e per ogni intero $n \geq 0$, $\mathcal{R}_n$ e $\mathcal{F}_n$, sono insiemi di simboli rispettivamente di predicati e funzioni di arità n.
L'insieme dei termini *del linguaggio è il più piccolo insieme di parole del linguaggio tale che:*

- *una variabile è un termine*
- *se $t_1, \ldots, t_n$ sono termini e se $f \in \mathcal{F}_n$, allora $f(t_1, \ldots, t_n)$ è un termine.*

L'insieme delle formule *del linguaggio è il più piccolo insieme di parole del linguaggio tale che:*

- *se $n \geq 0$, $t_1, \ldots, t_n$ sono termini e se $R \in \mathcal{R}_n$, allora $R(t_1, \ldots, t_n)$ e $\neg R(t_1, \ldots, t_n)$ sono formule dette formule atomiche*
- *se F e G sono formule, allora $F \wedge G$, $F \vee G$, $F \rightarrow G$ sono formule*
- *se F è una formula ed $x \in \mathcal{V}$, allora $\forall x F$ ed $\exists x F$ sono formule.*

Nella versione di LK che presentiamo la negazione $\neg$ non è un connettivo, e le formule sono quozientate rispetto alle leggi di De Morgan: $\neg\neg A = A$, $\neg(A \wedge B) = \neg A \vee \neg B$, $\neg(\exists x A) = \forall x \neg A$.
Un sequente Γ *($o \vdash \Gamma$) è un multinsieme di formule (un insieme di formule in cui viene precisato il numero di occorrenze di una stessa formula).*

Vi sono molte versioni (equivalenti) del calcolo dei sequenti LK di Gentzen. Una di queste è data dalle seguenti regole:

Gruppo identità: assioma e taglio:

$$(\text{Ax}) \quad \vdash A, \neg A \qquad\qquad (\text{cut}) \quad \frac{\vdash \Gamma, A \qquad \vdash \Delta, \neg A}{\vdash \Gamma, \Delta}$$

Gruppo logico: regole additive e moltiplicative:

Regole logiche moltiplicative:

$$(\wedge_m) \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge_m B} \qquad\qquad (\vee_m) \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee_m B}$$

Regole logiche additive:

$$(\vee_a) \frac{\vdash \Gamma, A}{\vdash \Gamma, A \vee_a B} \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \vee_a B}$$

$$(\wedge_a) \frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \ \wedge_a \ B}$$

Gruppo strutturale: contrazione ed indebolimento:

$$(\text{W}) \frac{\vdash \Gamma}{\vdash \Gamma, A}$$

$$(\text{C}) \frac{\vdash \Gamma, A, A}{\vdash \Gamma, A}$$

Regole per i quantificatori del primo ordine (y non è libera in Γ, t è un termine):

$$(\forall_1)\ \frac{\vdash \Gamma, A[y/x]}{\vdash \Gamma, \forall x\, A} \qquad (\exists_1)\ \frac{\vdash \Delta, A[t/x]}{\vdash \Delta, \exists x\, A}$$

B Complementi sulla logica lineare (Girard)

In quest'appendice introduciamo con precisione le formule di LL ed il calcolo dei sequenti lineare. Estendiamo anche la nozione di struttura di prova del capitolo 8 a tutta la logica lineare (definizione 15), senza però estendere il criterio di correttezza, e quindi senza enunciare l'analogo del teorema 4. È da notare che tali estensioni, pur essendo possibili, danno luogo ad una nozione di rete di prova meno "pura" ed elegante che nel caso del frammento MLL.
Per una trattazione precisa di queste nozioni si rinvia a [TdF00].

Definizione 14. *Le formule della logica lineare sono costruite come le formule della logica classica, a partire però da un insieme di connettivi diverso:*

- *i moltiplicativi $\wp$ (la "o" moltiplicativa, connettivo binario) e $\otimes$ (la "e" moltiplicativa, connettivo binario)*
- *gli additivi $\oplus$ (la "o" additiva, connettivo binario) e $\&$ (la "e" additiva, connettivo binario)*
- *gli esponenziali $!$ e $?$, entrambi connettivi unari.*

In LL (come nella versione di LK introdotta in precedenza) la negazione lineare $(\ )^\perp$ non è un connettivo, e le formule sono quozientate rispetto alle leggi di De Morgan: $A^{\perp\perp} = A$, $(A \otimes B)^\perp = A^\perp \wp B^\perp$, $(A \oplus B)^\perp = A^\perp \& B^\perp$, $(!A)^\perp = ?A^\perp$, $(\exists x A)^\perp = \forall x A^\perp$, $(\exists X A)^\perp = \forall X A^\perp$. Due formule A e B tali che $A = B^\perp$ vengono dette duali.

Il calcolo dei sequenti lineare è dato dalle seguenti regole:

Assioma e taglio:

$$(\text{Ax})\quad \vdash A, A^\perp \qquad (\text{cut})\ \frac{\vdash \Gamma, A \qquad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}$$

Regole moltiplicative:

$$(\otimes)\ \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \qquad (\wp)\ \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \wp B}$$

Regole additive:

$$(\oplus)\ \frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B} \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}$$

$$(\&)\ \frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B, \Delta}$$

Regole strutturali:

$$(?\text{W})\ \frac{\vdash \Gamma}{\vdash \Gamma, ?A}$$

$$(?\text{C})\ \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}$$

Regola di promozione o del !*:*

$$(!)\ \frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}$$

Regola di dereliction:

$$(?\text{de})\ \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}$$

Regole per i quantificatori del primo o del secondo ordine (Y *non è libera in* Γ*):*

$$(\forall)\ \frac{\vdash \Gamma, A[Y/X]}{\vdash \Gamma, \forall X\ A} \qquad (\exists)\ \frac{\vdash \Delta, A[T/X]}{\vdash \Delta, \exists X\ A}$$

Estendiamo ora la nozione di struttura di prova della definizione 7 del capitolo 8 alla logica lineare completa.

Definizione 15 (struttura di prova per LL). *Ai legami della definizione 7, aggiungiamo i seguenti:*

- *un legame* ! *o* !*-link ha una premessa ed una conclusione, quest'ultima etichettata da* $!A$ *se la premessa è etichettata da* A
- *un legame dereliction o de-link ha una premessa ed una conclusione, quest'ultima etichettata da* $?A$ *se la premessa è etichettata da* A
- *un legame indebolimento o* $?w$*-link non ha alcuna premessa ed ha una conclusione, etichettata da* $?A$ *per qualche formula* A
- *un legame contrazione o* $?co$*-link ha due premesse ed una conclusione, tutte e tre etichettate dalla stessa formula* $?A$ *per qualche formula* A
- *un legame porta ausiliaria o pax-link ha una premessa ed una conclusione, tutte e due etichettate dalla stessa formula* $?A$ *per qualche formula* A
- *un legame 1-plus o* $\oplus_1$ *(risp. 2-plus o* $\oplus_2$*) ha una premessa ed una conclusione tali che se* A *è l'etichetta della premessa, allora* $A \oplus B$ *(risp.* $B \oplus A$*) è l'etichetta della conclusione, per qualche formula* B
- *un legame with o* $\&$*-link ha due premesse ed una conclusione, tali che se* A *(risp.* B*) è l'etichetta della premessa sinistra (risp. destra) del legame, allora* $A\&B$ *è l'etichetta della sua conclusione*
- *un legame contrazione additiva o coad-link ha due premesse ed una conclusione, tutte e tre etichettate dalla stessa formula*
- *un legame per ogni o* $\forall$*-link (risp. un legame esiste o* $\exists$*-link) ha una premessa ed una conclusione tale che se* A *(risp.* $A[T/X]$*) è l'etichetta della premessa, allora* $\forall XA$ *(risp.* $\exists XA$*) è l'etichetta della conclusione.*

Un insieme di legami G *t.c. ogni arco è conclusione di esattamente un legame e premessa di al più un legame è una* struttura di prova *quando le condizioni seguenti sono soddisfatte:*

(1) scatola !*:*

(1.i) *ad ogni !-link n è associato un (unico) sottografo $B^!$ di G t.c. una almeno tra le conclusioni di $B^!$ sia la conclusione di n ed ogni altra conclusione di $B^!$ (potrebbero non essercene altre) sia conclusione di un pax-link. $B^!$ viene chiamata scatola esponenziale e si rapppresenta come in figura 3. Il nodo n si dice la porta principale della scatola.*

(1.ii) *ad ogni pax-link n è associata una scatola esponenziale $B^!$ di G tale che una tra le conclusioni di $B^!$ sia la conclusione di n (vedi figura 3). Il nodo n è una porta ausiliaria della scatola $B^!$.*

(2) scatola &:

(2.i) *ad ogni legame with n è associato un (unico) sottografo B di G t.c. una almeno tra le conclusioni di B sia la conclusione di n ed ogni altra conclusione di B (potrebbero non essercene altre) sia conclusione di un coad-link. B viene chiamata scatola additiva e si rapppresenta come in figura 4. Il nodo n si dice la porta principale della scatola.*

(2.ii) *ad ogni coad-link n è associata una scatola additiva B di G tale che una tra le conclusioni di B sia la conclusione di n (vedi figura 4). Il nodo n è una porta ausiliaria della scatola B.*

(3) condizione di inscatolamento:

due scatole, di qualsiasi tipo (additivo o esponenziale) sono disgiunte, oppure sono l'una contenuta nell'altra.

Diremo che un legame o un arco di una struttura di prova R ha *profondità* esponenziale (risp. additiva) n in R, se è contenuto in esattamente n scatole esponenziali (risp. additive) di R. Per una scatola B (additiva o esponenziale), diremo che essa ha *profondità* esponenziale (risp. additiva) n quando è contenuta in esattamente n scatole esponenziali (risp. additive) di R, tutte diverse da B. La *profondità* esponenziale (risp. additiva) di una struttura di prova è la profondità massima dei suoi legami.

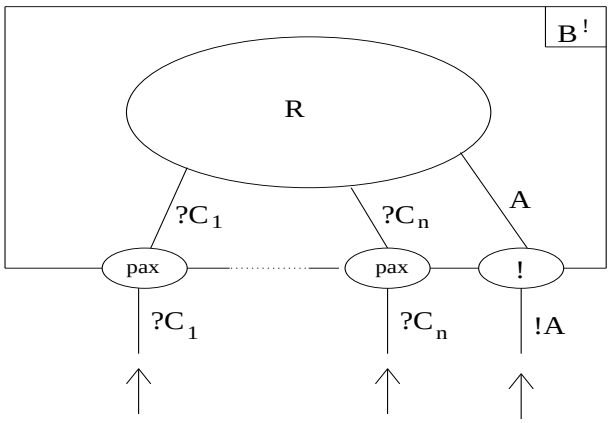

Figura 3. La scatola esponenziale $B^!$ della struttura di prova G.

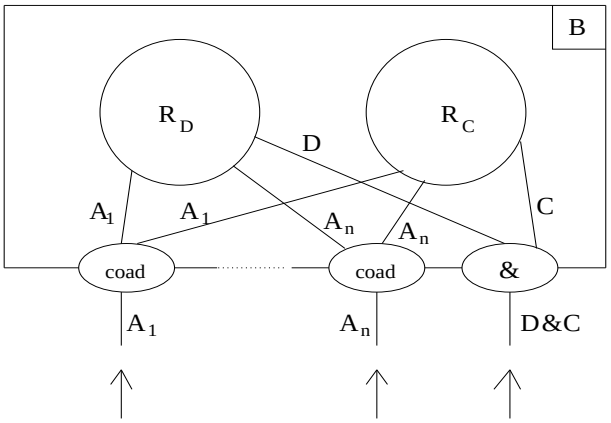

Figura 4. La scatola additiva B della struttura di prova G.

Osservazione 11. *Per le strutture di prova appena definite, esiste un criterio di correttezza che permette di dimostrare un teorema analogo al teorema 4 ed un teorema analogo al teorema 5 nel caso generale.*
Vale il teorema di forte normalizzazione (l'analogo del teorema 6), dimostrato da Girard in [Gir87a], ma non vale il teorema di confluenza (l'analogo del teorema 7), a causa del comportamento del connettivo additivo &. Per una discussione sull'argomento si veda [TdF00] e per le soluzioni proposte al problema si vedano [Gir96],[TdF03],[HvG03],[Lau99],[LTdF01].