Linguaggi Formali e Calcolabilità

Dispense per il Corso di Fondamenti dell'Informatica

Agostino Dovier

Roberto Giacobazzi

Author address:

Dipartimento Scientifico-Tecnologico, Università di Verona, Strada Le Grazie 3, 37134 Verona, Italy

E-mail address: (dovier+giaco)@sci.univr.it

Contents

Chapter 1. Introduzione al corso 7

Chapter 2. Notazione e concetti di base 9
1. Insiemi 9
2. Relazioni e funzioni 10
3. Cenni di Logica Matematica 13
4. Ordinamenti 15
5. Cardinalità di insiemi 15

Part 1. Linguaggi formali 17

Chapter 3. Automi a stati finiti 19
1. Alfabeti e Linguaggi 19
2. Automi 21
3. Automi deterministici 22
4. Automi non-deterministici 24
5. Equivalenza tra DFA e NFA 25
6. Automi con ε-transizioni 27
7. Equivalenza di ε-NFA e NFA 28
8. Automi con output 29

Chapter 4. Espressioni regolari 31
1. Operazioni sui linguaggi 31
2. Definizione formale 31
3. Equivalenza tra DFA e ER 32

Chapter 5. Proprietà dei linguaggi regolari 37
1. Il "Pumping Lemma" 37
2. Proprietà di chiusura 39
3. Risultati di decidibilità 40
4. Il teorema di Myhill-Nerode 41

5. Minimizzazione di DFA 44

Chapter 6. Grammatiche libere dal contesto 49
1. Definizione formale 49
2. Linguaggio generato 50
3. Alberi di derivazione 52
4. Ambiguità delle derivazioni 53
5. Semplificazione 55
6. Forma normale di Chomsky 59
7. Forma normale di Greibach 60

Chapter 7. Proprietà dei linguaggi liberi dal contesto 63
1. Il pumping lemma per i linguaggi CF 63
2. Proprietà di chiusura 65
3. Algoritmi di decisione 66

Chapter 8. Le grammatiche regolari 69

Chapter 9. La gerarchia di Chomsky 73
1. Grammatiche di tipo 0 73
2. Grammatiche di tipo 1 73
3. Gerarchia 74

Part 2. Teoria della calcolabilità 77

Chapter 10. Alcuni sistemi formali e la Tesi di Church 79
1. Nozione informale di algoritmo 80

Chapter 11. Macchine di Turing 85
1. Descrizione modellistica e matematica 85
2. Funzioni calcolabili da MdT 92
3. MdT generalizzate 94

Chapter 12. Funzioni parziali ricorsive di Kleene & Robinson 95
1. Funzioni primitive ricorsive 95
2. Diagonalizzazione 103
3. Funzioni parziali ricorsive 106

Chapter 13. Tesi di Church 111

Chapter 14. Aritmetizzazione e universalità 115

1. Enumerazione delle MdT 115
2. Macchina di Turing Universale 120

Chapter 15. Problemi insolubili 123

Chapter 16. Calcolabilità e Linguaggi di Programmazione 127

Chapter 17. Espressività nei linguaggi imperativi 129
1. Funzioni primitive ricorsive e programmi FOR 129
2. Funzioni parziali ricorsive e programmi WHILE 129

Chapter 18. Il linguaggio WHILE+ 131
1. Sintassi 131
2. Semantica 131
3. Espressività di WHILE+ 132
4. Compilazione, interpretazione e specializzazione 132

Part 3. Teoria della Ricorsione 133

Chapter 19. Teoria matematica della ricorsione 135

Chapter 20. Insiemi ricorsivi e ricorsivamente enumerabili 137

Chapter 21. Il teorema di ricorsione e sue conseguenze 147

Chapter 22. Teorema di ricorsione e programmazione 153
1. Teorema di Rice e proprietà di programmi 153
2. Funzionali ricorsivi e trasformazioni di programmi 153

Chapter 23. Riducibilità funzionale 169
1. Insiemi creativi e produttivi 172

Bibliography 177

CHAPTER 1

Introduzione al corso

I fondamenti (teorici) dell'informatica sono lo studio dei modelli per il calcolo. Tale disciplina, la cui nascita si può far risalire al 1930 (prima dell'esistenza del moderno calcolatore) negli studi dei logici Church, Gödel, Kleene, Post e Turing è alla base degli sviluppi pratici e teorici del calcolo mediante calcolatore. Ad esempio è possibile garantire in modo formale l'esistenza di una macchina universale (calcolatore con programma software) o caratterizzare rigorosamente la nozione di programma come sequenza/insieme di istruzioni. Si dimostra l'esistenza di funzioni non calcolabili da nessun calcolatore e, di conseguenza, di problemi non risolubili in modo completo e automatico mediante calcolatore. Lo studio formale dei linguaggi generabili a partire da un dato alfabeto è invece alla base dello sviluppo delle tecniche di traduzione dei moderni linguaggi di programmazione. Tali metodologie e risultati devono costituire, per un Informatico, ma più in generale per chi utilizza professionalmente un calcolatore, una base di conoscenze indispensabile come le nozioni fondamentali di algebra e di analisi lo sono per il matematico.

Scopo del testo è dunque quello di fornire gli strumenti formali e le nozioni fondamentali per studiare problemi trattabili e non mediante calcolatore. In letteratura vi sono molti ottimi testi sull'argomento (si consulti la Bibliografia). La riorganizzazione a "moduli" di 60/90 ore del sistema universitario italiano rende tuttavia molti di questi testi sovrabbondanti (per esempio il libro di Hopcroft-Ullman [**3**]) oppure un po' troppo avanzati (per esempio il libro di Rogers [**10**]). Si è pertanto cercato di fornire una presentazione il più possibile adatta ad un corso di 60/90 ore del III anno del corso di laurea in Informatica. Gli unici prerequisiti sono una conoscenza di base di teoria elementare degli insiemi e di logica. Riteniamo pertanto che il testo possa essere

utilizzato in un corso di Fondamenti teorici dell'Informatica nei Corsi di laurea in Matematica, Fisica, Statistica e Ingegneria. Sono necessarie circa 30 ore per ciascuna delle tre parti in cui è organizzato. Il contenuto di diversi capitoli della Parte 2, tuttavia, viene talvolta anticipato in altri corsi (per esempio Programmazione). Omettendo parte delle dimostrazioni dei paragrafi 6.5, 6.6 e 6.7, la Parte 1 può essere svolta in circa 20 ore. In tal modo gran parte del testo può costituire il programma di un modulo di 60 ore circa.

Si intende ringraziare gli studenti del corso di laurea in Informatica dell'Università di Verona per i loro commenti che hanno aiutato a chiarire e correggere alcune parti. Si ringraziano Carla Piazza ed Andrea Fusiello per le discussioni relative alla riduzione della while calcolabilità in repeat calcolabilità (§ ??). Un grazie particolare ad Antonella Meneghetti per l'attenta rilettura delle bozze.

Gli autori saranno grati a tutti coloro che avessero la gentilezza di segnalarci sviste ed errori o volessero comunque far pervenire i loro commenti sul contenuto e la forma del testo.

Verona, 1 Gennaio 2000

Agostino Dovier

Roberto Giacobazzi

CHAPTER 2

Notazione e concetti di base

In questo breve capitolo viene stabilita la notazione impiegata nel testo relativamente ai concetti base di matematica e logica utilizzati. Con ciò non si intende fornire una presentazione auto-contenuta di tali concetti, per la quale si rimanda ai testi classici, quali ad esempio [**2, 7**].

1. Insiemi

Con lettere latine maiuscole denoteremo in genere *insiemi* di oggetti. Alcuni insiemi hanno un nome particolare che li identifica univocamente: ad esempio $\mathbb{N}$ rappresenta l'insieme dei numeri naturali ($\mathbb{N} = \{0, 1, 2, 3, \dots\}$).

$x \in A$ significa che x è un *elemento* dell'insieme A. Mediante la rappresentazione *intensionale* di insiemi $\{x \mid E(x)\}$ si identifica l'insieme costituito dagli x che soddisfano (rendono vera) una data espressione E, la quale dipende da x ed ha valori booleani. Se gli elementi di un insieme hanno un indice e sono tutti rappresentabili come una sequenza (anche infinita) di elementi x_i al variare di $i \in I$, allora l'insieme costituito da questi oggetti è rappresentato con $\{x_i\}_{i \in I}$.

$A \subseteq B$ indica che A è un *sottoinsieme* di B, ovvero che ogni elemento di A è anche elemento di B. Con la notazione $A \subset B$ si denoterà l'inclusione *stretta* ovvero vale che $A \subseteq B$ e $A \neq B$.

Con $\wp(A)$ si denota l'*insieme delle parti* dell'insieme A, ovvero l'insieme costituito da tutti i suoi sottoinsiemi: $\wp(A) = \{X \mid X \subseteq A\}$.

Unione ed intersezione di insiemi sono rispettivamente denotati con i simboli $\cup$ e $\cap$.

Il simbolo $\setminus$ rappresenta la *differenza insiemistica*: $A \setminus B = \{x \mid x \in A \ \wedge \ x \notin B\}$.

$\bar{A}$ denota il *complemento* di A, ovvero $x \in \bar{A}$ se e solo se $x \notin A$. Se assumiamo (come faremo spesso nel seguito) che gli insiemi

sono insiemi di numeri naturali, allora considerando $\mathbb{N}$ come insieme *universo*, si ha che $\bar{A} = \mathbb{N} \setminus A$.

Dati due o più insiemi, è possibile costruire coppie, triple, etc. di oggetti. In generale una *n-upla* di oggetti $x_1, \ldots, x_n$ di un insieme A è rappresentata con $\langle x_1, \ldots, x_n \rangle$. L'insieme delle n-uple di elementi di A è rappresentato con A^n e corrisponde al *prodotto cartesiano* di A n-volte. Insiemi diversi possono essere messi in prodotto cartesiano:

$$A_1 \times A_2 \times \cdots \times A_n = \{\langle x_1, \ldots, x_n \rangle \mid x_1 \in A_1, \ldots, x_n \in A_n\}$$

indica l'insieme delle n-uple di oggetti appartenenti rispettivamente agli insiemi $A_1, \ldots, A_n$.

2. Relazioni e funzioni

Una relazione (binaria) è un sottoinsieme del prodotto cartesiano di due insiemi; dati A e B, $\mathcal{R} \subseteq A \times B$ è una relazione su A e B. Ad esempio, la relazione di ordinamento sui numeri naturali "$\leq$" $\subseteq \mathbb{N} \times \mathbb{N}$ è definita nel modo seguente: $\leq = \{\langle x, y \rangle \mid x \in \mathbb{N}, y \in \mathbb{N}, x \leq y\}$. Il fatto che $\langle x, y \rangle \in \leq$ viene solitamente indicato con $x \leq y$.

Dato un insieme S, una relazione $R \subseteq S \times S$ è una relazione di *equivalenza* se gode delle proprietà:

***riflessiva*:** per ogni $a \in S$ si ha che $a \; R \; a$,
***simmetrica*:** per ogni $a, b \in S$ si ha che se $a \; R \; b$ allora $b \; R \; a$), e
***transitiva*:** per ogni $a, b, c \in S$ si ha che se $a \; R \; b$ e $b \; R \; c$ allora $a \; R \; c$.

È possibile definire relazioni n-arie, ovvero $\mathcal{R} \subseteq A_1 \times \ldots \times A_n$. Sia $k < n$. Una relazione $\mathcal{R}$ n-aria è una *funzione* di k argomenti se, dati $x_1 \in A_1, \ldots, x_k \in A_k$, esiste uno ed un solo $z_{k+1} \in A_{k+1}, \ldots, z_n \in A_n$ tale che $\langle x_1, \ldots, x_k, z_{k+1}, \ldots, z_n \rangle \in \mathcal{R}$. In questo caso, la funzione è definita in $A_1 \times \cdots \times A_k$ ed ha valori in $A_{k+1} \times \cdots \times A_n$, ovvero è del tipo:

$$\mathcal{R} : A_1 \times \cdots \times A_k \longrightarrow A_{k+1} \times \cdots \times A_n$$

Nel caso $n = 2$ e $k = 1$, otteniamo la definizione usuale di funzione di 1 argomento: $\mathcal{R} : A_1 \longrightarrow A_2$ come relazione binaria $\mathcal{R} \subseteq A_1 \times A_2$.

Denoteremo come $\vec{x}$ una generica n-upla (vettore) di oggetti. La lunghezza di tale n-upla si evincerà dal contesto. Una funzione è *iniettiva* se $f(\vec{x}_1) = f(\vec{x}_2)$ implica che $\vec{x}_1 = \vec{x}_2$ per ogni x_1, x_2; è

suriettiva se per ogni elemento $\vec{y}$ del *codominio* (nel caso sopra $A_{k+1} \times \cdots \times A_n$) esiste $\vec{x}$ nel *dominio* (nel caso sopra $A_1 \times \cdots \times A_k$) tale che $f(\vec{x}) = \vec{y}$.

Una funzione può non essere definita su alcuni argomenti, ovvero se la funzione di k argomenti è data dalla relazione $\mathcal{R} \subseteq A_1 \times \cdots \times A_n$, $k < n$, allora possono esistere degli $x \in A_i$ argomenti di $\mathcal{R}$ (ovvero $i \leq k$) tali che nessuna n-upla $\langle \ldots, x, \ldots \rangle$ avente x nella i-esima posizione appartiene a $\mathcal{R}$. In questo caso diremo che la funzione $\mathcal{R}$ è *parziale*. Al contrario, una funzione sempre definita su ogni argomento è detta *totale*.

NOTA 2.1. Normalmente si usa parlare di funzioni $f : A \longrightarrow B$ e $f(x)$ è definita per ogni elemento x dell'insieme A (dominio). Ad esempio, la funzione $f(x) = 1000 \texttt{ div } x$, ove `div` ritorna il quoziente della divisione intera, è definita da $A = \mathbb{N} \setminus \{0\}$ a $B = \mathbb{N}$. In questo testo, per mantenere un'analogia con le funzioni calcolabili mediante un calcolatore (che potrebbe non fornire risultato per certi valori di input), si preferisce parlare di $f : \mathbb{N} \longrightarrow \mathbb{N}$ e parlare di funzioni parziali.

Nel seguito faremo largo uso di funzioni parziali e totali. Al fine di evitare confusione, saremo soliti rappresentare le funzioni secondo la notazione usuale:

$$\mathcal{R} : A_1 \times \cdots \times A_k \longrightarrow A_{k+1} \times \cdots \times A_n$$

identificando in questo modo gli insiemi su cui vengono considerati gli argomenti e gli insiemi su cui la funzione restituisce risultati. In particolare, rappresenteremo con lettere minuscole latine le funzioni fotali; ad esempio:

$$f : A_1 \times \cdots \times A_k \longrightarrow A_{k+1} \times \cdots \times A_n$$

rappresenta una funzione totale di k argomenti, mentre rappresenteremo con lettere minuscole greche le funzioni parziali:

$$\varphi : A_1 \times \cdots \times A_k \longrightarrow A_{k+1} \times \cdots \times A_n$$

rappresenta una funzione parziale di k argomenti. Poiché tra l'insieme delle funzioni parziali vi sono anche le funzioni totali, nel caso non vi siano ipotesi sulla totalità della funzione considerata, rappresenteremo con lettere greche minuscole una generica funzione.

Per indicare che una funzione φ è definita in x, scriveremo $\varphi(x)\downarrow$. Analogamente, per indicare che una funzione φ non è definita in x, scriveremo $\varphi(x)\uparrow$, o equivalentemente $\varphi(x) = \uparrow$. Il simbolo $\uparrow$ rappresenta la non definizione della funzione φ. Supponiamo quindi di ammettere come possibile valore della funzione anche il valore indefinito $\uparrow$. Questo ci permette di definire funzioni parziali del seguente tipo:

$$\varphi : A_1 \times \cdots \times A_k \longrightarrow (A_{k+1} \cup \{\uparrow\}) \times \cdots \times (A_n \cup \{\uparrow\})$$

Data dunque una generica funzione (parziale o totale) φ chiameremo *dominio* della funzione φ l'insieme su cui φ è definita, ovvero l'insieme:

$$dom(\varphi) = \{x \mid \varphi(x)\downarrow\}$$

Analogamente chiameremo *immagine* o *range* di φ l'insieme dei risultati definiti della funzione, ovvero:

$$range(\varphi) = \{x \mid \exists z.\, \varphi(z) = x \neq\uparrow\}$$

Un valore della funzione φ è un elemento di $range(\varphi)$.

In genere la definizione di una funzione $f(X, Y) = X + Y$ crea un'associazione (relazione) tra il nome f della funzione ed il corpo della funzione stessa $X+Y$. Eseguire f non significa altro che eseguire il suo corpo. In alcuni contesti è utile potersi riferire ad una funzione senza essere costretti ad associare ad essa un nome; il concetto di funzione ha infatti un suo significato a prescindere dal nome che gli viene assegnato con la definizione. A questo scopo viene introdotta la cosidetta λ-notazione. Per esempio la funzione che riceve due argomenti e ne restituisce la somma può essere definita con: $\lambda X, Y\,.\, X + Y$ dove λ è un simbolo speciale, generalmente seguito da una lista di variabili separate da una virgola (i *parametri* della funzione). Il corpo della funzione segue il punto: si tratta di un'espressione in cui possono apparire le variabili introdotte dopo il simbolo λ. Queste variabili risultano *legate* all'interno dell'espressione (bound variables), mentre si definiscono *libere* quelle variabili che appaiono nel corpo e non appaiono dopo il simbolo lambda.

L'applicazione di una funzione definita con la λ-notazione ha la seguente forma: alla definizione della funzione racchiusa fra parentesi seguono tante espressioni quanti sono i parametri. Ad esempio: $(\lambda X, Y\,.\, X+Y)7\ 4$ associa 7 a X e 4 a Y.

3. Cenni di Logica Matematica

Un *linguaggio del prim'ordine* $\mathcal{L}$ è costituito da

- un'insieme di *simboli relazionali* o *predicativi*,
- un'insieme di *simboli funzionali*,
- un'insieme di *simboli di costante* e
- da un'insieme di *simboli logici*.

I simboli logici presenti in ogni linguaggio sono:

- le parentesi ")", "(" e la virgola ","
- un insieme numerabile di variabili $v_0, v_1, \ldots$
- un insieme di connettivi: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- i quantificatori $\forall$ (per ogni) ed $\exists$ (esiste).

Si ricorda che l'implicazione e la doppia implicazione sono superflue in quanto $\varphi \rightarrow \psi$ è equivalente a $(\neg\varphi) \vee \psi$ mentre $\varphi \leftrightarrow \psi$ è equivalente a $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$. Anche uno tra $\vee$ e $\wedge$ è superfluo (si vedano le leggi di De Morgan più avanti), mentre uno solo dei due quantificatori è sufficiente. Tuttavia tali operatori, di uso comune, sono usualmente accettati per fornire maggiore chiarezza alle formule.[1]

Un *termine* di un linguaggio del prim'ordine è definito ricorsivamente nel seguente modo:

1. una variabile v è un termine;
2. un simbolo costante c è un termine;
3. se $t_1, \ldots, t_m$ sono termini e f è un simbolo funzionale m-ario, allora $f(t_1, \ldots, t_m)$ è un termine.

Se p è un simbolo relazionale n-ario di $\mathcal{L}$ e se $t_1, \ldots, t_n$ sono termini allora $p(t_1, \ldots, t_n)$ è una *formula atomica*. Una *formula* di un linguaggio del prim'ordine è definita ricorsivamente nel seguente modo:

1. una formula atomica è una formula;
2. se φ è una formula allora $(\neg\varphi)$ è una formula;
3. se φ e ψ sono formule, anche $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$ e $(\varphi \rightarrow \psi)$ sono formule;
4. se φ è una formula e v è una variabile allora $\forall v(\varphi)$ e $\exists v(\varphi)$ sono formule.

[1] Un solo operatore, ad esempio il nor definito come $\overline{\vee}(\varphi, \psi) = \neg(\varphi \vee \psi)$ è in realtà sufficiente, in quanto $\neg\varphi = \overline{\vee}(\varphi, \varphi)$ e $\varphi \vee \psi = \overline{\vee}(\overline{\vee}(\varphi, \psi), \overline{\vee}(\varphi, \psi))$.

Per semplificare la notazione, si assumono le comuni convenzioni per eliminare, qualora ciò non generasse ambiguità, alcune coppie di parentesi dalle formule.

Il significato (la semantica) di una formula del prim'ordine è stabilito qualora venga fornita una *interpretazione*, associata ad un insieme non vuoto detto *dominio* (ad esempio, l'insieme dei numeri naturali $\mathbb{N}$)

- per i simboli di costante (ad esempio, la costante c è il numero 5, d il numero 10),
- per i simboli di funzione (ad esempio, il simbolo funzionale binario f sta a significare il '+' tra numeri interi) e
- venga assegnata un'interpretazione ai predicati sul dominio, ovvero, in parole povere, un valore di verità (*vero* oppure *falso*) agli atomi costruiti con i simboli predicativi e gli oggetti del dominio (ad esempio, $f(5,5) = 10$ è vero).

Fissata una interpretazione per un linguaggio del prim'ordine, usando la semantica degli operatori, è possibile estendere l'interpretazione ad ogni formula chiusa (ovvero, ogni variabile che occorre in essa è presente come quantificata) del linguaggio. Ad esempio, nell'interpretazione accennata sopra, la formula $\exists v(v = f(c,c))$ è vera in quanto per $v = 10$ è soddisfatta, mentre la formula $\forall v(v = f(c,c))$ è falsa (per esempio, si prenda $v = 11$).

Per quanto concerne l'utilizzo della logica del prim'ordine nel prosieguo, sarà fondamentale saper negare una formula logica, ovvero data una formula φ, scrivere in modo equivalente ma più leggibile la formula $\neg\varphi$.

Relativamente alle formule che non hanno quantificatori come simboli più "esterni", le regole da utilizzare sono le seguenti:

Doppia negazione: $\neg(\neg\varphi) = \varphi$
De Morgan 1: $\neg(\varphi \wedge \psi) = (\neg\varphi) \vee (\neg\psi)$
De Morgan 2: $\neg(\varphi \vee \psi) = (\neg\varphi) \wedge (\neg\psi)$

Si osservi in particolare che$\neg(\varphi \to \psi)$ è equivalente a $\varphi \wedge \neg\psi$. Per quanto riguarda la semplificazione della **Negazione di quantificatori** valgono:

- $\neg\exists\varphi = \forall\neg\varphi$
- $\neg\forall\varphi = \exists\neg\varphi$

Pertanto, ad esempio,

$$\neg\forall n\exists y(r(y) \wedge y > n \wedge \exists u\exists v(y = u + v \wedge \forall i(r(u + v^i))))$$

ove r è un simbolo unario di predicato, risulta essere:

$$\exists n\forall y(\neg r(y) \vee y \leq n \vee \forall u\forall v(y \neq u + v \vee \exists i(\neg r(u + v^i))))$$

Esplicitando le implicazioni, quest'ultima diventa:

$$\exists n\forall y((r(y) \wedge y > n) \rightarrow \forall u\forall v(y = u + v \rightarrow \exists i(\neg r(u + v^i))))$$

4. Ordinamenti

5. Cardinalità di insiemi

Il concetto stesso di numero naturale
+++ Ordinali
+++ teorema di cantor

Part 1

Linguaggi formali

CHAPTER 3

Automi a stati finiti

In questo capitolo sarà presentato il concetto di automa a stati finiti. Si partirà dal formalismo più semplice (automi a stati finiti deterministici) per poi presentare diverse tipologie di automa, che si dimostreranno essere tutte equivalenti dal punto di vista delle potenzialità del loro utilizzo, ovvero del riconoscimento di un dato linguaggio.

1. Alfabeti e Linguaggi

Un *simbolo* è un'entità primitiva astratta che non sarà definita formalmente (come punto, linea, ecc.). Lettere e caratteri numerici sono esempi di simboli.

Una *stringa* (o *parola*) è una sequenza finita di simboli giustapposti (uno dietro l'altro). Ad esempio, se a, b, c sono simboli, *abcba* è una stringa.

La *lunghezza* di una stringa w, denotata come $|w|$, è il numero di *occorrenze* di simboli che compongono una stringa. Ad esempio, $|abcba| = 5$. La stringa vuota, denotata con ε è la stringa costituita da zero simboli: $|\varepsilon| = 0$.

Sia $w = a_1 \cdots a_n$ una stringa. Ogni stringa della forma:

- $a_1 \cdots a_j$, con $j \in \{1, \ldots, n\}$ è detta un *prefisso* di w;
- $a_i \cdots a_n$, con $i \in \{1, \ldots, n\}$ è detta un *suffisso* di w;
- $a_i \cdots a_j$, con $i, j \in \{1, \ldots, n\}$, $i \leq j$, è detta una *sottostringa* di w;
- ε è sia prefisso che suffisso che sottostringa di w.

Si osservi che se $n = 0$, allora $w = \varepsilon$. Dunque ε è sia prefisso che suffisso di ε.

Ad esempio, i prefissi di abc sono ε, a, ab, e abc. I suffissi sono ε, c, bc, e abc. Le sottostringhe sono:

$$\begin{array}{ccccc} & & \varepsilon & & \\ & a & ab & abc & \\ & & b & bc & \\ & & & c & \end{array}$$

Un prefisso, un suffisso o una sottostringa di una stringa, quando non sono la stringa stessa, sono detti *propri*.

ESERCIZIO 3.1. Sia data una stringa w di lunghezza $|w| = n$. Quanti sono i suoi prefissi, i suoi suffissi e le sue sottostringhe?

La *concatenazione* di due stringhe v e w è la stringa vw che si ottiene facendo seguire alla prima la seconda. La concatenazione è una operazione (associativa) che ammette come identità la stringa vuota ε.

Un *alfabeto* Σ è un insieme finito di simboli. Un *linguaggio formale* (in breve linguaggio) è un insieme di stringhe di simboli da un alfabeto Σ. L'insieme vuoto $\emptyset$ e l'insieme $\{\varepsilon\}$ sono due linguaggi formali di qualunque alfabeto. Con Σ^* verrà denotato il linguaggio costituito da tutte le stringhe su un fissato alfabeto Σ. Dunque

$$\Sigma^* = \{a_1 \cdots a_n : n \geq 0, a_i \in \Sigma\}$$

Ad esempio, se $\Sigma = \{0\}$, allora $\Sigma^* = \{\varepsilon, 0, 00, 000, \dots\}$; se $\Sigma = \{0, 1\}$, allora $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$.

ESERCIZIO 3.2. Si provi che se $\Sigma \neq \emptyset$, allora Σ^* è numerabile. Definendo inoltre:

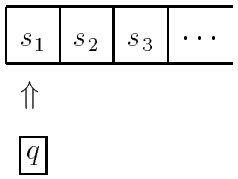
$$\begin{cases} \Sigma^0 &= \{\varepsilon\} \\ \Sigma^{n+1} &= \{ax : a \in \Sigma, x \in \Sigma^n\} \end{cases}$$

si osservi che $\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$. Qual è la cardinalità di Σ^i ?

2. Automi

Un automa a stati finiti è un modello matematico di un sistema avente input, ed eventualmente output, a valori discreti. Il sistema può essere in uno stato tra un insieme finito di stati possibili. L'essere in uno stato gli permette di tener traccia della storia precedente. Un buon esempio di automa a stati finiti è costituito dall'ascensore: l'input è la sequenza di tasti premuti, mentre lo stato è il piano in cui si trova.

Un automa a stati finiti si può rappresentare mediante una testina che legge, spostandosi sempre nella stessa direzione, un nastro di lunghezza illimitata contenente dei simboli. La testina si può trovare in un certo *stato*; a seconda dello stato q e del simbolo s_i letto, la testina si porta in un altro stato (o rimane nello stesso) e si sposta a destra per apprestarsi a leggere il simbolo sucessivo:

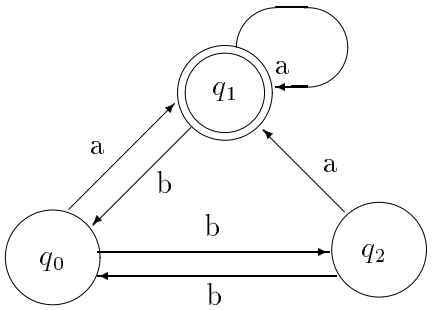


Quando la lettura dei simboli termina, a seconda dello stato raggiunto dalla testina, l'automa fornisce un risultato di accettazione o di refutazione della stringa (parola) letta.

Il comportamento dell'automa si definisce in maniera univoca mediante una tabella, detta *matrice di transizione*, come ad esempio:

	a	b
q_0	q_1	q_2
q_1	q_1	q_0
q_2	q_1	q_0

Un'automa siffatto si rappresenta bene anche con un grafo della forma seguente:



Nel prossimo paragrafo si fornirà una definizione formale del concetto ora esposto, necessaria per una trattazione precisa dell'argomento.

NOTA 3.3. Il fatto che, a differenza delle MdT (§ 11.1), la testina non possa produrre degli output (eventualmente sul nastro) non è essenziale (si vedano, ad esempio, le macchine di Moore e di Mealy (§ 8 e [**3**]). Intuitivamente, anche se scrivesse qualcosa sul nastro, non lo potrebbe più riutilizzare.

Neppure il permettere la bidirezionalità aumenterebbe di fatto la potenzialità del dispositivo (two-way automata [**3**]). Intuitivamente, la testina andrebbe su e giù ma sempre sugli stessi dati e con un controllo finito.

E' la somma delle due caratteristiche che permette di passare da questo formalismo al più potente formalismo di calcolo costituito dalla MdT.

3. Automi deterministici

Un *automa a stati finiti deterministico* (DFA) è una quintupla $\langle Q, \Sigma, \delta, q_0, F\rangle$ dove:

- Q è un insieme finito di *stati*;
- Σ è un alfabeto (alfabeto di input);
- $\delta : Q \times \Sigma \longrightarrow Q$ è la *funzione di transizione*;
- q_0 è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli *stati finali*.

NOTAZIONE 3.4. Useremo p, q, r con o senza pedici per denotare stati, P, Q, R, S per insiemi di stati, a, b con o senza pedici per denotare

simboli di Σ, x, y, z, u, v, w sempre con o senza pedici per denotare stringhe.

Dalla funzione δ si ottiene in modo univoco la funzione $\hat{\delta} : Q \times \Sigma^* \longrightarrow Q$ nel modo seguente:

$$\begin{cases} \hat{\delta}(q, \varepsilon) &= q \\ \hat{\delta}(q, wa) &= \delta(\hat{\delta}(q, w), a) \end{cases}$$

Una stringa x è detta essere *accettata* da un DFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ se $\hat{\delta}(q_0, x) \in F$. Il *linguaggio accettato da* M, denotato come $L(M)$ è l'insieme delle stringhe accettate, ovvero:

$$L(M) = \{x \in \Sigma^* : \hat{\delta}(q_0, x) \in F\}.$$

Un linguaggio L è detto *regolare* se è accettato da qualche DFA, ovvero se esiste M tale che $L = L(M)$.

ESEMPIO 3.5. $\emptyset$ e Σ^* sono linguaggi regolari. Sia $\Sigma = \{s_1, \ldots, s_n\}$: un automa M_0 che riconosce il linguaggio $\emptyset$ (ovvero: nessuna stringa è accettata) è il seguente:

	s_1	$\ldots$	s_n
q_0	q_0	$\ldots$	q_0

ove $F = \emptyset$. Infatti, poichè $\forall x \; x \notin \emptyset$, si ha che:

$$(\forall x \in \Sigma^*)(\hat{\delta}(q_0, x) \notin F).$$

Un automa per Σ^*, è invece l'automa M_1:

	s_1	$\ldots$	s_n
q_0	q_0	$\ldots$	q_0

ove $F = \{q_0\}$. Si dimostra facilmente infatti, per induzione su $|x|$ che

$$(\forall x \in \Sigma^*)(\hat{\delta}(q_0, x) = q_0).$$

ESERCIZIO 3.6. Si determini il linguaggio accettato dai seguenti automi rapresentati mediante la matrice di transizione (in tutti $F = \{q_1\}$).

1.

	0	1
q_0	q_1	q_2
q_1	q_1	q_1
q_2	q_1	q_0

2.

	0	1
q_0	q_2	q_0
q_1	q_2	q_0
q_2	q_1	q_2

ESERCIZIO 3.7. Si verifichi che i seguenti linguaggi, con $\Sigma = \{0, 1\}$, sono regolari:

1. l'insieme di tutte le stringhe aventi tre 0 consecutivi;
2. l'insieme di tutte le stringhe tali che il penultimo simbolo è 0;
3. l'insieme di tutte le stringhe tali che il terzultimo simbolo è 0;
4. l'insieme di tutte le stringhe tali che, se interpretate come numero intero (binario), sono divisibili per 2;
5. l'insieme di tutte le stringhe tali che, se interpretate come numero intero (binario), sono divisibili per 4;
6. l'insieme di tutte le stringhe tali che, se interpretate come numero intero (binario), sono divisibili per 5.

LEMMA 3.8. *Siano vx e wx due stringhe e $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ un DFA. Allora, $\hat{\delta}(q_0, v) = \hat{\delta}(q_0, w)$ implica $\hat{\delta}(q_0, vx) = \hat{\delta}(q_0, wx)$.*

PROOF. (Esercizio) □

LEMMA 3.9. *Sia $L \subseteq \Sigma^*$ un linguaggio accettato da un DFA M. Allora esiste $M' = \langle Q', \Sigma', \delta', q_0', F'\rangle$ tale che $\Sigma' = \Sigma$ e $L(M') = L(M)$.*

PROOF. (Esercizio) □

4. Automi non-deterministici

Un *automa a stati finiti non-deterministico* (NFA) è una quintupla $\langle Q, \Sigma, \delta, q_0, F\rangle$ dove Q, Σ, q_0 e $F \subseteq Q$ mantengono il significato visto

per gli automi deterministici, mentre la *funzione di transizione* δ è ora definita

$$\delta : Q \times \Sigma \longrightarrow \wp(Q)\,.$$

Si osservi che è ora ammesso: $\delta(q,a) = \emptyset$ per qualche $q \in Q$ ed $a \in \Sigma$. Anche per gli NFA dalla funzione δ si ottiene in modo univoco la funzione $\hat{\delta} : Q \times \Sigma^* \longrightarrow \wp(Q)$ nel modo seguente:

$$\begin{cases} \hat{\delta}(q,\varepsilon) &= \{q\} \\ \hat{\delta}(q,wa) &= \bigcup_{p\in\hat{\delta}(q,w)} \delta(p,a) \end{cases}$$

Una stringa x è detta essere *accettata* da un NFA $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ se $\hat{\delta}(q_0,x) \cap F \neq \emptyset$. Il *linguaggio accettato da* M è l'insieme delle stringhe accettate, ovvero:

$$L(M) = \{x \in \Sigma^* : \hat{\delta}(q_0,x) \cap F \neq \emptyset\}\,.$$

Mentre la rappresentazione mediante tabella degli automi non deterministici è profondamente diversa da quella per i deterministici (in ogni casella si deve inserire ora un insieme di stati), la rappresentazione a grafo rimane pressoché immutata. L'unica differenza è che da un nodo possono uscire più archi (o nessuno) etichettati dallo stesso simbolo.

Dal punto di vista invece del modello con testina e nastro, il non-determinismo va immaginato come la possibilità *contemporanea* di procedere la computazione in ognuno degli stati raggiunti. Si apre dunque un gran numero di computazioni virtuali parallele.

5. Equivalenza tra DFA e NFA

In questo paragrafo si mostrerà che i linguaggi accettati dai DFA e dagli NFA coincidono. Poiché un DFA si può vedere come un NFA in cui $\delta(q,a)$ restituisce sempre insiemi costituiti da un solo stato (detti anche *singoletti*), si ha che ogni linguaggio regolare è un linguaggio accettato da un qualche NFA. Per l'implicazione inversa ci serve il seguente teorema:

TEOREMA 3.10 (Rabin-Scott, 1959). *Sia* $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ *un NFA. Allora esiste un DFA* M' *tale che* $L(M) = L(M')$.

PROOF. Definisco $M' = \langle Q', \Sigma', \delta', q_0', F'\rangle$ come segue:

- $\Sigma' = \Sigma$;
- $Q' = \wp(Q)$ (sarebbe più preciso definire $Q' = \{q_1, \ldots, q_{2^{|Q|}}\}$ e poi stabilire una corrispondenza biunivoca fra tali stati e gli elementi di $\wp(Q)$. Tuttavia, con tale abuso sintattico la dimostrazione diventa molto più snella);
- $q'_0 = \{q_0\}$;
- $F' = \{P \subseteq Q : P \cap F \neq \emptyset\}$;
- $\delta'(P, a) = \bigcup_{p \in P} \delta(p, a)$, per $P \in \wp(Q)$.

Mostriamo per induzione sulla lunghezza della stringa di input x che

$$\hat{\delta}(q_0, x) = \hat{\delta}'(q'_0, x)$$

Base: Per $|x| = 0$ il risultato è banale, poichè $q'_0 = \{q_0\}$ e $x = \varepsilon$.
Passo: Supponiamo che l'ipotesi induttiva valga per tutte le stringhe x tali che $|x| \leq m$. Sia xa una stringa di lunghezza $m + 1$. Allora:

$$\begin{aligned}
\hat{\delta}'(q'_0, xa) &= \delta'(\hat{\delta}'(q'_0, x), a) && \text{Def. di } \hat{\delta}' \\
&= \delta'(\hat{\delta}(q_0, x), a) && \text{Ip. ind.} \\
&= \textstyle\bigcup_{p \in \hat{\delta}(q_0, x)} \delta(p, a) && \text{Def. di } \delta' \\
&= \hat{\delta}(q_0, xa) && \text{Def. di } \hat{\delta}
\end{aligned}$$

Il teorema segue dal fatto che $x \in L(M)$ sse[1] $\hat{\delta}(q_0, x) \cap F \neq \emptyset$ sse $\hat{\delta}'(q'_0, x) \cap F \neq \emptyset$ sse $\hat{\delta}'(q'_0, x) \in F'$ sse $x \in L(M')$. □

ESERCIZIO 3.11. Si determini il DFA equivalente al NFA:

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

ove $F = \{q_2\}$. Qual è il linguaggio accettato?

[1] Con "sse" si abbrevierà "se e solo se".

6. Automi con ε-transizioni

In questo paragrafo sarà presentato un terzo tipo di automa che estende il modello non-deterministico ma che, come sarà mostrato nel Teorema 3.12, ne è equivalente dal punto di vista dei linguaggi accettati.

Un *NFA con ε-transizioni* è una quintupla $\langle Q, \Sigma, \delta, q_0, F\rangle$ dove: Q, Σ, q_0 e $F \subseteq Q$ sono come per gli automi non deterministici, mentre la *funzione di transizione* δ è ora definita

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \longrightarrow \wp(Q)\,.$$

L'idea è che da uno stato è permesso passare ad un altro stato anche senza "leggere" caratteri di input.

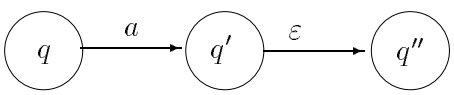
La costruzione della funzione $\hat{\delta} : Q \times \Sigma^* \longrightarrow \wp(Q)$ nel caso dei ε-NFA risulta leggermente più complessa che nei casi precedenti. Per far ciò si introduce la funzione *ε-closure* che, applicata ad uno stato, restituisce l'insieme degli stati raggiungibili da esso (compreso sè stesso) mediante ε-transizioni. La costruzione di tale funzione è equivalente a quella che permette di conoscere i nodi raggiungibili da un nodo in un grafo e può facilmente essere calcolata a partire dalla funzione δ (un arco $p \to q$ si ha quando $q \in \delta(p, \varepsilon)$). Il concetto di ε-closure si estende in modo intuitivo ad insiemi di stati:

$$\varepsilon\text{-closure}(P) = \bigcup_{p \in P} \varepsilon\text{-closure}(p)$$

$\hat{\delta}$ si può ora definire nel modo seguente:

$$\begin{cases} \hat{\delta}(q, \varepsilon) &= \varepsilon\text{-closure}(q) \\ \hat{\delta}(q, wa) &= \bigcup_{p \in \hat{\delta}(q,w)} \varepsilon\text{-closure}(\delta(p, a)) \end{cases}$$

Si noti che in questo caso $\hat{\delta}(q, a)$ può essere diverso da $\delta(q, a)$. Ad esempio, nell'automa:



Si ha che $\delta(q,a) = \{q'\}$, mentre $\hat{\delta}(q,a) = \bigcup_{p \in \hat{\delta}(q,\varepsilon)} \varepsilon\text{-closure}(\delta(p,a)) = \{q', q''\}$.

Definiamo dunque il *linguaggio accettato* dall'automa $L(M) = \{x \in \Sigma^* : \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$.

7. Equivalenza di ε-NFA e NFA

Ogni NFA è, per definizione, un caso particolare di un ε-NFA. Con il seguente teorema mostreremo che la classe dei linguaggi riconosciuti dagli automi ε-NFA non estende propriamente quella dei linguaggi riconosciuti da automi NFA:

TEOREMA 3.12. *Sia $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ un ε-NFA. Allora esiste un NFA M' tale che $L(M) = L(M')$.*

PROOF. Definisco $M' = \langle Q', \Sigma', \delta', q_0', F'\rangle$ come segue:

- $Q' = Q$,
- $\Sigma' = \Sigma$,
- $q_0' = q_0$,
- $F' = \begin{cases} F \cup \{q_0\} & \text{Se } \varepsilon\text{-closure}(q_0) \cap F \neq \emptyset \\ F & \text{altrimenti} \end{cases}$
- $\delta'(q,a) = \hat{\delta}(q,a)$.

Dobbiamo mostrare che

$$\hat{\delta}(q_0, x) \cap F \neq \emptyset \quad \text{sse} \quad \hat{\delta}'(q_0', x) \cap F' \neq \emptyset \quad (1)$$

Se $x = \varepsilon$, si ha che: $\hat{\delta}(q_0, \varepsilon) = \varepsilon\text{-closure}(q_0)$ per definizione.

D'altro canto: $\hat{\delta}'(q_0, \varepsilon) = \{q_0\}$. Per definizione di F', si ha che $q_0 \in F'$ sse $\varepsilon\text{-closure}(q_0) \neq \emptyset$.

Mostreremo per induzione su $|x| \geq 1$ che $\hat{\delta}'(q_0', x) = \hat{\delta}(q_0, x)$.
Base: $|x| = 1$. Allora $x = a$ per qualche simbolo $a \in \Sigma$. Ma allora $\hat{\delta}'(q_0', a) = \delta'(q_0', a) = \hat{\delta}(q_0, a)$ per definizione.

Passo: Assumiamo che la tesi valga per tutte le stringhe x tali che $1 \leq |x| \leq m$. Sia xa una stringa di lunghezza $m+1$. Allora:

$$\begin{aligned}
\hat{\delta}'(q_0', xa) &= \bigcup_{p \in \hat{\delta}'(q_0', x)} \delta'(p, a) && \text{def. di } \hat{\delta}' \\
&= \bigcup_{p \in \hat{\delta}'(q_0', x)} \hat{\delta}(p, a) && \text{def. di } \delta' \\
&= \bigcup_{p \in \hat{\delta}(q_0, x)} \hat{\delta}(p, a) && \text{ip. ind.} \\
&= \bigcup_{p \in \hat{\delta}(q_0, x)} \varepsilon\text{-closure}(\delta(p, a)) && \text{def. di } \hat{\delta} \text{ e } (*) \\
&= \hat{\delta}(q_0, xa) && \text{def. di } \hat{\delta}
\end{aligned}$$

$(*)$ si osservi che per definizione di $\hat{\delta}$, la ε-closure(p) è contenuta in $\hat{\delta}(q_0, x)$.

Per concludere la dimostrazione, si deve mostrare la proprietà (1), ovvero che $\hat{\delta}'(q_0', x)$ contiene uno stato di F' sse $\hat{\delta}(q_0, x)$ contiene uno stato di F.

Poiché $F \subseteq F'$ e $\hat{\delta}'(q_0', x) = \hat{\delta}(q_0, x)$ l'implicazione $(\leftarrow)$ deriva immediatamente.

Per l'altra implicazione, l'unico problema si avrebbe nel caso:

1. $q_0 \in \hat{\delta}'(q_0', x)$,
2. $q_0' \in F'$,
3. $q_0 \notin F$.

Poiché $\hat{\delta}'(q_0', x) = \hat{\delta}(q_0, x)$, si ha che $q_0 \in \hat{\delta}(q_0, x)$. Ma, per definizione di $\hat{\delta}$, anche ogni elemento della sua ε-closure appartiene a $\hat{\delta}(q_0, x)$. Tale ε-closure, poichè $q_0 \in F'$, interseca F. □

COROLLARIO 3.13. Le classi di linguaggi riconosciute da DFA, NFA e ε-NFA coincidono (linguaggi regolari).

PROOF. Immediato dai Teoremi 3.10 e 3.12. □

8. Automi con output

Gli automi visti fin qui, tutti equivalenti dal punto di vista della classe di linguaggi accettati (riconosciuti) non sono in grado di fornire alcun tipo di output se non il messaggio (booleano) del raggiungimento di uno stato finale o meno. Esistono in letteratura due tipi di automi provvisti di output: le macchine di *Moore* e quelle di *Mealy*. Tali

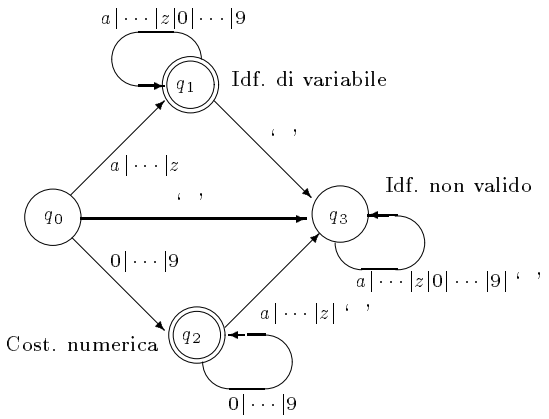
macchine sono automi a stati finiti deterministici (e dunque ereditano la classe di linguaggi accettati) che possiedono un alfabeto di output:

- le macchine di Moore forniscono in output un simbolo in funzione di ogni stato raggiunto. Poiché per una stringa di n elementi si raggiungono al più $n+1$ stati (compreso quello iniziale), si può ottenere un output di (al più) $n+1$ caratteri.
- le macchine di Mealy forniscono in output un simbolo ogni volta che avviene una transizione $\delta(q,a)$. In questo caso al più n caratteri saranno forniti in output.

Gli output possono essere utili per differenziare stati finali. Così, ad esempio, una stringa accettata può anche avere una più precisa classificazione (si veda l'Esempio 3.14).

Le macchine di Moore e Mealy vengono impiegate per la sintesi di reti sequenziali e sono formalismi tra loro equivalenti. Per maggiori dettagli, si veda [**3**].

ESEMPIO 3.14. Il seguente automa è una macchina di Moore che, oltre a riconoscere se una stringa sia o meno un elemento di una espressione numerica, segnala se si tratta di un numero intero o di un identificatore di variabile. L'output è rappresentato dalle scritte (Idf. di variabile, ecc.) posizionate accanto ai vari stati.



Si osservi come esso racchiuda in sè due DFA privi di output.

CHAPTER 4

Espressioni regolari

In questo capitolo verrà illustrato un modo alternativo per descrivere i linguaggi accettati da automi finiti (linguaggi regolari).

1. Operazioni sui linguaggi

Sia Σ un alfabeto e L, L_1, L_2 insiemi di stringhe di Σ^*. La *concatenazione di L_1 e L_2*, denotata come L_1L_2 è l'insieme:

$$L_1L_2 = \{xy \in \Sigma^* : x \in L_1, y \in L_2\}.$$

Definiamo ora

$$\begin{cases} L^0 &= \{\varepsilon\} \\ L^{i+1} &= LL^i \end{cases}$$

Si osservi che $L^0 = \{\varepsilon\} \neq \emptyset$. La *chiusura (di Kleene) di L*, denotata come L^* è l'insieme:

$$L^* = \bigcup_{i \geq 0} L^i$$

mentre la *chiusura positiva di L*, denotata come L^+ è l'insieme: $L^+ = \bigcup_{i \geq 1} L^i$ (si verifica immediatamente che $L^+ = LL^*$, dunque tale operatore, seppur comodo, non è essenziale). Si osservi come la definizione sia consistente con la nozione di Σ^* fin qui adoperata.

2. Definizione formale

Sia Σ un alfabeto. Le *espressioni regolari su Σ* e gli *insiemi* che esse denotano sono definiti ricorsivamente nel modo seguente:

1. $\emptyset$ è una espressione regolare che denota l'insieme vuoto.
2. ε è una espressione regolare che denota l'insieme $\{\varepsilon\}$.
3. Per ogni simbolo $a \in \Sigma$, a è una espressione regolare che denota l'insieme $\{a\}$.

4. Se r e s sono espressioni regolari denotanti rispettivamente gli insiemi R ed S, allora $(r+s)$, (rs), e (r^*) sono espressioni regolari che denotano gli insiemi $R \cup S$, RS, e R^* rispettivamente.

Se r è una espressione regolare, indicheremo con $L(r)$ il linguaggio denotato da r. Tra espressioni regolari valgono delle uguaglianze che permettono la loro manipolazione algebrica. Varrà che $r = s$ se e solo se $L(r) = L(s)$.

NOTAZIONE 4.1. Qualora non si crei ambiguità, nello scrivere espressioni regolari saranno omesse le parentesi. Le precedenze degli operatori sono le stesse dell'algebra, intepretando * come esponente e la concatenazione come prodotto (si veda Esercizio 4.2). E' ammesso usare l'abbreviazione r^+ per l'espressione regolare rr^*.

ESERCIZIO 4.2. Si provino o refutino le seguenti identità tra espressioni regolari:

1. $r + s = s + r$;
2. $(r + s) + t = r + (s + t)$;
3. $r(st) = r(st)$;
4. $r(s + t) = rs + rt$;
5. $(r + s)t = rt + st$;
6. $\emptyset^* = \varepsilon$;
7. $(r^*)^* = r^*$;
8. $(\varepsilon + r^*r) = r^*$;
9. $(\varepsilon + r)^* = r^*$;
10. $(r^*s^*)^* = (r + s)^*$;
11. $(rs + r)^*r = r(sr + r)^*$;
12. $s(rs + s)^*r = rr^*s(rr^*s)^*$;
13. $(r + s)^* = r^* + s^*$;
14. $(r^* + s^*)^* = (rs)^*$.

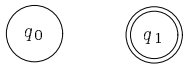
3. Equivalenza tra DFA e ER

TEOREMA 4.3 (McNaughton,Yamada, 1960). *Sia r una espressione regolare. Allora esiste un ε-NFA M tale che $L(M) = L(r)$.*

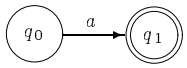
PROOF. Costruiremo un ε-NFA siffatto, con un unico stato finale, per induzione sulla complessità strutturale dell'espressione regolare r.
Base: Ci sono tre casi base:

- l'automa:

riconosce il linguaggio $\{\varepsilon\}$;
- l'automa



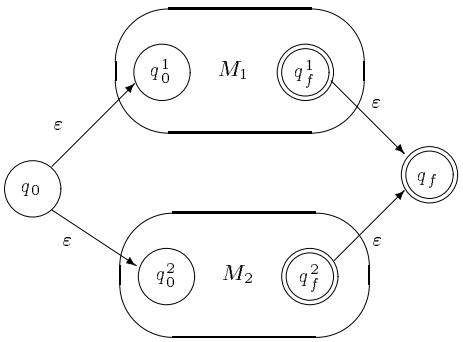
riconosce il linguaggio $\emptyset$;
- l'automa



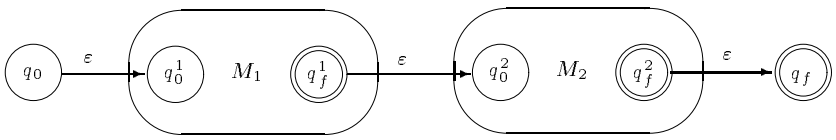
riconosce il linguaggio $\{a\}$.

Passo: Anche qui abbiamo tre casi da analizzare:

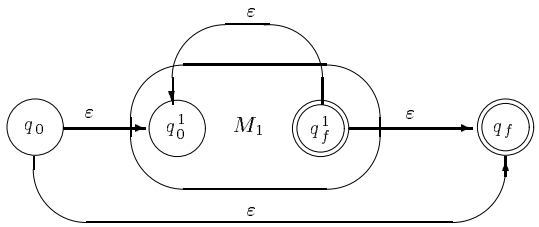
- $r = r_1 + r_2$. Per $i = 1, 2$, sia M_i, con stato iniziale q_0^i e stato finale q_f^i l'automa che riconosce $L(r_i)$. L'esistenza di tali automi è assicurata dall'ipotesi induttiva. Il seguente automa, con stato iniziale q_0 e stato finale q_f riconosce il linguaggio $L(r) = L(r_1) \cup L(r_2)$:



- $r = r_1 r_2$. Per $i = 1, 2$, sia M_i, con stato iniziale q_0^i e stato finale q_f^i l'automa che riconosce $L(r_i)$. L'esistenza di tali automi è assicurata dall'ipotesi induttiva. Il seguente automa, con stato iniziale q_0 e stato finale q_f riconosce il linguaggio $L(r_1)L(r_2)$:



- $r = r_1^*$. Sia M_1, con stato iniziale q_0^1 e stato finale q_f^1 l'automa che riconosce $L(r_1)$. L'esistenza di tale automa è assicurata dall'ipotesi induttiva. Il seguente automa, con stato iniziale q_0 e stato finale q_f riconosce $L(r) = (L(r_1))^*$:



Le dimostrazioni che tali automi riconoscono esattamente i linguaggi a loro assegnati è lasciata per esercizio. □

Si vuole ore mostrare il contrario, ovvero che preso a caso un automa M (senza perdita di generalità è sufficiente considerare un DFA), il linguaggio accettato da M è denotato da una espressione regolare. Si darà una dimostrazione costruttiva di questo fatto. Per una dimostrazione alternativa (e più rigorosa), si veda [**3**].

TEOREMA 4.4. *Sia M un DFA. Allora esiste una espressione regolare r tale che $L(M) = L(r)$.*

PROOF. (Traccia) Si costruirà, a partire da $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ un sistema di equazioni in cui ad ogni stato q_i viene associata una variabile Q_i. Si fornirà una metodologia per calcolare una soluzione a tale sistema. La soluzione calcolata assegnerà una espressione regolare r_i ad ogni variabile Q_i, che ne denota il linguaggio. Se $F = \{q_{j_1}, \ldots, q_{j_p}\}$, avremo che:

$$r = r_{j_1} + \cdots + r_{j_p}$$

e $L(r)$ è il linguaggio accettato dall'automa.

Per ogni $q_i \in Q$, siano:

- $q_{i_1}, \ldots, q_{i_h}$ gli stati diversi da q_i tali che esiste $a_{i_j} \in \Sigma$ t.c. $\delta(q_{i_j}, a_{i_j}) = q_i$;
- $b_{i_1}, \ldots, b_{i_k}$ i simboli tali che $\delta(q_i, b_{i_j}) = q_i$.

Assegnamo a q_i l'equazione:

$$Q_i \quad = \quad (Q_{i_1} a_{i_1} + \cdots + Q_{i_h} a_{i_h})(b_{i_1} + \cdots + b_{i_k})^*$$

Nel caso che q_i fosse q_0 bisogna aggiungere $+\varepsilon$ alla parte destra.

Un tale sistema si può risolvere mediante manipolazioni algebriche sulle espressioni regolari presenti ed usando il metodo di *sostituzione* (standard) qualora (come accade all'inizio) Q_i non compaia nella parte destra dell'equazione che lo definisce.

Se non fosse così, poichè le variabili Q_j compaiono sempre all'inizio delle espressioni presenti nelle parti destre delle equazioni, ci si riesce a ricondurre, mediante manipolazioni algebriche, ad una equazione della forma:

$$Q_i = Q_i s_1 + s_2$$

per opportuni s_1 ed s_2 in cui Q_i non compare. La soluzione di questa equazione è $Q_i = s_2 s_1^*$; infatti:

$$(s_2 s_1^*) s_1 + s_2 = s_2 s_1^+ + s_2 = s_2(s_1^+ + \varepsilon) = s_2 s_1^* .$$

Si applichi dunque la sostituzione $Q_i = s_2 s_1^*$ al resto del sistema. □

ESERCIZIO 4.5. Mediante la costruzione del Teorema 4.3 si identifichi un ε-NFA che riconosca il linguaggio denotato dall'espressione regolare $((0 + 1)^*0011(0 + 1)^*)^*$. Si passi poi, mediante le trasformazioni suggerite nei Teoremi 3.12 e 3.10 al corrispondente DFA. Di quanti stati si ha bisogno? Si cerchi di costruire direttamente un DFA che riconosca lo stesso linguaggio.

ESERCIZIO 4.6. Si determinino le espressioni regolari associate ai DFA finora presentati nel testo.

CHAPTER 5

Proprietà dei linguaggi regolari

Dai Capitoli 2 e 4 si è imparato che se un linguaggio è riconosciuto da un automa a stati finiti (equivalentemente, DFA, NFA, ε-NFA) allora è un linguaggio regolare. Argomenti di questo capitolo saranno quello i principali risultati e metodi che saranno utili per stabilire se un dato linguaggio sia o meno regolare.

1. Il "Pumping Lemma"

Il primo metodo che proponiamo e che viene spesso usato per mostrare che un linguaggio non è regolare deriva dal seguente risultato noto come *Pumping Lemma*:

LEMMA 5.1 (Bar-Hillel, Perles, Shamir, 1961). *Sia L un linguaggio regolare. Allora esiste una costante $n \in \mathbb{N}$ tale che per ogni $z \in L$, se $|z| \geq n$, esistono tre stringhe u, v, w tali che:*

1. $z = uvw$,
2. $|uv| \leq n$,
3. $|v| > 0$, *e*
4. *per ogni $i \geq 0$ vale che $uv^i w \in L$.*

PROOF. Sia $M = \langle \{q_0, \ldots, q_{n-1}\}, \Sigma, \delta, q_0, F \rangle$ DFA tale che $L = L(M)$ (esiste poichè L è regolare). Sia $z = a_1 \cdots a_m$, $m \geq n$, $z \in L$ (si noti l'arbitrarietà della scelta di z; se una tale z non esistesse, il lemma varrebbe banalmente). Per $i = 1, \ldots, n$ $(n \leq m)$ si consideri l'evoluzione degli stati $\hat{\delta}(q_0, a_1 \cdots a_i)$. In tal modo, considerando anche lo stato iniziale q_0, si raggiungono in tutto $n+1$ stati. Poiché gli stati dell'automa sono n per ipotesi, esiste (almeno) uno stato $\bar{q}$ raggiunto (almeno) due volte.

Dunque avremo una situazione del tipo

$$\begin{aligned}\hat{\delta}(q_0, a_1 \cdots a_{i_1}) &= \bar{q} \\ &= \hat{\delta}(q_0, a_1 \cdots a_{i_1} \cdots a_{i_2}) \quad i_2 > i_1\end{aligned}$$

A questo punto, si prenda $u = a_1 \cdots a_{i_1}$, $v = a_{i_1+1} \cdots a_{i_2}$, $w = a_{i_2+1} \cdots a_m$. E' immediato, usando il Lemma 3.8, mostrare che $\hat{\delta}(q_0, z) = \hat{\delta}(q_0, uv^i w)$ per ogni $i \geq 0$. Per costruzione si ha inoltre che: $|uv| \leq n$. □

COROLLARIO 5.2. n del Lemma 5.1 può essere preso come il numero di stati del più piccolo automa riconoscente L.

PROOF. Immediato dalla dim. del Lemma. □

Si osservi come il lemma asserisca la veridicità di una formula logica quantificata non banale, ovvero, per ogni linguaggio L, se L è regolare, allora vale:

$$(1.1)\quad \exists n \in \mathbb{N}\ \forall z \left(\begin{array}{l} (z \in L \wedge |z| \geq n) \rightarrow \exists u, v, w \\ \left(\begin{array}{l} z = uvw \wedge |uv| \leq n \wedge |v| > 0 \wedge \\ \forall i\,(i \in \mathbb{N} \rightarrow uv^i w \in L) \end{array} \right) \end{array} \right)$$

Il Pumping lemma viene largamente usato per mostrare che un dato linguaggio non è regolare (qualora non lo sia!). Per mostrare ciò, si assume che L non sia regolare e si deve mostrare che vale la negazione della formula (1.1), ovvero (si veda il Capitolo 3 per le tecniche di complementazione di una formula logica):

$$(1.2)\ \forall n \in \mathbb{N}\ \exists z \left(\begin{array}{l} z \in L \wedge |z| \geq n \wedge \\ \forall u, v, w\,(z = uvw \wedge |uv| \leq n \wedge |v| > 0 \\ \qquad \rightarrow \exists i\,(i \in \mathbb{N} \wedge uv^i w \notin L) \end{array} \right)$$

Nel prossimo esempio si illustrerà un corretto uso di questa tecnica, che dato l'annidamento di quantificazioni, può indurre ad errore.

ESEMPIO 5.3. Il linguaggio $L = \{0^i 1^i : i \geq 0\}$ non è regolare. Supponiamo, per assurdo, sia regolare e cerchiamo di verificare la formula (1.2). Si prenda un numero naturale n arbitrario e si scelga

(questo è il punto in cui bisogna avere la "fortuna" di scegliere una stringa "giusta" tra tutte quelle di lunghezza maggiore o uguale a n) la stringa $z = 0^n 1^n$. Per tale stringa vale che $z \in L$ e $|z| \geq n$.

Rimane da dimostrare che comunque si suddivida la stringa z in sottostringhe uvw (in modo tale cioè che $z = uvw$) tali da soddisfare i vincoli $|uv| \leq n$ e $|v| > 0$ esiste almeno un $i \geq 0$ tale per cui $uv^i w \notin L$.

I modi per partizionare z sono molti (per esercizio, si provi a pensare quanti) però, il fatto che $|uv| \leq n$ permette di ricondursi al seguente unico "schema di suddivisione":

- $u = 0^a$,
- $v = 0^b$,
- $w = 0^c 1^n$,

con $a + b + c = n$, $b > 0$. "Pompando" v si ottengono stringhe al di fuori del linguaggio, da cui l'assurdo. Ad esempio, con $i = 0$ vale che $uv^0 w = 0^a 0^c 1^n \notin L$, poichè $a + c < n$.

ESERCIZIO 5.4. Quali dei seguenti linguaggi sono regolari (si motivi la risposta)?

1. $\{0^{2n} : n \geq 1\}$;
2. $\{0^m 1^n 0^{m+n} : m \geq 1, n \geq 1\}$;
3. $\{0^n : n \text{ è primo}\}$.

2. Proprietà di chiusura

La definizione di linguaggio è una definizione di tipo insiemistico ($L \subseteq \Sigma^*$). Pertanto è lecito parlare di unione, intersezione, complementazione (etc.) di linguaggi. In questo paragrafo mostreremo che la proprietà di essere un linguaggio regolare si conserva sotto queste usuali operazioni.

TEOREMA 5.5. *I linguaggi regolari sono chiusi rispetto alle operazioni di unione, concatenazione e chiusura di Kleene.*

PROOF. Immediato dalla definizione di espressione regolare e dai Teoremi 4.3 e 4.4. □

TEOREMA 5.6. *I linguaggi regolari sono chiusi rispetto alla operazione di complementazione. Ovvero, se $L \subseteq \Sigma^*$ è regolare, anche $\bar{L} = \Sigma^* \setminus L$ è regolare.*

PROOF. Sia $M = \langle Q, \Sigma', \delta, q_0, F\rangle$ il DFA che riconosce L. Per il Lemma 3.9, possiamo assumere $\Sigma' = \Sigma$. Allora, banalmente, $M' = \langle Q, \Sigma, \delta, q_0, Q \setminus F\rangle$ riconosce $\bar{L}$. □

COROLLARIO 5.7. I linguaggi regolari sono chiusi rispetto all'intersezione.

PROOF. Immediato dal fatto che $L_1 \cap L_2 = \overline{(\bar{L}_1 \cup \bar{L}_2)}$. □

ESERCIZIO 5.8. Dati due DFA M_1 e M_2, si costruisca l'automa che riconosce $M_1 \cap M_2$ in maniera diretta. Di quanti stati avete bisogno?

ESERCIZIO 5.9. Si dimostri che l'insieme delle stringhe di 0 e 1 tali che: non vi sono mai tre zeri consecutivi e non vi sono mai tre 1 consecutivi, oppure vi sono almeno due zeri seguiti da due uni è un linguaggio regolare (Suggerimento: si suddivida il problema in problemi elementari e si usino i risultati di questo paragrafo per combinarli).

3. Risultati di decidibilità

In questo paragrafo saranno mostrati alcuni risultati relativi al problema di effettuare delle decisioni riguardanti linguaggi regolari.

TEOREMA 5.10 (Vuoto-infinito). *L'insieme delle stringhe accettate da un DFA M con n stati è:*

1. *non vuoto se e solo se accetta una stringa di lunghezza inferiore a n;*
2. *infinito se e solo se l'automa accetta una stringa di lunghezza ℓ, $n \leq \ell < 2n$.*

PROOF. 1. ($\leftarrow$) Se accetta una stringa (di lunghezza di inferiore a n) allora è banalmente non vuoto.
($\rightarrow$) Supponiamo sia non vuoto. Allora, per definizione M accetta almeno una stringa z, $|z| = m$.

Se $m < n$, la tesi è provata. Altrimenti come conseguenza del il pumping lemma, $z = uvw$ con $|v| \geq 1$, e $uv^0w = uw$ è accettata da M. Se $|uw| < n$ la tesi è provata, altrimenti si proceda iterativamente ripartendo con $z = uw$ (che ha lunghezza strettamente minore a m). Dopo al più $m - n$ iterazioni si otterrà una stringa di lunghezza inferiore a n;

2. ($\leftarrow$) Supponiamo M accetti una stringa z di lunghezza ℓ, $n \leq \ell < 2n$. Allora, per il pumping Lemma, $z = uvw$, $|v| \geq 1$ e $\{uv^iw : i \in \mathbb{N}\} \subseteq L(M)$. Ma $\{uv^iw : i \in \mathbb{N}\}$ è un insieme infinito.
   ($\rightarrow$) Sia $L(M)$ infinito. Allora esiste $z \in L(M)$ tale che $|z| = m \geq 2n$. Per il pumping lemma $z = uvw$ con $|uv| \leq n$ e $|v| \geq 1$ (e dunque $|uw| \geq n$). Per il pumping lemma, $z' = uw \in L(M)$. Se $|z'| < 2n$, allora la tesi è dimostrata. Altrimenti, si reiteri il procedimento partendo dalla stringa (più corta) $z' = uw$. In un numero finito di passi si trova la stringa cercata. □

COROLLARIO 5.11. Sia M DFA. Allora i problemi '$L(M) = \emptyset$' e '$L(M)$ è infinito' sono entrambi decidibili (cioè esiste una procedura effettiva per determinare la loro veridicità o falsità).

TEOREMA 5.12 (Equivalenza). *Dati due DFA M_1 e M_2, il problema di stabilire se $L(M_1) = L(M_2)$ è decidibile.*

PROOF. $L(M_1) = L(M_2)$ è insiemisticamente equivalente a

$$(L(M_1) \cap \overline{L(M_2)}) \cup (\overline{L(M_1)} \cap L(M_2)) = \emptyset\,.$$

Dai Teoremi 5.5 e 5.6 sappiamo esistere un automa M_3 tale che

$$L(M_3) = (L(M_1) \cap \overline{L(M_2)}) \cup (\overline{L(M_1)} \cap L(M_2))\,.$$

Il risultato segue dalla decidibilità del problema del vuoto (Corollario 5.11). □

4. Il teorema di Myhill-Nerode

Iniziamo il paragrafo richiamando alcune definizioni. Dato un insieme S, una relazione di equivalenza $R \subseteq S \times S$ (si veda § 2) induce (univocamente, a meno di ridenominazione degli indici) una *partizione* di $S = S_1 \cup S_2 \cup \ldots$ ove per ogni i $S_i \neq \emptyset$ e per ogni i, j, $i \neq j$, si ha che:

1. $S_i \cap S_j = \emptyset$;
2. $(\forall a, b \in S_i) a\ R\ b$;
3. $(\forall a \in S_i)(\forall b \in S_j) \neg(a\ R\ b)$.

Le varie S_i sono dette *classi di equivalenza*. Se $a \in S_i$ allora con la notazione $[a]_R$ (o semplicemente $[a]$ quando il contesto è chiaro)

si denota la classe S_i. Se S è partizionato in un numero finito di classi $S_1 \cup S_2 \cup \cdots \cup S_k$ allora R si dice di *indice finito* (k) su S. Date due relazioni di equivalenza R_1 e R_2 sullo stesso insieme S, R_1 è un *raffinamento* di R_2 se ogni classe di equivalenza della partizione indotta da R_1 è sottoinsieme di qualche classe di equivalenza della partizione indotta da R_2.

Ad esempio, se $S = \{2,3,4,5\}$, $P_1 = \{\{2,3,5\},\{4\}\}$ e $P_2 = \{\{2\},\{3,5\},\{4\}\}$ denotano le partizioni di S ottenute a partire dalle relazioni

(R_1) xR_1y sse x e y sono entrambi primi o uguali tra loro

(R_2) xR_2y sse x e y sono entrambi primi e dispari, o uguali tra loro

allora R_2 è un raffinamento di R_1.

ESERCIZIO 5.13. Dato un insieme S finito, quanti sono i modi distinti per partizionarlo?

Dato un linguaggio $L \subseteq \Sigma^*$, gli associamo la relazione $R_L \subseteq \Sigma^* \times \Sigma^*$ definita come segue:

$$x \; R_L \; y \quad \text{sse} \quad (\forall z \in \Sigma^*)(xz \in L \leftrightarrow yz \in L)\,.$$

Similmente, se $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ è un DFA, gli associamo la relazione $R_M \subseteq \Sigma^* \times \Sigma^*$ definita come:

$$x \; R_M \; y \quad \text{sse} \quad \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)\,.$$

Definiamo come $L_q = \{x \in \Sigma^* : \hat{\delta}(q_0, x) = q\}$ il linguaggio associato allo stato q dell'automa. Le classi di equivalenza di R_M sono esattamente i linguaggi associati ad ogni stato dell'automa M.

LEMMA 5.14. *R_L e R_M sono relazioni di equivalenza.*

PROOF. (Esercizio) □

Una relazione $R \subseteq \Sigma^* \times \Sigma^*$ che gode della proprietà:

$$x \; R \; y \quad \text{implica} \quad (\forall z \in \Sigma^*)(xz \; R \; yz)$$

si dice *invariante a destra* (rispetto alla concatenazione). Dal Lemma 3.8, si evince che R_M è invariante a destra. Inoltre R_L è anche invariante a destra: siano $x, y \in \Sigma^*$ tali che xR_Ly. Sia $z \in \Sigma^*$: dobbiamo mostrare

che xzR_Lyz. Se così non fosse, allora esisterebbe w tale che $xzw \in L$ e $yzw \notin L$ (o viceversa). Ma in tal caso, con $z' = zw$, mostreremmo che $x \not R_L y$: assurdo.

TEOREMA 5.15 (Myhill-Nerode, 1957–58). *I seguenti enunciati sono equivalenti:*

1. $L \subseteq \Sigma^*$ *è accettato da un qualche DFA;*
2. L *è l'unione di classi di equivalenza di* Σ^* *indotte da una relazione invariante a destra e di indice finito;*
3. R_L *è di indice finito.*

PROOF. (1) $\rightarrow$ (2) Sia L accettato da un DFA $M = \langle Q, \Sigma, \delta, q_0, F\rangle$. Mostreremo che R_M è la relazione che soddisfa il punto (2). Per definizione di linguaggio riconosciuto da un automa:

$$L = \bigcup_{q \in F} \{x \in \Sigma^* : \hat{\delta}(q_0, x) = q\}$$

Per il Lemma 3.8, R_M è invariante a destra, dunque i vari insiemi:

$$L_q = \{x \in \Sigma^* : \hat{\delta}(q_0, x) = q\}$$

costituiscono le classi di equivalenza della partizione indotta da R_M.

(2) $\rightarrow$ (3) Mostreremo che ogni relazione di equivalenza R che soddisfa (2) è un raffinamento di R_L. Sia $x \in \Sigma^*$, vogliamo mostrare che $[x]_R \subseteq [x]_{R_L}$. Sia $y \in [x]_R$ (dunque $x\ R\ y$). Poiché R è invariante a destra per ipotesi, allora $xz\ R\ yz$ per ogni $z \in \Sigma^*$. Poiché L è unione di classi di equivalenza di R, ciò implica che ogni qualvolta vRw si ha che $v \in L$ sse $w \in L$. Pertanto per ogni $z \in \Sigma^*$, $xz \in L$ sse $yz \in L$. Ma allora $x\ R_L\ y$ per definizione, dunque $y \in [x]_{R_L}$. L'indice di R_L è minore di quello di R, che per ipotesi è finito.

(3) $\rightarrow$ (1) Si costruisce un DFA $M' = \langle Q', \Sigma', \delta', q_0', F'\rangle$ che riconosce L. Sia

- Q' l'insieme (finito per ipotesi) di classi di equivalenza di R_L,
- Σ' lo stesso di L,
- $\delta'([x], a) = [xa]$ (la definizione ha senso indipendentemente dalla scelta di x in quanto R_L è invariante a destra),
- $q_0' = [\varepsilon]$,
- $F' = \{[x] : x \in L\}$.

Si tratta di mostrare che $L(M') = L$. Si verifica per induzione su $|y| \geq 0$ che $\hat{\delta}'([x], y) = [xy]$. Pertanto si ha che:

$$\hat{\delta}'(q_0', x) = \hat{\delta}'([\varepsilon], x) = [\varepsilon x] = [x]$$

e dunque

$$x \in L(M') \quad \text{sse} \quad \hat{\delta}'(q_0', x) \in F' \quad \text{sse} \quad [x] \in F \quad \text{sse} \quad x \in L\,.$$

□

5. Minimizzazione di DFA

Il Teorema 5.15 ha, tra le sue varie conseguenze, quella di stabilire che se un linguaggio L è riconosciuto da un DFA M allora ne esiste uno minimo M' (nel senso del numero degli stati) equivalente a M.

TEOREMA 5.16. *Per ogni linguaggio regolare L esiste un automa M con minimo numero di stati tale che $L = L(M)$, unico a meno di isomorfismo (ovvero ridenominazione di stati).*

PROOF. Sia L accettato da un DFA M. Σ^* è partizionato dalla relazione di equivalenza R_M negli insiemi di linguaggi accettati dai singoli stati di M.

Dal Teorema 5.15 ($1 \to 2$) sappiamo che R_M raffina R_L e pertanto il numero di stati di M è maggiore o uguale a quello di M' costruito come nella dimostrazione ($3 \to 1$) del Teorema 5.15.

Dunque tale automa ha il minor numero di stati possibile. Supponiamo che tale numero coincida con il numero di stati di M.

Definiamo la funzione f tra gli insiemi di stati dei due automi nel modo seguente:

$$f(q) = [x] \quad \text{sse} \quad \hat{\delta}(q_0, x) = q\,.$$

Mostriamo che questa è una buona definizione. Vogliamo essere sicuri che $f(q)$ è univoca. Supponiamo, per assurdo, $f(q) = [y_1]$ e $f(q) = [y_2]$. Allora $\hat{\delta}(q_0, y_1) = q$ e $\hat{\delta}(q_0, y_2) = q$. Ma allora $y_1\ R_M\ y_2$. Poiché R_M è raffinamento di R_L si ha anche $[y_1] = [y_2]$.

La funzione è suriettiva, in quanto per ogni $x \in \Sigma^*$ si ha che esiste $q = \hat{\delta}(q_0, x)$ tale che $f(q) = [x]$. Poiché $|Q| = |Q'|$ ciò implica che f è biiettiva, dunque vi è una funzione di rinomina dei nodi.

Per mostrare che i due automi sono isomorfi rimane da mostrare che il seguente diagramma commuta, per ogni $q \in Q$ e per ogni $a \in \Sigma$:

$$\begin{array}{ccc} q & \xrightarrow{f} & [x] \\ \downarrow a & & \downarrow a \\ q' & \xrightarrow{f} & [xa] \end{array}$$

In altri termini, dati $f(q) = [x]$, $\delta(q,a) = q'$, $\delta'([x],a) = [xa]$, dobbiamo mostrare che $f(q') = [xa]$:

$$\begin{array}{lll} f(q') = [xa] \text{ sse } q' = \hat{\delta}(q_0, xa) & & \text{def. di } f \\ \text{sse } q' = \delta(\hat{\delta}(q_0, x), a) & & \text{def. di } \hat{\delta} \\ \text{sse } q' = \delta(q, a) & & \text{vero per ipotesi.} \end{array}$$

□

In Fig. 1 è presentato un algoritmo che permette di ottenere automaticamente l'automa minimo corrispondente ad uno dato.

ESEMPIO 5.17. *La minimizzazione dell'automa:*

	0	1	
q_0	q_1	q_3	
q_1	q_2	q_3	
q_2	q_0	q_3	
q_3	q_3	q_3	F

conduce all'automa:

	0	1	
q_0	q_0	q_3	
q_3	q_3	q_3	F

procedure minimizza;

input: $M = \langle Q, \Sigma, \delta, q_0, F \rangle$;

output: $M' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ t.c. $L(M) = L(M')$ e
$(\forall M'' = \langle Q'', \Sigma, \delta'', q_0'', F'' \rangle)(L(M) = L(M'') \rightarrow |Q'| \leq |Q''|)$;

begin

$\Pi_{new} := \{Q - F, F\}$;

repeat

$\Pi := \Pi_{new}$;

$\Pi_{new} := \emptyset$;

for S in Π do

begin

partiziona S in $S_1, \ldots, S_m$

usando il valore di m più piccolo possibile

t.c. $p, q \in S_i$ sse

$(\forall a \in \Sigma)(\exists S' \in \Pi)(\delta(p, a) \in S' \leftrightarrow \delta(q, a) \in S')$

$\Pi_{new} := \Pi_{new} \cup \{S_1, \ldots, S_m\}$

end

until $\Pi_{new} = \Pi$;

for S in Π let $\bar{S} = \min_i\{q_i \in S\}$;

$Q' := \{\bar{S} : S \in \Pi\}$;

$\delta'(\bar{S}, a) := \min_i\{q_i : q_j \in S, \delta(q_j, a) = q_i\}$;

$F' := \{[S] : S \in \Pi, (\exists q \in S)(q \in F)\}$;

$q_0' := q_0$;

si eliminino ricorsivamente da Q' e F' gli stati diversi da q_0' privi di archi entranti e si eliminino da δ gli archi uscenti da essi.

end.

FIGURE 1. Algoritmo di minimizzazione

in un solo passo. Invece la minimizzazione dell'automa:

	0	1	
q_0	q_0	q_1	
q_1	q_1	q_2	
q_2	q_2	q_3	
q_3	q_3	q_3	F

restituisce l'automa stesso. Sono però necessarie 3 (= $|Q| - 1$) iterazioni.

ESERCIZIO 5.18 (Facoltativo). Si dimostri che l'algoritmo di Fig. 1 permette effettivamente di calcolare un automa equivalente e minimo.

ESERCIZIO 5.19. Si concretizzi in un linguaggio di programmazione ad alto livello l'algoritmo di Fig. 1 in modo tale che il problema sia risolto con complessità $O(|Q|^2|\Sigma|)$. Si mediti su come arrivare alla complessità $O(|Q| \log |Q||\Sigma|)$.

CHAPTER 6

Grammatiche libere dal contesto

Una grammatica è, intuitivamente, un insieme di regole che permettono di generare un linguaggio. Un ruolo fondamentale tra le grammatiche è costituito dalle grammatiche libere dal contesto mediante le quali vengono solitamente descritti i linguaggi di programmazione.

1. Definizione formale

Una *grammatica libera dal contesto* (CF) è una quadrupla $G = \langle V, T, P, S \rangle$, ove:

- V è un insieme finito di variabili (dette anche simboli *non terminali*),
- T è un insieme finito di simboli *terminali* ($V \cap T = \emptyset$),
- P è un insieme finito di produzioni; ogni *produzione* è della forma $A \to \alpha$,[1] ove:
  - $A \in V$ è una variabile, e
  - $\alpha \in (V \cup T)^*$.
- $S \in V$ è una variabile speciale, detta *simbolo iniziale.*

ESEMPIO 6.1. Sia G la grammatica seguente:

$$G = \{\{E\}, \{or, and, not, (,), 0, 1\}, P, E\}$$

[1]Oppure, equivalentemente, si può vedere come una coppia $\langle A, \alpha \rangle \in V \times (V \cup T)^*$.

ove P è costituito da:

$$\begin{aligned} E &\mapsto 0 \\ E &\mapsto 1 \\ E &\mapsto (E \; or \; E) \\ E &\mapsto (E \; and \; E) \\ E &\mapsto (not \; E) \end{aligned}$$

Come vedremo, G genererà le possibili espressioni booleane.

NOTAZIONE 6.2. Per le grammatiche saranno utilizzate le seguenti notazioni: lettere maiuscole A, B, C, D, E, S denotano variabili, S—a meno che non sia esplicitamente detto il contrario—il simbolo iniziale. $a, b, c, d, e, 0, 1$ denotano simboli terminali; X, Y, Z denotano simboli che possono essere sia terminali che variabili; u, v, w, x, y, z denotano stringhe di terminali, $\alpha, \beta, \gamma, \delta$ stringhe generiche di simboli (sia terminali che non). Se $A \to \alpha_1, \ldots, A \to \alpha_n \in P$, esprimeremo questo fatto scrivendo: $A \to \alpha_1 | \cdots | \alpha_n$. Con *A-produzione* denoteremo una generica produzione con la variabile A a sinistra.

La grammatica dell'esempio 6.1 può essere dunque schematicamente rappresentata da:

$$E \mapsto 0|1|(E \; or \; E)|(E \; and \; E)|(not \; E) \, .$$

2. Linguaggio generato

Le grammatiche servono a generare un linguaggio. Daremo ora le definizioni necessarie a definire il linguaggio generato da una grammatica $G = \langle V, T, P, S \rangle$. Definiremo le relazioni $\overset{G}{\Rightarrow}, \overset{G}{\Rightarrow}_i, \overset{G}{\Rightarrow}_* \subseteq (V \cup T)^* \times (V \cup T)^*$ nel seguente modo:

- se $A \to \beta \in P$ e $\alpha, \gamma \in (V \cup T)^*$, allora $\alpha A \gamma \overset{G}{\Rightarrow} \alpha \beta \gamma$. Diremo in questo caso che da $\alpha A \gamma$ *deriva immediatamente* $\alpha \beta \gamma$;
- se $\alpha_1, \ldots, \alpha_i \in (V \cup T)^*$, $i \geq 1$, e

$$\bigwedge_{j=1}^{i-1} \alpha_j \overset{G}{\Rightarrow} \alpha_{j+1} \, ,$$

allora $\alpha_1 \overset{G}{\Rightarrow}_i \alpha_m$. In questo caso diremo che da α_1 *deriva* α_m in i passi;

- se esiste i tale per cui $\alpha_1 \overset{G}{\Rightarrow}_i \alpha_m$, allora $\alpha_1 \overset{G}{\Rightarrow}_* \alpha_m$. Diremo in tal caso che da α_1 deriva α_m.

Si noti che $\overset{G}{\Rightarrow}_*$ è la chiusura transitiva e riflessiva della relazione $\overset{G}{\Rightarrow}$. In particolare vale che $\alpha \overset{G}{\Rightarrow}_* \alpha$ e $\alpha \overset{G}{\Rightarrow}_0 \alpha$ per ogni $\alpha \in (V \cup T)^*$.

Il *linguaggio generato da* G è:

$$L(G) = \{w \in T^* : S \overset{G}{\Rightarrow}_* w\}\,.$$

L è un liguaggio *libero dal contesto* (CF) se esiste una grammatica CF G tale che $L = L(G)$. Due grammatiche G_1 e G_2 sono equivalenti se $L(G_1) = L(G_2)$.

ESEMPIO 6.3. Σ^* è un linguaggio CF. Infatti, sia $\Sigma = \{s_1, \ldots, s_n\}$, allora Σ^* è generato da:

$$S \to \varepsilon | s_1 S | \cdots | s_n S\,.$$

ESEMPIO 6.4. La grammatica schematicamente definita da:

$$S \to ASB \,|\, \varepsilon, A \to 0, B \to 1$$

genera il linguaggio

$$\{0^n 1^n : n \geq 0\}$$

(dimostrare ciò per induzione su n come esercizio).

Per quanto dimostrato nell'esempio 5.3 usando il pumping lemma, il linguaggio $\{0^n 1^n : n \geq 0\}$ non è un linguaggio regolare. Pertanto l'insieme dei linguaggi regolari e quello dei linguaggi CF non sono lo stesso insieme. Vedremo, comunque, che il primo insieme è incluso nel secondo.

ESEMPIO 6.5. La grammatica dell'Esempio 6.1 genera tutte le espressioni booleane (si dimostri questo fatto per per induzione sulla struttura dell'espressione).

3. Alberi di derivazione

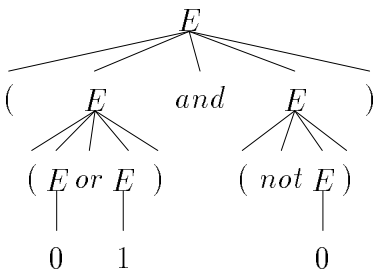
Le derivazioni generate da una grammatica vengono ben visualizzate mediante l'ausilio di strutture dati ad albero.

Sia $G = \langle V, T, P, S\rangle$ una grammatica CF. Un albero è un *albero di derivazione* (parse tree) per G se:

1. ogni vertice ha una *etichetta*, presa tra $V \cup T \cup \{\varepsilon\}$;
2. l'etichetta della radice appartiene a V;
3. se un vertice interno (ovvero, non una foglia) ha etichetta A, allora $A \in V$;
4. se un vertice n è etichettato con A e $n_1, \ldots, n_k$ sono (ordinatamente, da sinistra a destra) i vertici *figli* di A etichettati con $X_1, \ldots, X_k$, allora
$$A \to X_1 \cdots X_k \in P\,;$$
5. se un vertice n ha etichetta ε, allora n è una foglia ed è l'unico figlio di suo padre.

Si noti che la definizione di albero non impone che tutte le foglie siano etichettate con simboli terminali. L'albero avente un solo nodo etichettato da S è un caso particolare di albero di derivazione. Gli alberi verrano utilizzati per dare una controparte grafica delle derivazioni. Se vi sono foglie etichettate con simboli non terminali, allora l'albero rappresenta una derivazione *parziale*. Diremo che un albero *descrive* una stringa $\alpha \in (V \cup T)^*$ se α è proprio la stringa che possiamo leggere dalle etichette delle foglie da sinistra a destra.

ESEMPIO 6.6. La grammatica dell'esempio 6.1 genera, in particolare, la stringa $w = ((0\ or\ 1)\ and\ (not\ 0))$. La derivazione per w è rappresentata dall'albero seguente (che descrive w):



Dato un albero ed un suo nodo n, il *sottoalbero* identificato da quel nodo è costituito da quel nodo più tutti i suoi discendenti (figli, nipoti, ecc.), nonchè le varie etichette associate ad essi.

TEOREMA 6.7. *Sia $G = \langle V, T, P, S\rangle$ una grammatica CF. Allora $S \overset{G}{\Rightarrow}_* \alpha$ se e solo se esiste un albero di derivazione con radice etichettata S per G che descrive α.*

PROOF. Si proverà un enunciato più forte, ovvero: dato un qualunque simbolo non terminale $A \in V$, $A \overset{G}{\Rightarrow}_* \alpha$ se e solo se esiste un albero di derivazione per G che descrive α la cui radice è etichettata A.

($\rightarrow$) Proviamo l'enunciato per induzione sul numero i di passi di derivazione per $A \overset{G}{\Rightarrow}_i \alpha$.

Base: $i = 0$: $A \overset{G}{\Rightarrow}_0 \alpha$ implica, per definizione, che $\alpha = A$. Ma l'albero con unico nodo (radice) etichettato con A è un albero di derivazione (parziale) che descrive A stesso.

Passo: Supponiamo $A \overset{G}{\Rightarrow}_i \beta_1 B \beta_3 \overset{G}{\Rightarrow}_1 \underbrace{\beta_1\beta_2\beta_3}_{\alpha}$. Per ipotesi induttiva esiste un albero di derivazione T con radice etichettata A che descrive $\beta_1 B \beta_3$. Ma per definizione, $\beta_1 B \beta_3 \overset{G}{\Rightarrow}_1 \beta_1\beta_2\beta_3$ se e solo se esiste la produzione $B \rightarrow \beta_2$ in P. Ma allora, applicando la regola (4) di definizione di albero, dall'albero T si ottiene l'albero che soddisa i requisiti.

($\leftarrow$) Proviamo l'enunciato per induzione sul numero di nodi interni dell'albero di derivazione etichettato in A che descrive α. [Completare per esercizio] □

4. Ambiguità delle derivazioni

Se ad ogni passo di una derivazione la produzione è applicata al simbolo non terminale più a sinistra, allora la derivazione è detta *sinistra* (leftmost). Similmente, se viene applicata sempre a quello a destra, allora è detta *destra* (rightmost). Se $w \in L(G)$, dal Teorema 6.7 sappiamo che per essa esiste almeno un albero di derivazione con radice etichettata S. Si può dimostrare che, fissato un albero di derivazione con radice etichettata S, esattamente una derivazione sinistra (suggerimento: si visiti l'albero in preordine) ed una derivazione destra (simmetricamente) possono essere desunte.

Un albero di derivazione rappresenta dunque un certo insieme di possibili derivazioni distinte di una stessa stringa. Ci possono tuttavia essere più alberi di derivazione per la stessa parola:

ESEMPIO 6.8. Si consideri la grammatica

$$E \to E + E | E * E | 0 | 1 | 2 \, .$$

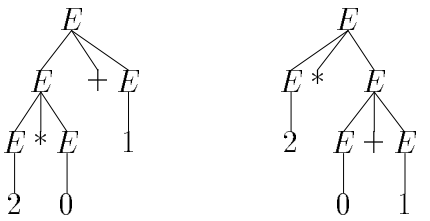
Una derivazione sinistra è:

$$E \overset{G}{\Rightarrow} E + E \overset{G}{\Rightarrow} E * E + E \overset{G}{\Rightarrow}_3 2 * 0 + 1 \, .$$

Un'altra derivazione sinistra per la stessa stringa è:

$$E \overset{G}{\Rightarrow} E * E \overset{G}{\Rightarrow} 2 * E \overset{G}{\Rightarrow} 2 * E + E \overset{G}{\Rightarrow}_2 2 * 0 + 1 \, .$$

Tuttavia gli alberi associati alle due derivazioni (che potrebbero essere visti come alberi per la computazione dell'espressione associata alla stringa generata) sono diversi:



Il primo è intuitivamente associato a $(2 * 0) + 1 = 1$ il secondo, invece, $2 * (0 + 1) = 2$.

Una grammatica CF tale per cui esiste una parola con più di un albero di derivazione con radice etichettata S è detta *ambigua*. Un linguaggio CF per cui ogni grammatica che lo genera è ambigua è detto essere *inerentemente ambiguo*. Un esempio di tali linguaggi è:

$$L = \{a^n b^n c^m d^m : n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n : n \geq 1, m \geq 1\} \, .$$

La dimostrazione di questa proprietà si può trovare in [**3**].

5. Semplificazione

Vi sono molti modi per restringere la forma delle produzioni permesse alle grammatiche CF senza alterare la classe dei linguaggi riconosciuti dall'insieme delle possibili grammatiche. In questo paragrafo si presenteranno due di queste trasformazioni, che conducono alla forma normale di Chomsky e a quella di Greibach.

Mostreremo che ogni linguaggio CF può essere generato da una grammatica G dalle seguenti due proprietà:

1. ogni variabile e ogni simbolo terminale di G compaiono in almeno una derivazione di una qualche parola di L;
2. non ci sono produzioni della forma $A \to B$ con A e B variabili.

Se, inoltre, $\varepsilon \notin L$, non avremo bisogno di produzioni $A \to \varepsilon$; sempre sotto tale ipotesi potremo richiedere che ogni produzione sia della forma $A \to BC$ o $A \to a$ (Chomsky) oppure, alternativamente, che ogni produzione sia della forma $A \to a\alpha$ ove α è una stringa di variabili (Greibach).

Nel prosieguo del capitolo alcune dimostrazioni tecniche ma dimostrabili immediatamente per induzione saranno lasciate per esercizio.

5.1. Eliminazione di simboli inutili. Sia $G = \langle V, T, P, S\rangle$ una grammatica CF. Un simbolo $X \in V \cup T$ è *utile* se esiste una derivazione

$$S \overset{G}{\Rightarrow}_* \alpha X \beta \overset{G}{\Rightarrow}_* w\,;$$

altrimenti è detto inutile. Mostreremo come simboli inutili si possano eliminare in due passi.

Innanzitutto si eliminano le variabili che non conducono in nessun modo ad una stringa di terminali.

LEMMA 6.9. *Data una grammatica CF $G = \langle V, T, P, S\rangle$ tale che $L(G) \neq \emptyset$, si può calcolare in modo effettivo una grammatica equivalente $G' = \langle V', T, P', S\rangle$ tale che per ogni $A \in V'$ esiste $w \in T^*$ tale che $A \overset{G}{\Rightarrow}_* w$.*

PROOF. Per calcolare V' usiamo il seguente operatore $\Gamma : \wp(V) \longrightarrow \wp(V)$:

$$\Gamma(W) = \{A \in V : (\exists \alpha \in (T \cup W)^*)(A \to \alpha \in P)\}$$

Γ è monotono[2] per definizione. Definiamo inoltre l'applicazione iterata di un operatore:

$$\begin{cases} \Gamma^0(W) &= W \\ \Gamma^{i+1}(W) &= \Gamma(\Gamma^i(W)) \end{cases}$$

Per induzione su $i \geq 0$, si mostri (esercizio) che:

$$\left\{ \begin{array}{l} \text{esiste un albero di derivazione che descrive } w \in T^* \\ \text{con radice etichettata con } A \text{ e di altezza} \leq i \end{array} \right\} \quad \text{sse} \quad A \in \Gamma^i(\emptyset)$$

Dunque, per il Teorema 6.7, si ha che $A \in \Gamma^i(\emptyset)$ per qualche i sse $A \stackrel{G}{\Rightarrow}_* w$.

Poiché V è finito, e Γ monotono, si ha che esiste $i \leq |V|$ tale per cui

$$\gamma^i(\emptyset) = \gamma(\gamma^i(\emptyset)) = \gamma^{i+1}(\emptyset) (= \gamma^{i+2}(\emptyset) = \cdots)$$

Pertanto si riesce a calcolare finitamente l'insieme $V' = \Gamma^{|V|}(\emptyset)$, e $P' = \{A \to \alpha \in P : A \in V' \wedge \alpha \in (V' \cup T)^*\}$. □

L'ipotesi che il linguaggio sia non vuoto è indispensabile Se infatti fosse $L = \emptyset$, allora l'applicazione del lemma in cascata fornirebbe una grammatica con $S \notin V'$ e dunque priva di significato in quanto S sarebbe il simbolo iniziale ma non apparterrebbe alla grammatica. In ogni caso, per definizione, ogni simbolo di una grammatica che genera un linguaggio vuoto è inutile. Si osservi che l'applicazione della procedura del lemma permette anche di determinare se il linguaggio riconosciuto da un automa sia vuoto o meno.

Fatto ciò eliminiamo le variabili che non sono mai raggiunte da S:

LEMMA 6.10. *Data una grammatica CF $G = \langle V, T, P, S\rangle$, si può calcolare in modo effettivo una grammatica equivalente $G' = \langle V', T', P', S'\rangle$ tale che per ogni $X \in (V' \cup T')$ esistono α e β in $(V' \cup T')^*$ per cui $S \stackrel{G}{\Rightarrow}_* \alpha X \beta$.*

PROOF. Definiamo un operatore $\Gamma : \wp(V \cup T) \longrightarrow \wp(V \cup T)$:

$$\Gamma(W) \;=\; \{X \in V \cup T \cup \{\varepsilon\} \;:\; (\exists A \in W)(A \to \alpha X \beta \in P)\} \cup \{S\}$$

[2] Ovvero $X \subseteq Y \to \Gamma(X) \subseteq \Gamma(Y)$.

Per induzione su $i \geq 0$, si mostra (esercizio) che:

$$\left\{ \begin{array}{l} \text{esiste un albero di derivazione (parziale)} \\ \text{con radice etichettata con } S \text{ di altezza} \leq i \\ \text{in cui } X \text{ compare come foglia} \end{array} \right\} \quad \text{sse} \quad X \in \Gamma^i(\{S\})$$

Pertanto, per il Teorema 6.7 si ha che $X \in \Gamma^i(\{S\})$ per qualche i se e solo se esistono α e β in $(V \cup T)^*$ per cui $S \stackrel{G}{\Rightarrow}_* \alpha X \beta$.

Γ è monotono, V e T sono finiti, dunque esiste $i \leq |V| + |T|$ tale che

$$\Gamma^i(\{S\}) = \Gamma^{i+1}(\{S\}) (= \Gamma^{i+2}(\{S\}) = \cdots)$$

Siano dunque:

- $V' = \Gamma^{|V|+|T|}(\{S\}) \cap V$;
- $T' = \Gamma^{|V|+|T|}(\{S\}) \cap T$;
- $P' = \{A \to \alpha \in P : A \in V' \wedge \alpha \in (V' \cup T')^*\}$.

□

TEOREMA 6.11. *Ogni linguaggio CF non vuoto è generato da una grammatica CF priva di simboli inutili.*

PROOF. Immediato, applicando prima il Lemma 6.9 e poi il Lemma 6.10. □

ESERCIZIO 6.12. Si cerchi una grammatica con linguaggio non vuoto tale per cui applicando prima la procedura del Lemma 6.10 e poi quella del Lemma 6.9 non si giunge all'eliminazione dei simboli inutili.

5.2. Eliminazione di ε-produzioni. Data una grammatica che genera un linguaggio CF L, si desidera eliminare le produzioni della forma $A \to \varepsilon$. Se $\varepsilon \in L$, ammetteremo la presenza della produzione $S \to \varepsilon$ qualora $\varepsilon \in L$.

Il metodo è quello di determinare quali variabili $A \in V$ sono tali che $A \stackrel{G}{\Rightarrow}_* \varepsilon$. Tali variabili sono dette *annullabili*. Ciò che poi si farà sarà quello di eliminare, tra le X_i di ogni produzione $A \to X_1 \cdots X_n$ zero, una, ... , o tutte le variabili annullabili. Se togliendo tutte le variabili annullabili da $X_1 \cdots X_n$ la stringa diventasse vuota, allora anche A sarà annullabile (ma non aggiungeremo la produzione $A \to \varepsilon$ (a meno che A non sia S).

TEOREMA 6.13. *Se $L = L(G)$ per qualche grammatica CF $G = \langle V, T, P, S\rangle$, allora $L \setminus \{\varepsilon\}$ è un linguaggio CF generato da una grammatica $G' = \langle V', T', P', S'\rangle$ senza simboli inutili e senza ε-produzioni.*

PROOF. Definiamo un operatore $\Gamma : \wp(V) \longrightarrow \wp(V)$:

$$\begin{aligned}\Gamma(W) \;=\; \{A \in V :\; & (\exists A \to \alpha \in P) \\ & \alpha \text{ privata delle variabili di } W = \varepsilon\}\end{aligned}$$

Anche qui, Γ è monotono, V è finito, dunque esiste $i \leq |V|$ tale che $\Gamma^i(\emptyset) = \Gamma^{i+1}(\emptyset)$. Siano dunque $N = \Gamma^{|V|}(\emptyset)$ e

$$\begin{aligned}P'' \;=\; \{A \to \alpha_1 \cdots \alpha_n :\; & A \to X_1 \cdots X_n \in P, \\ & \text{se } X_i \notin N, \text{ allora } \alpha_i = X_i, \\ & \text{se } X_i \in N, \text{ allora } \alpha_i = X_i \text{ o } \alpha_i = \varepsilon, \\ & \alpha_1 \cdots \alpha_n \neq \varepsilon\,\}\end{aligned}$$

E' immediato mostrare che $L(G) = L(\langle V, T, P'', S\rangle)$. Applichiamo dunque il Teorema 6.11 alla grammatica $\langle V, T, P'', S\rangle$. □

Ovviamente, se $\varepsilon \in L$, aggiungeremo la produzione $S \to \varepsilon$. Si osservi come questa procedura permetta di capire se $\varepsilon \in L$. E' infatti sufficiente vedere se esiste in P una produzione $S \to A_1 \cdots A_k$ con le $A_i \in N$ per $i = 1, \ldots, k$, oppure direttamente la produzione $A \to \varepsilon$.

5.3. Eliminazione di produzioni unitarie. Una produzione della forma $A \to B$, con A e B variabili, si dice *unitaria*.

TEOREMA 6.14. *Ogni grammatica CF senza ε è definita da una grammatica senza simboli inutili, senza ε-produzioni e senza produzioni unitarie.*

PROOF. Grazie al Teorema 6.13, possiamo partire da una grammatica senza simboli inutili e senza ε-produzioni. Innanzitutto identifichiamo tutte le coppie di variabili A, B che permettono derivazioni della forma $A \overset{G}{\Rightarrow}_* B$ (immediato, seguendo i cammini del grafo ottenuto prendendo come nodi le variabili e come archi le produzioni unitarie). Sia, per ogni variabile A, $\mathsf{seguenti}(A)$ l'insieme delle variabili B soddisfacenti la richiesta sopra.

A questo punto togliamo da P tutte le produzioni unitarie e, per ogni $A \in V$ e per ogni produzione (non unitaria) $B \to \beta \in P$ con $B \in \mathsf{seguenti}(A)$ aggiungo la produzione $A \to \beta$.

Dimostrare l'equivalenza dei linguaggi è immediato.

Tuttavia, la procedura potrebbe aver generato simboli inutili: applichiamo il Teorema 6.11 (si noti che produzioni unitarie non possono essere introdotte dalla procedura di eliminazione di simboli inutili) e otteniamo il risultato cercato. □

6. Forma normale di Chomsky

TEOREMA 6.15 (Chomsky, 1959). *Ogni linguaggio CF senza ε è generato da una grammatica in cui tutte le produzioni sono della forma $A \to BC$ e $A \to a$.*

PROOF. Per il Teorema 6.14 possiamo partire da una grammatica $G = \langle V, T, P, S\rangle$ senza ε-produzioni e senza produzioni unitarie.

Sia $A \to X_1 \cdots X_m \in P$, $m \geq 1$ (non vi sono ε-produzioni):

- se $m = 1$, allora, poichè non vi sono produzioni unitarie, $X_1 \in T$: è già nella forma cercata;
- se $m = 2$ e $X_1, X_2 \in V$, siamo nella forma cercata;
- se $m \geq 2$ e qualcuno degli $X_i \in T$, allora per ogni $X_i \in T$ introduco in V una nuova variabile, diciamo B_i, aggiungo la produzione $B_i \to X_i$ e rimpiazzo la produzione $A \to X_1 \cdots X_m$ con $A \to Y_1 \cdots Y_m$, ove $Y_i = X_i$ se $X_i \in V$, $Y_i = B_i$ altrimenti;
- se $m > 2$ e tutti gli X_i sono variabili, allora rimpiazzo la produzione $A \to X_1 \cdots X_m$ con le produzioni $A \to BX_3 \cdots X_m$, $B \to X_1X_2$, ove B è una nuova variabile da aggiungere a V.

E' immediato verificare che il procedimento termina e che l'insieme finale di produzioni (P') è nella forma normale di Chomsky.

Si tratta di verificare che le grammatica $G' = \langle V', T, P', S\rangle$ (ove V' si ottiene da V con l'aggiunta delle nuove variabili) ottenuta è equivalente a quella di partenza. Ovvero, per $A \in V$:

$$A \overset{G}{\Rightarrow}_* w \text{ sse } A \overset{G'}{\Rightarrow}_* w$$

Dimostriamo, per induzione su $i \geq 1$ che $A \overset{G}{\Rightarrow}_i w$ implica che esiste k tale per cui $A \overset{G'}{\Rightarrow}_{i+k} w$. [esercizio]

Viceversa, dimostriamo, per induzione su $i \geq 1$ che $A \overset{G'}{\Rightarrow}_i w$ implica che $A \overset{G}{\Rightarrow}_j w$ per qualche $j \leq i$. [esercizio] □

7. Forma normale di Greibach

Per raggiungere questa forma normale abbiamo bisogno di due lemmi preliminari.

LEMMA 6.16 (Unfolding). *Sia $G = \langle V, T, P, S\rangle$ una grammatica CF, sia $A \to \alpha_1 B \alpha_2$ una produzione di P e $B \to \beta_1 | \cdots | \beta_n$ l'insieme delle B-produzioni. Sia $G' = \langle V, T, P', S\rangle$ ottenuta da G eliminando tutte le produzioni $A \to \alpha_1 B \alpha_2$ da P e aggiungendo le produzioni $A \to \alpha_1\beta_1\alpha_2 | \cdots | \alpha_1\beta_n\alpha_2$. Allora $L(G) = L(G')$.*

PROOF. (Esercizio) □

LEMMA 6.17 (Eliminazione ricorsione sinistra). *Sia $G = \langle V, T, P, S\rangle$ una grammatica CF e sia $A \to A\alpha_1 | \cdots | A\alpha_m$ l'insieme delle A-produzioni di G per cui A è anche il simbolo più a sinistra della stringa di destra delle produzioni. Siano $A \to \beta_1 | \cdots | \beta_n$ le rimanenti A-produzioni. Sia $G' = \langle V \cup \{B\}, T, P', S\rangle$ la grammatica CF ottenuta aggiungendo una nuova variabile B a V e rimpiazzando tutte le A-produzioni con:*

1. *$A \to \beta_i | \beta_i B$ per $i \in \{1, \ldots, n\}$;*
2. *$B \to \alpha_i | \alpha_i B$ per $i \in \{1, \ldots, m\}$.*

Allora $L(G') = L(G)$.

PROOF. Prima o poi, la sequenza di derivazioni:

$$A \overset{G}{\Rightarrow} A\alpha_{i_1} \overset{G}{\Rightarrow} A\alpha_{i_2}\alpha_{i_1} \overset{G}{\Rightarrow} \cdots$$

deve terminare con un'applicazione del tipo $A \overset{G}{\Rightarrow} \beta_j$, altrimenti non si giungerà mai ad una stringa di terminali. Pertanto avremo una situazione del tipo:

$$A \overset{G}{\Rightarrow}_* \beta_j\alpha_{i_n} \cdots \alpha_{i_2}\alpha_{i_1}$$

La stessa situazione può essere ottenuta mediante:

$$A \overset{G}{\Rightarrow} \beta_j B \overset{G}{\Rightarrow} \beta_j\alpha_{i_n} B \overset{G}{\Rightarrow}_* \beta_j\alpha_{i_n} \cdots \alpha_{i_2}\alpha_{i_1}$$

L'implicazione inversa è lasciata per esercizio. □

TEOREMA 6.18 (Greibach, 1965). *Ogni linguaggio CF senza ε è generato da una grammatica in cui tutte le produzioni sono della forma $A \to a\alpha$, ove α è una stringa (eventualmente vuota) di simboli non terminali.*

PROOF. Sia G una grammatica in forma normale di Chomsky. Sia $V = \{A_1, \ldots, A_m\}$. Si costruirà in tre passi una grammatica G' in forma normale di Greibach equivalente a G.

1. Si applichi il seguente algoritmo:

for $k := 1$ to m do

begin

for $j := 1$ to $k - 1$ do

elimina le produzioni $A_k \to A_j\alpha$ mediante unfolding;

elimina le produzioni $A_k \to A_k\alpha$ mediante elim. ricors. sinistra

end;

Nuove variabili $B_1, \ldots, B_h$ sono state introdotte dalla fase di eliminazione della ricorsione sinistra. L'equivalenza tra G e la grammatica ottenuta deriva dai lemmi appena dimostrati.

A questo punto tutte le produzioni sono della forma:

$$\begin{aligned} A_i &\to A_j\alpha \quad j > i \\ A_i &\to a\beta \quad a \in T \\ B_i &\to \gamma \end{aligned}$$

Inoltre, poichè siamo partiti da una grammatica in forma normale di Chomsky, si osserva che α ha un prefisso della forma $A_{i_1} \cdots A_{i_h}$ seguito da un simbolo non terminale, seguito ancora da una sequenza eventualmente vuota di variabili. Per lo stesso motivo, β è una stringa di variabili. Per quanto riguarda le stringhe γ, esse iniziano con un terminale oppure con una coppia di A_iA_j, mai con un B_i.

2. Pertanto, ogni produzione $A_m \to \alpha$ ha la parte destra iniziante con un simbolo terminale. Posso effettuare l'unfolding delle

A_i-produzioni rimpiazzando A_m con ciascuna delle sue possibili parti destre. Posso iterare questo procedimento all'indietro, ottenendo produzioni della forma:

$$A_i \quad \rightarrow \quad a\beta \quad a \in T$$

per ogni $i = 1, \ldots, m$, con β stringa di variabili.

3. Si tratta ora di semplificare le produzioni $B_i \rightarrow \gamma$. Applicando la sostituzione sul primo (eventuale) simbolo A_i della stringa, si giunge al risultato. □

Un importante caso particolare di grammatiche in forma normale di Greibach sono le grammatiche CF in cui le produzioni hanno tutte la forma $A \rightarrow a$ oppure $A \rightarrow aB$. Tali grammatiche, studiate nel Capitolo 8, sono grammatiche lineari destre ed hanno l'importante caratteristica di generare esattamente i linguaggi regolari.

CHAPTER 7

Proprietà dei linguaggi liberi dal contesto

Per i linguaggi CF valgono delle proprietà simili a quelle studiate per i linguaggi regolari nel Capitolo 5.

1. Il pumping lemma per i linguaggi CF

LEMMA 7.1 (Bar-Hillel, Perles, Shamir, 1961). *Sia L un linguaggio CF. Allora c'è una costante $n \in \mathbb{N}$ (dipendente dal linguaggio L) tale che per ogni $z \in L, |z| \geq n$, esistono stringhe $uvwxy$ tali che*

1. $z = uvwxy$,
2. $|vx| \geq 1$,
3. $|vwx| \leq n$, *e*
4. *per ogni $i \geq 0$ vale che $uv^iwx^iy \in L$.*

PROOF. Sia $G = \langle V, T, P, S\rangle$ la grammatica che genera $L \setminus \{\varepsilon\}$. Senza perdita di generalità, possiamo assumere che G sia in forma normale di Chomsky (se $\varepsilon \in L$, potremmo costruire una nuova grammatica G' che estende G con un nuovo simbolo iniziale S' e le produzioni $S' \to \varepsilon$ e $S' \to S$ e modificare appena la dimostrazione—si noti che G' non è in forma normale. Poiché il significato del lemma è per stringhe "lunghe", non complicheremo la dimostrazione in questo modo).

Per induzione su $i \geq 1$, si mostra (esercizio) che se un albero di derivazione per una stringa $z \in T^*$ ha tutti i cammini di lunghezza minore o uguale a i, allora $|z| \leq 2^{i-1}$.

Sia $|V| = k$ ($k > 0$) e sia $n = 2^k$ ($n > 1$). Se $z \in L$ e $|z| \geq n$, allora ogni albero di derivazione per z deve avere un cammino di lunghezza almeno $k + 1$. Ma tale cammino ha almeno $k + 2$ nodi, tutti, eccetto l'ultimo, etichettati da variabili. Pertanto vi deve essere una variabile ripetuta nel cammino.

In particolare, dunque, vi devono essere due vertici v_1, v_2 tali che:

1. entrambi sono etichettati dalla stessa variabile, diciamo A;
2. v_1 è più vicino alla radice di v_2;
3. poichè $n \geq 2$, a v_1 è associata una produzione del tipo: $A \to BC$;
4. da v_1 alla foglia al più la lunghezza è $k+1$.

Si prenda ora il sottoalbero T_1 con radice in v_1; esso rappresenta una derivazione di una parola z_1 di lunghezza al più 2^k. Sia z_2 invece la parola relativa al sottoalbero T_2 di T_1 con radice in v_2. Possiamo scrivere $z_1 = vz_2x$. Inoltre almeno una tra v e x deve essere non vuota, in quanto a v_1 è associata una produzione del tipo: $A \to BC$ ed il sottoalbero T_2 è un sottoalbero di esattamente uno dei due sottoalberi individuati da B e C.

Allora abbiamo che:

$$A \overset{G}{\Rightarrow}_* vAx \quad \text{e} \quad A \overset{G}{\Rightarrow}_* z_2$$

ove $|vz_2x| \leq 2^k = n$. Ma allora ne segue anche che

$$A \overset{G}{\Rightarrow}_* v^i z_2 x^i$$

per ogni $i \geq 0$. Si ponga $w = z_2$ e u e y il prefisso e il suffisso di z necessari affinchè $z = uvwxy$. □

ESEMPIO 7.2. Il linguaggio $\{a^i b^i c^i : i \geq 1\}$ non è CF. Supponiamo per assurdo che lo sia. Come fatto nel caso dei linguaggi regolari, dobbiamo mostrare la veridicità del complemento della formula dimostrata nel lemma, ovvero che per ogni $n \geq 1$ sappiamo trovare una stringa $z \in L$, $|z| \geq n$ che soddisfa una certa condizione. Scegliamo $z = a^n b^n c^n$. Dobbiamo mostrare che per ogni quintupla di stringhe u, v, w, x, y tali che $a^n b^n c^n = uvwxy$, $|vx| \geq 1$ e $|vwx| \leq n$ esiste un $i \geq 0$ tale per cui $uv^i wx^i y \notin L$.

Vi sono molti modi di partizionare z in $uvwxy$ che soddisfano i requisiti. Tuttavia questi si possono raggruppare nelle seguenti casistiche:

1. $uvwx = a^h$ con $h \leq n$
2. $u = a^h$, $vwx = a^k b^m$ con $k > 0$, $k + m \leq n$
3. $u = a^n b^h$, $vwx = b^m$ con $m \leq n$
4. $u = a^n b^h$, $vwx = b^k c^m$ con $k + m \leq n$
5. $u = a^n b^n c^h$, $vwx = c^m$ con $m \leq n$

In ciascuna delle casistiche, tuttavia, si ha che $uv^0wx^0y = uwy \notin L$, in quanto vengono alterati al massimo due tra i numeri di ripetizioni dei tre tipi di carattere a, b e c.

2. Proprietà di chiusura

TEOREMA 7.3. *I linguaggi CF sono chiusi rispetto all'unione, alla concatenazione, ed alla chiusura di Kleene.*

PROOF. La dimostrazione è costruttiva. Siano $G_1 = \langle V_1, T_1, P_1, S_1\rangle$ e $G_2 = \langle V_2, T_2, P_2, S_2\rangle$ le grammatiche generanti i linguaggi L_1 e L_2. Per semplicità si assuma che V_1 e V_2 siano disgiunti (altrimenti si proceda a ridenominare le variabili). Sia S una nuova variabile.

Allora

- $G_\cup = \langle V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup P, S\rangle$, ove P è:
$$S \to S_1 | S_2$$
è la grammatica che genera $L_1 \cup L_2$.
- $G_\circ = \langle V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup P, S\rangle$, ove P è:
$$S \to S_1 S_2$$
è la grammatica che genera $L_1 L_2$.
- $G_* = \langle V_1 \cup \{S\}, T_1, P_1 \cup P, S\rangle$, ove P è:
$$S \to \varepsilon | S_1 S$$
è la grammatica che genera L_1^*. □

TEOREMA 7.4. *I linguaggi CF non sono chiusi rispetto all'intersezione.*

PROOF. Consideriamo $L_1 = \{a^i b^i c^j : i \geq 1, j \geq 1\}$ generato da:
$$S \to RC, R \to ab \,|\, aRb, C \to c \,|\, cC$$
e $L_2 = \{a^i b^j c^j : i \geq 1, j \geq 1\}$ generato da:
$$S \to AR, R \to bc \,|\, bRc, C \to c \,|\, cC$$
La loro intersezione è $\{a^i b^i c^i : i \geq 1\}$, mostrato essere non CF nell'esempio 7.2. □

COROLLARIO 7.5. I linguaggi CF non sono chiusi rispetto alla complementazione.

PROOF. Se lo fossero, allora lo sarebbero anche rispetto all'intersezione, in quanto $A \cap B = \overline{\overline{A} \cup \overline{B}}$, contraddicendo il Teorema 7.4. □

TEOREMA 7.6. *Se L è un linguaggio CF e L' è un linguaggio regolare, allora $L \cap L'$ è un linguaggio CF.*

PROOF. (omessa) □

3. Algoritmi di decisione

TEOREMA 7.7 (Vuoto, Finito, Infinito). *Data una grammatica CF $G = \langle V, T, P, S \rangle$, i problemi:*

1. *$L(G) = \emptyset$,*
2. *$L(G)$ è finito,*
3. *$L(G)$ è infinito,*

sono decidibili.

PROOF. 1) Come osservato nel paragrafo 5.1, se la grammatica ottenuta dall'applicazione dell'algoritmo presente nel Lemma 6.9 è tale che $S \notin V'$, allora il linguaggio è $\emptyset$. Altrimenti no.

2,3) Sia ora $L(G) \neq \emptyset$ e $G' = \langle V', T, P', S \rangle$ una grammatica equivalente a G in forma normale di Chomsky e priva di simboli inutili. Creiamo un grafo $\mathcal{G}$ avente:

- un nodo per ogni variabile $A \in V'$;
- un arco $\langle A, B \rangle$ ed un arco $\langle A, C \rangle$ per ogni produzione $A \to BC \in P'$.

Allora $L(G)$ $(= L(G'))$ è finito se e solo se $\mathcal{G}$ non ha cicli.

Se $\mathcal{G}$ non ha cicli, la finitezza è immediata: solo un numero finito di alberi di derivazione con radice etichettata da S possono essere costruiti.

Viceversa, supponiamo $\mathcal{G}$ abbia (almeno) un ciclo passante per un nodo associato ad una variabile A. Allora riusciamo a costruire un albero con un cammino dalla radice verso le foglie (qui è importante l'ipotesi che la grammatica non abbia simboli inutili) che attraversa due nodi etichettati con A. Ripetendo la dimostrazione del pumping lemma, si riescono a generare infiniti alberi di derivazione diversi. □

TEOREMA 7.8 (Appartenenza). *Data una grammatica CF $G = \langle V, T, P, S \rangle$, e una stringa z, il problema $z \in L(G)$ è decidibile.*

PROOF. Per il caso $z = \varepsilon$, dal Teorema 6.13 si ha che $\varepsilon \in L$ sse esiste $S \to A_1 \cdots A_n$ $(n \geq 0)$ in P tale per cui $A_i \in N$ per $i = 1, \ldots, n$.

Sia $z \neq \varepsilon$ e $G' = \langle V', T, P', S \rangle$ la grammatica equivalente a G in forma normale di Greibach. Allora, se z ha una derivazione, ne ha una di esattamente $|z|$ passi. Generiamo esaustivamente tutte le derivazioni di $|z|$ passi e verifichiamo se esiste una di queste che deriva z. □

L'algoritmo impiegato nella dimostrazione del Teorema 7.8 ha complessità esponenziale. Fortunatamente per chi necessita di riconoscere o refutare stringhe (ad esempio, per capire se un programma Pascal o C contiene degli errori sintattici) può avvalersi di metodi più efficienti. In particolare, l'algoritmo di Cocke-Younger-Kasami (1965–67) che è $O(|z|^3)$, oppure l'algoritmo di Valiant (1975) evente complessità $O(|z|^{2.8})$.

Si osservi che nel caso di linguaggi regolari invece tale problema viene deciso in tempo $O(n)$. Infatti, dato il DFA che accetta il linguaggio, la computazione avviene in tempo proporzionale alla lunghezza della stringa.

Per quanto riguarda il problema dell'equivalenza di due linguaggi liberi dal contesto, il problema è mostrato essere indecidibile in [**3**], Teorema 8.12.

Il problema corrispondente nel caso dei linguaggi regolari è invece decidibile (si confronti il Teorema 5.12).

CHAPTER 8

Le grammatiche regolari

In questo breve capitolo si studieranno due famiglie di grammatiche CF che generano esattamente i linguaggi regolari.

Una grammatica CF si dice *lineare destra* se ogni produzione è della forma:

$$A \to wB \quad w \in T^*, \text{ oppure}$$

$$A \to w \quad w \in T^*$$

più eventualmente $S \to \varepsilon$. Se invece tutte le produzioni sono della forma:

$$A \to Bw \quad w \in T^*, \text{ oppure}$$

$$A \to w \quad w \in T^*$$

più eventualmente $S \to \varepsilon$, si dice *lineare sinistra.*

LEMMA 8.1. *Data una grammatica lineare destra G esiste una grammatica G' equivalente (in forma normale di Greibach) tale che tutte le produzioni sono della forma:*

$$A \to aB \quad a \in T, \textit{ oppure}$$

$$A \to a \quad a \in T$$

più eventualmente $S \to \varepsilon$.

PROOF. Ogni produzione della forma $A \to a_1 a_2 \cdots a_n B$ viene sostituita dalle produzioni: $A \to a_1 C_1, C_1 \to a_2 C_2, \ldots, C_{n-1} \to a_n B$, ove $C_1, \ldots, C_n$ sono nuovi simboli non terminali. Similmente, $A \to a_1 a_2 \cdots a_n$ viene sostituita dalle produzioni: $A \to a_1 C_1, C_1 \to a_2 C_2, \ldots, C_{n-1} \to a_n$. □

TEOREMA 8.2. *Se L è generato da una grammatica lineare destra, allora L è un linguaggio regolare.*

PROOF. Grazie al Lemma 8.1 possiamo supporre che $G = \langle V, T, P, S\rangle$ tale che $L(G) = L$ sia nella forma semplificata descritta nell'enunciato di tale lemma. Costruiremo un NFA $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ che riconosce L.

- $Q = V \cup \{\bot\}$;
- $\Sigma = T$;
- $B \in \delta(A, a)$ sse $A \to aB \in P$, $\bot \in \delta(A, a)$ sse $A \to a \in P$;
- $q_0 = S$;
- $$F = \begin{cases} \{\bot, S\} & \text{se } S \to \varepsilon \in P \\ \{\bot\} & \text{altrimenti} \end{cases}$$

Si deve mostrare che $\hat{\delta}(S, w) \cap F \neq \emptyset$ se e solo se $S \overset{G}{\Rightarrow}_* w$, per ogni stringa w.

Iniziamo mostrando, per induzione su $|w| \geq 0$, che per ogni $A \in V$ (dunque $A \neq \bot$)

$$A \in \hat{\delta}(S, w) \quad \text{sse} \quad S \overset{G}{\Rightarrow}_* wA$$

Base: Sia $w = \varepsilon$. Per definizione, $\hat{\delta}(S, \varepsilon) = \{S\}$. Per la forma della grammatica $S \overset{G}{\Rightarrow}_* \varepsilon A$ sse $S = A$.
Passo: Sia $w = va$.

$$\begin{array}{lll} A \in \hat{\delta}(S, va) & \text{sse} \quad A \in \bigcup_{B \in \hat{\delta}(S,v)} \delta(B, a) & \text{def di } \hat{\delta} \\ & \text{sse} \quad A \in \bigcup_{B\,:\,S \overset{G}{\Rightarrow}_* vB} \delta(B, a) & \text{ip. induttiva} \\ & \text{sse} \quad S \overset{G}{\Rightarrow}_* vaA & \text{def di } \delta \end{array}$$

Da questo risultato, per definizione di F, si ha che:

1. $S \overset{G}{\Rightarrow}_* \varepsilon$ se e solo se $\hat{\delta}(S, \varepsilon) \cap F \neq \emptyset$;
2. $S \overset{G}{\Rightarrow}_* va$ se e solo se esiste A tale che $S \overset{G}{\Rightarrow}_* vA$ e $A \to a \in P$. Ciò accade se e solo se $A \in \hat{\delta}(S, v)$ e $\bot \in \delta(A, a)$, ovvero $\bot \in \hat{\delta}(S, va)$.

□

TEOREMA 8.3. *Se L è un linguaggio regolare, allora L è generato da una grammatica lineare destra.*

PROOF. Sia $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ il DFA che riconosce L. Costruiamo una grammatica lineare destra $G = \langle V, T, P, S\rangle$ tale che $L(G) = L$ nel modo seguente:

- $V = Q$;
- $T = \Sigma$;
- $$\begin{cases} A \to aB \in P & \text{sse } \delta(A,a) = B \\ A \to \varepsilon \in P & \text{sse } A \in F \end{cases}$$
- $S = q_0$.

Mostriamo che $\hat{\delta}(q_0, w) = p$ se e solo se $q_0 \overset{G}{\Rightarrow}_{|w|} wp$ per induzione su $|w|$. Il risultato segue dalle ε-transizioni per gli stati di F. □

Il Teorema 8.3 implica che ogni linguaggio regolare è anche CF.

Si possono mostrare risultati analoghi ai teoremi 8.2 e 8.3 per le grammatiche lineari sinistre (si veda [**3**]).

CHAPTER 9

La gerarchia di Chomsky

In questo capitolo riassumeremo brevemente dei risultati (alcuni dei quali presenti in questo testo) che permettono di classificare le grammatiche in base all'espressività dei linguaggi generabili.

1. Grammatiche di tipo 0

Le *grammatiche di tipo 0* (a struttura di frase) sono le grammatiche della forma $G = \langle V, T, P, S\rangle$ in cui P contiene produzioni

$$\alpha \to \beta$$

ove $\alpha \in (V \cup T)^+, \beta \in (V \cup T)^*$.

TEOREMA 9.1. *$L = L(G)$ per G grammatica a struttura di frase se e solo se L è un insieme ricorsivamente enumerabile.*

PROOF. (Si veda [**3**]) □

2. Grammatiche di tipo 1

Le *grammatiche di tipo 1* (dipendenti dal contesto) sono le grammatiche della forma $G = \langle V, T, P, S\rangle$ in cui P contiene produzioni

$$\alpha \to \beta$$

ove $\alpha \in (V \cup T)^+, \beta \in (V \cup T)^+$, e $|\alpha| \leq |\beta|$.

Per queste grammatiche esiste una forma normale per le produzioni, che devono essere del tipo:

$$\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$$

con $\beta \neq \varepsilon$.

TEOREMA 9.2. *Ogni linguaggio L generato da una grammatica dipendente dal contesto è ricorsivo.*

PROOF. (Si veda [**3**]) □

Tuttavia:

TEOREMA 9.3. *Ci sono insiemi ricorsivi che non sono generati da nessuna grammatica dipendente dal contesto.*

PROOF. (Si veda [**3**]) □

Nell'Esempio 7.2 abbiamo mostrato che $\{a^i b^i c^i : i \geq 1\}$ non è un linguaggio CF. Mostreremo ora che tale linguaggio è un linguaggio dipendente dal contesto.

ESEMPIO 9.4. La grammatica di tipo 1:

$$\begin{aligned} S &\rightarrow aSBC \,|\, aBC \\ CB &\rightarrow BC \\ bB &\rightarrow bb \\ bC &\rightarrow bc \\ cC &\rightarrow cc \\ aB &\rightarrow ab \end{aligned}$$

genera esattamente il linguaggio $\{a^i b^i c^i : i \geq 1\}$.

3. Gerarchia

Ricordiamo:

- dagli esempi 5.3 e 6.4 sappiamo che il linguaggio $\{a^i b^i : i \geq 0\}$ è un linguaggio CF ma non regolare. Sappiamo inoltre che le grammatiche (CF) lineari destre e sinistre generano esattamente i linguaggi regolari.
- Dagli esempi 7.2 e 9.4 sappiamo che il linguaggio $\{a^i b^i c^i : i \geq 1\}$ è un linguaggio dipendente dal contesto ma non CF.
- Dai Teoremi 9.3 e 9.1 sappiamo che la classe di linguaggi generati da grammatiche dipendenti dal contesto è inclusa strettamente nella classe di linguaggi generati da grammatiche a struttura di frase.

Pertanto viene indotta una gerarchia di grammatiche, nota come *gerarchia di Chomsky*, fatta di inclusioni proprie, che si può trovare in Figura 1.

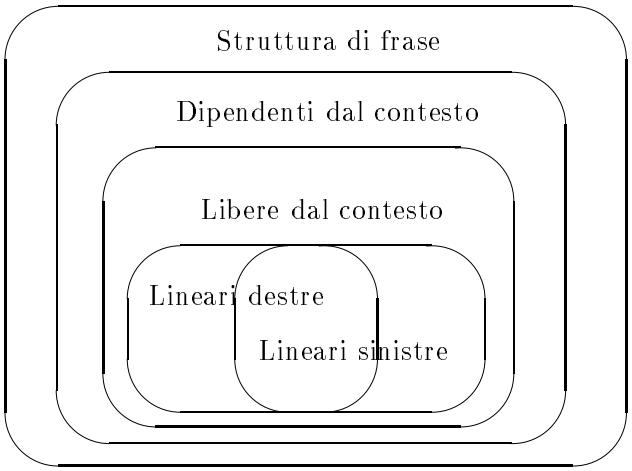


FIGURE 1. La gerarchia di Chomsky

Part 2

Teoria della calcolabilità

CHAPTER 10

Alcuni sistemi formali e la Tesi di Church

In questo capitolo presenteremo alcuni sistemi formali introdotti per studiare in modo rigoroso il concetto intuitivo di funzione calcolabile.

Un sistema formale $\mathcal{S}$ è, intuitivamente, un sistema di formule e regole per combinare formule, descritto mediante un insieme finito o numerabile S di simboli, e tale che per ogni regola $R \subseteq S^k$ per un qualche $k \in \mathbb{N}$, sia possibile stabilire in modo finito (decidibile) se una formula f è *conseguenza* di $f_1, \ldots, f_{k-1}$ secondo R, ovvero se $\langle f_1, \ldots, f_{k-1}, f \rangle \in R$. Il senso dei sistemi formali è quello di definire in modo rigoroso, mediante simboli e regole, l'impalcatura entro cui derivare asserzioni su una data realtà. Il calcolo proposizionale e la logica del prim'ordine sono esempi noti di sistemi formali.

In questo capitolo studieremo alcuni sistemi formali per descrivere in modo rigoroso la nozione intuitiva di calcolabilità. Ciò permetterà di stabilire al tempo stesso i limiti intrinseci dell'informatica moderna e studiare alcuni formalismi per rappresentare il "calcolo", apparentemente lontani tra loro, ma equivalenti nel potere espressivo riguardante ciò che è calcolabile.

La Parte 2 del testo, che riguarda questi argomenti, è strutturata nel modo seguente: nel par. 1 del presente capitolo sarà introdotta la nozione informale ed intuitiva di algoritmo, o funzione calcolabile. In seguito, nel Capitolo 11 sarà presentata la nozione di Macchina di Turing (MdT in breve), come primo sistema formale per definire il concetto di funzione calcolabile. L'importanza della MdT è, oltre che culturale, essendo una delle prime formalizzazioni del concetto di calcolabilità, anche tecnica: mediante MdT sono attualmente formalizzate le nozioni più importanti nella teoria moderna della complessità degli algoritmi. Nel Capitolo 12 introdurremo le funzioni parziali ricorsive, come modello matematico per le funzioni calcolabili. L'equivalenza tra

questi due modelli di calcolo ci porterà nel Capitolo 13, all'analisi della *Tesi di Church* e successivamente, nei Capitoli 14 e 15 all'enunciazione di alcuni risultati generali sulla calcolabilità ed alcuni problemi algoritmicamente insolubili.

1. Nozione informale di algoritmo

Discuteremo ora le caratteristiche che deve avere un algoritmo partendo da quella che è l'esperienza e l'idea intuitiva che tutti ne abbiamo. Innanzitutto, guardando superficialmente un programma osserviamo che esso ha, indipendentemente dal problema che intende risolvere, delle caratteristiche comuni.

a: Un programma è di lunghezza finita.
b: Esiste un agente di calcolo che porta avanti il calcolo eseguendo le istruzioni del programma.
c: L'agente di calcolo ha a disposizione una memoria dove vengono immagazzinati i risultati intermedi del calcolo.
d: Il calcolo avviene per passi discreti.
e: Il calcolo non è probabilistico.

I punti **a**–**c** hanno una ovvia interpretazione. Il punto **d** afferma che il calcolo non avviene mediante dispositivi analogici. Il punto **e** afferma che il calcolo non obbedisce a nessuna legge di probabilità. Associato al punto **a** parleremo di *espressioni simboliche* ovvero espressioni in un linguaggio di simboli che costituisce il linguaggio dei programmi.

Altre caratteristiche degli algoritmi sono:

f: Non deve esserci alcun limite finito alla lunghezza dei dati di ingresso.
g: Non deve esserci alcun limite alla quantità di memoria disponibile.

Mentre il punto **f** è ragionevole: ad esempio un algoritmo di somma deve poter funzionare per ogni possibile addendo, ovvero numero naturale, per il punto **g** è necessario un chiarimento. Per evidenziare la necessità di assumere una memoria illimitata facciano osservare che, limitandola, alcuni algoritmi noti per calcolare semplici funzioni non potrebbero funzionare. Ad esempio la funzione $\lambda x.\ x^2$ non sarebbe calcolabile poichè lo spazio di memoria necessario per calcolare il quadrato di x dipende da x, e per il punto **f**, esso deve essere illimitato.

Le seguneti osservazioni sono essenziali per comprendere la natura del calcolo e la sua complessità.

h: Deve esserci un limite finito alla complessità delle istruzioni eseguibili dal dispositivo.

i: Sono ammesse esecuzioni con un numero di passi finito ma illimitato.

Il punto **h**, in relazione al punto **a**, stabilisce la intrinseca finitezza del dispositivo di calcolo. Ad esempio, pensando ad un calcolatore, esso sarà in grado di calcolare solo indirizzando direttamente una parte finita della sua memoria (registri o memoria RAM) stabilendo quindi un limite nella complessità delle istruzioni eseguibili. Questo è un punto chiave nell'analisi che vedremo della nozione di effettiva calcolabilità. Un dispositivo di calcolo può effettivamente tenere traccia solo di un numero finito di simboli, ovvero esso può reagire (eseguendo un istruzione) tenendo conto di un numero finito di simboli ricordati. Questo venne intuitivamente giustificato da Turing che per primo analizzò formalmente il concetto di effettiva calcolabilità, con l'intrinseca limitazione della memoria umana. Tuttavia, non c'è limite alla memoria ausiliaria, come visto nel punto **g**. Per quanto riguarda i punti **i** e **l**, osserviamo che non essendo limitabile il numero di passi richiesti per eseguire un generico algoritmo (si pensi alla moltiplicazione o alla funzione $\lambda x.\ x^2$ discussa nel punto **g**), non è possibile stabilire a priori un limite massimo sul numero di passi nell'esecuzione di un algoritmo. Uno studio approfondito su come legare il numero di passi alla lunghezza dei dati è argomento trattato nella teoria della *complessità* degli algoritmi.

Secondo il modello precedente, un algoritmo definisce una funzione, ovvero una associazione $\{input\} \rightsquigarrow \{output\}$. Questa può essere rappresentata matematicamente come una funzione sui dati manipolati dall'algoritmo. Possiamo pertanto asserire che un algoritmo "calcola" funzioni. Diremo quindi che una funzione f è calcolabile se esiste un algoritmo ed un agente di calcolo tale che ad ogni input x restituisce come risultato del calcolo $f(x)$.

Tuttavia, una definizione intuitiva di questo tipo riguardante ciò che e' intuitivamente calcolabile non è soddisfacente. Supponiamo di

trattare solo funzioni sui numeri naturali.[1] Supponiamo di poter descrivere in questo modo tutti i processi di calcolo sui numeri naturali, ovvero che una funzione $f : \mathbb{N} \longrightarrow \mathbb{N}$ sia calcolabile se esiste un programma ed un agente di calcolo come descritti nei punti **a**–**i** sopra, tale che per ogni input $x \in \mathbb{N}$, l'algoritmo resituisce $f(x) \in \mathbb{N}$. Ovvero, secondo queste ipotesi, una funzione è calcolabile se esiste un programma ed un agente di calcolo che terminano restituendo comunque un risultato, per ogni input. Pur non assumendo limiti nel tempo e spazio impiegati dall'agente di calcolo per eseguire i suoi conti, le precedenti ipotesi stabiliscono che ciò che è calcolabile lo sia comunque, per ogni input, a seguito di un insieme potenzialmente illimitato (ma finito) di passi elementari di calcolo eseguiti dall'agente. Ovvero le funzioni considerate come "calcolabili" secondo le ipotesi **a**–**i** sono tutte *totali*, ovvero sempre definite per ogni input. Con queste ipotesi si dimostra che non si è mai in grado di descrivere tutto ciò che è "intuitivamente calcolabile". Ovvero, sarà sempre possibile costruire funzioni, intuitivamente calcolabili e totali, non esprimibili nel sistema formale adottato. Questo chiaramente rappresenterebbe un limite insormontabile alla nostra teoria, rendendola inadeguata a trattare la nozione di calcolabilità. Torneremo spesso nel seguito sul problema di stabilire in modo rigoroso cosa deve essere assunto come "calcolabile", in modo da definire formalmente una nozione di calcolabilità che sia adeguata e soddisfacente, ovvero che non ammetta limitazioni di questo tipo. Ci basti per ora il seguente argomento "informale" con il quale arriveremo alla necessità di estendere i precedenti punti **a**–**i** con una nuova ipotesi fondamentale.

È chiaro che, essendo i programmi finiti ed essendo possibile descrivere l'agente di calcolo mediante un insieme finito di simboli e regole che ne disciplinano il calcolo (programmi ed agenti costituiscono a tutti gli effetti un sistema formale), è possibile enumerare i programmi e gli agenti di calcolo. Sia dunque P_x l'x-esimo programma-agente che calcola una data funzione g_x. Per le ipotesi **a**–**i** la funzione g_x è chiaremente totale, ovvero sempre definita. Definiamo la funzione

[1] Questa non è una limitazione, poichè, come vedremo in seguito, ogni struttura dati, anche la più complessa, può essere messa in corrispondenza biunivoca con un sottoinsieme dei numeri naturali, modulo una opportuna codifica dei dati in numeri.

$h(x) = g_x(x) + 1$. È chiaro che la funzione h è calcolabile: tutti noi conosciamo un algoritmo (programma + agente di calcolo) per incrementare un naturale, ed inoltre per ipotesi g_x è calcolabile, essendo calcolata dal programma-agente P_x. Quindi, essendo h calcolabile, deve esistere un programma-agente che calcola h, e che avrà un dato indice nella numerazione. Sia questo indice x_0, ovvero h sia calcolata dal programma-agente P_{x_0}, ovvero $h = g_{x_0}$. Si ottiene in questo modo una evidente contraddizione:

$$g_{x_0} = h(x_0) = g_{x_0} + 1$$

Non è possibile che una funzione sempre definita sui naturali sia uguale al successore di se stessa!! La funzione h è quindi non calcolabile nel nostro formalismo scelto, che pertanto, essendo h intuitivamente calcolabile, non esprime tutto ciò che è intuitivamente calcolabile.

Per ovviare a questa limitazione, si assumerà che un processo di calcolo possa *non terminare*. Questo corrisponde all'intuizione che, un programma chiaramente non terminante del tipo:

$$read(x);\ \textbf{while}\ true\ \textbf{do}\ \cdots\ \textbf{endw}$$

corrisponde ad una funzione non definita su tutti gli argomenti, per la quale sappiamo pensare ad un programma (quello sopra) ed un agente di calcolo che la calcola (un computer).

È quindi necessario, per superare la precedente limitazione dei punti **a**–**i**, assumere che, oltre ai punti **a**–**i** vale la seguente ipotesi:

l: Sono ammesse esecuzioni con un numero di passi infinito (cioè non terminanti).

Il punto **l** rappresenta una necessità intrinseca, e come abbiamo visto irrinunciabile, nella trattazione matematica di ciò che è rffettivamente calcolabile. Secondo questa ipotesi, la funzione h descritta in precedenza può risultare indefinita su alcuni argomenti (ad esempio x_0), ovvero indicando con $\uparrow$ il simbolo di indefinito, la precedente uguaglianza può risultare valida (non contraddittoria):

$$\uparrow = g_{x_0} = h(x_0) = g_{x_0} + 1 = \uparrow .$$

CHAPTER 11

Macchine di Turing

In questo capitolo introduciamo le Macchine di Turing (MdT), ideate da Alan Turing negli anni '30 per rappresentare in modo formale una macchina in grado di eseguire algoritmi costituiti da passi elementari e discreti di calcolo, secondo la definizione informale data precedentemente. Turing, nella sua analisi del concetto di calcolabilità, introduce per primo la parola "computer", con il significato di una persona che fa un calcolo composto da passi elementari e discreti utilizzando un foglio di carta (memoria). Questa nozione di computer include tutte le caratteristiche di base dei calcolatori moderni (memoria, input/output, stato) e, come vedremo, modella in modo soddisfacente il concetto di effettiva calcolabilità.

1. Descrizione modellistica e matematica

Nell'eseguire un calcolo (manuale), dato un input, calcoliamo un risultato con l'ausilio della scrittura su carta. L'idea di Turing è quella di pensare ad una semplice macchina in grado di scrivere simboli su un nastro potenzialmente infinito, rappresentando il fatto (anche espresso nel punto **g**) che nei calcoli abbiamo solitamente a disposizione una quantità illimitata di carta dove registrare i risultati parziali del calcolo. L'organizzazione del nastro è fatta mediante *celle* che rappresentano la quantità di memoria unitaria. La macchina così pensata utilizzerà un alfabeto finito, col quale possiamo rappresentare una infinità di dati. Il corpo pensante della macchina (detto controllo) sarà realizzato mediante un *automa a stati finiti deterministico* (DFA). Questa restrizione modella perfettamente il punto **h** precedente, dove è stata assunta una complessità limitata nell'esecuzione delle istruzioni da parte della macchina. Tutto questo viene rappresentato matematicamente con una macchina detta *Macchina di Turing* (MdT in breve).

Una MdT è un dispositivo di calcolo rappresentabile con un *nastro* di lunghezza infinita nel quale vengono immagazzinati i dati o sequenze di simboli del calcolo. Uno di questi simboli è \$ rappresentante l'assenza di simboli (spazio bianco). Il controllo della MdT ha accesso al nastro attraverso una *testina* di lettura e scrittura che permette di leggere o scrivere un simbolo alla volta. Una MdT è pertanto costituita da due parti: il programma finito secondo cui verrà eseguito il calcolo e gli organi meccanici per lo scorrimento del nastro e il comando della testina. La struttura a stati finiti della parte di controllo modella una macchina con una memoria che potremmo dire a *breve termine*, finita, mentre il nastro modella una memoria a *lungo termine*, potenzialmente infinita. Questa distinzione tra memoria a breve e lungo termine è riscontrabile anche nelle architetture dei moderni calcolatori.

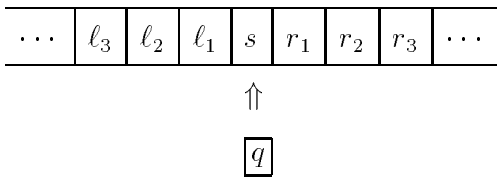
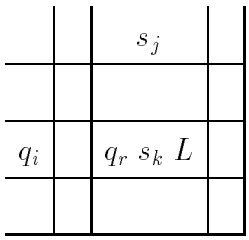


FIGURE 1. Il modello meccanico della Macchina di Turing

Ad ogni istante nel calcolo, il simbolo presente nella casella esaminata dalla testina rappresenta l'input alla macchina (in Figura 1, il simbolo s). In risposta la macchina può decidere di modificare il simbolo e/o spostare il nastro a sinistra o a destra della casella esaminata. Questo permette di avere in input un simbolo diverso per eseguire il *passo* successivo della computazione (il simbolo ℓ_1 se lo spostamento sarà a sinistra, r_1 se a destra). Inoltre la testina (che si trova nello stato q—memoria finita) può cambiare il suo stato scegliendolo da un insieme finito di stati possibili.

Il comportamento di una MdT è descrivibile mediante una tabella detta *matrice funzionale* della macchina in cui le righe rappresentano gli stati del controllo mentre le colonne rappresentano i simboli di ingresso. In una generica cella è descritta l'azione eseguita dalla

macchina nello stato corrispondente in riga e leggendo dal nastro il simbolo corrispondente in colonna.



L'azione compiuta, essendo il controllo nello stato q_i e leggendo il simbolo s_j, è in questo caso:

- portarsi in uno stato (interno) q_r
- scrivere il simbolo s_k
- spostare il nastro a destra R o a sinistra L.

Nel caso $s_j = s_k$, la macchina lascia inalterato il simbolo sul nastro. Se la casella corrispondente alla coppia $\langle s_j, q_i \rangle$ è vuota, la macchina si ferma. Una *computazione* di una MdT sarà dunque una sequenza di passi compiuti dalla macchina.

Più formalmente definiamo una MdT come segue.

DEFINIZIONE 11.1. Una *Macchina di Turing* M consiste di:

1. un alfabeto finito $\Sigma = \{s_0, \ldots, s_n\}$ con almeno due simboli distinti $s_0 = \$$ (blank) e $s_1 = 0$ (tally);
2. un insieme finito di stati $Q = \{q_0, \ldots, q_m\}$ tra i quali vi è lo stato iniziale q_0 (dunque Q è non vuoto);
3. un insieme finito non vuoto di istruzioni (o quintuple) $P = \{I_1, \ldots, I_p\}$ ognuna delle quali di uno dei seguenti 2 tipi base:
   - $q\ s\ q'\ s'\ R$
   - $q\ s\ q'\ s'\ L$

   tale che non esistono due istruzioni che iniziano con la medesima coppia q, s.

Si può dunque immaginare P come la descrizione "estensionale" (ovvero per casi) di una funzione (in generale) parziale $\delta : Q \times \Sigma \longrightarrow Q \times \Sigma \times \{R, L\}$. Tale funzione δ è detta *funzione di transizione*.

Si osservi la caratteristica *locale* delle istruzioni: il loro significato è univocamente determinato dalla parte di nastro immediatamente

adiacente alla testina e dallo stato in cui la macchina si trova. Lo stato in cui si trova l'intera MdT in un certo istante del calcolo è dunque esprimibile mediante la nozione di *descrizione istantanea* di una MdT, che rappresenta lo stato interno del controllo, il posizionamento della testina e il nastro.

DEFINIZIONE 11.2. Una *descrizione istantanea* (ID) della macchina è

$$v\ s\ q\ w$$

dove $v, w \in \Sigma^\star$ rappresentano i caratteri significativi (cioè diversi da \$) presenti sul nastro a sinistra ed a destra della testina, s è il simbolo letto dalla testina, essendo la macchina nello stato q.

Ad esempio, se la situazione è

$$\begin{array}{c} \ldots \$\ \$\ \$\ s_1 \ldots s_{i-1}\ \overset{s_i}{}\ s_{i+1} \ldots s_j\ \$\ \$\ \$ \ldots \\ \Uparrow \\ \boxed{q} \end{array}$$

allora $v = s_1 \ldots s_{i-1}$, $s = s_i$ e $w = s_{i+1} \ldots s_j$.

Vediamo ora come definire formalmente il generico passo di calcolo di una MdT. Definiamo una relazione tra descrizioni istantanee, ovvero una relazione $\vdash \subseteq ID \times ID$, che rappresenti l'esecuzione di un singolo passo di calcolo in una data MdT.

DEFINIZIONE 11.3. Il *successore* $\vdash$ è una "mappa" tra descrizioni istantanee definita come:

- $v\ r\ q\ (s\ w) \vdash v\ r'\ s\ q'\ w$ se $q\ r\ q'\ r'\ R$ è una istruzione di P;
- $v\ r\ q\ (s\ w) \vdash v\ q'\ r'\ s\ w$ se $q\ r\ q'\ r'\ L$ è una istruzione di P.

Una computazione sarà dunque definita come segue, ovvero come una sequenza finita di passi.

DEFINIZIONE 11.4. Una *computazione* è una sequenza *finita* di ID $\alpha_0, \ldots, \alpha_n$ tale che

- $\alpha_0 = v\ s\ q_0\ w$ e,
- $\alpha_i \vdash \alpha_{i+1}$ per ogni $i \in \{0, \ldots, n-1\}$.

Una computazione è detta *terminante* se esiste un $n \geq 0$ per cui $\alpha_n = v\ s\ q\ w$ e non vi è nessuna istruzione di P iniziante con q ed s (cioè $\delta(q, s)$ non è definita), i.e. $\alpha_n \nvdash$.

In accordo con il punto **l** della precedente sezione, esistono MdT che non terminano, ovvero che non calcolano nulla a partire da un dato input.

Osserviamo come una MdT così descritta soddisfa i requisiti per la definizione ragionevole di algoritmo.

- **a:** Le istruzioni della MdT sono finite.
- **b:** La MdT è l'agente di calcolo che esegue l'algoritmo.
- **c:** Il nastro rappresenta la memoria della macchina.
- **d:** La MdT opera in modo discreto.
- **e:** Ad ogni ID corrisponde una sola azione (determinismo della MdT).
- **f:** Non esiste alcun limite all'input, essendo il nastro illimitato.
- **g:** La capacità della memoria (nastro) è illimitata.
- **h:** Le operazioni eseguibili sono semplici e quindi di complessità limitata.
- **i:** Non esiste alcun limite al numero di istruzioni eseguite in quanto la medesima quintupla può essere usata più volte.
- **l:** Possono esistere MdT che non calcolano nulla generando una sequenza infinita di ID.

TEOREMA 11.5 (Determinismo). *Se* $\alpha \vdash \beta$ *e* $\alpha \vdash \gamma$ *con* $\alpha, \beta, \gamma \in ID$, *allora* $\beta \equiv \gamma$.

PROOF. Ovvio, poichè per definizione non esistono due quintuple $q\ s\ q'\ s'\ X$ aventi la medesima coppia q, s. □

ESEMPIO 11.6. Incremento di uno in binario.
Sia $\Sigma = \{\$, 0, 1\}$. Si vuole passare da una configurazione iniziale

$$\ldots \$\ \$\ \$\ s_n \ldots s_0\ \overset{\$}{}\ \$\ \$\ \$ \ldots$$

ad una finale

$$\ldots \$\ \$\ \$\ \overset{\$}{}\ s'_m \ldots s'_0\ \$\ \$\ \$ \ldots$$

ove

$$\sum_{i=0}^{m} 2^i \cdot s'_i = 1 + \sum_{i=0}^{n} 2^i \cdot s_i.$$

Una possibile definizione della funzione δ sarà la seguente:

δ	\$	0	1
q_0	q_1 \$ L		
q_1	q_2 1 L	q_2 1 L	q_1 0 L
q_2		q_2 0 L	q_2 1 L

ESEMPIO 11.7. Copia di una stringa unaria.
Sia $\Sigma = \{\$, 0\}$. Si vuole passare da una configurazione iniziale

$$\ldots \$\ \$\ \$\ \underbrace{0 \ldots 0}_{n}\ \ \$\ \ \$\ \$\ \$ \ldots$$
$$\Uparrow$$
$$\boxed{q_0}$$

ad una finale

$$\ldots \$\ \$\ \$\ \underbrace{0 \ldots 0}_{n}\ \ \$\ \ \underbrace{0 \ldots 0}_{n}\ \$\ \$\ \$ \ldots$$
$$\Uparrow$$
$$\boxed{q_f}$$

δ	0	1	2	$\cdots$	7	8	9	\$
q_0	$q_1\ 1\ R$	$q_1\ 2\ R$	$q_1\ 3\ R$	$\cdots$	$q_1\ 8\ R$	$q_1\ 9\ R$	$q_0\ 0\ L$	$q_1\ 1\ R$
q_1				$\cdots$				

FIGURE 2. MdT per l'incremento in base 10

Una possibile definizione della funzione δ sarà la seguente ($q_F \in \{q_1, q_7\}$):

δ	\$	0
q_0	$q_1\ \$\ L$	
q_1		$q_2\ \$\ R$
q_2	$q_3\ \$\ R$	$q_2\ 0\ R$
q_3	$q_4\ 0\ L$	$q_3\ 0\ R$
q_4	$q_5\ \$\ L$	$q_4\ 0\ L$
q_5	$q_6\ 0\ L$	$q_5\ 0\ L$
q_6	$q_7\ \$\ R$	$q_2\ \$\ R$
q_7		$q_7\ 0\ R$

Si provi, per esercizio, a definire δ supponendo di disporre di $\Sigma = \{\$, 0, 1\}$.

ESEMPIO 11.8. Definiamo una MdT che calcola il successore in base 10 di un numero. $S = \{0, \ldots, 9\}$, $Q = \{q_0, q_1\}$ essendo q_1 lo stato di terminazione. La matrice funzionale in figura 2 definisce la MdT desiderata che calcola il successore di un numero scritto sul nastro supponendo che la MdT inizi il calcolo essendo la testina posizionata sulla cifra meno significativa del numero da incrementare.

ESERCIZIO 11.9. Si scriva la funzione di transizione per le macchine di Turing soddisfacenti i seguenti requisiti:

1. si calcoli il successore di un numero decimale

2. decisione di quale sia la più lunga tra due stringhe di 0 poste una a sinistra ed una a destra della testina. Si scelga una opportuna forma di risposta (che tenga conto anche della possibilità che le due stringhe siano uguali);
3. si effettui la somma (in binario) di due stringhe non nulle di 0 e 1 poste entrambe a sinistra della testina ed inframmezzate da un \$;
4. si sposti la testina di n posizioni a destra rispetto a quella di partenza, ove n è letto da un input binario;
5. si calcoli il numero di occorrenze di 1 in una data sequenza binaria;
6. si ordini, con ripetizioni, una stringa in input di 0 e 1.

2. Funzioni calcolabili da MdT

Ad ogni MdT può essere facilmente associata una funzione che diremo *calcolata* dalla MdT. Poiché in questa parte del corso trattiamo solo funzioni sui naturali, è necessario fissare una opportuna codifica dei naturali come stringhe nell'alfabeto della MdT. Ad esempio è possibile utilizzare una codifica binaria nel caso $\Sigma = \{\$, 0, 1\}$ o una codifica decimale. Nella seguente definizione, assumeremo invece una codifica *unaria*, ovvero un generico numero $n \in \mathbb{N}$ sarà rappresentato come una sequenza di $n + 1$-simboli 1 consecutivi. Questa codifica è particolarmente conveniente, anche se del tutto inessenziale, per la seguente definizione di funzione calcolabile da una MdT.

DEFINIZIONE 11.10. Una funzione $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ è *Turing–calcolabile* se esiste una MdT tale che, partendo dalla configurazione iniziale

$$\ldots \$\,\$\,\$\,\underline{x_1}\,\$ \ldots \$\,\underline{x_n} \quad \$ \quad \$\,\$\,\$\ldots$$
$$\Uparrow$$
$$\boxed{q_0}$$

termina nella configurazione

$$\ldots \$\,\$\,\$\ \ldots \$\,\underline{f(x_1, \ldots, x_n)} \quad \$ \quad \$\,\$\,\$\ldots$$
$$\Uparrow$$
$$\boxed{q_f}$$

se $f(x_1, \ldots, x_n)$ è definita, non termina altrimenti; dove per ogni $(x_1, \ldots, x_n) \in \mathbb{N}^n$, $\underline{x_1}, \ldots, \underline{x_n}, \underline{f(x_1, \ldots, x_n)}$ sono le rappresentazioni in unario rispettivamente di $x_1, \ldots, x_n, f(x_1, \ldots, x_n)$ (mentre $q_f \in Q$ è tale per cui $\delta(q_F, \$)$ è indefinito).

Chiaramente, una funzione sui naturali Turing-calcolabile è parziale, poichè può non essere definita su alcuni (o tutti) gli argomenti. Vedremo in seguito il significato più profondo di funzione parziale nella teoria della calcolabilità.

ESEMPIO 11.11. Le seguenti funzioni sono Turing-calcolabili:

- $\lambda\, x.x$: $Q = \{q_0\}$; δ sempre indefinito.
- $\lambda\, x.0$:

δ	\$	0
q_0	q_1 \$ R	
q_1		

- $\lambda\, x.x + 1$:

δ	\$	0
q_0	q_1 0 R	
q_1		

- $\lambda\, x_1 \ldots x_n.x_i$:
  for $j = 0, \ldots, i-1$
  $\delta(q_j, 0) = (q_j, 0, L)$;
  $\delta(q_j, \$) = (q_{j+1}, \$, L)$.

ESERCIZIO 11.12. Si supponga di disporre delle definizioni per le macchine di Turing computanti le funzioni:

$f : \mathbb{N}^k \longrightarrow \mathbb{N}$,

$g_1, \ldots, g_k : \mathbb{N}^n \longrightarrow \mathbb{N}$.

Come esercizio si mostri che la funzione

$h : \mathbb{N}^n \longrightarrow \mathbb{N}$,

definita come

$h(x_1, \ldots, x_n) \stackrel{\text{def}}{=} f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$,
è Turing calcolabile.

NOTA 11.13. Si osservi che, se esiste una MdT computante la funzione f, allora ne esistono infinite calcolanti la stessa funzione (si aggiungano arbitrariamente stati o simboli 'inutili').

3. MdT generalizzate

Il modello di MdT che abbiamo visto è solo uno dei possibili modelli matematici per definire l'effettiva calcolabilità di funzioni. È infatti possibile definire una vasta gamma di MdT differenti ma equivalenti rispetto alla classe di funzioni Turing-calcolabili che esse possono indurre. Questo perchè qualunque MdT ottenuta anche con modifiche strutturali (per esempio aumentando il numero dei nastri o delle testine) se soddisfa le 10 condizioni che abbiamo definito per caratterizzare la nozione intuitiva di algoritmo, può essere simulata da una MdT definita come nella Definizione 11.1. I seguenti risultati, dei quali non riportiamo dimostrazione, forniscono un esempio di come si possa modificare il modello di MdT senza modificare l'insieme delle corrispondenti funzioni calcolabili. La potenza del formalismo delle MdT non viene dunque ridotta/aumentata modificando l'insieme dei simboli/stati o modificando la struttura stessa della macchina (numero di nastri e/o testine).

- Una MdT con n nastri ed m testine ($m \geq n$) può essere simulata da una MdT con 1 nastro ed una testina;
- Una MdT con n simboli ed m stati può essere simulata da una MdT con 2 stati, aumentando opportunamente il numero dei suoi simboli;
- Una MdT con n simboli ed m stati può essere simulata da una MdT con 2 simboli, aumentando opportunamente il numero dei suoi stati;
- Ogni MdT può essere simulata da una MdT che può solo scrivere e non rimpiazzare simboli sul nastro.

Per approfondimenti su queste tematiche, si consulti, ad esempio [**3**].

CHAPTER 12

Funzioni parziali ricorsive di Kleene & Robinson

Lo scopo primario di questa parte del corso è di studiare le funzioni sui numeri naturali, esprimibili in modo effettivo o algoritmico. La definizione intuitiva del concetto di funzione effettivamente calcolabile o funzione calcolabile mediante un algoritmo, ovvero mediante una sequenza discreta di passi elementari di calcolo di una MdT, fornita nel capitolo precedente, può sembrare limitativa rispetto a ciò che intuitivamente ci sembra effettivamente calcolabile. Per valutare la robustezza di questa formalizzazione del concetto di funzione calcolabile, nei seguenti due paragrafi studieremo diverse nozioni di calcolabilità che, come vedremo, si equivalgono e sono a loro volta tutte equivalenti alla calcolabilità mediante MdT.

Nel Paragrafo 1 studieremo la caratterizzazione di effettiva calcolabilità sui naturali nota con il termine di *funzione ricorsiva*. Arriveremo alla trattazione delle *funzioni parziali ricorsive*, dette anche *funzioni ricorsive generali*, di Kleene & Robinson, arricchendo via via con nuovi costrutti, o metodi di composizione, alcune funzioni elementari sui naturali, dette *funzioni ricorsive di base*.[1] Questi costrutti, come vedremo nel Cap. 16, corrispondono a costrutti noti ed ampiamente utilizzati nei moderni linguaggi di programmazione.

1. Funzioni primitive ricorsive

L'insieme dei numeri naturali è facilmente caratterizzabile mediante *induzione*, ovvero affermando che ogni numero naturale n è esprimibile come una iterazione della operazione elementare di *successore* applicata al valore costante 0: $n = S^n(0)$. Pertanto

$$\mathbb{N} = \{0, S(0), S(S(0)), S(S(S(0))), \dots\}$$

[1] Nel seguito per *funzione ricorsiva* intenderemo una funzione ricorsiva totale. Nel caso di funzioni parziali, parleremo di *funzioni parziali ricorsive*.

Nel seguito indicheremo spesso $S(x)$ con $x+1$. Questo procedimento induttivo per definire i numeri naturali si basa sull'idea che ogni numero naturale è definibile a partire da un numero più semplice (in questo caso minore nell'ordinamento usuale sui naturali) applicando l'operazione di successore. Questo modo di procedere è comune in matematica e informatica, e trova svariate applicazione nelle così dette *definizioni ricorsive* di funzioni. Una definizione ricorsiva di funzione è una definizione dove il valore della funzione per un dato argomento è direttamente correlato al valore della medesima funzione su argomenti "più semplici" o al valore di funzioni "più semplici". Ad esempio, la seguente funzione definisce ricorsivamente la successione di Fibonacci: 1, 1, 2, 3, 5, 8, 13, ...

$$\begin{aligned} f(0) &= 1 \\ f(1) &= 1 \\ f(x+2) &= f(x+1) + f(x). \end{aligned}$$

Al fine di comprendere meglio queste definizioni è pertanto dare un significato preciso al concetto informale di "più semplice". Identifichiamo le funzioni "più semplici" in assoluto con le funzioni costanti. Trattando i naturali, consideriamo dunque la costante 0 come funzione costante elementare. Supponiamo inoltre di avere a disposizione come funzione elementare l'operazione di successore, che ci permette di definire qualsiasi numero naturale, e la funzione identica. Queste assunzioni ci portano a definire il concetto di *funzione ricorsiva di base*.

DEFINIZIONE 12.1. Si dicono *funzioni ricorsive di base* le seguenti funzioni:

- la funzione costante 0: $\lambda x.\ 0$;
- La funzione successore S: $\lambda x.\ x+1$;
- La funzione identità, o i-esima proiezione, $\pi_i = \lambda x_1 \cdots x_n.\ x_i$ con $1 \leq i \leq n$.

Formalizziamo ora il meccanismo di definizione di funzioni per composizione e ricorsione primitiva. Questo meccanismo generalizza i precedenti esempi.

DEFINIZIONE 12.2. Una funzione $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ si dice:

- definita per *composizione* da $g_1, \ldots, g_k : \mathbb{N}^n \longrightarrow \mathbb{N}$ e $h : \mathbb{N}^k \longrightarrow \mathbb{N}$ se $f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$
- definita per *ricorsione primitiva* da $g : \mathbb{N}^{n-1} \longrightarrow \mathbb{N}$ e $h : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ se

$$\begin{cases} f(x_1, \ldots, x_{n-1}, 0) = g(x_1, \ldots, x_{n-1}) \\ f(x_1, \ldots, x_{n-1}, y+1) = h(x_1, \ldots, x_{n-1}, y, h(x_1, \ldots, x_{n-1}, y)) \end{cases}$$

Tutte e sole le funzioni definibili a partire dalle precedenti funzioni ricorsive di base mediante composizione e ricorsione primitiva definiscono l'insieme delle funzioni *primitive ricorsive*. L'idea è quella di costruire via via funzioni effettivamente calcolabili a partire dalle funzioni (banalmente) effettivamente calcolabili di base, ovvero 0 e S, costruendo da queste, per composizione e ricorsione primitiva, funzioni via via più complesse

DEFINIZIONE 12.3. La classe delle funzioni *primitive ricorsive* è la più piccola classe $\mathcal{P}$ (ovvero l'intersezione di tutte le classi) di funzioni contenenti le funzioni ricorsive di base e chiuse per composizione e ricorsione primitiva.

Segue direttamente dalla definizione che, per ogni funzione f, $f \in \mathcal{P}$ sse esiste una sequenza finita di funzioni $f_1, \ldots, f_n$, tale che: $f_n = f$ e per ogni funzione f_j, con $j \leq n$, o f_j è una funzione ricorsiva di base, oppure è ottenuta mediante composizione o ricorsione primitiva a partire da funzioni $f_{i_1}, \ldots, f_{i_k}$ con $i_1, \ldots, i_k < j$ e per le quali vale la stessa proprietà. La dimostrazione di questo fatto è lasciata per esercizio.

ESEMPIO 12.4. Consideriamo la sequenza di funzioni in ottenute come visto in precedenza:

$$\begin{aligned} f_1 &= \lambda x.\, x \\ f_2 &= \lambda x.\, x+1 \\ f_3 &= \lambda x_1 x_2 x_3.\, x_2 \\ f_4 &= f_2 \circ f_3 = \lambda x_1 x_2 x_3.\, x_2 + 1 \end{aligned}$$

$$\begin{aligned} f_5 &= \text{Tale che:} \\ & \qquad f_5(0, x_2) = f_1(x_2) \\ & \qquad f_5(y+1, x_2) = f_4(y, f_5(y, x_2), x_2) \\ f_6 &= f_5(f_1, f_1) \end{aligned}$$

È facile vedere che $f_6 = \lambda x.\, 2x$ e che $f_5 = \lambda xy.\, x + y$. Dimostrare per esercizio.

Le funzioni primitive ricorsive sono molto frequenti in matematica ed informatica. Il seguente esempio elenca alcune tra le più note funzioni primitive ricorsive.

ESEMPIO 12.5. Le seguenti funzioni sono ricorsive primitive (per semplicità si denoteranno numerali con numeri):

Zero: $0^n = \lambda x_1 \cdots x_n.\, 0$: definibile come $0(\pi_i(x_1, \ldots, x_n))$;
Costante: $c^n = \lambda x_1 \cdots x_n.c$: definibile come $\underbrace{S(\cdots S}_{c}(0^n(x_1, \ldots, x_n))\underbrace{)\cdots)}_{c}$;

Somma:

$$\begin{cases} +(x, 0) = x \\ +(x, y+1) = S(+(x, y)) \end{cases}$$

Rispetto alla notazione utilizzata:

- h è $+$, di arità 2,
- g è la funzione identica, di arità 1,
- $f(x, y, z)$, di arità 3, è la funzione ottenuta come $S(\pi_3(x, y, z))$.

Un esempio di 'computazione' nel formalismo che si sta definendo è il seguente:

$$\begin{aligned} +(S(S(0)), S(S(S(0)))) &= S(+(S(S(0)), S(S(0)))) \\ &= S(S(+(S(S(0)), S(0))) \\ &= S(S(S(+(S(S(0)), 0))))) \\ &= S(S(S(S(S(0))))) \end{aligned}$$

Moltiplicazione:

$$\begin{cases} \cdot(x,0) = 0 \\ \cdot(x,y+1) = +(\cdot(x,y),y)) \end{cases}$$

Potenza:

$$\begin{cases} x^0 = S(0) \\ x^{y+1} = \cdot(x^y,x)) \end{cases}$$

Iper-potenza:

$$\begin{cases} iper(x,0) = \pi_1(x) \\ iper(x,y+1) = iper(x,y)^x \end{cases}$$

Si osservi che $iper(x,y) = \underbrace{x^{\cdots^x}}_{y} = x^{(x^y)}$

Predecessore:

$$\begin{cases} pred(0) = 0 \\ pred(y+1) = \pi_1(y) \end{cases}$$

Differenza:

$$\begin{cases} -(x,0) = \pi_1(x) \\ -(x,y+1) = pred(-(x,y)) \end{cases}$$

Fattoriale:

$$\begin{cases} 0! = S(0) \\ (y+1)! = \cdot(S(y),y!) \end{cases}$$

Segno:

$$\begin{cases} sg(0) = 0 \\ sg(y+1) = S(0) \end{cases}$$

Definiamo inoltre $\overline{sg}(x) = -(1,sg(x))$.

Valore assoluto della differenza: $|-(x,y)| = +(-(x,y),-(y,x))$;

Minimo e massimo:

$$\left\{ \begin{array}{lcl} min(x,y) & = & -(x,-(x,y)) \\ max(x,y) & = & +(x,-(y,x)) \end{array} \right\}$$

Divisione intera: Assumiamo che $div(x,0) = mod(x,0) = 0$.

$$\begin{cases} mod(0,x) = 0 \\ mod(y+1,x) = S(mod(y,x)) \cdot sg(|-(x, S(mod(y,x)))|) \end{cases}$$

$$\begin{cases} div(0,x) = 0 \\ div(y+1,x) = div(y,x) + \overline{sg}(|-(x, S(mod(y,x)))|) \end{cases}$$

Produttoria:

Sia f una funzione ricorsiva primitiva binaria, allora definiamo la funzione $\Pi_{z<y} f(x,z) = \Pi_{z=0}^{y-1} f(x,z)$:

$$\begin{cases} \Pi_{z<0} f(x,z) = S(0) \\ \Pi_{z<y+1} f(x,z) = \cdot(f(x,y), \Pi_{z<y} f(x,z)) \end{cases}$$

Sommatoria:

Sia f una funzione ricorsiva primitiva binaria, allora definiamo la funzione $\Sigma_{z<y} f(x,z) = \Sigma_{z=0}^{y-1} f(x,z)$:

$$\begin{cases} \Sigma_{z<0} f(x,z) = 0 \\ \Sigma_{z<y+1} f(x,z) = +(f(x,z), \Sigma_{z<y} f(x,z)) \end{cases}$$

μ-operatore limitato di minimizzazione:

Sia f una funzione primitiva ricorsiva $n+1$ aria.

$$\begin{aligned} g(x_1,\ldots,x_n,y) &= \mu z < y.\, (f(x_1,\ldots,x_n,z) = 0) \\ &= \begin{cases} \text{il più piccolo } z \text{ minore di } y & \\ \text{tale che } f(x_1,\ldots,x_n,z) = 0 & \text{se tale } z \text{ esiste} \\ y & \text{altrimenti} \end{cases} \end{aligned}$$

Tale funzione è primitiva ricorsiva:

$$g(x_1,\ldots,x_n,y) = \Sigma_{v<y}(\Pi_{u\le v} sg(f(x_1,\ldots,x_n,u))).$$

ESERCIZIO 12.6. Si definiscano le seguenti funzioni ricorsive primitive:

1. la funzione che, dato x, fornisce il numero dei suoi divisori;
2. la funzione che, dato x, restituisce 1 se x è primo, 0 altrimenti;
3. la funzione p che, dato x, restituisce il p-esimo numero primo (si assuma $p(0) = 0$).

4. la funzione che restituisce l'esponente di $p(y)$ nella fattorizzazione di x;
5. la funzione che restituisce 1 se e solo se x è un cubo perfetto;
6. la funzione che restituisce 1 se e solo se x è ottenibile come somma di due cubi.

TEOREMA 12.7. *Le funzioni primitive ricorsive sono totali.*

PROOF. Per induzione strutturale sulla complessità (nel senso di numero di operazioni di composizione e ricorsione primitiva) delle funzioni definite in $\mathcal{P}$ osserviamo che le funzioni ricorsive di base sono totali, e la composizione di funzioni totali è totale. Resta da dimostrare che la composizione mediante ricorsione primitiva di funzioni totali è totale. Questo passo è lasciato per esercizio. □

I precedenti esempi ci lascerebbero pensare che la nozione di funzione primitiva ricorsiva catturi esattamente il concetto intuitivo di funzione calcolabile mediante un algoritmo. Questo è falso! Innanzitutto, il Teorema 12.7 afferma che le funzioni primitive ricorsive sono totali e, come vedremo, questa è una limitazione, ovvero una funzione calcolabile può essere indefinita su alcuni (o tutti) i valori di input. Le funzioni primitive ricorsive hanno inoltre un'ulteriore limitazione anche all'interno delle funzioni totali sui naturali. Per dimostrare questo fatto, è sufficente dare un testimone, ovvero una funzione evidentemente calcolabile e totale, ma non appartenente a $\mathcal{P}$. Una tale funzione è la *funzione di Ackermann*.

La funzione di Ackermann è una funzione in 3 argomenti, $ack : \mathbb{N}^3 \longrightarrow \mathbb{N}$, definita come segue:

$$\begin{cases} ack(0,0,y) = y \\ ack(0,x+1,y) = ack(0,x,y)+1 \\ ack(1,0,y) = 0 \\ ack(z+2,0,y) = 1 \\ ack(z+1,x+1,y) = ack(z,ack(z+1,x,y),y). \end{cases}$$

Per esercizio si verifichi che la funzione è totale (ed è ben definita, cioè non vi sono chiamate ricorsive ad oggetti 'più grandi').

Vediamo di capire il significato della definizione di ack. Dalle prime due equazioni segue che $ack(0, x, y) = y + x$. La terza e quarta definzione costituiscono le condizioni iniziali per la quinta che è la vera e propria chiamata ricorsiva di ack. Questa afferma che $\lambda xy.\ ack(z + 1, x, y)$ è ottenuta calcolando

$$y_0 = ack(z+1, 0, y) = \begin{cases} 0 & \text{se } z+1 = 1 \\ 1 & \text{se } z+1 > 1 \end{cases}$$

e quindi applicando la funzione $\lambda wy.\ ack(z, w, y)$ x volte, ovvero la sequenza:

$$\begin{array}{lcll} y_1 & = & ack(z+1, 1, y) & = & ack(z, y_0, y) \\ y_2 & = & ack(z+1, 2, y) & = & ack(z, y_1, y) \\ y_3 & = & ack(z+1, 3, y) & = & ack(z, y_2, y) \\ \ldots & & \ldots & & \ldots \\ y_x & = & ack(z+1, x, y) & = & ack(z, y_{x-1}, y) \end{array}$$

Quindi abbiamo:

$$\begin{array}{lcll} f(0, x, y) & = & y + x & \\ f(1, x, y) & = & y \cdot x & \\ f(2, x, y) & = & y^x & \\ f(3, x, y) & = & y^{y^{\cdot^{\cdot^{y}}}} \Big\} & x\text{-volte} \\ \ldots & & \ldots & \end{array}$$

Ad esempio, $ack(3, 3, 3) = 3^{27} > 10^{14}$.

TEOREMA 12.8. *ack non è ricorsiva primitiva* [**9**].

La dimostrazione del Teorema 12.8 è piuttosto complessa ed è basata sul fatto che ack cresce più velocemente di qualunque funzione ricorsiva primitiva. Questa funzione, universalmente accettata come funzione calcolabile non è primitiva ricorsiva pur essendo totale. Inoltre esistono infinite funzioni totali e chiaramente calcolabili che

non sono esprimibili mediante ricorsione primitiva: basta aggiungere alla funzione *ack* una generica costante.

2. Diagonalizzazione

In questa sezione studieremo uno degli strumenti fondamentali della teoria delle funzioni calcolabili: la *diagonalizzazione*. Il suo uso, già evidenziato nella precedente sezione 1, sarà frequente nel seguito. La tecnica di diagonalizzazione fu per la prima volta utilizzata da Cantor nel 1874 per dimostrare uno dei risultati fondamentali nella teoria classica degli insiemi, ovvero la non enumerabilità dei numeri reali (Si veda il Capitolo 2).

L'idea della diagonalizzazione è la seguente: Dato un insieme S numerabile, ed una funzione totale $d : S \longrightarrow S$ che non sia mai l'identità su S, ovvero tale che $\forall s \in S.\ d(s) \neq s$, costruiamo una matrice infinita di elementi di S secondo una data enumerazione:

$$\begin{array}{cccc} s_{0,0} & s_{0,1} & s_{0,2} & \dots \\ s_{1,0} & s_{1,1} & s_{1,2} & \dots \\ s_{2,0} & s_{2,1} & s_{2,2} & \dots \\ \dots & \dots & \dots & \dots \end{array}$$

Da questa costruzione, consideriamo la diagonale $s_{0,0}$, $s_{1,1}$, $s_{2,2}$, ... che viene trasformata da d nella sequenza:

$$d(s_{0,0}), d(s_{1,1}), d(s_{2,2}) \dots .$$

La caratteristica essenziale di questa sequenza è che essa differisce da ogni altra riga della matrice. Pertanto, data una qualsiasi enumerazione degli oggetti di un sistema formale dato, in questo modo sarà sempre possibile "costruire" un oggetto che non appartiene alla numerazione, ovvero esterno al sistema formale dato.

Come esempio di applicazione di questa tecnica, dimostreremo l'inadeguatezza della classe delle funzioni primitive ricorsive nell'esprimere ciò che intuitivamente viene ritenuto effettivamente calcolabile. Ovvero mediante un "argomento diagonale" dimostreremo che è possibile definire una classe di funzioni calcolabili effettivamente e non esprimibili mediante ricorsione primitiva. In realtà la diagonalizzazione ha un ben più

ampio campo di utilizzo in quanto è applicabile ad ogni sistema formale le cui istruzioni definite su un alfabeto dato, sono effettivamente enumerabili. L'inadeguatezza della classe delle funzioni primitive ricorsive per esprimere la calcolabilità era comunque già chiara dal fatto che la funzione calcolabile di Ackermann non è primitiva ricorsiva.

Consideriamo l'insieme delle funzioni primitive ricorsive $\mathcal{P}$ ed un formalismo, ovvero un insieme finito di simboli con il quale sia possibile descrivere tutte le possibili derivazioni associate alle funzioni in $\mathcal{P}$. Un siffatto alfabeto può essere ad esempio

$$\Sigma = \{a, b, c, \ldots, z, 0, 1, 2, \ldots 9, (,)\}$$

Ogni derivazione associata ad una funzione in $\mathcal{P}$ è quindi descrivibile con una opportuna sequenza finita di simboli in Σ^*. È inoltre facile definire una procedura che verifichi se una data stringa $\sigma \in \Sigma^*$ corrisponde ad una derivazione legittima per una funzione in $\mathcal{P}$. È pertanto possibile enumerare tutte le derivazioni legittime per funzioni primitive ricorsive esaminando le stringhe in Σ di lunghezza 1, quindi quelle di lunghezza 2 etc. Sia quindi Q_x l'$(x+1)$-esima derivazione in questa lista e sia g_x la funzione di cui Q_x è derivazione. Definiamo la funzione h come segue:

$$h(x) = g_x(x) + 1$$

Evidentemente abbiamo un algoritmo per calcolare h: dato x, generiamo la lista delle derivazioni fino a Q_x, quindi utilizziamo Q_x per calcolare $g_x(x)$ e sommiamo 1. h così costruita non è primitiva ricorsiva altrimenti, se lo fosse, avremmo $h = g_{x_0}$ per un qualche x_0, e quindi l'assurdo:

$$g_{x_0}(x_0) = h(x_0) = g_{x_0}(x_0) + 1$$

Si noti l'analogia di questa dimostrazione (informale) con la dimostrazione di Cantor sulla non enumerabilità di $\wp(\mathbb{N})$. È pertanto possibile "costruire", mediante diagonalizzazione, una funzione a tutti gli effetti calcolabile che non è primitiva ricorsiva.

Il problema posto dalla diagonalizzazione sembra limitare in modo intrinseco e insuperabile ogni definizione matematica di funzione calcolabile codificabile con un alfabeto finito di simboli. Infatti, per diagonalizzazione sembra possibile costruire funzioni calcolabili che sfuggono ad ogni definizione formale, basata ad esempio su un alfabeto finito, di funzione calcolabile.

L'idea fondamentale che permette di superare la difficoltà indotta dalla diagonalizzazione su un dato sistema formale è quella di ammettere algoritmi, o insiemi di istruzioni per calcolare sia funzioni totali che *parziali*. Per comprendere meglio questo aspetto fondamentale da cui è iniziato lo studio della teoria della calcolabilità, cerchiamo di applicare un argomento diagonale al caso di sistemi formali per esprimere funzioni parziali. Osserveremo che un sistema formale che caratterizzi un sottoinsieme opportuno di funzioni parziali, non è soggetto a limitazioni dovute ad argomenti diagonali.

Sia ψ_x la funzione parziale corrispondente all'$(x+1)$-esimo insieme di istruzioni P_x in un dato sistema formale per funzioni parziali, e sia x_0 tale che ψ_{x_0} è la funzione parziale φ definita dalle seguenti istruzioni: trova P_x, calcola $\psi_x(x)$ e, se e quando si ottiene un risultato, restituisci come output di φ il valore $\psi_x(x)+1$. Trattandosi di funzioni parziali, l'equazione

$$\psi_{x_0}(x_0) = \varphi(x_0) = \psi_{x_0}(x_0) + 1$$

non genera contraddizioni, poichè $\varphi(x_0)$ può non essere definita.

Questo metodo di procedere, ovvero considerare la teoria della effettiva calcolabilità come una teoria di funzioni parziali è alla base degli approcci di Kleene, Church e Turing sviluppati negli anni '30. Tutti i sistemi formali che vedremo infatti hanno in comune una descrizione in un opportuno sistema formale, ovvero mediante un opportuno insieme di simboli e regole, di ciò che sono gli algoritmi ed una conseguente caratterizzazione del sottoinsieme delle funzioni parziali calcolabili in quel sistema formale, ovvero calcolabile dagli algoritmi così descritti. Abbiamo già visto un esempio di questo modo di procedere nella descrizione delle MdT come sistema formale per esprimere algoritmi e nella funzioni calcolabili da una MdT come descrizione di opportune funzioni parziali. Queste considerazioni generali sulla natura parziale

delle funzioni calcolabili giustificano l'assunzione **I** nella descrizione intuitiva di ciò che è effettivamente calcolabile. Indicheremo nel seguito con $f(x)\downarrow$ il fatto che $f(x)$ è definita mentre con $f(x)\uparrow$ il fatto che $f(x)$ non è definita.

3. Funzioni parziali ricorsive

Introduciamo ora un metodo generale per costruire funzioni parziali a partire da funzioni totali (ad esempio primitive ricorsive).

DEFINIZIONE 12.9. Sia f una funzione totale $n+1$ aria, allora definiamo la funzione

$$\begin{aligned}\varphi(x_1,\ldots,x_n,y) &= \mu z.\,(f(x_1,\ldots,x_n,z)=0)\\ &= \begin{cases}\text{il più piccolo } z \text{ t.c. } f(x_1,\ldots,x_n,z)=0 & \text{se } z \text{ esiste}\\ \uparrow & \text{altrimenti}\end{cases}\end{aligned}$$

Tale operatore tra funzioni è detto μ-operatore ed una funzione così definita è detta definita per *minimizzazione* o μ-ricorsione da f.

Chiaramente una funzione definita per minimizzazione è in generale, per definizione, parziale.

NOTA 12.10. Si osservi l'ipotesi che f sia una funzione totale. Questa ipotesi può essere rilasciata definendo la minimizzazione, o μ-ricorsione nel modo seguente: Sia ψ una funzione parziale; φ è definita per μ-ricorsione da ψ se

$$\varphi(x_1,\ldots,x_n,y)=\mu z.\,(\forall y<z.\,\psi(x_1,\ldots,x_n,y)\downarrow\ \wedge\ \psi(x_1,\ldots,x_n,z)=0)$$

La rimozione dell'ipotesi di totalità nella Definizione 12.9 provocherebbe un chiaro non senso. Infatti, per determinare il minimo z tale che $f(x_1,\ldots,x_n,z)=0$ si ricorre al seguente algoritmo (facilmente descrivibile con una MdT) calcolando la sequenza

$$f(x_1,\ldots,x_n,0), f(x_1,\ldots,x_n,1),\ldots$$

fino a che non si trova il primo valore z tale che $f(x_1,\ldots,x_n,z)=0$. Se f fosse parziale, ovvero il suo calcolo potesse non convergere, la procedura così descritta non sarebbe effettiva. Inoltre, come dimostrato

da Kleene nel '52, le funzioni parziali ricorsive non sono chiuse rispetto allo schema[2]

$$\varphi(x_1, \ldots, x_n, y) = \mu z.\ (\psi(x_1, \ldots, x_n, z) = 0).$$

DEFINIZIONE 12.11. La classe delle funzioni *parziali ricorsive* è la minima classe $\mathcal{PR}$ di funzioni contenente $\mathcal{P}$ e chiusa per μ-ricorsione.

ESEMPIO 12.12. Si osservi come si possa utilizzare il μ-operatore per la definizione delle seguenti funzioni:

- logaritmo (intero):
  $$\lfloor \log_a x \rfloor = \mu y.\ (leq(x, a^{(y+1)}) = 0),$$
  ove leq è una funzione ricorsiva primitiva tale che $leq(x, y) = 0$ se e solo se $x < y$;
- radice n-esima (intera):
  $$\lfloor \sqrt[n]{x} \rfloor = \mu y.\ (leq(x, (y+1)^n) = 0);$$

Il seguente risultato dimostra l'equivalenza delle funzioni calcolabili da MdT e le funzioni parziali ricorsive di Kleene & Robinson.

TEOREMA 12.13. *$f : \mathbb{N} \longrightarrow \mathbb{N}$ è parziale ricorsiva se e solo se è Turing-calcolabile.*

PROOF. La dimostrazione che, se φ è parziale ricorsiva, allora è Turing-calcolabile viene lasciata al lettore (suggerimento: si dimostri per induzione sulla struttura delle funzioni—per la base ed un caso induttivo si vedano l'esempio 11.11 e l'esercizio 11.12). Mostriamo invece l'implicazione inversa e cioé che se φ è Turing-calcolabile allora φ è parziale ricorsiva.

Sia $\varphi : \mathbb{N} \longrightarrow \mathbb{N}$ una funzione calcolabile da una MdT Z. L'obiettivo è quello di codificare la MdT Z come una funzione sui naturali che sia parziale ricorsiva. Per semplicità assumiamo, senza incorrere in limitazioni, che la MdT Z sia definita con 2 simboli, che corrispondono ai

[2]Tale risultato si può mostrare facilmente ma utilizzando risultati e nozioni che vedremo in seguito nel Cap. 20. In ogni modo, se A è un insieme ricorsivamente enumerabile ma non ricorsivo, allora definendo:

$$\psi(x, y) = 0 \ \Leftrightarrow\ (y = 0 \wedge x \in A) \vee y = 1$$

si ha che ψ è parziale ricorsiva; ma definendo $f(x) = \mu y.\ (\psi(x, y) = 0)$, abbiamo che f nonpuò essere parziale ricorsiva altrimenti, se lo fosse, essendo chiaramente totale, sarebbe ricorsiva e poichè $f(x) = 0 \ \Leftrightarrow\ x \in A$ anche A sarebbe ricorsivo.

numeri 0 e 1 rispettivamente:

$$\begin{aligned} \Sigma &= \{0,1\} \\ Q &= \{q_0, \ldots, q_k\} \end{aligned}$$

Con queste ipotesi possiamo rappresentare una generica ID come una tupla di numeri, associando un numero nell'intervallo $[0,k]$ a ogni simbolo di stato in Q. Definiamo $ar : \mathrm{ID} \longrightarrow \mathbb{N}^4$ tale che, per ogni $\alpha \in \mathrm{ID}$ tale che se

$$\alpha = \cdots\ b_2\ b_1\ b_0\ s\ q_h\ c_0\ c_1\ c_2\ \cdots$$

con $s \in \{0,1\}$, $h \in [0,k]$ e $\{b_i, c_i\}_{i\in\mathbb{N}} \subseteq \Sigma$, allora: $ar(\alpha) = (h,s,m,n)$ dove

$$\begin{aligned} m &= \textstyle\sum_{i=0}^{\infty} b_i 2^i = \sum_{i=0}^{h_m} b_i 2^i \\ n &= \textstyle\sum_{i=0}^{\infty} c_i 2^i = \sum_{i=0}^{h_n} c_i 2^i \end{aligned}$$

ove h_m (h_n) è h_n è il massimo intero i per cui $b_i \neq 0$ (rispettivamente $c_i \neq 0$). In questo modo, poichè il numero di 1 sul nastro non può essere infinito (siamo partiti con un numero finito di 1 ed abbiamo effettuato un numero finito di passi), per ogni ID in una computazione di Z, segue che ar associa in modo univoco ad ogni ID una quadrupla di naturali (aritmetizzazione di ID).

Una computazione di una MdT è definita come una sequenza (anche infinita) di ID: $\alpha_1 \vdash \alpha_2 \vdash \ldots$. Definiamo dunque una funzione che esprime un passo di transizione tra ID, ovvero una funzione che manipola quadruple di naturali. A tal proposito, definiamo le seguenti funzioni: $\Delta : Q \times \Sigma \longrightarrow Q$, $\nabla : Q \times \Sigma \longrightarrow \Sigma$ e $\mathcal{X} : Q \times \Sigma \longrightarrow \{0,1\}$ tali che:

- $\Delta(q,s)$ è lo stato che la MdT Z assume trovandosi nello stato q con simbolo in lettura s;
- $\nabla(q,s)$ è il simbolo che la MdT Z produce trovandosi nello stato q con simbolo in lettura s;
- $\mathcal{X}(q,s)$ è lo spostamento a destra ($=1$) o sinistra ($=0$) compiuto dalla MdT Z trovandosi nello stato q con simbolo in lettura s.

Si noti che Δ, ∇ e $\mathcal{X}$ sono funzioni definite dalla matrice funzionale di Z, e sono chiaramente funzioni primitive ricorsive (lo si dimostri

per esercizio). Possiamo quindi definire le trasformazioni compiute eseguendo un singolo passo della MdT Z sulle quadruple rappresentanti una generica ID di Z:

$$\begin{aligned} q' &= \Delta(q,s) \\ s' &= (m \ mod \ 2)(1-\mathcal{X}(q,s)) + (n \ mod \ 2)\mathcal{X}(q,s) \\ m' &= (2m+\nabla(q,s))\mathcal{X}(q,s) + (m \ div \ 2)(1-\mathcal{X}(q,s)) \\ n' &= (2n+\nabla(q,s))(1-\mathcal{X}(q,s)) + (n \ div \ 2)\mathcal{X}(q,s). \end{aligned}$$

Queste funzioni sono chiaramente primitive ricorsive, essendo la composizione di funzioni primitive ricorsive. Possiamo dunque rappresentare una transizione $\alpha \vdash \beta$ nel modo seguente: Se $ar(\alpha) = (q,s,m,n)$ allora $ar(\beta) = (q',s',m',n')$.

Per rappresentare l'effetto di t-transizioni o passi di calcolo della MdT Z, definiamo le seguenti funzioni di 5 argomenti:

- $P_\Delta(t,q,s,m,n)$ è lo stato ($\in [0,k]$) ottenuto dopo t-passi partendo da una ID α tale che $ar(\alpha) = (q,s,m,n)$;
- $P_\nabla(t,q,s,m,n)$ è il simbolo ($\in \{0,1\}$) ottenuto dopo t-passi partendo da una ID α tale che $ar(\alpha) = (q,s,m,n)$;
- $P_M(t,q,s,m,n)$ è il valore ($\in \mathbb{N}$) rappresentante il nastro a sinistra della testina ottenuto dopo t-passi partendo da una ID α tale che $ar(\alpha) = (q,s,m,n)$;
- $P_N(t,q,s,m,n)$ è il valore ($\in \mathbb{N}$) rappresentante il nastro a destra della testina ottenuto dopo t-passi partendo da una ID α tale che $ar(\alpha) = (q,s,m,n)$.

È facile vedere che, ad esempio, P_Δ può essere definita nel modo seguente:

$$\begin{aligned} P_\Delta(0,q,s,m,n) &= q \\ P_\Delta(t+1,q,s,m,n) &= P_\Delta(t,q',s',m',n') \end{aligned}$$

Supponiamo che la MdT Z abbia uno stato $q_1 \in Q$ di terminazione, ovvero tale che le uniche caselle vuote della matrice funzionale di z siano in corrispondenza della riga q_1. È chiaro che ogni MdT può essere trasformata in modo tale da avere uno stato di terminazione di questo tipo. Pertanto, la MdT Z termina il suo calcolo nel primo (minimo) t

tale che: $P_\Delta(t, q_0, s_0, m_0, n_0) = 1$, essendo α_0 la configurazione iniziale tale che $ar(\alpha_0) = (q_0, s_0, m_0, n_0)$. Ovvero, il numero di passi necessario per terminare, è definibile mediante μ-ricorsione come:

$$\mu t.\ (P_\Delta(t, q_0, s_0, m_0, n_0) - 1 = 0).$$

Sia $x \in \mathbb{N}$ un numero naturale di input. Assumiamo che la MdT inizi il suo calcolo avendo il numero x sul nastro alla destra della sua testina codificato come una sequenza di x-zeri seguita da 1 (rappresentata matematicamente dal numero 2^x). Assumiamo anche che il risultato, se calcolato, sia rappresentato nel medesimo modo sul nastro della macchina. È dunque chiaro che, avendo un modo algoritmico primitivo ricorsivo dato da una funzione C per rappresentare come potenza di 2 il numero alla destra della testina nella ID terminale, la funzione φ calcolata da Z è esprimibile dalla seguente funzione parziale ricorsiva:

$$\varphi = \lambda x.\ C(P_N(\mu t.\ (P_\Delta(t, 0, 0, 0, 2^x) - 1 = 0), 0, 0, 0, 2^x)).$$

Per esercizio si trovi la funzione C che chiude la dimostrazione e si dimostri che è primitiva ricorsiva. □

COROLLARIO 12.14 (Forma normale di Kleene). Per ogni funzione $\varphi \in \mathcal{PR}$ (o equivalentemente Turing calcolabile) esistono $f, g \in \mathcal{P}$ tali che

$$\varphi = \lambda x.\ f((\mu t.\ g(t, x) = 0), x)\,.$$

CHAPTER 13

Tesi di Church

Negli stessi anni in cui venivano proposti i vari formalismi per definire la effettiva calcolabilità, ad esempio le MdT, le funzioni ricorsive, il λ-calcolo etc., veniva dimostrata anche la loro l'equivalenza. Si dimostrava cioè che i diversi sistemi permettono di calcolare esattamente la stessa classe di funzioni. Abbiamo visto degli esempi nelle sezioni precedenti riguardanti le MdT e le funzioni parziali ricorsive di Kleene & Robinson.

Questi risultati (in particolare l'equivalenza del λ-calcolo con le funzioni parziali ricorsive) portarono Church e Turing nel '36 a formulare la seguente Tesi fondamentale:

> **Tesi di Church:** *La classe delle funzioni "intuitivamente calcolabili" coincide con la classe delle funzioni Turing calcolabili*

Si tratta chiaramente di una tesi indimostrabile, vista l'impossibilità di maneggiare formalmente (= matematicamente) la nozione di "intuitivamente calcolabile". Accettando la Tesi di Church si osserva immediatamente la notevole importanza che assumono le funzioni calcolabili da MdT, o equivalentemente, le funzioni parziali ricorsive o λ-definibili. Queste classi di funzioni vengono a rappresentare, in virtù della Tesi di Church, la classe delle funzioni calcolabili effettivamente mediante un algoritmo. La Tesi di Church ha pertanto notevole interesse metamatematico. Lo stesso Gödel scrisse nel '46 le seguneti osservazioni riguardanti la Tesi di Church, in riferimento ad una presentazione di Tarski sul medesimo argomento:

> *Tarski has stressed in his lecture (and I think justly) the great importance of the concept of general recursiveness*[1]

[1] ... che noi chiamiamo ricorsività *a la* Kleene & Robinson.

> *(or Turing's computability). It seems to me that this importance is largely due to the fact that with this concept one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, i.e. one not depending on the formalism chosen. [...] By a kind of miracle it is not necessary to distinguish orders and the diagonal procedure does not lead outside the defined notion.* [K. Gödel, Princeton 1946]

La Tesi di Church permette di limitare l'estensione di ciò che è effettivamente calcolabile: se dimostriamo che una funzione non è parziale ricorsiva, allora, in virtù della Tesi di Church, essa non è calcolabile in nessun modo effettivo. Analogamente, dare un algoritmo per una funzione corrisponde a dimostrare, per la Tesi di Church, che essa è una funzione parziale ricorsiva. Questo utilizzo della Tesi di Church è fondamentale in questo corso. Infatti, sarà sufficiente dimostrare la realizzabilità di un algoritmo per affermare l'esistenza di una MdT o di una funzione parziale ricorsiva corrispondente, senza dover esibire una tale macchina o una funzione per calcolarlo. Questo ci permetterà di sviluppare un certo numero di strumenti a supporto della teoria delle calcolabilità, senza dover esibire esplicitamente complesse MdT o funzioni parziali ricorsive. Questa caratteristica della Tesi di Church darà un aspetto apparentemente informale alle dimostrazioni di effettiva calcolabilità di funzioni. Dimostrazioni di questo tipo, che si basano su una non banale applicazione della Tesi di Church, saranno dette *dimostrazioni mediante Tesi di Church*. Pertanto, ogni volta che una funzione risulterà "palesemente" calcolabile, la si accetterà come effettivamente calcolabile da uno dei formalismi prescelti, senza dover costruire la MdT o altra funzione corrispondente. Nel seguito dunque considereremo equivalenti i concetti di calcolabilità: "effettiva" (ovvero associata all'esistenza di una MdT), "algoritmica" (ovvero associata all'esistenza di un algoritmo che soddisfi i requisiti **a–l**) o "ricorsiva" (ovvero associata ad una funzione ricorsiva parziale che la calcola).

A questo punto della trattazione, è bene distinguere, per chiarezza, tra due nozioni diverse di calcolabilità. Infatti è possibile definire una funzione g per cui sappiamo esistere un algoritmo ma per la quale non

sappiamo dare l'algoritmo che la calcola. Consideriamo le funzioni definite nel modo seguente:

$$f(x) = \begin{cases} 1 & \text{se esattamente } x \text{ '5' consecutivi appaiono nella} \\ & \text{espansione decimale di } \pi \\ 0 & \text{altrimenti} \end{cases}$$

$$g(x) = \begin{cases} 1 & \text{se almeno } x \text{ '5' consecutivi appaiono nella espansione} \\ & \text{decimale di } \pi \\ 0 & \text{altrimenti} \end{cases}$$

È chiaro che g è calcolabile: dato un algoritmo che genera l'espansione decimale di π, g sarà o la funzione costante $g(x) = 1$, nel caso vi sia un numero arbitrariamente grande di occorrenze successive di '5' in π, che è chiaramente una funzione calcolabile; oppure sarà la funzione calcolata da un qualche algoritmo che, fissato un k (ed esempio il massimo numero di '5' consecutivi in π), calcola:

$$g(x) = \begin{cases} 1 & \text{se } x \leq k \\ 0 & \text{se } x > k \end{cases}$$

In questo secondo caso esiste sicuramente un tale k che permette di individuare l'algoritmo cercato, ma ci è impossibile sapere quale sia il k giusto. In entrambi i casi la funzione g è calcolabile (ed é in particolare primitiva ricorsiva), ma in generale non sappiamo costruire effettivamente un algoritmo per calcolarla. Al contrario, per f non siamo in grado a tutt'oggi di affermare nulla sulla sua calcolabilità effettiva. Pertanto, mentre per g è possibile invocare la Tesi di Church, ed affermare che esiste una MdT che la calcola, anche se non si sa quale, per f questo non è possibile. Accettando quindi di considerare calcolabile ogni funzione per la quale esiste un algoritmo che la calcoli, possiamo nel seguito utilizzare la Tesi di Church ogni qual volta sia chiara l'esistenza di un algoritmo per calcolare una data funzione.

CHAPTER 14

Aritmetizzazione e universalità

Aritmetizzazione significa semplicemente traduzione nel linguaggio dell'aritmetica. Il primo utilizzo dell'aritmetizzazione è dovuto a Gödel nel 1931 nella dimostrazione del risultato fondamentale di incompletezza dell'aritmetica; da qui anche il termine equivalente di Gödelizzazione. Nel seguito considereremo le MdT come esempio ed applicheremo alle MdT il concetto di aritmetizzazione. Per la Tesi di Church, sappiamo che questa non è una restrizione, ed analoghe aritmetizzazioni possono essere definite per ogni sistema formale equivalente alle MdT.

1. Enumerazione delle MdT

Sia $\Sigma = \{\$, 0\}$. Quante sono le macchine di Turing 'distinte' descrivibili? Si supponga $Q = \{q_0\}$. Si tratterà di riempire la tabella

	$\$$	0
q_0		

in tutti i modi (significativi) possibili, ossia ogni singola casella potrà essere

1. *vuota* (si immagini di scrivere $\$\ \$\ \$$);
2. $q_0\ \$\ R$;
3. $q_0\ \$\ L$;
4. $q_0\ 0\ R$;
5. $q_o\ 0\ L$.

Dunque ci sono esattamente $25 = 5 \times 5$ macchine di Turing distinte ad un solo stato.

Si può pensare a questo punto di definire un ordinamento $\lhd$ sulle macchine di Turing ad uno stato definibili; fissato un ordinamento

su Σ (poniamo $\$ <_\Sigma 0$), sia inoltre $L <_M R$ e si consideri la cella a sinistra più importante di quella a destra, essendo M insieme dei possibili movimenti della testina, cioé $M = \{L, R\}$.

Si assuma che $\$\,\$\,\$$ sia minore di qualunque terna $q_0\, s\, m$, allora si avrà (in una estensione lessicografica i cui parametri sono, nell'ordine *cella* e *contenuto*):

$$\langle \$\$\$, \$\$\$\rangle \lhd \langle \$\$\$, q_0\$L\rangle \lhd \cdots \lhd \langle \$\$\$, q_0 0R\rangle \lhd \langle q_0\$L, \$\$\$\rangle \lhd \cdots \lhd \langle q_0 0R, q_0 0R\rangle.$$

Più in generale, sia $Q = \{q_0, \ldots, q_{n-1}\}$, $\Sigma = \{s_0 = \$, s_1 = 0\}$, $m_0 = L$,$m_1 = R$; si definisca inoltre un ordinamento tra le $2 \cdot n$ celle presenti (per righe, colonne, a zig zag etc.). Definendo esplicitamente l'ordinamento per ogni cella nel modo seguente:

- $\$\,\$\,\$ \prec q_i\, s\, m$ per ogni $q_i \in Q$, $s \in \Sigma$, $m \in \{L, R\}$;
- $q_{i_1}\, s_{j_1}\, m_{k_1} \prec q_{i_2}\, s_{j_2}\, m_{k_2}$ se $i_1 < i_2$ oppure ($i_1 = i_2$ e $j_1 < j_2$) oppure ($i_1 = i_2$ e $j_1 = j_2$ e $k_1 < k_2$).

si ha un buon ordine sull'insieme delle macchine di Turing aventi n stati. Ovviamente esse sono in tutto

$$(n \cdot 2 \cdot 2 + 1)^{2n} = (4 \cdot n + 1)^{2n},$$

dunque ci saranno 25 MdT ad uno stato, 9^4 MdT a due stati, 13^6 MdT a tre stati, ... Si può dunque pensare di associare un numero naturale (un *indice*) ad ogni macchina di Turing in maniera biunivoca:

$1 \div 25$	MdT ad uno stato
$26 \div 25 + 9^4$	MdT a due stati
$25 + 9^4 + 1 \div 25 + 9^4 + 13^6$	MdT a tre stati
...	...

all'interno di ogni gruppo, poi ci si basa sulla relazione di ordinamento. In generale, con un semplice algoritmo, dato $x \in \mathbb{N}$ si può risalire alla MdT x-esima.

ESEMPIO 14.1. Assumendo un ordinamento delle celle per cui la più importante è quella in alto a sinistra, la macchina di Turing n.4399 ha due stati ($Q = \{q_0, q_1\}$), e la sua funzione di transizione è la

seguente:

δ	\$	0
q_0	q_1 \$ R	
q_1		

Si osservi che tale MdT computa la funzione $\lambda x.0$.

Una differente aritmetizzazione delle MdT si basa sulle proprietà della decomposizione in fattori primi dei numeri naturali. Analogamente al caso precedente, è possibile, in modo algoritmico, passare da una MdT ad un numero naturale che la aritmetizza, e viceversa è possibile determinare se un numero è o no l'artimetizzazione di una MdT. Consideriamo ancora una generica MdT definita su un alfabeto finito:

$$\begin{aligned} \Sigma &= \{s_0, \ldots, s_n\} \\ Q &= \{q_0, \ldots, q_m\} \\ X &= \{L, R\} \end{aligned}$$

ed associamo, ad ogni simbolo $x \in \Sigma \cup Q \cup X$ un numero dispari maggiore di 1: $a(x) = 2k + 1$ per qualche $k \geq 1$, tale che se $x \neq y$ allora $a(x) \neq a(y)$. Ad esempio:

$$\begin{array}{cccccccc} R & L & s_0 & q_1 & s_1 & q_2 & \ldots \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \ldots \\ 3 & 5 & 7 & 9 & 11 & 13 & \ldots \end{array}$$

Definiamo *numero di Gödel* di una espressione $E \in (\Sigma \cup Q \cup X)^*$ il numero:

$$gn(E) = \prod_{i=1}^{k} p_i^{a(x_i)}$$

essendo k il numero dei simboli in E, p_i l'i-esimo numero primo e x_i l'i-esimo simbolo di E. Per l'unicità della decomposizione in fattori primi, per ogni espressione esiste uno ed un solo numero di Gödel corrispondente; inoltre tale numero è pari nell'ipotesi E sia una stringa di almeno un simbolo.

Poichè le quintuple di una MdT sono a tutti gli effetti espressioni in $(\Sigma \cup Q \cup X)^*$, è possibile in modo algoritmico associare ad ogni quintupla un corrispondente numero di Gödel, e viceversa verificare se un dato numero è numero di Gödel di una quintupla legittima.

Definiamo numero di Gödel di una MdT Z, ovvero di una sequenza di quintuple $Z = \{E_1, \ldots, E_h\}$ il numero:

$$\prod_{i=1}^{h} p_i^{gn(E_i)}$$

Questo numero è anch'esso unico, per l'unicità della decomosizione in fattori primi, e non risulta confondibile con $gn(E)$ per una qualche espressione $E \in (\Sigma \cup Q \cup X)^*$, poichè, al contrario di gn, è ottenuto mediante esponenziazione pari. Analogamente, due sequenze con lo stesso numero di Gödel sono identiche (verificare per esercizio).

In generale è possibile definire infinite aritmetizzazioni diverse per MdT, e quindi funzioni parziali ricorsive. L'artimetizzazione scelta non è quindi sostanziale nello sviluppo della teoria della calcolabilità. Essenziale è che, come dimostrato in precedenza, esistano metodi algoritmici per codificare MdT all'interno dei numeri naturali (suoi argomenti di calcolo). La possibilità di vedere i numeri come al tempo stesso rappresentanti gli argomenti del calcolo e le MdT che eseguono il calcolo, permette di applicare ai sistemi formali analizzati fino ad ora, uno dei principi alla base dell'informatica moderna, ovvero quello di poter passare programmi come argomenti ad altri programmi. In tutto ciò la scelta della particolare aritmetizzazione è inessenziale.[1]

Nel seguito dunque indicheremo con P_x l'insieme di istruzioni di una MdT di indice x, ed x il corrispondente *indice* o *numero di Gödel* della macchina, in una data aritmetizzazione. Il seguente risultato è banale in seguito ad una qualsiasi aritmetizzazione delle MdT.

[1] La libertà di poter applicare MdT a MdT (autoapplicazione o referenziazione) condurrebbe immediatamente a paradossi tipo *paradosso di Russell* nella teoria che stiamo sviluppando. Ancora una volta, l'assunzione fondamentale che le funzioni calcolabili siano essenzialmente parziali, permette di superare tali paradossi, rendendo la teoria della calcolabilità una teoria consistente. Si veda in proposito la Nota 15.4.

TEOREMA 14.2. *Ci sono $\aleph_0$ macchine di Turing distinte o funzioni parziali ricorsive, e $\aleph_0$ funzioni ricorsive.*

PROOF. Segue banalmente dal fatto che tutte le funzioni costanti sono ricorsive. Pertanto ci sono almeno $\aleph_0$ funzioni ricorsive. □

Segue dunque per la Tesi di Church che:

TEOREMA 14.3. *Ci sono esattamente $\aleph_0$ funzioni calcolabili.*

Indicheremo nel seguito con φ_x la funzione parziale ricorsiva di indice o numero di Gödel x. Chiaramente sappiamo che esiste un algoritmo per passare da x a P_x e/o φ_x e viceversa. Pertanto, con φ_x indicheremo indifferentemente l'x-esima MdT o funzione parziale ricorsiva in una data aritmetizzazione.

Il seguente risultato dimostra che la cardinalità delle funzioni calcolabili è strettamente minore della cardinalità di tutte le possibili funzioni sui naturali. Si tratta di una rivisitazione del Teorema di Cantor, nel contesto delle funzioni calcolabili. Ne consegue che l'insieme delle funzioni calcolabili è strettamente contenuto (ne rappresenta una piccolissima parte) nell'insieme di tutte le funzioni, ovvero nell'insieme di tutti i possibili *problemi* esprimibili mediante funzioni sui naturali. In particolare, potendo selezionare una funzione in $\mathbb{N} \longrightarrow \mathbb{N}$, è facile dimostrare che la probabilità di selezionarne una calcolabile è 0.

TEOREMA 14.4. *Esistono funzioni $f : \mathbb{N} \longrightarrow \mathbb{N}$ non Turing calcolabili.*

PROOF. Sappiamo che, dal Teorema di Cantor (vedi Cap. 2) che:

$$\begin{aligned} |\{f : \mathbb{N} \longrightarrow \mathbb{N}\}| &\geq |\{f : \mathbb{N} \longrightarrow \{0,1\}\}| \\ &= |\wp(\mathbb{N})| \\ &> |\mathbb{N}| \\ &\geq |\{\, f : \mathbb{N} \longrightarrow \mathbb{N} \mid f \text{ è Turing calcolabile}\}| \end{aligned}$$

□

Rispetto a quanto visto nelle sezioni precedenti, abbiamo pertanto dimostrato le inclusioni tra classi di funzioni come evidenziate nella Figura 1.

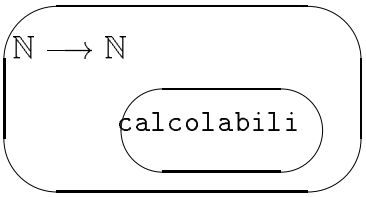


FIGURE 1. Diagramma di inclusione tra insiemi di funzioni

2. Macchina di Turing Universale

Per quanto si è illustrato nella sezione 1, dato un numero naturale x si può facilmente risalire ad una tabella definente la funzione di transizione della x-esima macchina di Turing. D'altro canto ogni singola macchina di Turing è un oggetto provvisto di un compito fisso. Una volta definita la funzione di transizione non vi è modo di modificarla.

Il passo immediatamente successivo (che fornisce la base teorica al concetto di calcolatore programmabile) è il seguente: si supponga di disporre della macchina di Turing con un unico input che, in base alle considerazioni precedenti, possiamo vedere come una coppia $\langle x_1, x_2 \rangle$, con il seguente comportamento:

- x_1 è l'indice di una macchina di Turing P_{x_1};
- x_2 è la rappresentazione di un input che va immaginato come input per la macchina P_{x_1}.

La macchina in questione dapprima calcola la funzione di transizione della macchina P_{x_1}, memorizzando questa funzione. Suppone poi (cioè lo memorizza opportunamente) di essere nello stato q_0. Si posiziona dunque in corrispondenza del primo carattere di input (chiamiamolo s) e va a vedere—sulla tabella memorizzata—cosa farebbe P_{x_1}, ossia computa $\delta_{P_{x_1}}(q_0, s)$. Procede dunque nella simulazione andando a modificare il carattere sulla zona di nastro dedicata all'input, lo stato nella opportuna zona in cui va memorizzato, e sposta idealmente la testina, in base al risultato di $\delta_{P_{x_1}}(q_0, s)$.

Una tale MdT si riesce ovviamente a costruire (anche se non può essere considerato un facile esercizio) e dunque avrà un suo indice nella

enumerazione delle MdT; sia esso u. Si avrà pertanto:

$$\forall x_1\, x_2.\ P_u(\langle x_1, x_2\rangle) = P_{x_1}(x_2)$$

ove il simbolo = sta a significare che, se una macchina termina, allora terminano entrambe fornendo lo stesso risultato, altrimenti entrambe ciclano. Come interessante ed immediata conseguenza avremo, ad esempio, $P_u(\langle u, u\rangle) = P_u(u)$.

Più in generale possiamo dimostrare il seguente risultato, che rappresenta una prima dimostrazione mediante Tesi di church.

TEOREMA 14.5. *Esiste un indice z tale che per ogni x e y,*

$$\varphi_z(x,y) = \begin{cases} \varphi_x(y) & \textit{se } \varphi_x(y) \textit{ è definita} \\ \uparrow & \textit{altrimenti} \end{cases}$$

PROOF. Sia P_x l'insieme di istruzioni di una MdT di indice x. Sappiamo che esiste un metodo effettivo per ottenere P_x da x. Applichiamo P_x all'input y e, quando questa termina, prendiamo il risultato come output di una funzione di 2 argomenti $\psi(x,y)$. Pertanto abbiamo che:

$$\psi(x,y) = \begin{cases} \varphi_x(y) & \text{se } \varphi_x(y) \text{ converge} \\ \uparrow & \text{se } \varphi_x(y) \text{ diverge} \end{cases}$$

Applicando la Tesi di Church, possiamo concludere che ψ è parziale ricorsiva e pertanto esiste un indice z tale che $\psi = \varphi_z$. □

La funzione φ_z ottenuta nella dimostrazione del precedente teorema è detta *funzione parziale universale* e corrisponde alla MdT universale vista in precedenza. Chiaramente, il precedente teorema può essere generalizzato a funzioni di $k \geq 1$ variabili, in modo tale che, per ogni funzione di $k \geq 1$ variabili, esiste una funzione di $k+1$ variabili che gioca il ruolo di funzione universale parziale.

ESERCIZIO 14.6. Formulare e dimostrare il precedente teorema per funzioni di $k \geq 1$ variabili.

Il seguente risultato, noto con il nome *Teorema s-m-n* e dovuto a Kleene, sarà fondamentale nel seguito. Anch'esso rappresnta un esempio di dimostrazione basata sulla Tesi di Church.

TEOREMA 14.7 (s-m-n). *Per ogni $m, n \geq 1$ esiste una funzione ricorsiva s_n^m di $m+1$ variabili tale che per ogni $x, y_1, \ldots, y_m$ abbiamo che:*

$$\lambda z_1 \cdots z_n.\ \varphi_x(y_1, \ldots, y_m, z_1, \ldots, z_n) = \varphi_{s_n^m(x, y_1, \ldots, y_m)}$$

PROOF. Consideriamo per semplicità il caso $m = n = 1$. La dimostrazione per il caso più generale per $m, n \geq 1$ è simile. Fissati x_0 e y_0, la funzione $\lambda z.\ \varphi_{x_0}(y_0, z)$ è chiaramente una funzione calcolabile in una sola variabile. Per ogni x_0 e y_0, la funzione $\lambda z.\ \varphi_{x_0}(y_0, z)$ dipenderà costruttivamente da x_0 e y_0, pertanto esisterà per la Tesi di Church, una funzione ricorsiva (totale!) s tale che: $\lambda z.\ \varphi_{x_0}(y_0, z) = \varphi_{s(x_0, y_0)}$. □

Il Teorema s-m-n gioca un ruolo essenziale nello sviluppo della teoria della ricorsività, e spesso verrà utilizzato, come la Tesi di Church, tacitamente. Il seguente risultato è una semplice applicazione del Teorema s-m-n.

TEOREMA 14.8. *Esiste una funzione ricorsiva g in due argomenti tale che per ogni x e y: $\varphi_{g(x,y)} = \varphi_x \circ \varphi_y$.*

PROOF. Esercizio. □

CHAPTER 15

Problemi insolubili

In questo Capitolo analizzeremo alcuni problemi classici non risolvibili nella teoria della effettiva calcolabilità sviluppata fino ad ora. Il più classico problema insolubile è il *problema della terminazione* posto da Turing.

> *Esiste una procedura effettiva tale che dati x e y determina se $\varphi_x(y)$ è definita o no?*

Il seguente risultato preliminare dimostra l'insolubilità di un problema correlato.[1]

LEMMA 15.1. *Non esiste una funzione ricorsiva g tale che per ogni x:*

$$g(x) = \begin{cases} 1 & se\ \varphi_x(x) \downarrow \\ 0 & se\ \varphi_x(x) \uparrow \end{cases}$$

PROOF. Supponiamo per assurdo che esista una MdT di indice i_0 tale che $g = \varphi_{i_0}$. Consideriamo la funzione

$$g'(x) = \begin{cases} \uparrow & \text{se } \varphi_x(x) \downarrow \\ 0 & \text{se } \varphi_x(x) \uparrow \end{cases}$$

Per definizione abbiamo che:

$$g'(x) = \begin{cases} \uparrow & \text{se } g(x) = 1 = \varphi_{i_0}(x) \\ 0 & \text{se } g(x) = 0 = \varphi_{i_0}(x) \end{cases}$$

[1] Si ricorda che con $\varphi_x(y) \downarrow$ si intende che la macchina di Turing x-esima sull'input y termina o equivalentemente, che la funzione ricorsiva parziale x-esima è definita per l'input x. Con $\varphi_x(y) \uparrow$ si intende l'opposto di tale affermazione.

Allora esiste i_1 tale che $\varphi_{i_1} = g'$. Ma allora abbiamo che:

$$\varphi_{i_1}(i_1) \uparrow \Leftrightarrow \varphi_{i_1}(i_1) \downarrow$$

Che è assurdo. Dunque non esiste un tale indice i_1, ovvero non esiste l'indice i_0 che calcoli g, ovvero g non è ricorsiva. □

In base al Lemma 15.1, segue il teorema:

TEOREMA 15.2. *Non esiste una funzione ricorsiva ψ tale che per ogni x e y:*

$$\psi(x,y) = \begin{cases} 1 & se\ \varphi_x(y) \downarrow \\ 0 & se\ \varphi_x(y) \uparrow \end{cases}$$

PROOF. Se esistesse allora esisterebbe un indice x_0 tale che $\varphi_{x_0}(x) = \psi(x,x)$ che è assurdo per il Lemma 15.1 □

Il problema della terminazione è stato, storicamente, uno dei primi problemi matematici dimostrati come insolubili, ovvero per i quali non esiste una procedura in grado di "decidere" il problema per ogni possibile input, e rappresenta, per la sua semplicità ed importanza pratica e teorica, uno dei risultati più importanti ottenuti in matematica nel 20-esimo secolo.

Il segunete risultato, dovuto a Kleene, dimostra la non solvibilità di un altro semplice problema correlato alle funzioni ricorsive, ovvero non è possibile decidere se una funzione parziale ricorsiva data è totale. Il problema è chiaramente legato alla non decidibilità della terminazione.

TEOREMA 15.3. *Non esiste una funzione ricorsiva f tale che per ogni x:*

$$f(x) = \begin{cases} 1 & se\ \varphi_x\ è\ totale \\ 0 & se\ \varphi_x\ non\ è\ totale \end{cases}$$

PROOF. La dimostrazione utilizza un argomento diagonale. Supponiamo che esista una tale funzione f. Allora definiamo una funzione che associ ad ogni naturale una corrispondente funzione ricorsiva.

$$\begin{aligned} g(0) &= \mu y.\,(f(y) = 1) \\ g(x+1) &= \mu y\ (y > g(x) \wedge f(y) = 1). \end{aligned}$$

Poichè sappiamo che esistono ω funzioni ricorsive totali, anche g sarà totale. Quindi è ricorsiva. Definiamo una funzione h tale che $h = \lambda x.\ \varphi_{g(x)}(x) + 1$. Per la definizione di f segue che h è totale, e quindi per la Tesi di Church ricorsiva. Sia dunque $h = \varphi_{z_0}$. Sia y_0 tale che $g(y_0) = z_0$. Questo indice esiste per definizione di g. Allora $h(y_0) = \varphi_{g(y_0)}(y_0) + 1$. Ma $\varphi_{g(y_0)}(y_0) = h(y_0)$ per definizione di y_0. Poichè h è totale, questa è una contraddizione. □

NOTA 15.4. Argomenti diagonali e insolubilità sono nozioni strettamente legate tra loro. Ritorniamo brevemente sulla questione delle funzioni parziali. Abbiamo visto che ricorrendo alle funzioni parziali si possono superare i limiti imposti da argomenti di tipo diagonale ai sistemi formali introdotti per definire il concetto di effettiva calcolabilità. La necessità di utilizzare funzioni parziali in luogo di funzioni totali segue anche dalla possibilità, dimostrata in precedenza, di definire macchine universali o funzioni ricorsive universali. Questa possibilià, associata alla ipotesi di totalità delle funzioni calcolabili, conduce infatti direttamente a paradossi simili al paradosso di Russell.[2] L'aritmetizzazione infatti non impone nessuna distinzione tra funzioni (MdT) e argomenti, analogamente come nella teoria degli insiemi classica non vi è distinzione tra insiemi e loro argomenti. Questa mancanza di struttura permette una indisciplinata possibilità di definizioni di insiemi o funzioni, che conduce ai ben noti paradossi.

È noto che è possibile codificare coppie, terne etc. con numeri naturali. Ad esempio la seguente funzione $g : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ è noto essere iniettiva e chiaramente calcolabile:

$$g(x, y) = \frac{1}{2}(x^2 + 2xy + y^2 + 3x + y).$$

Essa permette di associare in modo univoco, ad ogni coppia di naturali, un numero naturale corrispondente. Quindi per la Tesi di Church esisterà un indice x_0 tale che, se $\psi(x, y)$ è la funzione parziale universale vista in precedenza, allora:

$$\psi(x, y) = \varphi_{x_0}(g(x, y))$$

[2] Il paradosso di Russell nella teoria classica degli insiemi è ottenuto considerando come definibile nella teoria stessa l'insieme $A = \{x \mid x \notin x\}$. Quindi $x \in A \Leftrightarrow x \notin x$, da cui segue che $x \in A \Leftrightarrow x \notin A$.

Inoltre, essendo sia g che φ_{x_0} calcolabili, esisterà un indice y_0 tale che $\varphi_{y_0}(x) = \varphi_{x_0}(g(x,x))$. Sarà quindi lecito definire la funzione:

$$f(x) = \begin{cases} 1 & \text{se } \varphi_{y_0}(x) = \varphi_x(x) = 0 \\ 0 & \text{se } \varphi_{y_0}(x) = \varphi_x(x) \neq 0 \end{cases}$$

Anche questa funzione avrà un opportuno indice z_0. Segue quindi che $\varphi_{z_0}(z_0) = 0$ se e solo se $\varphi_{z_0}(z_0) \neq 0$, che è chiaramente una contraddizione. Per superare questo paradosso, del tutto analogo al paradosso di Russell, analogamente a quanto visto per superare i limiti imposti dalla diagonalizzazione, basterà ancora assumere che la funzione calcolabile f, calcolata dalla z_0-esima MdT, diverga in z_0, essendo quindi parziale ricorsiva. La funzione

$$\psi(x,y) = \begin{cases} 1 & \text{se } \varphi_x(y) \downarrow \\ \uparrow & \text{se } \varphi_x(y) \uparrow \end{cases}$$

è dunque necessariamente parziale ricorsiva, ed infatti, l'algoritmo che la calcola è facilmente identificabile con la MdT universale vista in precedenza.

Esercizio 15.5. Dimostrare che i seguenti problemi non ammettono soluzione ricorsiva:

- Decidere se, per ogni x: φ_x è costante;
- Decidere se per ogni x, y e z: $\varphi_x(y) = z$;
- Decidere se per ogni x e y: $\varphi_x = \varphi_y$.

CHAPTER 16

Calcolabilità e Linguaggi di Programmazione

In questo capitolo vedremo alcune applicazioni tra le più importanti, della teoria elementare della calcolabilità vista nel precedente capitolo al progetto e implementazione dei moderni linguaggi di programmazione. In particolare metteremo in evidenza come l'espressività (in senso lato) di un linguaggio dipenda dalla possibilità offerta da quest'ultimo di codificare uno dei formalismi affrontati per definire il concetto di effettiva calcolabilità. Questi problemi hanno dato origine, negli anni '60, alla ricerca di linguaggi di programmazione ad alto livello in grado di esprimere tutte le funzioni effettivamente calcolabili. Questa ricerca ha dato origine a una vasta famiglia di linguaggi ome ALGOL, PASCAL, entrambi progenitori dei più moderni linguaggi C, C++, e Java. Inoltre vedremo come l'esistenza (almeno teorica) di strumenti ampiamente utilizzati nella moderna pratica informatica per manipolare programmi, come i compilatori, gli interpreti, e i programmi di specializzazione, dipenda dai risultati di universalità visti nella caratterizzazione delle funzioni calcolabili.

CHAPTER 17

Espressività nei linguaggi imperativi

+++ il concetto di espressività

1. Funzioni primitive ricorsive e programmi FOR

Introduciamo un semplice linguaggio denominato FOR, per scrivere programmi che manipolano numeri naturali. I costrutti di base del linguaggio sono: l'assegnamento, la composizione e l'iterazione determinata mediante il comando `for`.

+++ sintassi

+++ semantica intuitiva

Il seguente risultato caratterizza le funzioni esprimibili nel linguaggio FOR. In paricolare, segue da quanto visto nella Cap. 12 che il linguaggio FOR è inadeguato a rappresentare le funzioni intuitivamente calcolabili, così come le funzioni primitive ricorsive non rappresentano tutte le funzioni algoritmicamente calcolabili. Limitarsi pertanto a scrivere programmi in FOR implica limitare fortemente l'espressività dei nostri algoritmi.

TEOREMA 17.1 (Meyer & Ritchie). *Una funzione è* `for` *calcolabile sse è primitiva ricorsiva.*

PROOF. +++ Odifreddi 71 □

2. Funzioni parziali ricorsive e programmi WHILE

Come abbiamo visto, il linguaggio FOR è inadeguato, come le funzioni primitive ricorsive, a rappresentare la classe delle funzioni algoritmicamente calcolabili. Estendiamo dunque FOR con un costrutto più generale (ovvero che può simulare il comando `for`). Questo ci permette di definire il linguaggio WHILE.

+++ sintassi di WHILE

+++ semantica intuitiva e semantica formale

ESERCIZIO 17.2. Definite i costrutti di FOR in WHILE. È possibile simulare il comando `if _ then _ else` utilizzando solo il comando `while`?

TEOREMA 17.3. *Le funzioni* WHILE *calcolabili sono tutte e sole le funzioni parziali ricorsive.*

+++ considerazioni sui programmi strutturati in WHILE e i programmi goto (Boem-Jacopini).

CHAPTER 18

Il linguaggio WHILE+

Abbiamo fino ad ora definito il concetto di effettiva calcolabilità su una semplicissima struttura dati: i numeri naturali $\mathbb{N}$. Come vedremo, questa apparente restrizione in realtà si dimostra essere perfettamente equivalente ad ogni definizione di calcolabilità data su strutture dati più evolute, quali stringhe, alberi etc. Ovvero i numeri naturali sono sufficenti per rappresentare ogni possibile struttura dati su cui si vogliono definire algoritmi.

Al fine di evitare l'aritmetizzare dei programmi, in questo capitolo studieremo una estensione del linguaggio WHILE visto in precedenza per manipolare naturali, con strutture dati più complesse. WHILE+ è un linguaggio ad alto livello che, includendo WHILE, è adeguato per rappresentare tutte le funzioni calcolabili ed inoltre premette di manipolare semplici strutture dati. Le strutture dati impiegate in WHILE+ sono molto simili a quelle impiegate in SCHEME e LISP. Questa estensione ci permetterà di evitare l'aritmetizzazione (operazione di scarso interesse informatico e preminente interesse matematico) permettendo quindi di considerare direttamente i programmi come dati. Questo, come abbiamo visto è il passo fondamentale per definire il concetto di funzione universale.

1. Sintassi

+++ alberi su insiemi finiti (2.1.1)
+++ sintassi BNF di WHILE+ (2.1.2)

2. Semantica

+++ semantica intuitiva (2.1.3)
+++ semplice semantica formale: Store, espressioni, comandi (2.2)

3. Espressività di WHILE+

+++ calcolabilità in WHILE+ (5.1)
+++ Compilazione in WHILE+ di MdT (Cap.8)

4. Compilazione, interpretazione e specializzazione

+++ Programmi come dati (da Von Neumann in poi)
+++ simulazione (3.1)
+++ sintassi concreta di WHILE+ (3.2)
+++ compilazione (3.3)
+++ interpretazione (3.4)
+++ universalità in WHILE+ (4.1)
+++ specializzazione (3.6)
+++ proiezioni di futamura (6.5)

Part 3

Teoria della Ricorsione

CHAPTER 19

Teoria matematica della ricorsione

Abbiamo visto nel Cap. 10 diverse formalizzazioni per definire ciò che noi riteniamo effettivamente calcolabile. In questo capitolo studieremo alcuni risultati classici della teoria della ricorsività. L'equivalenza tra le diverse definizioni di calcolabilità e la Tesi di Church ci permette di studiare l'effettiva calcolabilità in un modo per così dire "più astratto". In particolare la Tesi di Church ci permette di astrarre dalla particolare MdT o WHILE-program, e trattare in modo formale il concetto (informale) di funzione effettivamente calcolabile, indipendentemente dal formalismo adottato per rappresentarla. Al solito, nel seguito di queste note, assumeremo di avere fissato una enumerazione $\{\varphi_i\}_{i\in\mathbb{N}}$ delle MdT.

In questo capitolo studieremo le proprietà ricorsive di insiemi di numeri naturali. Questo ci permetterà di studiare, analogamente a quanto visto per le funzioni calcolabili, gli insiemi (di numeri) effettivamente costruibili in modo algoritmico. È evidente che trattare insiemi di numeri naturali non è per nulla limitativo. Un analogo ragionamento si può applicare ad insiemi di strighe su di un alfabeto, ovvero ad un linguaggio, o ad insiemi di alberi sintattici, rappresentanti WHILE+-programs. È infatti noto l'isomorfismo tra $\mathbb{N}$ e rispettivamente Σ^* e $\mathbb{D}_A$, essendo Σ un dato alfabeto e A un dato insieme di *atomi*. Da questo consegue che studiare proprietà degli elementi di $\wp(\mathbb{N})$ equivale a studiare le proprietà degli elementi di $\wp(\Sigma^*)$ e $\wp(\mathbb{D}_A)$, e che ogni definizione/proprietà che vedremo per insiemi di naturali, si applica senza restrizione alcuna a linguaggi (ovvero elementi di $\wp(\Sigma^*)$) e a insiemi di alberi sintattici. In questo senso, il passaggio da funzioni ad insiemi è naturale e rappresenta una astrazione successiva verso una più astratta definizione di problema ricorsivamente risolvibile.

Il materiale contenuto in questo capitolo e nei seguenti è tratto in larga parte dai seguenti testi di riferimento [**1, 5, 4, 8, 10**].

CHAPTER 20

Insiemi ricorsivi e ricorsivamente enumerabili

Consideriamo un insieme $I \subseteq \mathbb{N}$. Ci chiediamo se I è costruibile in modo algoritmico, ovvero se è possibile generare gli elementi di I mediante una funzione che sia effettivamente calcolabile.

DEFINIZIONE 20.1. Un insieme $I \subseteq \mathbb{N}$ è detto *ricorsivamente enumerabile* (r.e.) se esiste una funzione parziale ricorsiva ψ tale che $I = dom(\psi)$.

Nel seguito indicheremo con $RE \subseteq \wp(\mathbb{N})$ la classe degli insiemi ricorsivamente enumerabili. Pertanto $A \in RE$ sse A è r.e. Inoltre, poichè sappiamo enumerare le MdT e quindi le funzioni parziali ricorsive, nel seguito indicheremo con W_x l'insieme r.e. associato alla funzione φ_x secondo una opportuna Gödelizzazione, ovvero $W_x = dom(\varphi_x)$ essendo φ_x l'x-esima MdT nella numerazione fissata.

Sia $A \in RE$. La funzione parziale:

$$\psi_A(x) = \begin{cases} 1 & \text{se } x \in A \\ \uparrow & \text{se } x \notin A \end{cases}$$

è detta *funzione semicaratteristica* dell'insieme A. È chiaro che $A = dom(\psi_A)$. Inoltre è evidente che, essendo per ipotesi A r.e. , la verifica della condizione $x \in A$ è semidecidibile, ovvero se $A = W_y$, $x \in A$ sse $\varphi_y(x) \downarrow$. Sia dunque $P_y \stackrel{\text{def}}{=} \textbf{read } x; C; \textbf{write } z$ un programma WHILE+ equivalente a φ_y. È pertanto naturale definire il seguente programma: $\textbf{read } x; C; \textbf{write } 1$. Questo programma corrisponde chiaramente ad una MdT, ed implementa la funzione ψ_A. Pertanto abbiamo dimostrato che un insieme è detto r.e. sse ha una funzione semicaratteristica parziale ricorsiva.

All'interno della famiglia degli insiemi r.e. , esistono insiemi con caratteristiche ricorsive più forti. Per questi insiemi vale la condizione

aggiuntiva che la loro funzione caratteristica (che è sempre una funzione totale) è ricorsiva, ovvero esiste un algoritmo in grado di *decidere* l'appartenenza o meno di un elemento all'insieme.

DEFINIZIONE 20.2. Un insieme A è detto *ricorsivo* se esiste una funzione ricorsiva (totale) f_A tale che:

$$f_A(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{se } x \notin A \end{cases}$$

ESEMPIO 20.3. I seguenti sono insiemi ricorsivi:

- L'insieme $\{0, 2, 4, 6, \dots\}$ dei numeri pari;
- $\mathbb{N}$ e $\emptyset$;
- Ogni insieme finito (Dimostrare per esercizio);
- Ogni insieme A tale che $\bar{A}$ è finito (Dimostrare per esercizio).

Esisteranno dunque $\aleph_0$ insiemi r.e. di cui una parte di uguale cardinalità sarà ricorsiva. Si nota infatti subito della definizoni che

$$\boxed{A \text{ ricorsivo} \Rightarrow A \in RE}$$

Dato A insieme ricorsivo, basta infatti definire la funzione parziale e chiaramente ricorsiva:

$$\varphi(x) = \begin{cases} 1 & \text{se } f_A(x) = 1 \\ \uparrow & \text{altrimenti} \end{cases}$$

ESERCIZIO 20.4. Dimostrare che i seguenti insiemi sono ricorsivi:

- $\{ x \in \mathbb{N} \mid \exists n \exists p \text{ primo } . \, x = p^n \}$;
- $\{ \langle x, y \rangle \in \mathbb{N} \times \mathbb{N} \mid x^4 + y^6 = 1238 \}$;
- $\{ x \in \mathbb{N} \mid \exists n. \, x = 2n + 1 \}$.

TEOREMA 20.5 (Teorema di Post). *Un insieme A è ricorsivo se solo se A e $\bar{A}$ sono r.e.*

PROOF. Sia $A \subseteq \mathbb{N}$ e $\bar{A} = \mathbb{N} \setminus A$.

($\Rightarrow$). Se A è ricorsivo allora sia f_A la funzione caratteristica (totale) ricorsiva. Definiamo:

$$\psi(x) = \begin{cases} 1 & \text{se } f_A(x) = 1 \\ \uparrow & \text{altrimenti} \end{cases}$$

È chiaramente una funzione parziale ricorsiva. Ad esempio il seguente frammento di programma implementa ψ: Sia $F_A \stackrel{\text{def}}{=} \textbf{read}\ x; C_A; \textbf{write}\ y$.

G : **read** x;
C_A
while $y = 0$ **do** $y := y$ **endw**;
write y

A ha quindi una funzione semicaratteristica parziale ricorsiva. Analogo discorso si applica a $\bar{A}$, essendo $f_{\bar{A}} = 1 - f_A$ la corrispondente funzione caratteristica. Abbiamo pertanto dimostrato che A e $\bar{A}$ sono r.e.

($\Leftarrow$). Supponiamo A e $\bar{A}$ r.e. , con funzioni semicaratteristiche ψ_A e $\psi_{\bar{A}}$. Definiamo:

$$f(x) = \begin{cases} 1 & \text{se } \psi_A(x)\downarrow \\ 0 & \text{se } \psi_{\bar{A}}(x)\downarrow \end{cases}$$

È banale osservare che, dato $x \in \mathbb{N}$: o $x \in A$ o $x \in \bar{A}$, allora eseguendo a turno un'istruzione della MdT che calcola ψ_A e un'istruzione della MdT che calcola $\psi_{\bar{A}}$, prima o poi una delle due MdT termina, rendendo la funzione f totale ricorsiva. □

Si osservi come la ricorsività di $\mathbb{N}$ segua anche dal fatto che sia $\bar{\mathbb{N}} = \emptyset$ che $\mathbb{N}$ sono r.e.

Il seguente teorema caratterizza gli insiemi r.e. come la famiglia degli insiemi che possono essere effettivamente enumerati da un algoritmo, ovvero i cui elementi possono essere "listati" in modo effettivo come "output" di un programma. Gli insiemi r.e. possono dunque essere visti come liste, eventualmente infinite, di elementi generabili da un algoritmo.

Il teorema di caratterizzazione fa uso di una metodologia effettiva per esplorare il campo di convergenza di una funzione parziale, ovvero l'insieme degli elementi su cui una funzione parziale converge (è definita). Questa metodologia per esplorare il campo di convergenza di una funzione parziale è detta a *coda di colomba* (dovetail) e sarà spesso utilizzata nel seguito. In essa ha un ruolo essenziale il concetto

FIGURE 1. Dovetail

di passo di calcolo o derivazione. Come abbiamo visto nel Cap. 10 ogni sistema formale introdotto definisce il concetto di calcolo come sequenza discreta di passi elementari. Pertanto, il procedimento a dovetail è perfettamente accettabile come metodo algoritmico per esplorare il campo di definizione (o convergenza) di una funzione parziale ricorsiva.

Sia dunque φ_x una funzione parziale ricorsiva. Applicare la tecnica dovetail a φ_x significa costruire l'insieme L in modo induttivo, tale che:

Passo 0**:** Si faccia un passo di calcolo per $\varphi_x(0)$, e se termina allora $L := \varphi_x(0)$;

Passo $n+1$**:** Si faccia: un passo nel calcolo di $\varphi_x(0), \ldots, \varphi_x(n)$ ove il calcolo non sia già terimato al passo precedente, e per ogni m tale che $\varphi_x(m)$ converge, si pone $L := L \cup \varphi_x(m)$.

In pratica si esplora il grafico della funzione seguendo un andamento a zig-zag, come evidenziato nella figura 1.

L'insieme L è stato dunque effettivamente costruito ed i suoi valori corrispondono al codominio definito di φ_x.

Siamo ora in grado di dimostrare il seguente teorema di Kleene di caratterizzazione degli insiemi r.e.

TEOREMA 20.6 (Teorema di caratterizzazione degli insiemi r.e.). *Le seguenti condizioni sono equivalenti:*

1. $A \in RE$;
2. *esiste* $\varphi \in \mathcal{PR}$: $A = range(\varphi)$;
3. $A = \emptyset$ *oppure esiste una funzione* $f \in R$ *tale che* $A = range(f)$.

PROOF. Consideriamo una data numerazione delle MdT (o delle funzioni parziali ricorsive) $\{\varphi_x\}_{x\in\mathbb{N}}$.

$(1 \Rightarrow 2)$. Sia $A \in RE$. Per definizione, esisterà φ_x parziale ricorsiva t.c. $A = dom(\varphi_x)$. Definiamo:

$$\delta(y) = \begin{cases} y & \text{se } \varphi_x(y)\downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

Ovvero $\delta(y) \stackrel{\text{def}}{=} y + 0\varphi_x(y)$. È chiaro che δ è parziale ricorsiva e $range(\delta) = dom(\varphi_x) = A$.

$(2 \Rightarrow 3)$. Sia $A = range(\varphi_x)$ e $A \neq \emptyset$. Applicando il dovetail alla funzione:

$$\psi(y) = \begin{cases} \varphi_x(y) & \text{se } \varphi_x(y)\downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

poichè $A = range(\varphi_x) \neq \emptyset$, esiste $n_0 \in \mathbb{N}$ tale che all'n_0-esimo stadio del dovetail si rivela una terminazione. Definiamo la funzione f induttivamente sullo stadio n-esimo del dovetail, in modo tale che f abbia valori tutti e soli nella lista L generata dal dovetail di ψ.

- Se ψ converge in $\{m_0, \ldots, m_{j_0}\}$ con $0 \leq j_0 \leq n_0$ allora per ogni $i \in [0, j_0]$: $f(i) = \psi(m_i)$;
- Supponiamo che per definire $f(j_p)$ si sia usato lo stadio n_p-esimo del dovetail. Possiamo avere i seguenti casi:
  - Se nell'n_p+1-esimo passo del dovetail non si rivelano nuove terminazioni, allora $f(j_p + 1) = f(j_p)$.
  - Se nell'n_p+1-esimo passo del dovetail ψ converge con input $\{p_0, \ldots, p_k\}$ con $0 \leq k \leq n_p + 1$, allora per ogni $i \in [0, k]$: $f(j_p + 1 + i) = \psi(p_i)$.
- Iteriamo il precedente punto ponendo $j_{p+1} = j_p + 1 + k$.

La procedura che abbiamo descritto è effettiva e definisce una funzione ricorsiva totale f tale che $A = range(f)$.

$(3 \Rightarrow 1)$. Se $A = \emptyset$ allora $A = dom(\lambda x.\uparrow)$. Sappiamo che $\lambda x.\uparrow \in \mathcal{PR}$, e quindi A è r.e. Se $A = range(f)$ con $f \in \mathcal{P}$, basta porre:

$$\varphi(x) = \mu z.(f(z) - x = 0)$$

$\varphi \in \mathcal{PR}$ ed inoltre è immediato osservare che $\varphi(x)\downarrow \Leftrightarrow \exists z.\ f(z) = x$, quindi $A = dom(\varphi)$ che dimostra che A è r.e. □

ESEMPIO 20.7. $\mathbb{N}$ è r.e. , poichè $\lambda x.\ x \in \mathcal{R}$ e $\mathbb{N} = range(\lambda x.\ x)$.

COROLLARIO 20.8. Esiste una funzione totale ricorsiva f tale che $W_x = range(\varphi_{f(x)})$.

È chiaro che non è detto che $\varphi_{f(x)}$ sia totale, pur essendolo f.

TEOREMA 20.9. *Le seguenti condizioni sono equivalenti:*

1. *A è ricorsivo;*
2. *$A = \emptyset$ oppure $A = range(f)$ con $f \in \mathcal{R}$ non decrescente.*

PROOF. $(\Rightarrow)$. Sia $A \neq \emptyset$ e A ricorsivo. Poichè $A \subseteq \mathbb{N}$, esiste $a = min(A)$ minimo valore tra quelli contenuti in A. Definiamo:

$$\begin{aligned} f(0) &= a \\ f(n+1) &= \begin{cases} n+1 & \text{se } n+1 \in A \\ f(n) & \text{altrimenti} \end{cases} \end{aligned}$$

È chiaro che f è totale ricorsiva non decrescente e che $A = range(f)$.

$(\Leftarrow)$. Se A è finito, allora A è banalmente ricorsivo. Supponiamo che A sia infinito e $A = range(f)$ con f totale ricorsiva non decrescente. Sia $x \in A$ e $z_x = \mu y.\ (f(y) > x)$. È chiaro che un tale z_x esiste (perchè?) e che

$$x \in A \ \Leftrightarrow\ x \in \{f(0), \ldots, f(z)\}$$

È quindi decidibile l'appartenenza di x ad A, ovvero A è ricorsivo (definire la funzione caratteristica di A). □

TEOREMA 20.10. *Ogni insieme r.e. infinito ha un sottoinsieme ricorsivo infinito.*

PROOF. Sia $A \subseteq \mathbb{N}$ tale che $A = range(f)$ con f funzione ricorsiva totale e $|A| = \omega$. Definiamo la funzione g tale che:

$$\begin{aligned} g(0) &= f(0) \\ g(n+1) &= f(\mu y.\, f(y) > g(n)). \end{aligned}$$

$g(n+1)$ è dunque il più piccolo elemento di A più grande di $g(n)$. È evidente che g è crescente, quindi $range(g)$ è ricorsivo per il Teorema 20.9. Inoltre $range(g) \subseteq A$, poichè per definizione di g: $range(g) \subseteq range(f) = A$. □

ESERCIZIO 20.11. Si dica se i seguenti insiemi sono ricorsivi o r.e. :

1. $A = range(f)$ con
$$\begin{aligned} f(0) &= n_0 \\ f(n+1) &= 2f(n). \end{aligned}$$
2. $B = range(g)$ con
$$\begin{aligned} g(0) &= n_0 \\ g(n+1) &= h(n)g(n). \end{aligned}$$
essendo h una generica funzione ricorsiva (totale).
3. Data una funzione ricorsiva f, dimostrare che l'immagine inversa di f e W_x, con $x \in \mathbb{N}$, ovvero l'insieme $f^{-1}(W_x) = \{\, y \mid f(y) \in W_x \,\}$, è r.e.
4. Sia $D = range(\psi)$ con:
$$\begin{aligned} \psi(0) &= n_0 \\ \psi(n+1) &= \varphi_n(n)\psi(n). \end{aligned}$$
D è ricorsivo?

Fino a questo momento abbiamo visto insiemi ricorsivi e r.e. Tuttavia non abbiamo ancora visto un esempio di un insieme che sia r.e. ma non ricorsivo. Questo insieme (se esiste) rappresenta l'insieme che discrimina le due classi di insiemi che abbiamo definito. È chiaro che

un insieme di questo tipo deve essere r.e. ma non ricorsivo, ovvero l'appartenenza ad esso deve essere un predicato semidecidibile. È naturale chiedersi se questo insieme possa essere in qualche modo correlato alla non decidibilità della terminazione. Al solito, fissiamo una numerazione delle MdT o delle funzioni parziali ricorsive $\{\varphi_x\}_{x\in\mathbb{N}}$.

DEFINIZIONE 20.12. Definiamo $K = \{\, x \mid \varphi_x(x)\downarrow \,\}$.

È chiaro per la definizione sopra che $K = \{\, x \mid x \in W_x \,\}$[1].

TEOREMA 20.13. *K è r.e. ma non ricorsivo.*

PROOF. definiamo la funzione ψ_K tale che:

$$\psi_K = \begin{cases} 1 & \text{se } \varphi_x(x)\downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

ψ_K è parziale ricorsiva (perchè?) e semicaratteristica di K, quindi K è r.e.

Supponiamo K ricorsivo. Allora $\bar{K}$ è r.e. Sia y_0 tale che $\bar{K} = W_{y_0}$. Segue che:

$$y_0 \in W_{y_0} = \bar{K} \;\Leftrightarrow\; y_0 \in K$$

che è chiaramente assurdo. □

ESERCIZIO 20.14.
1. Dimostrare che $K_2 = \{\, \langle x, y\rangle \mid x \in W_y \,\}$ è r.e. ma non ricorsivo;
2. Dimostrare che le seguenti proprietà caratterizzano gli insiemi finiti;

[1]Questo insieme ricorda l'insieme utilizzato per contraddire la teoria classica degli insiemi, ovvero il *Paradosso di Russell* [**11**]. Se infatti fosse possibile definire un insieme del tipo: $M = \{\, x \mid x \notin x \,\}$ allora otterremmo immediatamente un paradosso poichè $M \in M$ sse $M \notin M$. L'idea utilizzata per risolvere questo paradosso nella teoria degli insiemi è quella di non ammettere M come insieme, ovvero di definire una "disciplina" per la costruzione di insiemi che vieti una costruzione tipo M. Si osservi come la teoria della ricorsione è immune da questo tipo di paradossi. Infatti $\bar{K} = \{\, x \mid x \notin W_x \,\}$, l'insieme dei numeri non appartenenti agli insiemi r.e. da essi generati, non può essere r.e. , altrimenti, essendolo K per il Teorema 20.13, K sarebbe ricorsivo (Teorema di Post) il che per il Teorema 20.13 è assurdo. Quindi gli insiemi r.e. non includono $\bar{K}$, ovvero non esiste y_0 tale che $\bar{K} = W_{y_0}$

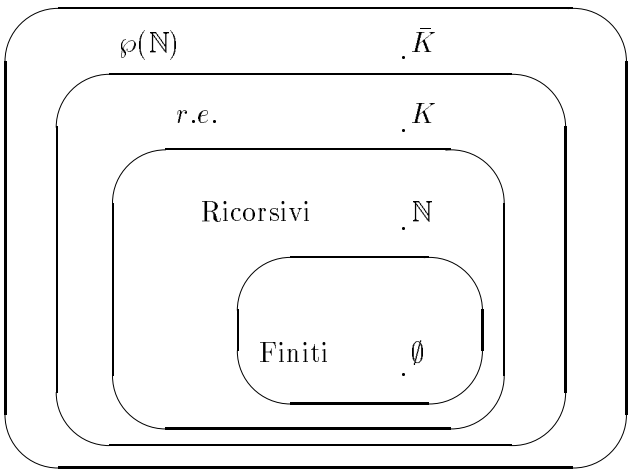


FIGURE 2. La gerarchia degli insiemi Ricorsivi

- tutti i sottoinsiemi sono r.e.
- tutti i sottoinsiemi sono ricorsivi

3. Dimostrare che esistono funzioni ricorsive (totali) f e g tali che: $W_x \cap W_y = W_{f(x,y)}$ e $W_x \cup W_y = W_{g(x,y)}$;
4. Dimostrare che A è ricorsivo infinito sse esiste una funzione f ricorsiva crescente (i.e. $\forall n.\ f(n) < f(n+1)$) tale che $A = range(f)$.

Abbiamo quindi la seguente rappresentazione in "classi" di risolvibilità degli insiemi di numeri naturali. La classe degli insiemi ricorsivi corrisponde alla classe degli insiemi *decidibili*, quella degli insiemi r.e. corrisponde alla classe degli insiemi semidecidibili, mentre gli insiemi non r.e. (che sono la maggior parte di $\wp(\mathbb{N})$) sono gli insiemi per cui non è possibile dare alcun metodi effettivo per verificare sia l'appartenenza che la non appartenenza di un elemento a questi insiemi. Valgono quindi le seguenti inclusioni, mostrate anche in Figura 2, tra le varie classi di insiemi viste fino ad ora:

$$ricorsivi \ \subset \ r.e. \ \subset \ non - r.e.$$

CHAPTER 21

Il teorema di ricorsione e sue conseguenze

I teoremi di ricorsione, detti anche di punto fisso, sono tra i risultati più importanti che si incontrano nello studio della teoria della ricorsività.

TEOREMA 21.1 (Teorema di ricorsione di Kleene). *Sia t una funzione ricorsiva (totale). Esiste $n_0 \in \mathbb{N}$ tale che $\varphi_{n_0} = \varphi_{t(n_0)}$.*

PROOF. Sua u un indice di una generica MdT φ_u. Definiamo la seguente funzione parziale:

$$\psi(u,x) = \begin{cases} \varphi_{\varphi_u(u)}(x) & \text{se } \varphi_u(u)\downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

È chiaro che la funzione così definita è parziale ricorsiva (si descriva a parole una MdT che "calcola" ψ). Quindi, per il Teorema s-m-n: esiste una funzione ricorsiva (totale) g tale che: $\psi(u,x) = \varphi_{g(u)}(x)$. Sia ora t una funzione ricorsiva (totale). Poichè la composizione di funzioni ricorsive è ricorsiva, $t \circ g$ è ricorsiva. Quindi esiste $v \in \mathbb{N}$ tale che $t \circ g(v) = \varphi_v(v)$. Vista la genericità di u, sostituendo u con v nella definizione di $\psi(u,x)$ otteniamo:

$$\varphi_{g(v)}(x) = \begin{cases} \varphi_{t(g(v))}(x) & \text{se } \varphi_v(v) = t(g(v))\downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

Poichè $t \circ g$ è totale, $\varphi_{g(v)}(x) = \varphi_{t(g(v))}(x)$. Posto quindi $n_0 = g(v)$, abbiamo che $\varphi_{n_0} = \varphi_{t(n_0)}$. □

Il significato intuitivo del teorema di ricorsione di Kleene sarà più chiaro in seguito, in particolare nella Sezione 2. Per ora ci basti utilizzarlo come potente metodo per dimostrare proprietà di ricorsività sugli elementi di $\wp(\mathbb{N})$. Uno dei risultati più importanti, dimostrabili

mediante il teorema di ricorsione, è il Teorema di Rice. Esso afferma che ogni proprietà non banale che rappresenti ciò che viene calcolato dalle MdT (o dai programmi, vedi Sezione 22) non è ricorsiva.

Definiamo cosa si intende per *proprietà* sulle MdT. Sia al solito $\{\varphi_i\}_{i\in\mathbb{N}}$ una enumerazione delle MdT.

DEFINIZIONE 21.2. Una proprietà sulle MdT è un qualsiasi sottoinsieme di $\{\varphi_i\}_{i\in\mathbb{N}}$.

È evidente che una proprietà sulle MdT è univocamente identificata dall'insieme degli indici delle MdT in essa contenuti. Nel seguito quindi non faremo distinzione tra proprietà intesa come insieme di indici (e quindi di numeri) di MdT o proprietà intesa come insieme di MdT. Nel primo caso, ovviamente potremo parlare di proprietà ricorsive (o decidibili), r.e. (o semidecidibili), e proprietà non r.e.

DEFINIZIONE 21.3. Sia dunque $\Pi \subseteq \{\varphi_i\}_{i\in\mathbb{N}}$. Π è estensionale (o chiusa per eguaglianza estensionale) se per ogni $x, y \in \mathbb{N}$:

$$x \in \Pi \wedge \varphi_x = \varphi_y \ \Rightarrow y \in \Pi$$

Intuitivamente una proprietà è estensionale quando questa "parla" di cosa calcolano le MdT in essa contenute, e non di come queste macchine sono "fatte".

Il seguente risultato permette di costruire un numero arbitrario di MdT tra loro equivalenti.

LEMMA 21.4 (Tecnica del Padding). $\forall i, k.\ \exists j :\ \varphi_j = \varphi_i \wedge j > k$.

PROOF. Siano dati i e k e sia φ_i l'i-esima MdT. Se $i > k$ allora ponendo $j = i$ si ha la tesi. Se $i \leq k$ allora è sufficente aggiungere a φ_i un numero sufficente di quintuple (con stati mai raggiungibili dal calcolo) in modo tale che si ottenga φ_j con $j > k$. □

Mediante la tecnica del padding si osserva che esistono infinite MdT equivalenti ad una MdT data. Pertanto, una proprietà estensionale sulle MdT non può che essere un insieme infinito.

ESEMPIO 21.5. Le seguenti proprietà sono tutte estensionali (dimostrare per esercizio):

- Dato $F \subseteq \mathcal{PR}$, $\{\, i \mid \varphi_i = \psi \ \wedge \psi \in F \,\}$;
- $\{\, i \mid \varphi_i = \lambda x.\, x \,\}$

- $\{ i \mid \varphi_i$ è definitivamente $\geq 3 \}$.

Al contrario, la proprietà $\{157\}$ che include solo la MdT di indice 157 non è estensionale. Infatti esistono infinite MdT equivalenti ad una data MdT, e pertanto l'insieme $\{157\}$ non può essere estensionale.

TEOREMA 21.6 (Teorema di Rice). *Sia* Π *una proprietà estensionale.* Π *è ricorsiva sse è banale (ovvero* $\Pi = \emptyset$ *oppure* $\Pi = \mathbb{N}$*).*

PROOF. ($\Leftarrow$). Banale.

($\Rightarrow$). Supponiamo che Π sia ricorsivo, ovvero che abbia una funzione caratteristica ricorsiva:

$$g(x) = \begin{cases} 1 & \text{se } x \in \Pi \\ 0 & \text{se } x \notin \Pi \end{cases}$$

Supponiamo che Π non sia banale. Allora esistono $a, b \in \mathbb{N}$ tali che $a \in \Pi$ e $b \notin \Pi$. Definiamo la funzione

$$h(x) = \begin{cases} b & \text{se } x \in \Pi \\ a & \text{se } x \notin \Pi \end{cases}$$

Essendo g ricorsiva per ipotesi, chiaramente h è ricorsiva (totale). Per il Teorema di ricorsione di Kleene, esiste n_0 tale che $\varphi_{n_0} = \varphi_{h(n_0)}$. Da questo fatto segue immediatamente l'assurdo:

- Se $n_0 \in \Pi$ allora essendo Π estensionale, $h(n_0) \in \Pi$, ma questo è assurdo poichè $h(n_0) = b$ e $b \notin \Pi$.
- Se $n_0 \notin \Pi$ allora essendo Π estensionale, $h(n_0) \notin \Pi$, ma questo è assurdo poichè $h(n_0) = a$ e $a \in \Pi$.

Quindi Π non può essere ricorsivo. □

Il Teorema di Rice dunque afferma che ogni funzione parziale ricorsiva o MdT ammette un insieme infinito non ricorsivo di indici di funzioni o macchine equivalenti. Questo insieme può essere in generale r.e. o non r.e. In particolare osserviamo che i seguenti problemi, di chiaro interesse informatico, non sono decidibili in seguito al Teorema di Rice.

1. Data $\psi \in \mathcal{PR}$, ed una MdT M, non è in generale decidibile se M "calcola" ψ. Infatti se $j = gn(M)$ è l'indice della MdT M, allora questo problema equivale a decidere se $j \in \{ i \mid \varphi_i = \psi \}$

che è una proprietà estensionale non banale ($\{ i \mid \varphi_i = \psi \} \neq \emptyset$ poichè esiste una MdT che calcola la funzione parziale ricorsiva ψ essendo $\psi \in \mathcal{PR}$ e $\{ i \mid \varphi_i = \psi \} \neq \mathbb{N}$ poichè non tutte le MdT calcolano la stessa funzione) delle MdT, e quindi per il Teorema di Rice è una proprietà non ricorsiva.

2. Date M_1 e M_2, non è decidibile se esse sono equivalenti (ovvero calcolano la stessa funzione). Dimostrare questo per esercizio applicando il teorema di Rice.

È necessario fare molta attenzione ad utilizzare il Teorema di Rice. Esso infatti ha come ipotesi il fatto che l'insieme di cui si vuole studiare la ricorsività sia *estensionale*. Alcuni insiemi possono trarre in inganno, come nel seguente esempio.

ESEMPIO 21.7. Il seguente esempio, ideato da G. Longo [**5**], non può essere risolto invocando il Teorema di Rice.

Definiamo una nozione *debole* di equivalenza tra MdT (o programmi) come segue:

$$\varphi_i \sim \varphi_j \;\Leftrightarrow\; \forall x.\, (\varphi_i(x)\downarrow \;\wedge \varphi_j(x)\downarrow \Rightarrow\; \varphi_i(x) = \varphi_j(x))$$

Ci chiediamo se l'insieme $D = range(\psi)$ con $\psi(i) = \mu j.\ (\varphi_i \sim \varphi_j)$ è ricorsivo.

In questo caso, l'utilizzo del Teorema di Rice porterebbe ad affermare erroneamente che D non è ricorsivo. Si osserva infatti che il Teorema di Rice non è applicabile a questo caso: l'insieme D infatti non è estensionale. Poichè infatti $= \Rightarrow \sim$, se $i \in D$ e $\varphi_k = \varphi_i$ con $k > i$ (k è facilmente ottenibile con la tecnica del Padding), allora sicuramente, $k \notin D$ (dimostrare per esercizio).

Al contrario, l'insieme D è ricorsivo. Si osserva infatti che la funzione sempre indefinita $\lambda x.\ \uparrow$ è debolmente equivalente ad ogni altra funzione parziale ricorsiva, ovvero, per ogni $i \in \mathbb{N}$: $\varphi_i \sim \varphi_u$ con u indice della più piccola MdT nella numerazione delle MdT $\{\varphi_n\}_{n\in\mathbb{N}}$, tale che $\varphi_u = \lambda x.\ \uparrow$. Quindi, $D \subseteq [0, u]$, ovvero D è un insieme fintito e quindi ricorsivo.

ESERCIZIO 21.8.
1. Dire (giustificando formalmente) per quali $A \subseteq \mathbb{N}$ il seguente insieme $\{ i \mid W_i = A \}$ è ricorsivo.
2. Dire (giustificando formalmente) se $\{ i \mid \forall x.\, \varphi_i(x) \uparrow \}$ è ricorsivo, r.e. o non r.e.

3. Dimostrare che $\{ i \mid \varphi_i \text{ è definitivamente } \geq 3 \}$ è non ricorsivo.
4. Dimostrare che $\{ i \mid \exists x.\, \varphi_i(x) = x \}$ è non ricorsivo.

Il Teorema di Rice è assai disarmante di fronte a proprietà estensionali che riguardano MdT (quindi programmi). Di fatto, ogni proprietà non banale che tratti ciò che calcolano le MdT è non ricorsiva (non decidibile). Tuttavia, è possibile rappresentare alcune proprietà estensionali non ricorsive (ma r.e.) in modo decidibile, ricorrendo a proprietà su MdT non estensionali decidibili.

DEFINIZIONE 21.9. Sia Π una proprietà. $A \subseteq \mathbb{N}$ è una rappresentazione di Π se per ogni $\varphi_i \in \Pi$ esiste $j \in A$ tale che $\varphi_i = \varphi_j$ e viceversa, per ogni $j \in A$, esiste $\varphi_i \in \Pi$ tale che $\varphi_i = \varphi_j$.

Ogni insieme $A \subseteq \mathbb{N}$ può essere esteso in modo estensionale ad una proprietà $\Pi(A)$ tale che $A \subseteq \Pi(A)$ e A rappresenta $\Pi(A)$. Basta definire $\Pi(A) = \{ i \mid \exists j \in A.\, \varphi_i = \varphi_j \}$.

TEOREMA 21.10. *Sia A un insieme r.e. Esiste una rappresentazione ricorsiva della proprietà $\Pi(A)$.*

PROOF. Se $A = \emptyset$ allora basta porre $B = A = \emptyset$. B è chiaramente una rappresentazione ricorsiva di $\Pi(A) = \emptyset$. Supponiamo che $A \neq \emptyset$. Per il Teorema 20.6, esiste una funzione ricorsiva (totale) f tale che $A = range(f)$. Definiamo g tale che:

$$\begin{aligned} g(0) &= f(0) \\ g(n+1) &= j_n \qquad \text{dove } \varphi_{j_n} = \varphi_{f(n+1)} \text{ e } j_n > g(n) \end{aligned}$$

Questa funzione è ben definita per il Lemma del Padding (j_n così costruito esiste sempre, basta porre $i = f(n+1)$ e $k = g(n)$ nel Lemma del Padding). È chiaro che g è crescente e ricorsiva. Quindi per il Teorema 20.9 $B = range(g)$ è ricorsivo. Inoltre, è chiaro che B è una rappresentazione per $\Pi(A)$ (segue banalmente per costruzione, verificare per esercizio). □

CHAPTER 22

Teorema di ricorsione e programmazione

1. Teorema di Rice e proprietà di programmi

2. Funzionali ricorsivi e trasformazioni di programmi

In questa sezione analizzeremo la possibilità, derivante dalla teoria della calcolabilità, di definire programmi in modo intensionale, ovvero implicito. Il materiale contenuto in questa sezione è una rielaborazione tratta da [**6, 10**]. L'idea è quella di dare significato in termini di programmi, ovvero funzioni calcolabili, a costrutti ampiamente utilizzati nei linguaggi di programmazione. Nel caso particolare dei funzionali ricorsivi, essi forniscono un modello matematico all'interno della teoria della calcolabilità per dare semantica a definizioni ricorsive di programmi. Costrutti linguistici di questo tipo sono presenti nei linguaggi di programmazione a partire dagli anni '60, con i linguaggi LISP e Algol, quest'ultimo progenitore di Pascal e C, e più di recente in Prolog. L'introduzione delle procedure ricorsive di Algol e le funzioni definite ricorsivamente (e.g. di LISP) hanno rappresntato un notevole passo avanti verso una programmazione ad alto livello, tipica dei linguaggi moderni. Basti pensare la naturale traduzione in un linguaggio di programmazione di una funzione semplice come il fattoriale:

$$f(n) = \begin{cases} 1 & \text{se } n = 0 \\ n\ f(n-1) & \text{Altrimenti} \end{cases}$$

che tradotta in Pascal diventa:

function $f\ (n : integer) : integer$;
begin
if $n = 0$ **then** $f := 1$ **else** $f := n * f(n-1)$
end

Il linguaggio WHILE e WHILE+ visto precedentemente non ammetteva procedure/funzioni ricorsive. Tuttavia esso era sufficentemente potente per rappresentare tutte le funzioni clacolabili. Da quanto dimostrato nei precedenti capitoli, ad ogni programma WHILE corrisponde una funzione parziale ricorsiva, e viceversa. Ci chiediamo ora, cosa succede de arricchiamo il nostro linguaggio con procedure/funzioni ricorsive. Chiaramente non è possibile estendere la classe delle funzioni da esso calcolabili, poichè questa già corrisponde alla classe delle funzioni Turing calcolabili. Quello che succede è una rricchimento del potere espressivo del linguaggio: Ad esempio è possibile descrivere algoritmi complessi in modo più succinto (e.g. il fattoriale). La struttura dei comandi con procedura é la seguente:

procedure P;
begin
corpo procedura
end

Nel corpo della procedura è possibile utilizzare il nome della procedura stessa. È evidente che il significato intuitivo di una procedura è quella di un programma, quindi di una funzione parziale ricorsiva. Pertanto, una procedura ricorsiva[1] può essere vista come una funzione che *dipende* da una data funzione incognita, ovvero una funzione che manipola funzioni, detta anche *funzionale*. Mentre un programma WHILE corrisponde ad una funzione parziale ricorsiva, l'estensione del linguaggio WHILE con funzioni e procedure ricorsive, permette di

[1] Attenzione all'uso in diversi contesti del termine "ricorsivo".

definire funzioni che ammettono come parametro altre funzioni, ovvero *funzionali*. Si consideri il seguente

In questa sezione daremo un modello basato sulla teoria della calcolabilità per programmi con funzioni e procedure ricorsive. In particolare dimostreremo come il Teorema di ricorsione di Kleene permetta di trovare un programma (e.g. WHILE) senza procedure equivalente ad uno avente procedure ricorsive. Al solito utilizzeremo indistintamente programmi WHILE e MdT. Il Teorema di ricorsione di Kleene (detto anche primo teorema di ricorsione) afferma che, se t è una funzione ricorsiva (totale), allora esiste un indice n_0 tale che

$$\varphi_{t(n_0)} = \varphi_{n_0}$$

Data dunque una numerazione delle MdT (o dei programmi WHILE): $\{\varphi_n\}_{n\in\mathbb{N}}$, la funzione t può essere vista come una trasformazione effettiva di indici di MdT, ovvero come una funzione che trasforma programmi. Sia dunque $\tau : L - Prog \to L - Prog$, una funzione sui programmi scritti in un dato linguaggio di programmazione (e.g. $L =$ WHILE). Ci chiediamo inizialmente: cosa vuole dire trasformare effettivamente un programma, ovvero un oggetto $P \in L - Prog$? Sappiamo che non è restrittivo considerare le funzioni calcolabili come funzioni sui naturali. Ad esempio, la seguente funzione appare a prima vista come una trasformazione effettiva di programmi (funzioni parziali ricorsive):

$$\Sigma(\varphi) = \lambda x. \begin{cases} 0 & \text{se } x = 0 \\ 1 & \text{se } x = 1 \\ \varphi(x-1) + x & \text{altrimenti} \end{cases}$$

Infatti, se φ è una funzione parziale ricorsiva, il risultato calcolato da Σ è anch'esso una funzione parziale ricorsiva poichè alla funzione φ vengono applicati solo schemi ricorsivi. Tale funzione è facilmente definibile in un qualsiasi linguaggio di programmazione che ammette procedure o funzioni ricorsive. Per definire più formalmente ciò che intendiamo per trasformazione effettiva di programmi, è necessario ricorrere alla seguente definizione.

DEFINIZIONE 22.1. Data una funzione ricorsiva di aritmetizzazione dei programmi in $L - Prog$ nei naturali, ovvero $gn : L - Prog \to \mathbb{N}$,

iniettiva con inversa gn^{-1} anch'essa chiaramente ricorsiva, $\tau : L - Prog \to L - Prog$ è una *trasformazione effettiva* di $L - Prog$ se esiste una funzione ricorsiva totale ψ tale che il seguente diagramma commuta, ovvero tale che $\tau = gn^{-1} \circ \psi \circ gn$.

$$\begin{array}{ccc} L - Prog & \xrightarrow{\quad\tau\quad} & L - Prog \\ \Big\downarrow gn & & gn^{-1} \Big\uparrow \\ \mathbb{N} & \xrightarrow{\quad\psi\quad} & \mathbb{N} \end{array}$$

Ci chiediamo pertanto quale sia il significato della funzione Σ descritta precedentemente, o più in generale di una trasformazione effettiva di programmi. È possibile far corrispondere una funzione parziale ricorsiva ad ogni procedura ricorsiva definita implementando una trasformazione effettiva di programmi, nello stesso modo in cui la funzione fattoriale $!n$ corrisponde alla funzione ricorsiva descritta all'inizio $f(n)$? Si tratta evidentemente di trovare una funzione parziale ricorsiva φ (o programma), se esiste, tale che data una trasformazione effettiva di programmi τ: $\tau(\varphi) = \varphi$. Dalla precedente definizione segue che, fissata una numerazione delle MdT (o equivalentemente degli $L - Prog$), $\{\varphi_n\}_{n\in\mathbb{N}}$, una trasformazione $\tau : \{\varphi_n\}_{n\in\mathbb{N}} \to \{\varphi_n\}_{n\in\mathbb{N}}$ è effettiva se esiste una funzione ricorsiva $t : \mathbb{N} \to \mathbb{N}$ tale che

$$\tau(\varphi_i) = \varphi_{t(i)}.$$

Il Teorema di ricorsione di Kleene assicura l'esistenza di un programma "punto fisso" per ogni trasformazione effettiva di programmi, ovvero un programma di indice n_0 tale che sia soluzione dell'equazione ricorsiva sui programmi: $\tau(X) = X$. L'equazione $\tau(X) = X$ modella in questo caso una procedura ricorsiva di nome X definita in termini di X stesso. La dimostrazione di questo fatto è banale in quanto, per il Teorema di Kleene, esiste n_0 tale che $\tau(\varphi_{n_0}) = \varphi_{t(n_0)} = \varphi_{n_0}$. Il Teorema di ricorsione di Kleene afferma dunque che esiste sempre un programma privo di procedure ricorsive che da significato (semantica) alla definizione (procedura) ricorsiva $\tau(X) = X$. Tuttavia, il Teorema di Kleene, non ci fornisce nessuno strumento per "costruire" il programma soluzione dell'equazione $\tau(X) = X$. Al fine di caratterizzare la struttura dei programmi ottenibili come soluzioni di equazioni ricorsive del tipo $\tau(X) = X$, con τ trasformazione effettiva, è necessario

studiare la nozione di *funzionale ricorsivo*. L'idea è di estendere la nozione di ricorsività vista fino ad ora (sui numeri naturali) a funzioni che operano su funzioni, ovvero a funzionali.

2.1. Funzioni come insiemi. Un funzionale è semplicemente una funzione (nel senso standard del termine) in cui le variabili hanno valori sullo spazio delle funzioni. Manipolare funzioni può non essere agevole, in particolare considerando il fatto che fino ad ora abbiamo lavorato esclusivamente con numeri e insiemi di numeri. Tuttavia è possibile associare ad ogni funzione sui naturali, un opportuno insieme di numeri. Data una aritmetizzazione delle coppie di naturali: $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, è possibile *immergere* le funzioni sui naturali in $P\omega$. Ogni funzione parziale ψ può infatti essere messa in corrispondenza biunivoca con il suo grafico, ovvero con l'insieme

$$\hat{\psi} = \{ \langle n, m \rangle \mid m = \psi(n) \}.$$

ESERCIZIO 22.2. Descrivere il grafico delle seguenti funzioni parziali ricorsive $\lambda x. \uparrow$, $\lambda x.\ 1$, $\lambda x.\ x$, e

$$\lambda x. \begin{cases} x & \text{se } x \text{ è pari} \\ \uparrow & \text{altrimenti} \end{cases}$$

È possibile generalizzare il concetto di *grafico di funzione* e caratterizzare gli insiemi che corrispondono a possibili grafici di funzioni.

DEFINIZIONE 22.3. Un insieme $I \in P\omega$ è detto *univoco* se esiste una funzione parziale ψ tale che $I = \hat{\psi}$.

Ovvero, un insiem è univoco se per ogni coppia $\langle x, y \rangle, \langle x, z \rangle \in I$ si ha che $y = z$. Questa definizione corrisponde al fatto che l'insieme rappresenta esattamente il grafico di una funzione. Nel seguito chiameremo con $\mathcal{U}$ la classe degli insiemi univoci, ovvero:

$$\mathcal{U} = \{ \hat{\psi} \mid \psi : \mathbb{N} \to \mathbb{N} \cup \{\uparrow\} \}$$

È evidente che $\mathcal{U} \subset P\omega$.

ESERCIZIO 22.4. Dimostrare che:

- ψ è parziale ricorsiva sse $\hat{\psi}$ è r.e.

- se f è una funzione ricorsiva totale allora $\hat{f}$ è un isieme ricorsivo. Il viceversa vale?
- Esiste una funzione ricorsiva (totale) f tale che per ogni $n \in \mathbb{N}$:
  - $W_{f(x)}$ è univoco;
  - $W_{f(x)} \subseteq W_x$;
  - Se W_x è univoco allora $W_{f(x)} = W_x$.

Si osservi che non tutti gli insiemi r.e. sono univoci. Basti pensare all'insieme finito $\{\langle 1,1\rangle, \langle 1,2\rangle\}$.

Il passo successivo della nostra costruzione è quello di fornire una *struttura* all'insieme $P\omega$, in modo tale che due insiemi univoci (ovvero due funzioni) possano essere confrontati tra loro in base alla quantità di informazione che essi contengono. Il concetto di informazione associata ad un insieme univoco può ragionevolmente corrispondere a ciò che la funzione corrispondente è in grado di dare come output. È evidente che con queste ipotesi, la funzione parziale ricorsiva sempre indefinita $\lambda x.\uparrow$ ha il minimo di informazione, poichè ad ogni possibile input non è associato nessun output, ovvero ha grafico vuoto.

Consideriamo l'insieme $P\omega$ ordinato secondo la relazione binaria di inclusione insiemistica, ovvero la struttura parzialmente ordinata $(P\omega, \subseteq)$. L'interpretazione di questo ordinamento passando da insiemi a funzioni (ovvero insiemi univoci) corrisponde all'idea intuitiva secondo la quale ad una funzione ϕ è associata più informazione di una funzione ψ quando il grafico di ψ è contenuto in quello di ϕ. Petanto, definiamo l'ordinamento sulle funzioni derivante dalla interpretazione di $(\mathcal{U}, \subseteq)$:

$$\psi \sqsubseteq \phi \text{ sse } \hat{\psi} \subseteq \hat{\phi}.$$

ESEMPIO 22.5. È evidente che $\emptyset$ è un insieme univoco, ovvero è il grafico della funzione $\lambda x.\uparrow$. È banale dimostrare che $\emptyset$ è il minimo elemento in $(\mathcal{U}, \subseteq)$, ovvero per ogni funzione parziale ψ: $\lambda x.\uparrow \sqsubseteq \psi$.

L'insieme parzialmente ordinato $(\mathcal{U}, \subseteq)$ ha una importante proprietà di *completezza*, ovvero ogni catena (anche infinita) di insiemi univoci ha un estremo superiore anch'esso univoco. Un insieme parzialmente ordinato con questa caratteristica è detto CPO (complete partial order).

TEOREMA 22.6. *$(\mathcal{U}, \subseteq)$ è un CPO.*

PROOF. Sia $\{I_n\}_{n\in\mathbb{N}}$ una catena di insiemi univoci, ovvero tali che per ogni $n \in \mathbb{N}$: $I_n \in \mathcal{U}$ e $I_n \subseteq I_{n+1}$. A questa catena corrisponde una catena di funzioni parziali ordinate tra loro mediante $\sqsubseteq$ ed il cui grafico sono gli elementi di $\{I_n\}_{n\in\mathbb{N}}$.

$$\begin{array}{ccccccc} I_1 & \subseteq & I_2 & \subseteq & I_3 & \subseteq & \ldots \\ \downarrow & & \downarrow & & \downarrow & & \ldots \\ \psi_1 & \sqsubseteq & \psi_2 & \sqsubseteq & \psi_3 & \sqsubseteq & \ldots \end{array}$$

Consideriamo l'insieme $I = \bigcup_{i\in\mathbb{N}} I_i$. Dimsotriamo che I è univoco: Supponiamo che $\langle n, m\rangle, \langle n, m'\rangle \in I$. Allora, per definizione, esistono die insiemi I_{i_1} e I_{i_2} tali che $\langle n, m\rangle \in I_{i_1}$ e $\langle n, m'\rangle \in I_{i_2}$. Sia $j = max(i_1, i_2)$. Poichè $\{I_n\}_{n\in\mathbb{N}}$ è una catena, $I_{i_1} \subseteq I_j$ e $I_{i_2} \subseteq I_j$. Se fosse $m \neq m'$ avremmo dunque un assurdo poichè un elemento I_j della catena non sarebbe univoco, quindi $m = m'$ ovvero I è univoco. □

2.2. Continuità. Abbiamo visto come sia possibile dare una struttura di ordinamento alle funzioni parziali sui naturali in base al loro grado di definizione e che tale struttura, che corrisponde alla classe degli insiemi univoci ordinati per inclusione insiemistica, è un CPO con un elemento minimo corrispondente alla funzione sempre indefinita. In questa sezione analizzeremo una nozione di *continuità* per funzionali. Essa ci permetterà di definire un metodo costruttivo per caratterizzare le soluzioni alle equazioni ricorsive $\tau(X) = X$ con τ trasformazione effettiva di programmi. Il concetto di continuità che analizzeremo ora, introdotto da D. Scott negli anni '70, è alla base di tutti i modelli matematici per la semantica dei linguaggi di programmazione.

Intuitivamente, una funzione è continua se essa preserva i *limiti*. Per definire formalmente questo concetto è necessario avere una nozione ben definita di limite. Una definizione formale di questo tipo non può prescindere da una analisi precisa di una topologia sulle funzioni parziali. Questo chiaramente esula dagli scopi del corso, pertanto ci limiteremo a dare una idea solo intuitiva del concetto di continuità di Scott sullo spazio delle funzioni parziali (o insiemi univoci). Consideriamo l'insieme $\mathcal{U}$. Sappiamo che esso è un CPO, ovvero ogni catena

di funzioni in esso rappresentate (mediante grafico) ha un estremo superiore corrispondente alla funzione che ha come grafico l'unione dei grafici presenti nella catena. Analogamente, $P\omega$ è un CPO poichè ogni catena di insiemi ha un estremo superiore corrispondente all'unione degli insiemi della catena. Assumiamo pertanto che il *limite* della catena $\{I_i\}_{i\in\mathbb{N}}$ sia $\bigcup_{i\in\mathbb{N}} I_i$. Tali punti esistono per completezza sia in $\mathcal{U}$ che ovviamente in $P\omega$.

DEFINIZIONE 22.7. Una funzione $\phi : P\omega \to P\omega$ è detta continua se per ogni $S \in P\omega$:

$$\phi(S) = \bigcup_{F\subseteq_f S} \phi(F).$$

Per comprendere a fondo questa definizione, è importante osservare che ogni insieme $S \in P\omega$ può essere visto come il limite di una catena di insiemi finiti, ovvero esiste una catena $\{I_i\}_{n\in\mathbb{N}}$ tale che per ogni $i \in \mathbb{N}$: $|I_i| < \omega$, $I_i \subseteq_f S$ e $\bigcup_{i\in\mathbb{N}} I_i = S$. Questa proprietà degli insiemi ci permette di affermare che una funzione è continua quando essa preserva il limite di ogni catena (anche infinita) di insiemi finiti.

ESEMPIO 22.8. La seguente funzione non è continua:

$$\psi(X) = \begin{cases} \emptyset & \text{se } X \text{ è finito} \\ \mathbb{N} & \text{altrimenti} \end{cases}$$

infatti,

$$\psi(\mathbb{N}) = \mathbb{N} \neq \emptyset = \bigcup_{F\subseteq_f \mathbb{N}} \psi(F).$$

TEOREMA 22.9. *Se $\phi : P\omega \to P\omega$ è continua ψ è monotona, ovvero $A \subseteq B \Rightarrow \psi(A) \subseteq \psi(B)$.*

PROOF. Siano $A \subseteq B$ con $A, B \in P\omega$. Poichè $A \subseteq B$, $\{F\}_{F\subseteq_f A} \subseteq \{F\}_{F\subseteq_f B}$. Pertanto, per continuità:

$$\begin{aligned} \psi(A) \cup \psi(B) &= \textstyle\bigcup_{F\subseteq_f A} \psi(F) \cup \bigcup_{F\subseteq_f B} \psi(F) \\ &= \textstyle\bigcup_{F\subseteq_f B} \psi(F) \\ &= \psi(B). \end{aligned}$$

È chiaro che $\psi(A) \cup \psi(B) = \psi(B)$ sse $\psi(A) \subseteq \psi(B)$.

□

2.3. Costruzione di funzionali ricorsivi. Consideriamo insiemi univoci, ovvero grafici di funzioni. Fino a questo punto non abbiamo dato nessuna ipotesi sulla effettività delle funzioni in $(\mathcal{U}, \subseteq)$. In questa sezione introdurremo un metodo per definire funzioni su $\mathcal{U}$, ovvero funzionali.

DEFINIZIONE 22.10. Sia $\{F_i\}_{i\in\mathbb{N}}$ una fissata numerazione degli insiemi finiti di $P\omega$. La funzione $\cdot : \mathcal{U} \times P\omega \to P\omega$ è detta applicazione se per ogni $A \in \mathcal{U}$ e $B \in P\omega$:

$$A \cdot B = \{\, m \mid \exists F_n \subseteq B.\, \langle n, m\rangle \in A \,\}$$

L'idea intuitiva di applicazione di un insieme univoco A ad un dato insieme B è quella che $A \cdot B$ tappresenta l'insieme dei riusltati ottenibili applicando la funzione parziale il cui grafico è A a porzioni finite dell'input B, secondo una data copertura mediante insiemi finiti di B.

Siamo ora in grado di definire matematicamente una classe significativa di trasformazioni effettive sui programmi.

DEFINIZIONE 22.11. $\psi : P\omega \to P\omega$ è un *operatore di enumerazione* se esiste $A \in RE \cap \mathcal{U}$ tale che per ogni $X \in P\omega$: $\psi(X) = A \cdot X$.

Un operatore di enumerazione corrisponde dunque all'applicazione di un insieme r.e. che deve essere grafico di una funzione.

LEMMA 22.12. *Esiste una funzione ricorsiva (totale) h tale che per ogni $i, j \in \mathbb{N}$ t.c. $W_i \in \mathcal{U}$: $W_i \cdot W_j = W_{h(i,j)}$.*

PROOF. Per esercizio. □

Gli operatori di enumerazione hanno importanti proprietà, tra le quali la continuità che come vedremo giocherà un ruolo essenziale per definire un metodo costruttivo che caratterizzi le soluzioni all'equazione di punto fisso $\tau(X) = X$, e quindi una versione costruttiva del Teorema di Kleene.

LEMMA 22.13.

1. *Se ϕ e ϕ' sono osperatori di enumerazione, anche $\phi \circ \phi'$ è un operatore di enumerazione;*

2. *Se ϕ è un operatore di enumerazione e $B \in RE$, anche $\phi(B) \in RE$;*
3. *Se ϕ è un operatore di enumerazione, ϕ è continuo.*

PROOF.

1. Siano A e A' insiemi r.e. corrispondenti rispettivamente a ϕ e ϕ'. Al solito fissiamo una numerazione degli insiemi finiti di naturali: $\{D_i\}_{i\in\mathbb{N}}$. Definiamo:

$$C \stackrel{\text{def}}{=} \{\, \langle n, m\rangle \mid m \in A \cdot (A' \cdot D_n) \wedge n \in \mathbb{N} \,\}.$$

Si dimostri per esercizio che C è r.e. . Si osservi che:

$$\begin{aligned}
& C \cdot B \\
= \;& \{\, m \mid \exists D_q \subseteq_f B.\, m \in A \cdot (A' \cdot D_q) \,\} \\
= \;& \{\, m \mid \exists D_q \subseteq_f B, D_n \subseteq_f A' \cdot D_q.\, \langle n, m\rangle \in A \,\} \\
= \;& \{\, m \mid \exists n, D_q \subseteq_f B.\, \forall d \in D_n.\, \exists D_t \subseteq_f D_q.\, \langle t, d\rangle \in A' \wedge \langle n, m\rangle \in A \,\} \\
= \;& \{\, m \mid \exists n, D_q \subseteq_f B.\, \forall d \in D_n.\, \langle q, d\rangle \in A' \wedge \langle n, m\rangle \in A \,\} \\
= \;& \{\, m \mid \exists D_n \subseteq_f A' \cdot B.\, \langle n, m\rangle \in A \,\} \\
= \;& A \cdot (A' \cdot B).
\end{aligned}$$

Basta dunque considerare l'insieme r.e. C come definente l'operatore di enumerazione $\phi \circ \phi'$.

2. Dimostrare per esercizio.
3. È immediato dimostrare le seguenti equivalenze:

$$\begin{aligned}
m \in \textstyle\bigcup_{D_n \subseteq_f B} \phi(D_n) \;& \Leftrightarrow\; m \in \textstyle\bigcup_{D_n \subseteq_f B} A \cdot D_n \\
& \Leftrightarrow\; \exists D_n \subseteq_f B.\, m \in A \cdot D_n \\
& \Leftrightarrow\; \exists D_n \subseteq_f B, D_p \subseteq_f D_n.\, \langle p, m\rangle \in A \\
& \Leftrightarrow\; \exists D_n \subseteq_f B.\, \langle n, m\rangle \in A \\
& \Leftrightarrow\; m \in A \cdot B \\
& \Leftrightarrow\; m \in \phi(B).
\end{aligned}$$

□

Nel seguito utilizzeremo lettere greche maiuscole per denotare funzionali.

DEFINIZIONE 22.14. Una funzione $\Phi : P\omega \to P\omega$ è un funzionale se per ogni $X \in \mathcal{U}$: $\Phi(X) \in \mathcal{U}$. Φ è detto funzionale ricorsivo se è un funzionale ed è un operatore di enumerazione.

Il seguente risultato caratterizza una soluzione *minima* rispetto alla quantità di informazione in essa contenuta per ogni equazione $\Phi(X) = X$ con Φ funzionale ricorsivo. Ricordiamo che un insieme X tale che $\Phi(X) = X$ è detto *punto fisso* di Φ. All'interno della classe di tutti i punti fissi, se questa non è vuota, determineremo in modo costruttivo, ovvero come limite di approssimazioni successive finite, il *minimo punto fisso* di Φ.

TEOREMA 22.15. *Sia Φ un funzionale ricorsivo. Esiste una funzione parzaile φ tale che*

1. $\Phi(\hat{\varphi}) = \hat{\varphi}$;
2. *φ è parziale ricorsiva;*
3. *Se esiste ψ tale che $\Phi(\hat{\psi}) = \hat{\psi}$ allora $\varphi \sqsubseteq \psi$.*

PROOF. Sia $\Phi = \lambda X.\ A \cdot X$ un funzionale ricorsivo con A r.e. univoco.

1. Definiamo una catena di insiemi unicoci, ovvero una catena in $\mathcal{U}$, tale che il suo limite sia un punto fisso di Φ. Consideriamo la successione di insiemi $\{\Phi^i(\emptyset)\}_{i\in\mathbb{N}}$, dove poniamo $\Phi^0(\emptyset) = \emptyset$. Si dimostra per induzione che $\{\Phi^i(\emptyset)\}_{i\in\mathbb{N}}$ è una catena.
   - $\emptyset \subseteq A \cdot \emptyset = \Phi(\emptyset)$ poichè $\emptyset$ è contenuto in ogni insieme.
   - Per il Lemma 22.13, Φ è continuo e quindi monotono. È pertanto immediato osservare che $\Phi^{i+1}(\emptyset) = \Phi(\Phi^i(\emptyset)) \subseteq \Phi(\Phi^{i+1}(\emptyset)) = \Phi^{i+2}(\emptyset)$.

   Poichè $(\mathcal{U}, \subseteq)$ è un CPO, $\bigcup_{i\in\mathbb{N}} \Phi^i(\emptyset)$ è univoco. Pertanto esiste una funzione parziale φ tale che $\hat{\varphi} = \bigcup_{i\in\mathbb{N}} \Phi^i(\emptyset)$. È immediato

osservare che per monotonia di Φ, $\hat{\varphi}$ è un punto fisso di Φ:

$$\begin{aligned}\Phi(\hat{\varphi}) &= \Phi(\bigcup_{i\in\mathbb{N}} \Phi^i(\emptyset)) \\ &= \bigcup_{i\in\mathbb{N}} \Phi(\Phi^i(\emptyset)) \\ &= \bigcup_{i\in\mathbb{N}} \Phi^{i+1}(\emptyset) \\ &= \bigcup_{i\in\mathbb{N}} \Phi^i(\emptyset) \\ &= \hat{\varphi}.\end{aligned}$$

2. Dimostriamo che φ ottenuta nel punto precedente è una funzione parziale ricorsiva. Per quanto osservato nell' Esercizio 22.4, φ è parziale ricorsiva sse $\bigcup_{i\in\mathbb{N}} \Phi^i(\emptyset)$ è r.e. Per induzione dimostriamo che per ogni $i \in \mathbb{N}$: $\Phi^i(\emptyset)$ è r.e. .
   - $\Phi^0(\emptyset) = \emptyset$ è ovviamente r.e.
   - Supponiamo che $\Phi^i(\emptyset)$ sia r.e. . Poichè $\Phi^{i+1}(\emptyset) = \Phi(\Phi^i(\emptyset)) = A \cdot \Phi^i(\emptyset)$, e A è r.e. , segue che, per il Lemma 22.12, anche $\Phi^{i+1}(\emptyset)$ è r.e. .

   Poichè l'unione anche infinita di insiemi r.e. è r.e. , abbiamo che φ è parziale ricorsiva.
3. Sia ψ un qualsiasi punto fisso di Φ, ovvero $\Phi(\hat{\psi}) = \hat{\psi}$. Dimostriamo per induzione che per ogni $i \in \mathbb{N}$: $\Phi^i(\emptyset) \subseteq \hat{\psi}$.
   - È ovvio che $\emptyset \subseteq \hat{\psi}$.
   - Assumiamo che $\Phi^i(\emptyset) \subseteq \hat{\psi}$. Poichè ψ è punto fisso di Φ e Φ è monotona in quanto continua, abbiamo che: $\Phi^{i+1}(\emptyset) = \Phi(\Phi^i(\emptyset)) \subseteq \Phi(\hat{\psi}) = \hat{\psi}$.

   Pertanto: $\bigcup_{i\in\mathbb{N}} \Phi^i(\emptyset) \subseteq \hat{\psi}$. Questo dimostra che ogni altro punto fisso è comunque più grande nell'ordinamento, ovvero che φ è il minimo punto fisso di Φ.

□

Secondo questo risultato, il minimo punto fisso del funzionale ricorsivo Φ è ottenuto come limite di una successione di insiemi rappresentanti il grafico di funzioni sempre più definite, partendo dall'insieme vuoto $\emptyset$ rappresentante la funzione sempre indefinita $\lambda x.\uparrow$. Il limite di questo proceso è una funzione che è il minimo punto fisso del funzionale. Il teorema precedente applica, nella costruzione del minimo punto fisso, una tecnica ampiamente utilizzata nello studio della

semantica formale dei linguaggi di programmazione. In particolare il punto 1 e 3 del teorema sono istanze di un risultato più generale sui punti fissi di funzioni monotone su insiemi parzialmente ordinati, risultato dovuto a Tarski.

Concludiamo questa parte con un importante risultato che caratterizza la classe delle trasformazioni effettive sui programmi in termini di funzionali ricorsivi. Mediante questo risultato, dovuto a Myhill e Shepherdson, è posibile dare una caratterizzazione costruttiva al Teorema di ricorsione di Kleene.

TEOREMA 22.16 (Myhill-Shepherdson). *$\Phi : P\omega \to P\omega$ è un funzionale ricorsivo sse esiste una funzione ricorsiva (totale) $g : \mathbb{N} \to \mathbb{N}$ estensionale tale che per ogni $i \in \mathbb{N}$: $\Phi(\hat{\varphi}_i) = \hat{\varphi_{g(i)}}$.*

PROOF. ($\Rightarrow$). Sia Φ un funzionale ricorsivo univocamente determinato da un insieme r.e. W_z. Sia φ_i una funzione parziale ricorsiva. Allora esiste una funzione ricorsiva t tale che $\hat{\varphi}_i = W_{t(i)}$, essendo $\hat{\varphi}_i$ r.e. . Pertanto, $\Phi(\hat{\varphi}_i) = \Phi(W_{t(i)})$. Per il Lemma 22.12, esiste una funzione ricorsiva h tale che $W_z \cdot W_{t(i)} = W_{h(z,t(i))}$ e $W_{h(z,t(i))} \in \mathcal{U}$ poichè per ipotesi Φ è un funzionale ricorsivo. Pertanto esiste una funzione ricorsiva f tale che $W_{h(z,t(i))} = \varphi_{f(\hat{h(z,t(i))})}$. Ponendo $g(i) = f(h(z, t(i)))$, abbiamo la tesi, in quanto g così definita è estensionale: se $\varphi_i = \varphi_j$ allora $\hat{\varphi}_i = \hat{\varphi}_j$, da cui $W_z \cdot \hat{\varphi}_i = W_z \cdot \hat{\varphi}_j$. Da questo segue che $W_{h(z,t(i))} = W_{h(z,t(j))}$. Essendo $W_{h(z,t(i))}$ e $W_{h(z,t(j))}$ univoci, per quanto osservato in precedenza: $\varphi_{f(\hat{h(z,t(i))})} = \varphi_{f(\hat{h(z,t(j))})}$, da cui segue che $\varphi_{g(i)} = \varphi_{g(j)}$.
($\Leftarrow$). . . . □

Per il Teorema 22.16 abbiamo dunque che, se $g : \mathbb{N} \to \mathbb{N}$ è ricorsiva estensionale, allora una funzione di indice n_0 tale che $\varphi_{g(n_0)} = \varphi_{n_0}$ è sempre ottenibile come minimo punto fisso (secondo il Teorema 22.15) del funzionale ricorsivo corrispondente Φ tale che $\Phi(\hat{\varphi}_i) = \varphi_{g(i)}$. Inoltre tale funzione è la meno definita tra tutte le funzioni soluzione della medesima equazione ricorsiva.

ESEMPIO 22.17. Consideriamo il funzionale descritto all'inizio del capitolo:

$$\Sigma(\varphi) = \lambda x. \begin{cases} 0 & \text{se } x = 0 \\ 1 & \text{se } x = 1 \\ \varphi(x-1) + x & \text{altrimenti} \end{cases}$$

È evidente che si tratta di un funzionale ricorsivo. Per verificare questo, basti osservare che, data una qualsiasi MdT φ_i, è possibile costruire una MdT che calcola $\Sigma(\varphi_i)$. Inoltre questa MdT dipende in modo effettivo da φ_i, ovvero $\Sigma(\varphi_i) = \varphi_{g(i)}$, con g ricorsiva. Inoltre, tale g non può che essere estensionale, in quanto se $\varphi_i = \varphi_j$ allora $\Sigma(\varphi_i) = \Sigma(\varphi_i)$. Il fatto che Σ corrisponda ad un funzionale ricorsivo segue dunque per il Teorema di Myhill-Shepherdson.

È facile a questo punto trovare la funzione parziale ricorsiva punto fisso di Σ. Intuitivamente questa corrisponde alla funzione ottenibile applicando Σ ripetutamente alla funzione sempre indefinita $\lambda x.\uparrow$. Ad essa corrisponderà una catena di insiemi univoci $\{\Sigma^i(\emptyset)\}_{i\in\mathbb{N}}$ che per semplicità non calcoliamo in questa sede.

step	**cetena**	**funzione**
0.	$\Sigma^0(\emptyset) = \emptyset$	$\lambda x.\uparrow$
1.	$\Sigma^1(\emptyset) = \Sigma(\emptyset)$	$\lambda x.\ \begin{cases} 0 & \text{se } x = 0 \\ 1 & \text{se } x = 1 \\ \uparrow & \text{altrimenti} \end{cases}$
2.	$\Sigma^2(\emptyset) = \Sigma(\Sigma(\emptyset))$	$\lambda x.\ \begin{cases} 0 & \text{se } x = 0 \\ 1 & \text{se } x = 1 \\ 3 & \text{se } x = 2 \\ \uparrow & \text{altrimenti} \end{cases}$
3.	$\Sigma^3(\emptyset) = \Sigma(\Sigma(\Sigma(\emptyset)))$	$\lambda x.\ \begin{cases} 0 & \text{se } x = 0 \\ 1 & \text{se } x = 1 \\ 3 & \text{se } x = 2 \\ 6 & \text{se } x = 3 \\ \uparrow & \text{altrimenti} \end{cases}$
...	...	...

È facile osservare che il minimo punto fisso di Σ è la funzione ricorsiva di sommatoria $\lambda x.\ \sum_{i=1}^{x} i$.

CHAPTER 23

Riducibilità funzionale

L'inclusione insiemistica, intesa come relazione tra insiemi, non da nessuna informazione circa la ricorsività di un insieme. In particolare, il fatto che un insieme sia contenuto in un dato insieme ricorsivo non implica in alcun modo la ricorsività del primo: la ricorsività di $\mathbb{N}$ non implica la ricorsività di ogni suo sottoinsieme (e.g. $K \subset \mathbb{N}$). Anzi, come si è visto, la maggior parte dei sottoinsiemi di $\mathbb{N}$ non è nemmeno r.e. In questa sezione studieremo il concetto di *riducibilità funzionale*. Questo concetto permette di correlare tra loro insiemi in modo tale che la ricorsività (o la non ricorsività) di un insieme possa essere propagata agli insiemi ad esso correlati mediante riduzione funzionale.

Consideriamo l'insieme $K_2 = \{ \langle x, y \rangle \mid y \in W_x \}$. K_2 ha una struttura molto simile a K, che sappiamo essere un insieme non ricorsivo. Ci chiediamo se anche K_2 è non ricorsivo. Si osservi che K_2 rappresenta una generalizzazione di K: $x \in K$ sse $x \in W_x$ sse $\langle x, x \rangle \in K_2$. Chiaramente K_2 è r.e. (mostrare la funzione semicaratteristica). Pertanto, se volessimo dimostrare che K_2 è ricorsivo, dovremmo dimostrare che $\bar{K}_2$ è anch' esso r.e. Chiaramente $x \in \bar{K}$ sse $\langle x, x \rangle \in \bar{K}_2$. Si osservi che la funzione $\lambda x.\ \langle x, x \rangle$, che "traduce" l'appartenenza di x a $\bar{K}$ in quella di $\langle x, x \rangle$ a $\bar{K}_2$ è totale ricorsiva. Pertanto, esiste una funzione totale ricorsiva f tale che

$$x \in \bar{K} \ \Leftrightarrow \ f(x) \in \bar{K}_2$$

È evidente che, se fosse K_2 ricorsivo, essendo f ricorsiva, avremmmo un modo per "decidere" se $x \in \bar{K}$, il che è assurdo.

Quello che abbiamo visto è un metodo per *ridurre* la decidibilità del predicato di appartenenza ad un insieme alla decidibilità del medesimo predicato ad un altro insieme. Questo esempio conduce alla seguente definizione.

DEFINIZIONE 23.1. Sia $A, B \subseteq \mathbb{N}$. A è (funzionalmente) riducibile a B, denotato con $A \preceq B$ se esiste una funzione (totale) ricorsiva f tale che

$$x \in A \Leftrightarrow f(x) \in B$$

Nel caso $A \preceq B$ con funzione ricorsiva f, diremo che A si riduce a B via f.

ESEMPIO 23.2. Da quanto abbiamo visto in precedenza, $K \preceq K_2$ via $\lambda x.\, \langle x, x \rangle$.

La relazione $\preceq \subseteq \wp(\mathbb{N}) \times \wp(\mathbb{N})$ è simmetrica e transitiva (dimostrare per esercizio). Pertanto, possiamo definire una relazione di equivalenta $\equiv$ tra insiemi di numeri, in modo tale che $A \equiv B$ sse $A \preceq B$ e $B \preceq A$. Si dimostri per esercizio che $\equiv$ è una relazione di equivalenza su $\wp(\mathbb{N})$. Denoteremo con $[A]_{\equiv}$ la classe di equivalenza di A, ovvero l'insieme $\{ B \mid A \equiv B \}$.

TEOREMA 23.3. *Per ogni $A \in RE$, $A \preceq K_2$.*

PROOF. Sia $A \in RE$. Allora esiste y_0 tale che $A = W_{y_0}$. $x \in A = W_{y_0}$ sse $\langle y_0, x \rangle \in K_2$. Definiamo dunque $f'_{y_0} = \lambda x.\, \langle y_0, x \rangle$. È chiaro che $x \in A$ sse $f'_{y_0} \in K_2$. □

Pertanto, ogni insieme r.e. è riducibile a K_2. Questo significa che K_2 ha un ruolo speciale all'interno della classe degli insiemi r.e. : ogni elemento di questa classe è riducibile a K_2.

DEFINIZIONE 23.4. $C \in RE$ è completo se per ogni $A \in RE$: $A \preceq C$.

K_2 è chiaramente completo. Il seguente teorema dimostra che anche K è completo. Pertanto, $K \equiv K_2$, ed entrambi giocano un ruolo speciale (massimale) all'interno della classe degli insiemi r.e. È inoltre banale osservare che A è completo sse $A \in [K]_{\equiv}$.

TEOREMA 23.5. *K è completo.*

PROOF. Basta dimostrare che $K_2 \preceq K$. Infatti, in questo modo, poichè ogni insieme r.e. è riducibile a K_2 (vedi Teorema 23.3), allora per transitività, ogni insieme r.e. è anche riducibile a K, dimostrando la completezza di K.

Sia $\langle x, y\rangle \in K_2$. Per definizione, ciò è vero sse $y \in W_x$. Definiamo la funzione

$$\psi(x,y,z) = \begin{cases} 1 & \text{se } \varphi_x(y)\downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

È chiaro che ψ è parziale ricorsiva, quindi, per il teorema s-m-n, esiste una funzione ricorsiva (totale) g tale che: $\psi(x,y,z) = \varphi_{g(x,y)}(z)$. Si noti come, fissato $x, y \in \mathbb{N}$, la funzione $\lambda z.\ \varphi_{g(x,y)}(z)$ sia costante, ovvero indipendente dal valore dell'argomento z. Si osserva che $\langle x, y\rangle \in K_2$ sse $\varphi_{g(x,y)}(z)\downarrow$ per $z \in \mathbb{N}$ qualsiasi. Quindi, a maggior ragione $\langle x, y\rangle \in K_2$ sse $\varphi_{g(x,y)}(g(x,y))\downarrow$. Questo dimostra che $\langle x, y\rangle \in K_2$ sse $g(x,y)K\downarrow$, con g funzione ricorsiva. □

ESEMPIO 23.6. Dimostriamo che $K \preceq \{\, x \mid |W_x| < \omega \,\}$. Definiamo la funzione:

$$\psi(x,y) = \begin{cases} 1 & \text{se } \varphi_x(x) \text{ non converge in meno di } y \text{ passi} \\ \uparrow & \text{altrimenti} \end{cases}$$

Questa funzione è parziale ricorsiva (si descriva a parole una possibile MdT che calcola ψ). Per il teorema s-m-n, esiste g ricorsiva (totale) tale che $\psi(x,y) = \varphi_{g(x)}(y)$. Sia $x \in K$. Allora, per definizione di K: $\varphi_x(x)\downarrow$. $\varphi_x(x)\downarrow$ sse esiste $n \in \mathbb{N}$, $n \geq 1$, tale che $\varphi_x(x)$ converge in n passi. Per definizione, $\varphi_x(x)$ converge in n passi sse $dom(\psi) = W_{g(x)} = \{0, \ldots, n-1\}$, ovvero $g(x) \in \{\, x \mid |W_x| < \omega \,\}$. Pertanto $x \in K$ sse $g(x) \in \{\, x \mid |W_x| < \omega \,\}$, da cui la tesi.

Vediamo alcune proprietà fondamentali della nozione di riduzione funzionale.

TEOREMA 23.7. *Sia* $A, B \subseteq \mathbb{N}$.

1. $A \preceq B \ \Rightarrow\ \bar{A} \preceq \bar{B}$;
2. $A \preceq B$ *e* $B \in RE \ \Rightarrow\ A \in RE$.
3. $A \preceq B$ *e* B *ricorsivo* $\Rightarrow$ A *ricorsivo.*

PROOF. Il primo punto è banale. Dimostriamo che se $A \preceq B$ e $B \in RE$ allora $A \in RE$. Sia ψ_B la funzione semicaratteristica

parziale ricorsiva di B e supponiamo che $A \preceq B$ via f. Consideriamo la funzione parziale ricorsiva $\psi_A = \lambda x.\ \psi_B(f(x))$. Allora abbiamo che

$$\psi_A = \begin{cases} 1 & \text{se } f(x) \in B \\ \uparrow & \text{se } f(x) \notin B \end{cases}$$

Poichè $x \in A$ sse $f(x) \in B$, ψ_A è la funzione semicaratteristica di A, e quindi $A \in RE$. Il terzo punto ha una dimostrazione del tutto analoga al secondo punto, considerando la funzione caratteristica di B. □

Abbiamo quindi dimostrato che la proprietà di *essere ricorsivo* o *ricorsivamente enumerabile* si propaga verso il basso secondo l'ordinamento $\preceq$ tra insiemi.

ESERCIZIO 23.8. Dimostrare che:

- $A \in RE$, $A \neq \emptyset$ e $A \neq \mathbb{N}$:

$$A \text{ ricorsivo } \Leftrightarrow\ A \preceq \bar{A}$$

- Se A, B sono ricorsivi non banali, allora $A \equiv B$.
- $A \in [B]_{\equiv}$ e A ricorsivo allora per ogni $X \in [B]_{\equiv}$, X è ricorsivo.

1. Insiemi creativi e produttivi

Abbiamo visto che all'interno della classe degli insiemi r.e. , esistono insiemi (e.g. K) che hanno la proprietà per cui ogni altro insieme r.e. è ad essi riducibile. Questi insiemi, detti completi, rappresentano quindi la classe degli insiemi r.e. , quasi fossero dei rappresentanti "canonici" di questa classe.

In questa sezione caratterizzeremo gli insiemi completi. Caratterizzare gli insiemi completi significa dare una condizione che determini in modo univoco se un dato insieme è o meno completo, senza dovere ricorrere ad altri insiemi r.e.

DEFINIZIONE 23.9. Un insieme $A \subseteq \mathbb{N}$ è detto produttivo se esiste una funzione (totale) ricorsiva f_A, detta funzione produttiva di A, tale che:

$$\forall x \in \mathbb{N}.\ (W_x \subseteq A\ \Rightarrow f_A(x) \in A - W_x)$$

Un insieme è detto creativo se esso è r.e. ed il suo complemento è produttivo, ovvero se:

$$A \in RE \text{ e } \forall x \in \mathbb{N}.\ (W_x \subseteq \bar{A} \ \Rightarrow f_A(x) \in \bar{A} - W_x)$$

Intuitivamente un insieme è produttivo se per ogni tentativo di enumerarlo in modo effettivo mediante un algoritmo di indice x (ovvero per ogni $W_x \subseteq A$), esiste una trasformazione effettiva di x in un punto di A che sfugge all'enumerazione. È evidente che un insieme produttivo non può essere r.e. , altrimenti avremmo l'assurdo che $A = W_{x_0}$ per un certo x_0 e al tempo stesso $A - W_{x_0} \neq \emptyset$, poichè $f(x_0) \in A - W_{x_0}$. Al contrario, un insieme creativo è sempre, per definizione, r.e.

TEOREMA 23.10. *Sia* $A \subseteq \mathbb{N}$.

1. *A produttivo* $\Rightarrow$ *A non r.e.*
2. *A creativo* $\Rightarrow$ *A non ricorsivo*

PROOF. Immediato per definizione. □

L'insieme $K = \{\, x \mid x \in W_x \,\}$ ancora una volta è rappresentativo per la classe di insiemi creativi, come dimostrato nel seguente teorema.

TEOREMA 23.11. *K è creativo e $\bar{K}$ è produttivo.*

PROOF. Dimostriamo che $\bar{K}$ è produttivo. Consideriamo la funzione ricorsiva di base $id = \lambda x.\ x$. Osserviamo che, se $W_x \subseteq \bar{K}$ allora $id(x) = x \notin W_x$. Infatti, se $x \in W_x$, allora per definizione di K: $x \in K$. Poichè $W_x \subseteq \bar{K}$, allora $x \in \bar{K}$ che sarebbe assurdo. Quindi $id(x) = x \notin W_x$. Inoltre, se $x \in K$ allora avremmo $x \in W_x \subseteq \bar{K}$, ovvero $x \in \bar{K}$, che è assurdo poichè x non può essere al tempo stesso in K e $\bar{K} = \mathbb{N} - K$. Quindi se $W_x \subseteq \bar{K}$ allora $id(x) = x \in \bar{K} - W_x$, ovvero $\bar{K}$ è produttivo. La dimostrazione che K è creativo è immediata poichè $K \in RE$ e $\bar{K}$ è produttivo. □

Il seguente teorema dimostra una proprietà fondamentale degli insiemi creativi e produttivi, ovvero che la produttività e la creatività di un insieme viene ereditata per riduzione funzionale.

TEOREMA 23.12. *Siano* $A, B \subseteq \mathbb{N}$.

1. *A produttivo e* $A \preceq B$ $\Rightarrow$ *B produttivo*
2. *A creativo e* $A \preceq B$ $\Rightarrow$ $(B \in RE \ \Rightarrow$ *B creativo*$)$

PROOF. Dimostriamo il punto (2) e lasciamo il punto (1) per esercizio. La dimostrazione è del tutto analoga al caso creativo.

(2). Sia A creativo e $B \in RE$ tale che $A \preceq B$ via f. Dato $x \in \mathbb{N}$, consideriamo l'insieme r.e. : $f^{-1}(W_x) = \{ y \mid f(y) \in W_x \}$. Essendo r.e. , esiste una funzione parziale ricorsiva $\psi(x, y)$ tale che

$$\psi(x,y) = \begin{cases} 1 & \text{se } f(y) \in W_x \\ \uparrow & \text{altrimenti} \end{cases}$$

Per il teorema s-m-n, esiste una funzione totale ricorsiva g tale che: $\psi(x, y) = \varphi_{g(x)}(y)$. Inoltre, per definizione di ψ, fissato $x \in \mathbb{N}$: $dom(\varphi_{g(x)}) = W_{g(x)} = f^{-1}(W_x)$.
Dimostriamo ora che B è creativo. Per fare ciò basta dimostrare che $\bar{B}$ è produttivo. Sia $W_x \subseteq \bar{B}$. Poichè $A \preceq B$ via f, $f^{-1}(\bar{B}) = \bar{A}$. Quindi, $W_{g(x)} = f^{-1}(W_x) \subseteq f^{-1}(\bar{B}) = \bar{A}$, ovvero $W_{g(x)} \subseteq \bar{A}$. Essendo A creativo, esiste h ricorsiva (totale) tale che $h(g(x)) \in \bar{A} - W_{g(x)}$. Poichè $A \preceq B$ via f, $h(g(x)) \in \bar{A} - W_{g(x)}$ implica che $f(h(g(x))) \in \bar{B}$. Inoltre, $f(h(g(x))) \notin W_x$, altrimenti, se $f(h(g(x))) \in W_x$ allora avremmo che $h(g(x)) \in W_{g(x)}$ che sarebbe assurdo, poichè abbiamo appena dimostrato che $h(g(x)) \in \bar{A} - W_{g(x)}$. Quindi, $f(h(g(x))) \in \bar{B} - W_x$. La tesi segue dal fatto che la composizione di funzioni ricorsive è ricorsiva, e quindi $f \circ h \circ g$ è una funzione ricorsiva e produttiva per $\bar{B}$, da cui consegue la creatività di B. □

Il seguente teorema, che chiude la parte sulla ricorsività, caratterizza gli insiemi completi come tutti e soli gli insiemi creativi, ovvero aventi un complemento produttivo. Gli insiemi completi hanno dunque complementi non r.e. !

TEOREMA 23.13. *Sia $A \in RE$.*

$$A \text{ completo} \Leftrightarrow A \text{ creativo}$$

PROOF. ($\Rightarrow$). Sia $A \in RE$. A è completo sse per ogni $X \in RE$: $X \preceq A$. Poichè K è creativo e A completo, $K \preceq A$. Per il Teorema 23.12, anche A è creativo.

($\Leftarrow$). Sia A creativo con funzione f. Sia $B \in RE$. Definiamo la funzione:

$$\psi(x,y,z) = \begin{cases} 0 & \text{se } y \in B \text{ e } z = f(x) \\ \uparrow & \text{altrimenti} \end{cases}$$

Per il teorema s-m-n, esiste una funzione totale $\hat{g}$ tale che $\psi(x,y,z) = \varphi_{\hat{g}(x,y)}(z)$. Quindi:

$$W_{\hat{g}(x,y)} = \begin{cases} \{f(x)\} & \text{se } y \in B \\ \emptyset & \text{se } y \notin B \end{cases}$$

La funzione totale $\hat{g}(x,y)$ è ulteriormente specializzabile mediante il teorema s-m-n: $\hat{g}(x,y) = \varphi_{s(y)}(x)$ con s ricorsiva (totale). la funzione $\hat{g}$ è ricorsiva totale, quindi anche $\varphi_{s(y)}(x)$ è ricorsiva totale. Per il Teorema di ricorsione di Kleene, esiste $n_0 \in \mathbb{N}$ tale che

$$\varphi_{\varphi_{s(y)}(n_0)} = \varphi_{n_0}$$

Osserviamo che: $W_{\hat{g}(x,y)} = W_{\varphi_{s(y)}(x)}$ e che $W_{\varphi_{s(y)}(n_0)} = W_{n_0}$.

...

□

Come conseguenza di questo fatto, osserviamo che tutti gli insiemi creativi sono tra loro equivalenti secondo $\equiv$. Inoltre, se A è creativo, allora la classe di equivalenza $[A]_{\equiv}$ è massima nell'ordinamento indotto da $\preceq$ sulle classi di equivalenza di insiemi r.e.

ESERCIZIO 23.14.

1. Studiare la relazione $\preceq$ o $\equiv$ esistente tra i seguenti insiemi:
   - $\{ x \mid x \text{ è primo} \}$;
   - $\{ x \mid x \text{ è pari} \}$;
   - $\{ x \mid W_x = \emptyset \}$;
   - $\{ x \mid |W_x| = \omega \}$;
   - $\{ x \mid |W_x| < \omega \}$;
   - $\{ x \mid \varphi_x \text{ è totale} \}$.
2. Dimostrare che $\{ x \mid \varphi_x \text{ è totale} \}$ è produttivo (Suggerimento: dimostrare che $\bar{K} \preceq \{ x \mid \varphi_x \text{ è totale} \}$).
3. Dimostrare che $\{ i \mid \varphi_i(i) = 0 \}$ è produttivo.
4. Dimostrare che $\{ i \mid W_i \text{ è ricorsivo} \}$ è produttivo.

5. Dimostrare che $\{ x \mid \exists y.\, x \in W_y \ \wedge \ y \in W_x \}$ è creativo.

Bibliography

[1] M. Aiello, A. Albano, G. Attardi, and U. Montanari. *Teoria della Computabilità, logica, teoria dei linguaggi formali.* Materiali didattici ETS, Pisa, 1976.

[2] H. B. Enderton. *A mathematical introduction to logic.* Academic Press, 1973. 2^{nd} printing.

[3] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computations.* Addison-Wesley, 1979.

[4] N.D. Jones. *Computability and Complexity.* MIT Press, 1997.

[5] G. Longo. *Metodi per il trattemento dell'informazione.* Servizio Editoriale Universitario di Pisa, 1984.

[6] G. Longo and P. Giannini. Ricorsività e continuità: il teorema di myhill-shepherdson per gli operatori ricorsivi ed il i teorema di ricorsione. Servizio Editoriale Universitario di Pisa, 1980.

[7] E. Mendelson. *Introduction to Mathematical Logic.* Van Nostrand, Princeton, N. J., 1979.

[8] P. Odifreddi. *Classical Recursion Theory.* Studies in Logic and the Foundations of Mathematics. North-Holland, 1989.

[9] R. Péter. *Rekursive funktionen.* Akadémiai Kiadó, Budapest, 1953.

[10] H. J. Rogers. *Theory of Recursive Functions and Effective Computability.* The MIT Press, 1988.

[11] J.R. Shoenfield. Axioms of set theory. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 321–344. North-Holland, Amsterdam, 1977.