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

Tesi di Laurea in Matematica

presentata da

Silvia Lanaro

LOGICA LINEARE E TEMPO ELEMENTARE

(SINTESI)

Relatore

Prof. Lorenzo Tortora de Falco

Il Candidato

Il Relatore

ANNO ACCADEMICO 2001 - 2002

Febbraio 2003

1 Sintesi

Questa tesi si propone di mostrare come sia possibile affrontare i problemi di teoria della complessità in ambito logico.

La teoria della complessità si pone la questione di classificare i problemi in termini di risorse (in genere spazio o tempo) necessarie e/o sufficienti a risolverli: si ottengono così le *classi di complessità* e ciò viene fatto relativamente ad un dato modello di calcolo. Il modello di calcolo su cui si poggia la teoria della complessità è quello della macchina di Turing [1]. Questo perchè, pur non essendo l'unico modello di calcolo esistente, esso presenta un vantaggio enorme: un passo elementare di calcolo di una MdT è effettivamente elementare, in quanto può essere simulato da una macchina vera in un tempo costante.

Il modello della MdT presenta però un grave inconveniente: è metematicamente inelegante e pertanto non permette di stabilire le proprietà matematiche delle classi di complessità.
Proprio per ovviare a tale problematica l'informatica (teorica) contemporanea si interessa di trovare delle definizioni alternative [2] delle classi di complessità. Un possibile approccio alla soluzione di questo problema viene fornito dalla teoria della dimostrazione, che è quella parte della logica matematica il cui oggetto di studio sono le dimostrazioni.

Lo scopo che ci siamo prefissati in questa tesi è di descrivere, facendo riferimento all'articolo di Danos e Joinet [7], il sistema logico ELL, detto *la logica lineare elementare*. In tale sistema è possibile rappresentare tutte e sole le funzioni calcolabili da una MdT in tempo elementare.

I due ingredienti principali necessari alla comprensione dell'articolo sono:

- il λ-calcolo (per la relazione tra teoria della dimostrazione e informatica), per cui abbiamo fatto riferimento al libro di Krivine [16] e al libro di Girard, Taylor e Lafont [11].

[1] nel seguito invece di macchina di Turing scriveremo MdT
[2] cioè che non facciano riferimento al modello della MdT

- la logica lineare e i proof-net (per capire come costruire ELL), per cui abbiamo fatto riferimento all'importante articolo di Girard [12], alla tesi di dottorato di Tortora de Falco [8] e alla tesi di laurea di Claudia Barcaglioni [2].

Prima di affrontare questioni riguardanti la complessità dei calcoli (effettuati per la risoluzione di un problema) è necessario distinguere ciò che è calcolabile da ciò che non lo è.

È questo l'oggetto di studio della teoria della *calcolabilità*. Tradizionalmente vi sono tre approcci diversi a questa questione:

1. il primo è di tipo assiomatico e si basa sulla nozione di *funzione ricorsiva*.

2. il secondo più concreto (in esso si definisce una nozione di passo elementare di calcolo) si deve a Turing il quale definì una macchina astratta, che gli permise di definire l'insieme delle *funzioni Turing-calcolabili*.

3. il terzo (in realtà il primo da un punto di vista cronologico) Church definì il λ-calcolo e la nozione di *funzione λ-definibile*.

Le tre nozioni di "funzione calcolabile" derivanti da questi tre modelli (ricorsiva, Turing-calcolabile, λ-definibile) coincidono; pertanto è comunemente accettata l'affermazione seguente (nota col nome di "tesi di Church"): "le funzioni intuitivamente calcolabili sono tutte e sole le funzioni ricorsive (o Turing-calcolabili, o λ-definibili)".

Dopo aver stabilito la calcolabilità di un problema ci si pone la domanda: qual è il costo di risoluzione di un problema in termini di risorse di calcolo (spazio o tempo) utilizzate? La teoria della complessità classifica i problemi in base alle risorse di calcolo necessarie a risolverli.

Nel fare questa distinzione essa fa riferimento al modello della macchina di Turing all'interno del quale vengono definite le seguenti classi di complessità:

1. $P = \bigcup_k Time(n^k)$ (risp. $fP = \bigcup_k fTime(n^k)$), che rappresenta l'insieme dei linguaggi decisi (risp. delle funzioni calcolate) da una macchina di Turing in tempo polinomiale.

2. $EXP = \bigcup_k Time(2^{n^k})$, che rappresenta l'insieme dei linguaggi decisi da una macchina di Turing in tempo esponenziale.

3. $\mathcal{E}_{time} = \{f : \mathbb{N}^p \longrightarrow \mathbb{N}$ totali tali che $\exists k_f \in \mathbb{N}$ e tale che $f \in fTime(g(n))$ dove g è una torre di esponenziali di altezza fissata $k_f\}$.

Come già detto il modello di calcolo della MdT presenta il grosso vantaggio di avere in se definita una nozione di passo di calcolo evidentemente meccanizzabile. Ma presenta anche l'inconveniente di fornire una descrizione del calcolo poco rigorosa da un punto di vista matematico. Di conseguenza è estremamente difficile dimostrare proprietà non banali di questo modello. Di questo ha molto sofferto (e soffre ancora) la teoria della complessità algoritmica, almeno fino ai suoi recentissimi sviluppi.

Si cercano pertanto definizioni alternative delle classi di complessità.

Una prima soluzione a questa questione è di tipo assiomatico:

1. Dimostreremo che è possibile definire assiomaticamente la classe $\mathcal{E}_{time}$ come segue:

   Definizione 1.1. $\mathcal{E}_{time}$ è la più piccola classe di funzioni contenente: le proiezioni, le costanti, l'addizione, l'esponenziale ed è chiusa rispetto alla composizione e alla ricorsione limitata.

   dove:

   Definizione 1.2. Date $f_1, \cdots, f_n : \mathbb{N}^p \longrightarrow \mathbb{N}$ e $g : \mathbb{N}^n \longrightarrow \mathbb{N}$ diremo che f è ottenuta da $f_1 \cdots f_n, g$ per **composizione** quando $f : \mathbb{N}^p \longrightarrow \mathbb{N}$ tale che $f(\overrightarrow{x}) = g(f_1(\overrightarrow{x}), \cdots, f_n(\overrightarrow{x}))$

   Definizione 1.3. La funzione $f : \mathbb{N}^{p+1} \longrightarrow \mathbb{N}$ è ottenuta da $g : \mathbb{N}^p \longrightarrow \mathbb{N}, h : \mathbb{N}^{p+2} \longrightarrow \mathbb{N}, k : \mathbb{N}^{p+1} \longrightarrow \mathbb{N}$ per ricorsione limitata quando:

   - $f(\overrightarrow{x}, 0) = g(\overrightarrow{x})$
   - $f(\overrightarrow{x}, y+1) = h(\overrightarrow{x}, y, f(\overrightarrow{x}, y))$
   - $f(\overrightarrow{x}, y) \leq k(\overrightarrow{x}, y)$

Osservazione 1. Nelle definizioni precedenti si ha $\overrightarrow{x} = (x_1, \cdots, x_p)$.

2. Daremo, anche un breve accenno della caratterizzazione della classe fP dovuta a Bellantoni-Cook ([3]).

Una seconda soluzione possibile è fornita dalla teoria della dimostrazione. In questa prospettiva si colloca la tesi.

Iniziamo a vedere come sia possibile rappresentare classi di complessità in un sistema logico.

Ripartiamo dal λ-calcolo puro, introdotto da Church negli anni '30. Lo studio di tale sistema portò successivamente alla definizione di un altro sistema: il λ-calcolo tipato semplice. Il λ-calcolo tipato semplice ha suscitato un grande interesse a causa dei suoi rapporti con i linguaggi di programmazione. In questo contesto si può pensare ai termini[3] come ai programmi e il tipo di un termine si può considerare come una specifica, cioè ciò che il programma fa astrattamente. A priori è un commento della forma: "questo programma calcola la somma tra due interi". Per i termini del λ-calcolo esiste una nozione di calcolo, detta la β-riduzione, che corrisponde ad un passo di esecuzione del programma.

In maniera completamente indipendente dal λ-calcolo tipato semplice si sviluppa la logica classica del primo ordine e il sistema della deduzione naturale che sono due sistemi logici in cui le dimostrazioni corrispondono al naturale procedere del ragionamento logico-matematico.
Per tali sistemi si definisce una procedura tale che, se π è una derivazione di un sequente S da $\emptyset$ produce in un numero finito di passi una derivazione ψ di S da $\emptyset$ tale che:

1. ψ non contiene alcuna regola di taglio.
2. Se π è senza regole strutturali, anche ψ è senza regole strutturali.

Questo risultato, che si può dire abbia segnato la nascita della teoria della dimostrazione è dovuto a Gentzen e prende il nome di **Teorema di eliminazione del taglio**.

[3] i termini sono gli oggetti di cui si occupa il λ-calcolo

Negli anni 60 fu scoperto un legame profondo tra la teoria della dimostrazione e l'informatica teorica: **l'isomorfismo di Curry-Howard**. Questa è una corrispondenza tra le dimostrazioni del sistema della deduzione naturale (ristretto al frammento $(\wedge, \rightarrow)$) e i termini del λ-calcolo tipato semplice. Grazie a tale isomorfismo si ha che ad ogni dimostrazione π di un sequente $\vdash A$ corrisponde un termine t di tipo A e applicare a π la procedura di eliminazione del taglio corrisponde ad applicare una successione di β-riduzioni al termine t

$$\begin{array}{ccccc} & \pi & \stackrel{cut-el}{\longrightarrow} & \pi' & \\ (CH) & \downarrow & & \downarrow & (CH) \\ & t & \stackrel{\beta}{\longrightarrow} & t' & \end{array}$$

dove CH sta per corrispondenza Curry-Howard.

Tale corrispondenza ci permette di vedere le dimostrazioni come programmi (come accennato in precedenza), infatti ogni linguaggio di programmazione funzionale ha come "nucleo" il λ-calcolo tipato semplice. Pertanto una dimostrazione è un programma la cui esecuzione corrisponde ad una successione di passi di eliminazione del taglio[4].

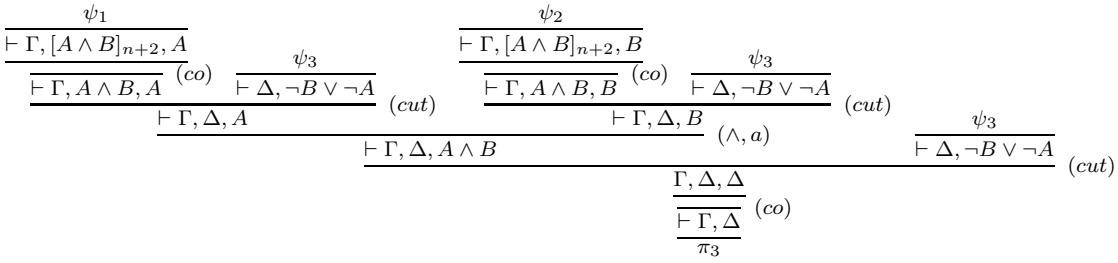
In un contesto logico la complessità di un calcolo può, quindi, misurarsi in termini di passi elementari della procedura di eliminazione del taglio.

Osservando le regole di riduzione delle derivazioni si vede che i passi di che fanno esplodere la complessità sono quelli relativi alla riduzione della contrazione. Vediamo ad esempio uno di tali passi: (co)-$(\wedge, a)$ sinistra:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\psi_1}{\vdash \Gamma, [A \wedge B]_{n+2}, A} \quad \dfrac{\psi_2}{\vdash \Gamma, [A \wedge B]_{n+2}, B}}{\vdash \Gamma, [A \wedge B]_{n+3}}\,(\wedge, a)}{\vdash \Gamma, A \wedge B}\,(co) \qquad \dfrac{\psi_3}{\vdash \Delta, \neg B \vee \neg A}}{\vdash \Gamma, \Delta}\,(cut)}{\pi_3}$$

[4]i passi di eliminazione del taglio si ottengono tramite l'applicazione di una serie di regole di riduzione (o trasformazione) delle derivazioni

si riduce in ψ:



Come si vede tale regola di trasformazione copia parti della dimostrazione di partenza.

Tali passi di eliminazione del taglio non permettono di controllare la complessità della riduzione. Da qui nasce l'esigenza di limitarli; per fare ciò, useremo come strumento *la logica lineare*.

In realtà, in questa tesi noi faremo vedere come la logica lineare nasca naturalmente dallo studio della semantica coerente della logica minimale[5]. In particolare dalla dimostrazione della validità dell'equazione

$$X \to Y = !X \multimap Y$$

dove $\to$ è l'implicazione minimale ! e $\multimap$ sono due operazioni definite sugli spazi coerenti. Queste operazioni possono essere internalizzate (cioè hanno un significato logico) e portano alla definizione dei connettivi della **logica lineare**.

Si introduce, poi, il calcolo dei sequenti lineare in cui spiccano le regole per i connettivi esponenziali che sono:

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

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

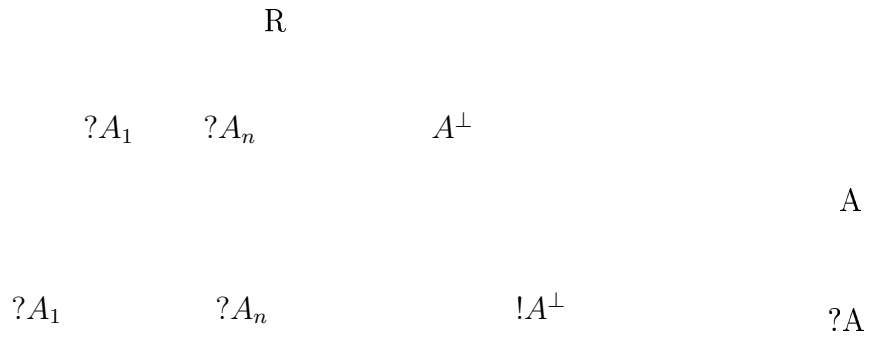
[5]La logica minimale è un frammento della logica classica in cui si limitano le regole strutturali

Come si osseva da queste regole, contrazione e weakening possono essere applicate solo su formule esponenziate, il che ne permette un maggior controllo.

A questo punto si deve definire una procedura di eliminazione del taglio per la logica lineare. Per definire tale procedura introduciamo i **proof-net** che sono una nuova classe di grafi che forniscono una rappresentazione più geometrica del calcolo. Si introducono le strutture di dimostrazione i cui nodi sono le regole della logica lineare, mentre gli archi sono etichettati dalle formule della logica lineare. Le strutture di dimostrazione sono proof-net sse soddisfano una proprietà puramente geometrica detta **criterio di correttezza** (su cui però non ci soffermiamo in questa sintesi).

Per tali proof-net è possibile definire una procedura di eliminazione del taglio che normalizza fortemente detta **lazy cut-elimination**. Per i proof-net un passo elementare di riduzione è la deformazione di un grafo.

Uno dei passi di riduzione è quello della dereliction, indicato con $[de]$.
Il passo elementare di riduzione $[de]$ consiste nell'apertura della "scatola esponenziale" nel modo seguente:



R

$?A_1$ $?A_n$ $A^{\perp}$

A

Possiamo osservare che ad ogni applicazione di una regola di promozione è associata una "scatola esponenziale $B^{!}$". Questo passo cancella la scatola esponenziale, il che produce una variazione di profondità [6] delle regole presenti nel grafo.

Girard in [14] mostra che è possibile modificare il sistema logico della logica lineare in maniera tale che la profondità delle regole rimanga costante durante la procedura di eliminazione del taglio. Questo "invariante" della riduzione è tra i principali ingredienti che permettono di ottenere per il sistema ELL una "buona" eliminazione del taglio.

Pertanto per ottenere un sistema con una eliminazione del taglio "buona" si devono limitare le regole per gli esponenziali, in particolare la regola di dereliction, e la si deve limitare in maniera tale che durante il processo di eliminazione del taglio la profondità delle regole rimanga costante.

Per studiare ELL partiamo dal calcolo dei sequenti di ILL (logica lineare intuizionista) di cui mettiamo in evidenza solo le regole per gli esponenziali:

$$\frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\ (!) \qquad \frac{\Gamma, A \vdash C}{\Gamma, !A \vdash C}\ (de)$$

$$\frac{\Gamma \vdash C}{\Gamma, !A \vdash C}\ (!w) \qquad \frac{\Gamma, !A, !A \vdash C}{\Gamma, !A \vdash C}\ (!co)$$

e definiamo in ILL una procedura di eliminazione del taglio, il **protocollo-q**,

[6] dove la profondità di una regola corrisponde al numero di scatole in cui tale regole è contenuta

che dimostreremo può essere simulato in un numero finito di passi di riduzione dei proof-net.

Le dimostrazioni di IELL sono le dimostrazioni di ILL che soddisfano la **condizione di stratificazione**:

1. Ogni occorrenza a sinistra di una formula esponenziale $!A$, principale in una regola di dereliction, ha un solo discendente in una regola di promozione.
2. Ogni occorrenza a sinistra di una formula $!A$, principale in un assioma, non ha alcun discendente in una regola di promozione.

In IELL il protocollo-q lascia invariata la profondità di ogni regola diversa dal cut. La profondità del cut non diminuisce durante la riduzione. Pertanto si possono eliminare i tagli in ordine di profondità crescente. Riferendoci a questa strategia di normalizzazione dimostriamo il seguente teorema:

Teorema 1.4. *IELL normalizza in tempo elementare; cioè c'è una funzione* $\theta : \mathbb{N} \longrightarrow \mathbb{N}$ *tale che:*

1. *Per ogni dimostrazione* π *di taglia* s *e profondità* d, $\theta(s,d)$ *limita il massimo numero di passi di eliminazione del taglio in* π.
2. $\theta(.,y)$ *è ricorsiva elementare per ogni* y.

Conseguenza di questo teorema è che in IELL sono rappresentabili *solo* le funzioni Turing-calcolabili in tempo elementare (cioè le funzioni di $\mathcal{E}_{time}$), dove la nozione di rappresentabilità è data dalla seguente definizione

Definizione 1.5. Sia f una funzione k-aria da interi a interi, si ha che f è rappresentabile (o programmabile) in IELL, se esiste un intero $p \geq 0$ e una dimostrazione $\dfrac{\overset{\pi}{\underbrace{N, \cdots, N}}}{\text{al } + k \text{ volte}} \vdash !^p N$ in IELL tale che $\forall n_1, \cdots, n_k$:

$$\pi(\omega_{n_1}, \cdots, \omega_{n_k}) \quad q - normalizza \quad in \quad !^p \omega_{f(n_1,\cdots,n_k)}$$

Se $p = 0$ π si dice piatta, se $p \neq 0$ π si dice obliqua.

dove ω_m è la dimostrazione associata all'intero m e $\pi(\omega_{n_1}, \cdots, \omega_{n_k})$ sta per π tagliata con ciascuna della ω_i.

Per concludere, rimane da dimostrare il seguente teorema:

Teorema 1.6. *Ogni funzione ricorsiva elementare da interi a interi è rappresentabile in IELL.*

Riferimenti bibliografici

[1] V.M. Abrusci. Logica classica. Dispense del corso di logica matematica, a.a. 2000-2001.

[2] C. Barcaglioni. Le dimostrazioni logihe come costruzioni : spazi coerenti ed esperienze, Febbraio 2001. Tesi di laurea in matematica.

[3] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational complexity*, 2:97–110, 1992.

[4] T. Brauner. Introduction to linear logic. *BRICS*, 1996.

[5] A. Cobham. The intrinsic computational difficulty of functions. *International Congress for Logic, Methodology, and the Philosophy of Science*, 1964.

[6] V. Danos. *La logique linéaire appliquée á l'étude de divers processus de normalisation (principalment du lambda-calcul).* PhD thesis, Paris 7, 1990.

[7] V. Danos and J.B. Joinet. Linear logic and elementary time. *In Procedings of the 1st international workshop on implicit computational complexity(ICC'99)*, 1999. Santa Barbara (Calif.).

[8] L. Tortora de Falco. *Réseaux cohérence et expériences obsessionnelles.* PhD thesis, Paris 7, 2000.

[9] L. Tortora de Falco. Teoria della ricorsività. Dispense del corso di IN3, corso di laurea in matematica, a.a. 2001-2001.

[10] L. Tortora de Falco. Additives of linear logic and normalization. *Theoretical computer science*, 294/3:489–524, febbraio-2003.

[11] J. Y. Girard, P. Tayolor, and Y. Lafont. *Proof and types.* Cambridge University Press, 1989.

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

[13] J.Y. Girard. Linear logic: its syntax and semantics. *Advances in linear logic*, London Mathematical Society Lecture Note Series 222:Cambridge University Press 1–42, 1995.

[14] J.Y. Girard. Light linear logic. *Information and Computation*, 14(3), 1998.

[15] J.B. Joinet. *Etude de la normalisation du calcul des sèquents à travers la logique linèaire.* PhD thesis, Paris 7, 1990.

[16] J.L. Krivine. *Lambda-calcul : types et models.* Masson, 1990.

[17] D. Leivant. A foundational delineation of computation feasibility. *Sixth Annual IEEE Symposium on Logic in Computer Science*, 1991.

[18] C.H. Papadimitriou. *computational complexity.* Addison-Wesley, 1994.

[19] D. Prawitz. *Natural deduction.* Almqvist and Wiksell, 1965.

[20] M.E. Szabo. *Collected papers of Gherard Gentzen.* North-Holland Publishing company, 1969. Amsterdam-London.