Università degli Studi di Verona
Corso di Laurea in Informatica

Fondamenti dell'Informatica

Appunti di Lorenzo Di Berardino

ottobre 2024 – gennaio 2025

Indice

1 Introduzione

1.1 Il problema dell'informatica

L'informatica si occupa di risolvere problemi. Un problema è una domanda, un'interrogazione, un quesito. La soluzione di un problema è un algoritmo, ovvero una sequenza finita di passi che, se seguiti correttamente, risolvono il problema. Poiché ogni dato può essere rappresentato da numeri naturali, tramite una biiezione, è sufficiente considerare funzioni del tipo:

$$f\colon \mathbb{N} \to \mathbb{N} \tag{1.1}$$

per modellare tutti i possibili problemi.

Definizione (Funzione calcolabile). *Una funzione f si dice* calcolabile *se esiste un algoritmo che, dato un input x, restituisce $f(x)$.*

Proposizione 1. *Nell'insieme delle funzioni $\mathbb{N} \to \mathbb{N}$, la parte delle funzioni calcolabili è strettamente contenuta.*

Resta da capire quale sia la proporzione tra le funzioni calcolabili e quelle non calcolabili. Prima alcune definizioni:

Definizione (Alfabeto). *Un* alfabeto Σ *è un insieme finito di simboli.*

Esempio. Si consideri l'alfabeto $\Sigma = \{0, 1\}$. Le stringhe componibili da Σ sono:

$$\Sigma^* = \{\, \varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots \}. \tag{1.2}$$

In questa scrittura, l'insieme Σ^* è detto *insieme delle sequenze di lunghezza finita sull'alfabeto*; la stringa ε è, invece, la stringa vuota (analoga dell'*insieme vuoto* per gli insiemi).

Poiché non vi è limite alla lunghezza delle stringhe, l'insieme Σ^* è infinito, e la sua cardinalità vale:

$$|\Sigma^*| = |\mathbb{N}| = \aleph_0. \tag{1.3}$$

Anche in questo caso è possibile ordinare le sequenze sull'i-esimo carattere, ottenendo una biiezione con $\mathbb{N}$. ◇

Definizione (Algoritmo). *Un* algoritmo*, o* programma*, è una sequenza finita di simboli su un alfabeto Σ.*

Un ultimo teorema prima della dimostrazione:

Teorema 1 (di Cantor). *Per ogni insieme S, si ha che $|S| < |\wp(S)|$.*

Insieme S	Insieme delle parti $\wp(S)$	$\lvert S\rvert$	$\lvert\wp(S)\rvert$
$\emptyset$	$\{\emptyset\}$	0	1
$\{0\}$	$\{\emptyset, \{0\}\}$	1	2
$\{0, 1\}$	$\{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$	2	4
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\{0, 1, \ldots, n-1\}$	$\{\emptyset, \{0\}, \{1\}, \ldots, \{0, 1, \ldots, n-1\}\}$	n	2^n

Tabella 1.1 *Cardinalità di insiemi finiti e dei loro sottoinsiemi.*

Il teorema è facilmente verificabile con alcuni insiemi, come mostrato in Tabella 1.1.

Dimostrazione (Teorema 1). Supponiamo per assurdo che $|S| = |\wp(S)|$. Allora esiste una funzione biunivoca $f\colon S \to \wp(S)$ che mappa ogni elemento di S in un sottoinsieme di S stesso. Prendiamo l'insieme $A = \{\, x \in S \mid x \notin f(x) \,\} \subseteq S$. Dato che $\wp(S)$ contiene ogni sottoinsieme di S e A ne è uno, $A \in \wp(S)$. Pertanto, deve necessariamente esistere un $a \in S$ tale che $f(a) = A$.

Ma se $a \in A$, allora $a \notin f(a) = A$ per costruzione di A; simmetricamente, se $a \notin A$, allora $a \in f(a) = A$. In entrambi i casi si è raggiunta una contraddizione. Si conclude, quindi, che non può esistere una biiezione tra S e $\wp(S)$ perché A non è ottenibile da nessun a e, di conseguenza, la cardinalità di S è strettamente inferiore a quella di $\wp(S)$. □

Dimostrazione (Proposizione 1). Innanzitutto, una funzione del tipo (1.1) può essere interpretata come un insieme di coppie ordinate:

$$f \coloneqq \{\, (x_i, y_i) \mid \forall i \in [1, n]\colon\ y_i = f(x_i) \,\}. \tag{1.4}$$

Pertanto, f è un sottoinsieme di $\mathbb{N} \times \mathbb{N}$ e, se si considerano tutte le possibili funzioni f_i:

$$\left|\bigcup_{i \geqslant 1} f_i\right| = |f\colon \mathbb{N} \times \mathbb{N}| = |\wp(\mathbb{N} \times \mathbb{N})| \tag{1.5}$$

dove $\wp(\mathbb{N} \times \mathbb{N})$ indica l'insieme delle parti di $\mathbb{N} \times \mathbb{N}$, ovvero tutti i sottoinsiemi di $\mathbb{N} \times \mathbb{N}$.

Dato che $|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$,[1] si ha anche che:

$$|\wp(\mathbb{N} \times \mathbb{N})| = |\wp(\mathbb{N})|. \tag{1.6}$$

Per la relazione (1.5), dunque, vi sono tante funzioni quanti sottoinsiemi di $\mathbb{N}$:

$$|f\colon \mathbb{N} \times \mathbb{N}| = |\wp(\mathbb{N})|. \tag{1.7}$$

Dato che un algoritmo è una sequenza finita di simboli, e si è detto che l'insieme delle stringhe costruito su un alfabeto è numerabile, si ha che:

$$|\text{Programmi}| = |\mathbb{N}| \iff |\text{Problemi calcolabili}| = |\mathbb{N}|. \tag{1.8}$$

Si può dire che l'insieme degli algoritmi è *equipotente* a $\mathbb{N}$.

Si è quindi ottenuto il seguente risultato:

$$|\text{Problemi calcolabili}| = |\text{Programmi}| = |\mathbb{N}| < |\wp(\mathbb{N})| = |f\colon \mathbb{N} \to \mathbb{N}| = |\text{Problemi}|. \tag{1.9}$$

□

[1] Un modo equivalente per dimostrare la numerabilità di un insieme è quello di definire una relazione di ordinamento sui suoi elementi. In questo caso, è sufficiente ordinare le coppie rispetto al primo numero e poi rispetto al secondo.

Dopo aver dimostrato che la cardinalità dei programmi è strettamente inferiore a quella di tutte le funzioni, ovvero di tutti i problemi, ci si può chiedere quanto sia questa differenza.

Il Teorema 1 stabilisce che la cardinalità dell'insieme delle parti è esponenziale rispetto a quella dell'insieme su cui viene calcolato. Inoltre, la cardinalità dell'insieme delle parti di $\mathbb{N}$ è la stessa di $\mathbb{R}$, secondo *l'ipotesi del continuo*, che è non numerabile. Pertanto, i programmi sono in quantità numerabile, mentre quella di tutti i problemi è non numerabile.

1.2 Induzione matematica

Per utilizzare il principio di induzione su un insieme, è necessario che su questo sussista una relazione d'ordine ben fondata.

Definizione (Relazione d'ordine ben fondata). *Una relazione d'ordine $R \subseteq A \times A$ su un insieme A si dice* ben fondata *quando è transitiva e antisimmetrica.*

Si ricorda che la *transitività* è definita come:

$$\forall a, b, c \in A \colon a \; R \; b \land b \; R \; c \implies a \; R \; c, \tag{1.10}$$

mentre l'*antisimmetria* come:

$$\forall a, b \in A \colon a \; R \; b \implies \neg(b \; R \; a). \tag{1.11}$$

La caratteristica di riflessività:

$$\forall a \in A \colon a \; R \; a \tag{1.12}$$

non deve essere soddisfatta in quanto la possibilità di confrontare un elemento con se stesso permetterebbe di costruire catene discendenti infinite.

Si può quindi definire formalmente l'induzione.

Definizione (Induzione matematica). *Sia A un insieme con una relazione d'ordine $<$ ben fondata. Sia π una proprietà definita sugli elementi di A. Allora vale la seguente corrispondenza:*

$$\forall a \in A \colon \pi(a) \iff \forall a \in A \colon \big((\forall b \in A . b < a \colon \pi(b)) \implies \pi(a)\big). \tag{1.13}$$

La (1.13) è equivalente a:

$$\forall a \in A \colon \pi(a) \iff \begin{cases} \forall a \in \textsc{Base} \colon \pi(a) \\ [\forall b \in A . b < a \land b \notin \textsc{Base} \colon \pi(b)] \implies \pi(a). \end{cases} \tag{1.14}$$

I

Linguaggi formali

2 Linguaggi regolari

Definizione (Stringa). *Una* stringa*, o* parola*, è una sequenza finita di simboli su un alfabeto* Σ*.*

Definizione (Lunghezza di una stringa). *La* lunghezza $|x|$ *di una stringa* x *è il numero di occorrenze di simboli al suo interno.*

Definizione (Linguaggio). *Un* linguaggio L *su un alfabeto* Σ *è un sottoinsieme delle stringhe di* Σ^**.*

Alcuni esempi di linguaggi sono $\emptyset$, dato che è sottoinsieme di qualsiasi insieme, e $\{\varepsilon\}$, la stringa vuota. Quest'ultimo compone, appunto, il linguaggio L che contiene solo la stringa vuota.

Poiché L è, di fatto, un insieme, è possibile eseguire su di esso operazioni insiemistiche:

Complemento $L \subseteq \Sigma^*$: $\neg L = \Sigma^* \setminus L$. (2.1)

- $\Sigma^* \setminus L \equiv \neg L = \overline{L}$.
- $\overline{\emptyset} = \Sigma^* \setminus \emptyset = \Sigma^*$.
- $\overline{\{\varepsilon\}} = \{\, x \in \Sigma^* \mid x \neq \varepsilon \,\} = \Sigma^* \setminus \{\varepsilon\}$.

Unione $L_1, L_2 \subseteq \Sigma^*$: $L_1 \cup L_2 = \{\, x \in \Sigma^* \mid x \in L_1 \vee x \in L_2 \,\}$. (2.2)

Intersezione $L_1, L_2 \subseteq \Sigma^*$: $L_1 \cap L_2 = \{\, x \in \Sigma^* \mid x \in L_1 \wedge x \in L_2 \,\}$. (2.3)

Alcune operazioni sono invece proprie dei linguaggi:

Concatenazione $L_1, L_2 \subseteq \Sigma^*$: $L_1 \cdot L_2 = \{\, xy \in \Sigma^* \mid x \in L_1, y \in L_2 \,\}$. (2.4)

- $L_1 = \{\, 0^n \mid n \in \mathbb{N} \,\}$, $L_2 = \{\, 1^n \mid n \in \mathbb{N} \,\}$: $L_1 \cdot L_2 = \{\, 0^n 1^m \mid n, m \in \mathbb{N} \,\}$.

Stella di Kleene[1] $L \subseteq \Sigma^*$:

$$L^n = \begin{cases} L^0 = \{\varepsilon\} \\ L^{n+1} = L \cdot L^n \end{cases}, \qquad L^* = \bigcup_{n \in \mathbb{N}} L^n. \tag{2.5}$$

- $L = \{\, 0^n 1^n \mid n \in \mathbb{N} \,\}$: $L^0 = \{\varepsilon\}$,
  $L^1 = L \cdot L^0 = L$,
  $L^2 = L \cdot L = \{\, x_1 x_2 \mid x_1 \in L, x_2 \in L \,\} = \{\, 0^n 1^n 0^m 1^m \mid n, m \in \mathbb{N} \,\}$.

[1] La stella di Kleene corrisponde all'insieme di tutte le possibili concatenazioni degli elementi di un linguaggio.

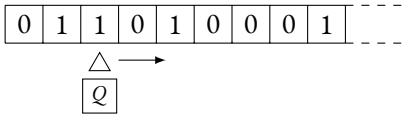


Figura 2.1 *Modello di automa a stati finiti deterministico.*

2.1 Automi a stati finiti deterministici

I linguaggi regolari sono tali per cui esiste un automa a stati finiti deterministico che li riconosce. In generale, la struttura dell'automa, che permette di rispondere alla domanda "$x \in L$?", determina la classe del linguaggio.

Un automa è composto da un nastro di lunghezza illimitata ma in sola lettura, e dunque non possiede nessuna alcun tipo di memoria *attiva*. Il nastro viene letto mediante una testina e scorre in una sola direzione. L'automa possiede inoltre un'unità di controllo, descritta attraverso lo stato in cui si trova. Un programma in esecuzione sull'unità, infine, si occupa di stabilire le transizioni di stato.

Ora una definizione più formale di automa.

Definizione (Automa a stati finiti deterministico). *Un* Deterministic Finite Automaton (DFA) *è rappresentato da una quintupla:*

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle \tag{2.6}$$

dove:

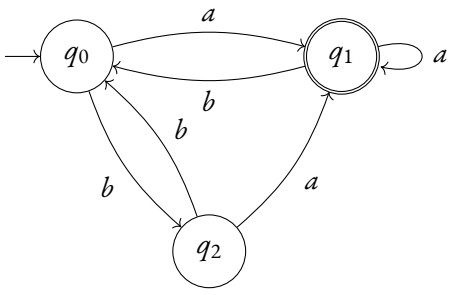
- Q *è l'insieme (finito) degli stati. Descrive le caratteristiche delle stringhe che fanno raggiungere uno stato finale.*
- Σ *è l'alfabeto (finito) di simboli del linguaggio che l'automa riconosce.*
- $\delta \colon Q \times \Sigma \to Q$ *è la funzione di transizione. Descrive come evolve ogni stato in funzione di un simbolo letto.*
- $q_0 \in Q$ *è lo stato iniziale. Nel contesto delle stringhe rappresenta la stringa vuota* ε.
- $F \subseteq Q$ *è l'insieme degli stati finali. Rappresenta le caratteristiche delle stringhe di* L.

Poiché la funzione δ acquisisce in ingresso un solo simbolo, è necessario estenderla per poter leggere stringhe. Questo viene eseguito mediante la funzione $\hat{\delta}$, definita come segue:

$$\hat{\delta} \colon Q \times \Sigma^* \to Q, \qquad \begin{cases} \hat{\delta}(q, \varepsilon) = q \\ \hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a). \end{cases} \tag{2.7}$$

Questa definizione è induttiva sulla lunghezza di x e su $a \in \Sigma$.

Esempio. Se si considera il seguente diagramma degli stati:



si può individuare l'automa che rappresenta. In particolare, si ha:

- $\Sigma = \{a, b\}$,
- $Q = \{q_0, q_1, q_2\}$,
- q_0 è lo stato iniziale,
- $F = \{q_1\}$.

Le corrispondenze per le transizioni di stato possono essere rappresentate più agevolmente in forma tabellare:

δ	Σ	
Q	a	b
q_0	q_0	q_2
q_1	q_1	q_0
q_2	q_1	q_0

L'automa descritto dal diagramma accetta tutte le stringhe che terminano con a. Il linguaggio riconosciuto è dunque $L = \{ x \in \Sigma^* \mid x = ya, y \in \Sigma^* \}$. ◇

Definizione (Linguaggio riconosciuto da un DFA). *Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un* DFA. *Sia x una stringa su Σ. Allora M accetta x se $\hat{\delta}(q_0, x) \in F$. Il linguaggio riconosciuto da M è:*

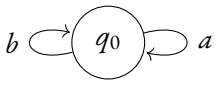
$$L(M) = \{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F \}. \tag{2.8}$$

Definizione (Linguaggio regolare). *Un linguaggio L si dice* regolare *se esiste un* DFA *M tale che:*

$$L = L(M). \tag{2.9}$$

Esempi.

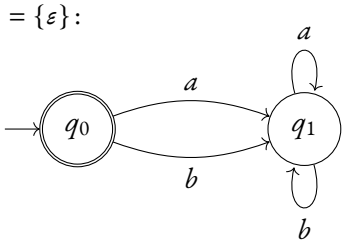
- $L = \emptyset$:



$Q = \{q_0\}$ $\quad \Sigma = \{a, b\}$ $\quad q_0$ $\quad F = \emptyset$

δ	a	b
q_0	q_0	q_0

- $L = \{\varepsilon\}$:



$Q = \{q_0, q_1\}$ $\quad \Sigma = \{a, b\}$ $\quad \hat{\delta}(q_0, \varepsilon) = q_0 \in F$
q_0 $\quad F = \{q_0\}$ $\quad \hat{\delta}(q_0, x \neq \varepsilon) = q_1 \notin F$

◇

Di conseguenza, un modo per dimostrare che un linguaggio è regolare è costruire un DFA che lo riconosca. L'uguaglianza (2.9) viene dimostrata facendo vedere la duplice inclusione $L \subseteq L(M)$ e $L(M) \subseteq L$. Queste due ultime relazioni sono equivalenti a:

$$\begin{aligned} & L(M) \subseteq L \\ \iff \quad & x \in L(M) \implies x \in L \\ \iff \quad & \hat{\delta}(q_0, x) \in F \implies x \in L \\ \iff \quad & x \notin L \implies \hat{\delta}(q_0, x) \notin F. \end{aligned} \tag{2.10a}$$

$$\begin{aligned} & L \subseteq L(M) \\ \iff \quad & x \in L \implies x \in L(M) \\ \iff \quad & x \in L \implies \hat{\delta}(q_0, x) \in F. \end{aligned} \tag{2.10b}$$

Riassumendo:

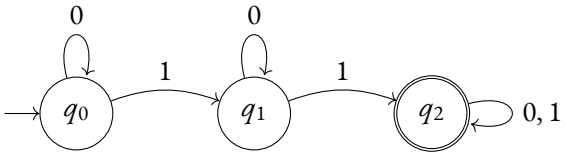
$$\begin{cases} x \in L \implies \hat{\delta}(q_0, x) \in F & \text{(2.11a)} \\ x \notin L \implies \hat{\delta}(q_0, x) \notin F. & \text{(2.11b)} \end{cases}$$

Dal momento che le condizioni da dimostrare sono analoghe, verranno portate avanti insieme ed è quindi necessario che la base dell'induzione sia la stessa per entrambe. In particolare, è richiesto di trovare una lunghezza tale per cui esistano stringhe in L e stringhe non in L. I passi sono dunque:

Base Si considera la lunghezza minima di x e, per lunghezze inferiori o uguali a quella, si dimostra che valgono la (2.11a) e la (2.11b).

Passo induttivo Si ipotizza che per ogni x tale che $|x| \leqslant n$ valgano la (2.11a) e la (2.11b) e si dimostra che valgono anche per $|x| = n + 1$.

Esempio. Si consideri il linguaggio $L = \{ x \in \Sigma^* \mid \text{“}x \text{ contiene almeno due 1”} \}$ definito sull'alfabeto binario $\Sigma = \{0, 1\}$. L'automa a stati finiti che riconosce L è il seguente:



δ	0	1
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_2	q_2

(2.12)

È comunque necessario dimostrare che vengono riconosciute tutte e sole le stringhe di L. In particolare, bisogna dimostrare che $L = L(M) = \{ x \mid \hat{\delta}(q_0, x) = q_2 \}$. Si procede per induzione su $|x|$. Si nota subito che non è possibile considerare $|x| = 0$[2] o $|x| = 1$, in quanto:

- $|x| = 0$: $x = \varepsilon \notin L$, $\hat{\delta}(q_0, \varepsilon) = q_0 \neq q_2$.
- $|x| = 1$: $x = 0 \notin L$, $\hat{\delta}(q_0, 0) = q_0 \neq q_2$,
  $x = 1 \notin L$, $\hat{\delta}(q_0, 1) = q_1 \neq q_2$.

ovvero non vi sono stringhe della stessa lunghezza – inferiore a 2 – sia appartenenti a L che non.

Il punto di partenza deve essere necessariamente $|x| = 2$. Infatti, per tale lunghezza (minima) vi sono stringhe con questa caratteristica:

- $|x| = 2$: $x = 00 \notin L$, $\hat{\delta}(q_0, 00) = q_0 \neq q_2$,
  $x = 01 \notin L$, $\hat{\delta}(q_0, 01) = q_1 \neq q_2$,
  $x = 10 \notin L$, $\hat{\delta}(q_0, 10) = q_1 \neq q_2$,
  $x = 11 \in L$, $\hat{\delta}(q_0, 11) = q_2$.

La dimostrazione delle implicazioni (2.11a) e (2.11b) può avvenire mostrando proprietà più forti:

1. $x \in L \implies \hat{\delta}(q_0, x) = q_2$,
2. $x \notin L \wedge$ “x non contiene 1” $\implies \hat{\delta}(q_0, x) = q_0$,
3. $x \notin L \wedge$ “x contiene esattamente un 1” $\implies \hat{\delta}(q_0, x) = q_1$.

Base

- $|x| = 0$: $x = \varepsilon \implies x \notin L \wedge$ “x non contiene 1”: $\hat{\delta}(q_0, \varepsilon) = q_0$.

[2] In realtà, non è mai possibile considerare $|x| = 0$ come caso base perché l'unica stringa di tale lunghezza, ε, o appartiene a un linguaggio o non vi appartiene. Servono, invece, lunghezze tali per cui esistono stringhe sia appartenenti che non.

- $|x| = 1$: $x = 0 \implies x \notin L \wedge$ "x non contiene 1": $\hat{\delta}(q_0, 0) = q_0$,
  $x = 1 \implies x \notin L \wedge$ "x contiene esattamente un 1": $\hat{\delta}(q_0, 1) = q_1$.
- $|x| = 2$: $x = 11 \implies x \in L$: $\hat{\delta}(q_0, 11) = q_2$.

Dato che sono stati enumerati tutti gli stati dell'automa, si può affermare che le tre proprietà sono verificate per $|x| \leqslant 2$.

Passo induttivo Per ipotesi induttiva, si suppone che valgano le proprietà 1, 2 e 3:

$$\forall x \in \Sigma^*: |x| \leqslant n. \tag{2.13}$$

Si deve mostrare che valgono anche per $|x| = n + 1$. Dunque, se $|x| = n + 1$ e $x = va$, con $|v| = n$ e $a \in \Sigma$, si hanno i tre casi:

1. $x \in L \implies \hat{\delta}(q_0, x) = q_2$.

   1a. $x \in L \wedge a = 0$: se $x \in L$, allora x contiene almeno due 1. Quindi, se $a = 0$, i due 1 devono essere contenuti in v. Per l'ipotesi induttiva Hp(1), $\hat{\delta}(q_0, v) = q_2$. Pertanto:

   $$\hat{\delta}(q_0, v0) = \delta(\hat{\delta}(q_0, v), 0) = \delta(q_2, 0) = q_2 \in F. \tag{2.14}$$

   1b. $x \in L \wedge a = 1$: se $x \in L$, i due 1 possono trovarsi entrambi in v (ed eventualmente essercene di più), oppure essere uno in v e uno in a.

   - Se $v \in L$, cioè v contiene almeno due 1, si ricade nel caso precedente:

   $$\hat{\delta}(q_0, v1) = \delta(\hat{\delta}(q_0, v), 1) = \delta(q_2, 1) = q_2 \in F. \tag{2.15}$$

   - Se $v \notin L$, cioè v contiene al più un 1, e l'altro si trova in a, allora per l'ipotesi induttiva Hp(3):

   $$\hat{\delta}(q_0, v1) = \delta(\hat{\delta}(q_0, v), 1) = \delta(q_1, 1) = q_2 \in F. \tag{2.16}$$

2. $x \notin L \wedge$ "x non contiene 1" $\implies \hat{\delta}(q_0, x) = q_0$.
   Per le condizioni imposte, deve verificarsi che $x = v0$, $v \notin L$ e non contiene 1. Per l'ipotesi induttiva Hp(2):
   $$\hat{\delta}(q_0, v0) = \delta(\hat{\delta}(q_0, v), 0) = \delta(q_0, 0) = q_0 \tag{2.17}$$

3. $x \notin L \wedge$ "x contiene esattamente un 1" $\implies \hat{\delta}(q_0, x) = q_1$.
   In questo caso, poiché $x \notin L$, l'unico 1 presente può trovarsi in v oppure in a, ma non in entrambi.

   3a. Se $x = v1$, allora $v \notin L$ e non contiene 1. Per l'ipotesi induttiva Hp(2):

   $$\hat{\delta}(q_0, v1) = \delta(\hat{\delta}(q_0, v), 1) = \delta(q_0, 1) = q_1. \tag{2.18}$$

   3b. Se $x = v0$, allora $v \notin L$ e contiene esattamente un 1. Per l'ipotesi induttiva Hp(3):

   $$\hat{\delta}(q_0, v0) = \delta(\hat{\delta}(q_0, v), 0) = \delta(q_1, 0) = q_1. \tag{2.19}$$

◇

2.2 Automi a stati finiti non deterministici

Definizione. *Un* Non-deterministic Finite Automaton (NFA) *è rappresentato da una quintupla:*

$$N = \langle Q, \Sigma, \delta, q_0, F \rangle, \tag{2.20}$$

in modo analogo a un DFA*, con la differenza che:*

$$\delta \colon Q \times \Sigma \to \wp(Q). \tag{2.21}$$

L'insieme dei valori raggiungibili da uno stato e una stringa di ingresso è un sottoinsieme dell'insieme delle parti di Q, potenzialmente anche vuoto.

Anche per i NFA è possibile definire induttivamente la funzione estesa $\hat{\delta}$:

$$\hat{\delta}(q, x) = \begin{cases} \hat{\delta}(q, \varepsilon) = \{q\} \\ \hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q,x)} \delta(p, a). \end{cases} \tag{2.22}$$

Il linguaggio riconosciuto da un NFA $N = \langle Q, \Sigma, \delta, q_0, F \rangle$ è:

$$L(N) = \{\, x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset \,\}, \tag{2.23}$$

ovvero l'insieme delle stringhe che portano l'automa in uno stato finale, anche seguendo percorsi diversi.

Teorema 2 (di Rabin–Scott).

a. Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un DFA*. Allora esiste un* NFA *$N = \langle Q', \Sigma, \delta', q_0, F \rangle$ tale che i due linguaggi riconosciuti coincidono, ovvero $L(M) = L(N)$.*

b. Sia $N = \langle Q, \Sigma, \delta, q_0, F \rangle$ un NFA*. Allora esiste un* DFA *$M = \langle Q', \Sigma, \delta', q_0, F' \rangle$ tale che i due linguaggi riconosciuti coincidono, ovvero $L(N) = L(M)$.*

Questo teorema stabilisce l'assoluta equivalenza tra DFA e NFA. La differenza tra le due tipologie di automi risiede nelle proporzioni di utilizzo delle risorse temporali e spaziali.

Dimostrazione (Teorema 2). Poiché il teorema garantisce l'equivalenza, è necessario dimostrare entrambe le direzioni:

a. L'unica modifica da produrre affinché si possa considerare un DFA come un NFA è la definizione della funzione di transizione. In particolare, se si considera la funzione δ di un DFA:

$$\delta_{\text{D}} \colon Q \times \Sigma \to Q, \tag{2.24}$$

si può modificare l'insieme dei valori restituiti, considerando insiemi composti da un solo stato:

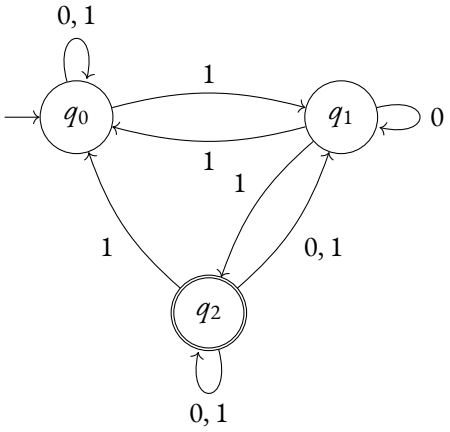
$$\delta_{\text{N}} \colon Q \times \Sigma \to \{Q\}. \tag{2.25}$$

Infatti, si può vedere un DFA come un caso particolare di NFA in cui l'insieme dei valori restituiti è sempre costituito da un solo elemento.

b. La dimostrazione nell'altro senso è meno banale. Infatti, le modifiche devono essere più estese:

- $Q' = \wp(Q)$,
- $q'_0 = \{q_0\}$,
- $F' = \{\, P \subseteq Q' \mid P \cap F \neq \emptyset \,\}$,
- $\delta'(P, a) = \bigcup_{q \in P} \delta(q, a)$. □

Esempio. Per illustrare il Teorema 2, si consideri il seguente NFA definito su $\Sigma = \{0, 1\}$:



Una stringa presente nel linguaggio è sicuramente 010010, perché esiste almeno un percorso che arriva a q_2 da q_0 consumandola interamente. In particolare: $q_0 \to_0 q_0 \to_1 q_1 \to_0 q_1 \to_0 q_1 \to_1 q_2 \to_0 q_2$.

La traduzione da un NFA a un DFA avviene innanzitutto esplicitando la funzione di transizione δ:

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

A questo punto, eseguendo le seguenti associazioni tra stati:[3]

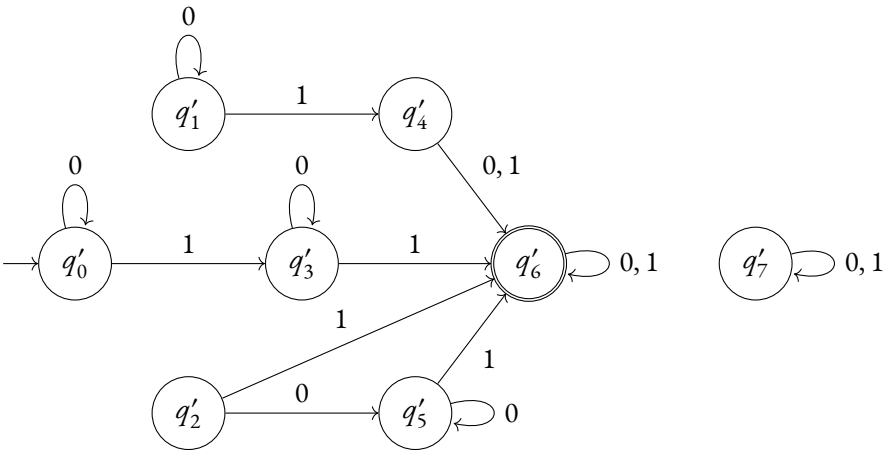
- $q'_0 = \{q_0\}$,
- $q'_1 = \{q_1\}$,
- $q'_2 = \{q_2\} \in F'$,
- $q'_3 = \{q_0, q_1\}$,
- $q'_4 = \{q_0, q_2\} \in F'$,
- $q'_5 = \{q_1, q_2\} \in F'$,
- $q'_6 = \{q_0, q_1, q_2\} \in F'$,
- $q'_7 = \emptyset$.

si può costruire la tabella di transizione per il DFA:

Q'	Q	0	1
q'_0	$\{q_0\}$	$q'_0 = \{q_0\}$	$q'_3 = \{q_0, q_1\}$
q'_1	$\{q_1\}$	$q'_1 = \{q_1\}$	$q'_4 = \{q_0, q_2\}$
q'_2	$\{q_2\}$	$q'_5 = \{q_1, q_2\}$	$q'_6 = \{q_0, q_1, q_2\}$
q'_3	$\{q_0, q_1\}$	$q'_3 = \{q_0, q_1\}$	$q'_6 = \{q_0, q_1, q_2\}$
q'_4	$\{q_0, q_2\}$	$q'_6 = \{q_0, q_1, q_2\}$	$q'_6 = \{q_0, q_1, q_2\}$
q'_5	$\{q_1, q_2\}$	$q'_5 = \{q_1, q_2\}$	$q'_6 = \{q_0, q_1, q_2\}$
q'_6	$\{q_0, q_1, q_2\}$	$q'_6 = \{q_0, q_1, q_2\}$	$q'_6 = \{q_0, q_1, q_2\}$
q'_7	$\emptyset$	$q'_7 = \emptyset$	$q'_7 = \emptyset$

[3] Si noti che gli stati finali del DFA sono quelli che contengono almeno uno stato finale del NFA.

Si ottiene dunque il seguente DFA:



Dal momento che lo stato iniziale è q'_0 e i rami associati a q'_2, q'_1 e q'_7 non sono mai raggiungibili, si può semplificare il DFA ottenendo il medesimo dell'esempio (2.12), ovvero mantenendo solo gli stati q'_0, q'_3 e q'_6. ◇

Dimostrazione (Teorema 2). In generale, per dimostrare che $L(M) = L(N)$ bisogna verificare che:

a. $\hat{\delta}(q_0, x) = \hat{\delta}'(q'_0, x)$. Per induzione su $|x|$:

Base $|x| = 0, x = \varepsilon$: $\hat{\delta}(q_0, \varepsilon) = \{q_0\}$ (NFA),
$\delta'(q'_0, \varepsilon) = q'_0 = \{q_0\}$ (DFA).

Passo induttivo Per ipotesi induttiva: $\forall x \in \Sigma^*. |x| \leqslant n$: $\hat{\delta}(q_0, x) = \hat{\delta}'(q'_0, x)$.

$$\begin{aligned}
\hat{\delta}'(q'_0, va) &= \delta'(\hat{\delta}'(q'_0, v), a) && \text{definizione di } \delta' \text{ per il DFA equivalente} \\
&= \delta'(\hat{\delta}(q_0, v), a) && \text{ipotesi induttiva} \\
&= \bigcup_{q \in \hat{\delta}(q_0, v)} \delta(q, a) && \text{definizione di } \delta \text{ per un NFA} \\
&= \hat{\delta}(q_0, va).
\end{aligned} \tag{2.26}$$

b. $x \in L(N) \iff x \in L(M)$:

$$\begin{aligned}
x \in L(N) &\iff \hat{\delta}(q_0, x) \cap F \neq \emptyset && \text{definizione di stato finale per un NFA} \\
&\iff \hat{\delta}'(q'_0, x) \cap F \neq \emptyset && \text{per il punto a.} \\
&\iff \hat{\delta}'(q'_0, x) \in F' && \text{definizione di } F' \text{ per un DFA} \\
&\iff x \in L(M).
\end{aligned} \tag{2.27}$$

□

2.3 Automi a stati finiti non deterministici con ε-transizioni

Definizione. *Un* Non-deterministic Finite Automaton with ε-transitions (ε-NFA) *è rappresentato da una quintupla:*

$$N_\varepsilon = \langle Q, \Sigma, \delta, q_0, F \rangle. \tag{2.28}$$

La principale differenza rispetto a un NFA tradizionale è la possibilità di spostarsi tra gli stati senza consumare la stringa di input. Se nel caso deterministico e non-deterministico le definizioni di δ sono:

$$\delta_{\text{D}} \colon Q \times \Sigma \to Q, \tag{2.29}$$

$$\delta_{\text{N}} \colon Q \times \Sigma \to \wp(Q). \tag{2.30}$$

con questa tipologia di automi:

$$\delta_{\varepsilon} \colon Q \times (\Sigma \cup \{\varepsilon\}) \to \wp(Q), \tag{2.31}$$

per cui si ammette la possibilità di fornire la stringa vuota per transitare in un nuovo stato.

La funzione estesa $\hat{\delta}$ viene definita come:

$$\hat{\delta}(q, x) = \begin{cases} \hat{\delta}(q, \varepsilon) = \varepsilon\text{-closure}(q) \\ \hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q,x)} \varepsilon\text{-closure}(\delta(p, a)) \end{cases} \tag{2.32}$$

dove ε-closure(q) è l'insieme degli stati raggiungibili da q tramite archi etichettati con ε (tra cui anche q stesso). Inoltre, si definisce la ε-closure su un insieme $P \subseteq Q$ come:

$$\varepsilon\text{-closure}(P) = \bigcup_{q \in P} \varepsilon\text{-closure}(q). \tag{2.33}$$

L'insieme delle stringhe accettate da N_ε è:

$$L(N_\varepsilon) = \{\, x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset \,\}. \tag{2.34}$$

Teorema 3 (Equivalenza tra ε-NFA e NFA).

a. Sia $N_\varepsilon = \langle Q, \Sigma, \delta, q_0, F \rangle$ *un* ε-NFA. *Allora esiste un* NFA $N = \langle Q', \Sigma, \delta', q_0, F' \rangle$ *tale che* $L(N) = L(N_\varepsilon)$.

b. Sia $N = \langle Q, \Sigma, \delta, q_0, F \rangle$ *un* NFA. *Allora esiste un* ε-NFA $N_\varepsilon = \langle Q', \Sigma, \delta', q_0, F' \rangle$ *tale che* $L(N_\varepsilon) = L(N)$.

Dimostrazione (Teorema 3). Anche in questo caso la dimostrazione richiede entrambi i versi. L'esistenza di un ε-NFA dato un NFA è banale, dato che il secondo è un caso particolare del primo in cui non vi sono archi etichettati con ε.

L'altra direzione si dovrebbe comporre di tre passaggi:

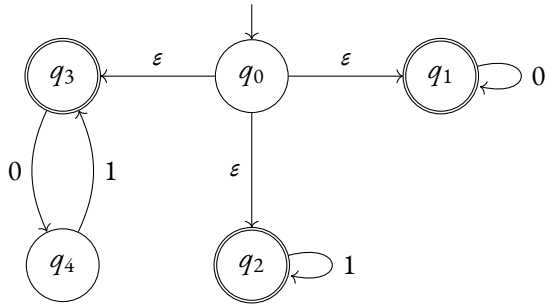
- Costruzione di un NFA $N = \langle Q', \Sigma, \delta', q_0, F' \rangle$,
- Dimostrazione che $\hat{\delta}(q_0, x) \cap F = \hat{\delta}'(q_0, x) \cap F'$,
- Dimostrazione che $\hat{\delta}(q_0, x) = \hat{\delta}'(q_0', x)$ tramite il punto precedente.

Si preferisce invece illustrare la possibilità di traduzione da ε-NFA a NFA attraverso un esempio. □

Esempio. La costruzione del NFA avviene eseguendo le seguenti trasformazioni:

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

Si consideri l'ε-NFA N_ε definito come:

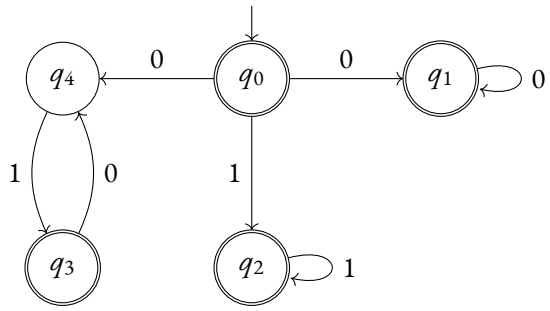


Poiché l'insieme degli stati raggiungibili da q_0 tramite archi etichettati con ε è $Q = \{ q_0, q_1, q_2, q_3 \}$, e tra questi vi è almeno uno stato finale, si ha che $F' = F \cup \{q_0\}$.

La funzione di transizione δ' si costruisce seguendo per primi i percorsi etichettati con ε e cercando, in coda, una transizione contenente 0 o 1, in base alla colonna corrispondente:

δ'	0	1
q_0	$\{q_1, q_4\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\emptyset$	$\{q_3\}$

Si ottiene dunque il seguente NFA:



◇

2.4 Espressioni regolari

Un modo alternativo per definire un linguaggio regolare è attraverso le *espressioni regolari*.

Definizione (Espressione regolare). *Una Regular Expression* (RE) *è definita induttivamente come:*

Base

- $\emptyset$ *è una* RE *che denota il linguaggio* $L = \emptyset$,
- ε *è una* RE *che denota il linguaggio* $L = \{\varepsilon\}$,
- a *è una* RE *che denota il linguaggio* $L = \{a\}$.

Passo induttivo *Siano* r *e* s *due* RE *che denotano i linguaggi* $L(r)$ *e* $L(s)$ *rispettivamente. Allora:*

- $r + s$ *è una* RE *che denota l'unione dei due linguaggi* $L(r) \cup L(s)$,

- $r \cdot s$ *è una* RE *che denota la concatenazione dei due linguaggi* $L(r) \cdot L(s)$,
- r^* *è una* RE *che denota la stella di Kleene* $L(r)^*$ *sul linguaggio* $L(r)$.

Teorema 4 (Equivalenza tra automi ed espressioni regolari).

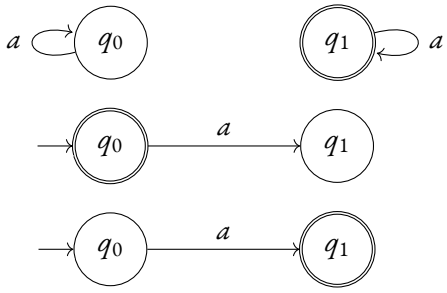
a. Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ *un* DFA. *Allora esiste una* RE r *tale che* $L(M) = L(r)$.

b. Sia r *una* RE. *Allora esiste un* ε-NFA $N_\varepsilon = \langle Q, \Sigma, \delta, q_0, F \rangle$ *tale che* $L(r) = L(N_\varepsilon)$.

Dimostrazione (Teorema 4).

Base Sia $a \in \Sigma$.

- $\emptyset$:
- ε:
- a:



Per ogni elemento nella base si è costruito un ε-NFA equivalente a r.

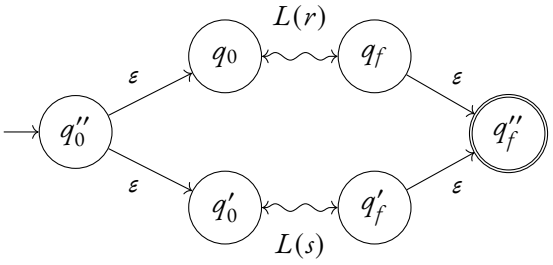
Passo induttivo Per ipotesi induttiva:

$$\forall r \in \text{RE}: \exists N_\varepsilon . L(r) = L(N_\varepsilon). \tag{2.35}$$

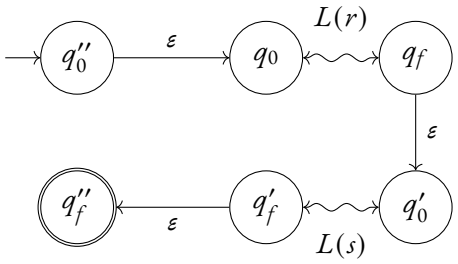
Si suppone, senza perdita di generalità, che nell'automa N_ε vi sia un solo stato iniziale e uno finale. In caso contrario, è sufficiente introdurli e collegarli con transizioni aventi etichetta ε a quelli già presenti.

Si trattano quindi le tre operazioni separatamente:

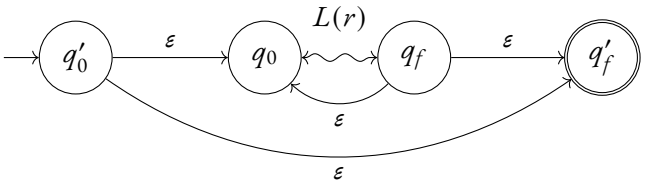
- Unione $L(r) \cup L(s) = L(r + s)$:



- Concatenazione $L(r) \cdot L(s) = L(r \cdot s)$:



- Stella di Kleene $L(r)^* = L(r^*)$:



□

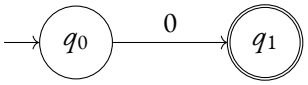
Esempio. Si vuole costruire l'ε-NFA corrispondente all'espressione regolare $r = (0^* + 1^* + (01)^*)$.

I linguaggi individuati dai singoli termini sono:

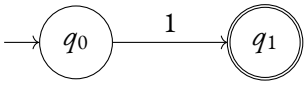
$$\begin{aligned} L(0^*) &= \{\, 0^n \mid n \in \mathbb{N} \,\}, \\ L(1^*) &= \{\, 1^n \mid n \in \mathbb{N} \,\}, \\ L((01)^*) &= \{\, (01)^n \mid n \in \mathbb{N} \,\}. \end{aligned} \tag{2.36}$$

Si comincia dunque con la rappresentazione di questi, secondo la definizione data:

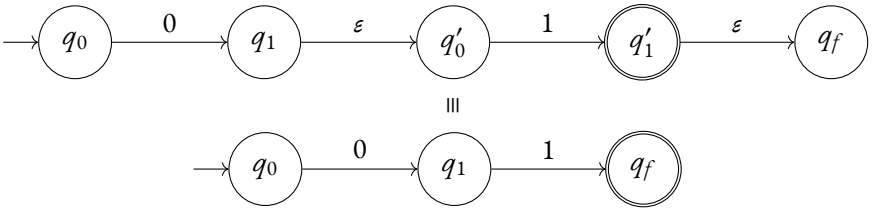
- 0:



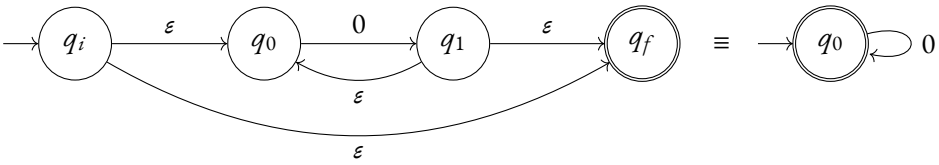
- 1:



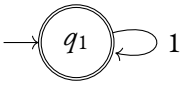
- 01:



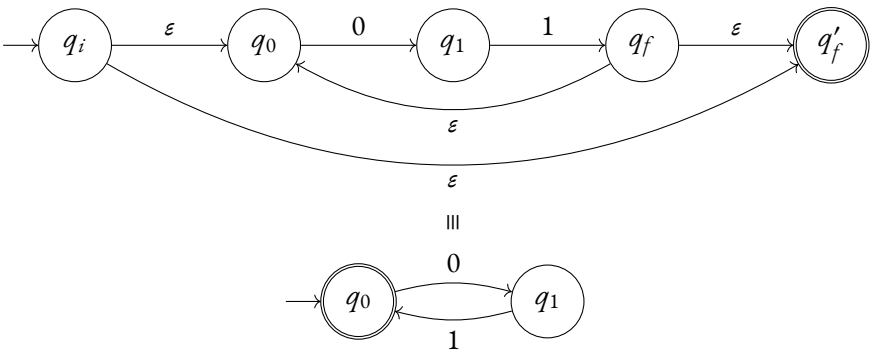
- 0^*:



- 1^*:

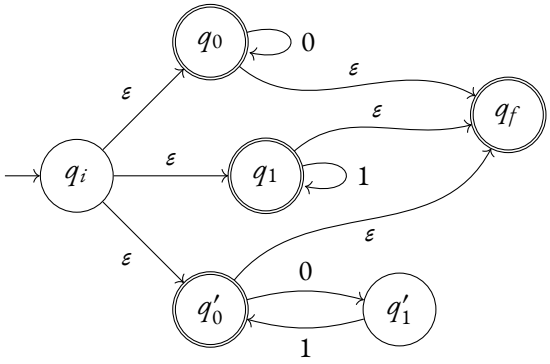


- $(01)^*$:



Infine, si combinano i tre linguaggi ottenuti:

- $0^* + 1^* + (01)^*$:



A partire da questa rappresentazione, si può costruire prima un NFA e poi un DFA equivalente. ◇

I linguaggi regolari godono di alcune proprietà di chiusura. Sia LR l'insieme dei linguaggi regolari:

$$\text{LR} = \{\, L \subseteq \Sigma^* \mid \text{“}L \text{ è un linguaggio regolare”} \,\}. \tag{2.37}$$

LR è chiuso rispetto all'*unione*, alla *concatenazione* e alla *stella di Kleene*:[4]

$$L_1, L_2 \in \text{LR} \implies \begin{cases} L_1 \cup L_2 \in \text{LR} \\ L_1 \cdot L_2 \in \text{LR} \\ L_1^* \in \text{LR}. \end{cases} \tag{2.38}$$

Inoltre, LR è chiuso anche rispetto alla *complementazione*:[5]

$$L \in \text{LR} \implies \overline{L} \in \text{LR}. \tag{2.39}$$

Per le leggi di De Morgan, la chiusura rispetto all'unione e alla complementazione implica anche la chiusura rispetto all'*intersezione*:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} \implies L_1 \cap L_2 \in \text{LR}. \tag{2.40}$$

Nota. La proprietà di chiusura rispetto all'unione e all'intersezione si applica solo quando il numero di insiemi che si stanno considerando è finito. Questo significa che, anche se i singoli L_n sono regolari, le seguenti composizioni non lo sono necessariamente:

$$\bigcup_{n \in \mathbb{N}} L_n, \qquad \bigcap_{n \in \mathbb{N}} L_n. \tag{2.41}$$

○

[4] La chiusura rispetto a queste operazioni deriva direttamente dall'equivalenza con le espressioni regolari.

[5] L'operazione di complementazione si traduce in un DFA nella semplice ridefinizione dell'insieme degli stati finali come $F' = Q \setminus F$.

2.5 Automi minimi

Il Teorema 5 di Myhill-Nerode fornisce la garanzia dell'esistenza di un automa minimo per un linguaggio regolare. Per l'enunciazione del teorema sono però necessarie alcune definizioni preliminari.

Definizione (Partizione). *Sia S un insieme. Sia $P = \{ S_1, S_2, \ldots, S_n \}$ una collezione di $n \leqslant |S|$ sottoinsiemi di S tale che:*

$$\forall i \in [1, n] : S_i \subseteq S, \tag{2.42}$$

$$\forall i \neq j \in [1, n] : S_i \cap S_j = \emptyset, \tag{2.43}$$

$$\bigcup_{i=1}^{n} S_i = S. \tag{2.44}$$

Allora P si dice partizione *di S.*

Definizione (Partizione indotta da R). *Sia $R \subseteq S \times S$ una relazione di equivalenza[6] su S. Si definisce* classe di equivalenza *la partizione indotta da R. L'insieme S viene suddiviso in sottoinsiemi disgiunti tali che:*

$$\forall x \in S : [x]_R = \{ y \in S \mid x \; R \; y \}. \tag{2.45}$$

Le classi di equivalenza di R costituiscono una partizione di S, raggruppando elementi che sono in relazione tra loro.

Definizione (Relazione di indice finito). *Una relazione R si dice* di indice finito *se l'insieme delle classi di equivalenza indotte da R possiede cardinalità finita.*

Definizione (Relazione di equivalenza indotta da L). *La relazione $R_L \subseteq \Sigma^* \times \Sigma^*$ è detta* indotta da L *se:*

$$\forall z \in \Sigma^* : xz \in L \iff yz \in L. \tag{2.46}$$

Ovvero, due stringhe x e y sono in relazione R_L se, per ogni loro prolungamento comune z, entrambe o nessuna di esse appartiene al linguaggio L.

Definizione (Relazione di equivalenza indotta da un automa). *La relazione R_M, dove M è l'automa $\langle Q, \Sigma, \delta, q_0, F \rangle$, è definita come:*

$$x \; R_M \; y \iff \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y). \tag{2.47}$$

Esempio. Nell'automa (2.12) si possono individuare tre classi di equivalenza:

- $\hat{\delta}(q_0, x) = q_0$: $[\varepsilon]_{R_M} = [0]_{R_M} = [00]_{R_M} = [000]_{R_M} = \ldots$
  $[\varepsilon]_{R_M} = \{ x \mid \hat{\delta}(q_0, x) = q_0 \} = \{ 0^n \mid n \in \mathbb{N} \}.$
- $\hat{\delta}(q_0, x) = q_1$: $[01]_{R_M} = [0010]_{R_M} = [0100]_{R_M} = \ldots$
  $[1]_{R_M} = \{ x \mid \hat{\delta}(q_0, x) = q_1 \} = \{ 0^i 10^j \mid i, j \in \mathbb{N} \}.$
- $\hat{\delta}(q_0, x) = q_2$: $[11]_{R_M} = [001010]_{R_M} = [010001]_{R_M} = \ldots$
  $[11]_{R_M} = \{ x \mid \hat{\delta}(q_0, x) = q_2 \} = \{ 0^i 10^j 1 \mid i, j \in \mathbb{N} \} \cdot \Sigma^*.$

◇

[6] Una relazione si dice di equivalenza quando è transitiva, simmetrica e riflessiva.

Definizione (Relazione invariante destra). *La relazione R si dice invariante destra se:*

$$\forall x, y \in \Sigma^*\colon (x \ R \ y \implies \forall z \in \Sigma^*\colon xz \ R \ yz). \tag{2.48}$$

Lemma 1. *R_L e R_M sono relazioni di equivalenza invarianti destre. Inoltre, R_M è di indice finito.*

Teorema 5 (di Myhill-Nerode). *Sia $L \subseteq \Sigma^*$ un linguaggio. I seguenti enunciati sono equivalenti:*

1. *L è regolare.*
2. *L è unione di classi di equivalenza su Σ^* indotte da una relazione di equivalenza $R \subseteq \Sigma^* \times \Sigma^*$ di indice finito e invariante destra.*
3. *R_L è di indice finito.*

Dimostrazione (Teorema 5). Poiché le tre affermazioni sono equivalenti, si possono dimostrare circolarmente, ovvero si dimostra che $1 \implies 2 \implies 3 \implies 1$.

$1 \implies 2$ Sia L un linguaggio regolare. Allora, per il Teorema 2, esiste un DFA $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ tale che $L = L(M)$. Si è detto che R_M è una relazione di equivalenza che partiziona le stringhe rispetto agli stati ed è di indice finito (poiché sono anch'essi finiti). Inoltre, è anche invariante destra per proprietà dei DFA. Si può quindi definire L come:

$$\begin{aligned} L &= \left\{ [x]_{R_M} \mid \hat{\delta}(q_0, x) \in F \right\} \\ &= \bigcup_{q \in F} \left\{ [x]_{R_M} \mid \hat{\delta}(q_0, x) = q \right\}. \end{aligned} \tag{2.49}$$

ovvero come unione di classi di equivalenza finite e indotte dalla relazione R_M, che è invariante destra e di indice finito.

$2 \implies 3$ Si assume come ipotesi il secondo enunciato. Sia $y \in [x]_R$, ovvero $x \ R \ y$. Poiché R è invariante destra, per ipotesi vale:

$$\forall z \in \Sigma^*\colon xz \ R \ yz. \tag{2.50}$$

Inoltre, sempre per ipotesi, L è definito come unione di classi di equivalenza di R, quindi si deve avere che:

$$\begin{aligned} v \ R \ w &\implies (v \in L \iff w \in L) \\ &\implies \forall z \in \Sigma^*\colon (vz \in L \iff wz \in L) \\ &\implies w \in [v]_{R_L} \\ &= v \ R_L \ w. \end{aligned} \tag{2.51}$$

Poiché $v \ R \ w \implies v \ R_L \ w$, o equivalentemente $\forall x \in \Sigma^*\colon [x]_R \subseteq [x]_{R_L}$, si dice che R_L è un *raffinamento* di R, dunque accorpa più elementi, con conseguenti meno classi. Inoltre, l'indice di R è finito per ipotesi, quindi lo è anche quello di R_L, dato che il primo è un limite superiore del secondo.

$3 \implies 1$ Sia $L \subseteq \Sigma^*$ un linguaggio. Per ipotesi, R_L è di indice finito e vale:

$$x \ R_L \ y \iff (\forall z \in \Sigma^*\colon xz \in L \iff yz \in L). \tag{2.52}$$

Per dimostrare che L è regolare si costruisce un automa che lo accetti nel seguente modo:

$$M' = \begin{cases} Q' = \{\, [x]_{R_L} \mid x \in \Sigma_L^* \,\} & \text{Classi di equivalenza di } R_L \\ \Sigma' = \Sigma_L \\ \delta'([x]_{R_L}, a) = [xa]_{R_L} \\ q_0' = [\varepsilon]_{R_L} \\ F' = \{\, [x]_{R_L} \mid x \in L \,\}. & \text{Classi delle stringhe in } L \end{cases} \tag{2.53}$$

Per la funzione di transizione δ, poiché R_L è invariante destra, la sua definizione non dipende dalla scelta di x. Si può quindi dimostrare per induzione che $\hat{\delta}'([x]_{R_L}, y) = [xy]_{R_L}$:

Base $y = \varepsilon$:
$$\hat{\delta}'([x]_{R_L}, \varepsilon) = \delta'([x]_{R_L}, \varepsilon) = [x\varepsilon]_{R_L} = [x]_{R_L}. \tag{2.54}$$

Passo induttivo Per ipotesi vale:
$$\forall y \in \Sigma^* . |y| \leqslant n : \hat{\delta}'([x]_{R_L}, y) = [xy]_{R_L}. \tag{2.55}$$

Sia ora $y = va$ con $|v| = n$ e $a \in \Sigma'$:
$$\hat{\delta}'([x]_{R_L}, va) = \delta'(\hat{\delta}'([x]_{R_L}, v), a) = \delta'([xv]_{R_L}, a) = [xva]_{R_L} = [xy]_{R_L}. \tag{2.56}$$

Ora si può usare la (2.56) per dimostrare che $L = L(M')$:

$$\begin{aligned} x \in L(M') &\iff \hat{\delta}'(q_0', x) \in F' \\ &\iff \hat{\delta}'([\varepsilon]_{R_L}, x) \in F' \\ &= [\varepsilon x]_{R_L} \in F' \\ &= [x]_{R_L} \in F' \\ &\iff x \in L. \end{aligned} \tag{2.57}$$

Il numero di classi di R_L è il numero minimo di stati che l'automa M' deve avere per accettare L. Ogni altro automa M, dunque, avrà un numero di stati almeno uguale a quello di M'. □

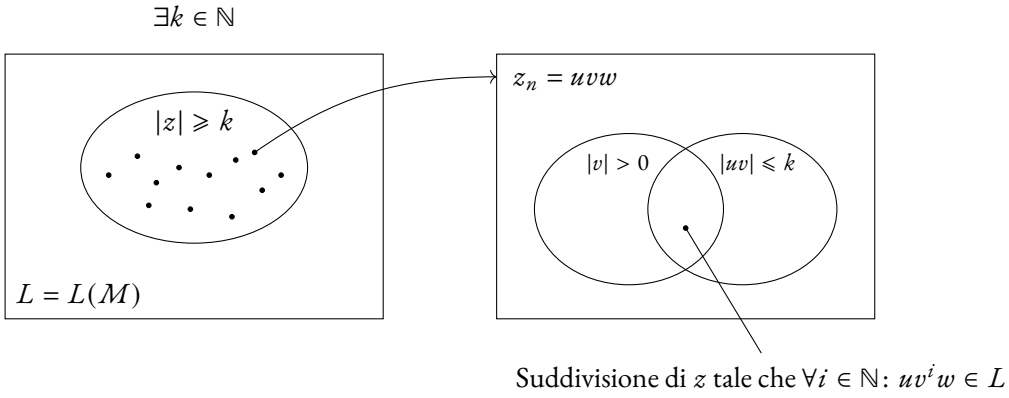
2.6 Pumping Lemma per linguaggi regolari

Se L è un linguaggio regolare, sotto determinate condizioni, si può ripetere un numero illimitato di volte una parte di una stringa di L e ottenere ancora una stringa in L. Questa proprietà è formalizzata dal *Pumping Lemma*, il quale fornisce una condizione necessaria alla regolarità di un linguaggio:

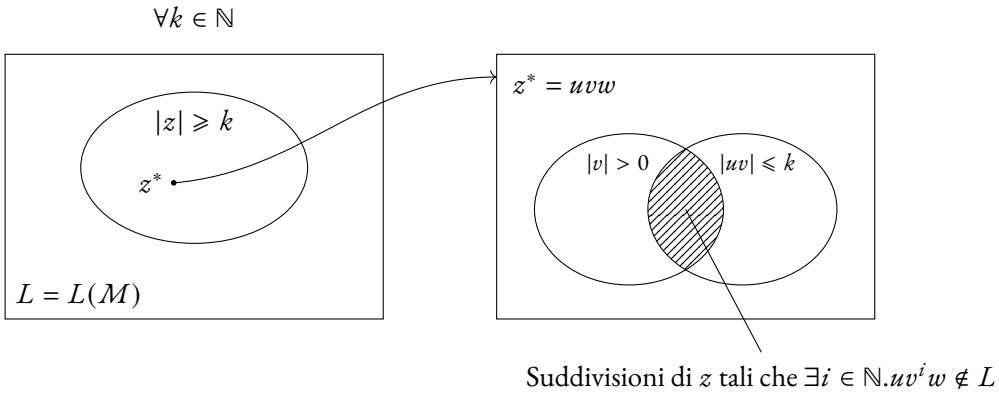
$$\begin{aligned} L \in \mathrm{LR} &\implies \mathrm{PL}(L), \\ \neg\mathrm{PL}(L) &\implies L \notin \mathrm{LR}. \end{aligned} \tag{2.58}$$

Lemma 2 (Pumping Lemma per linguaggi regolari). *Sia L un linguaggio regolare. Allora esiste un intero $k \in \mathbb{N}$ tale che, per ogni $z \in L$ tale che $|z| \geqslant k$, esiste una suddivisione di z in tre sottostringhe uvw tali che:*

1. $|uv| \leqslant k$,
2. $|v| > 0$,
3. $\forall i \in \mathbb{N} : uv^i w \in L$



(A) *Implicazione diretta.*



(B) *Implicazione negata.*

Figura 2.2 *Rappresentazione grafica del Pumping Lemma per linguaggi regolari.*

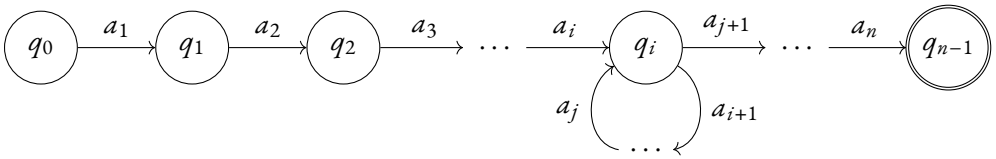
Dimostrazione (Lemma 2). Per il Teorema 2, se L è regolare esiste un DFA associato che riconosce il linguaggio. La tesi da dimostrare è:

$$\begin{aligned} &\exists k \in \mathbb{N}. \forall z \in L. |z| \geqslant k : \\ &\exists u, v, w \in \Sigma^*. z = uvw \wedge |uv| \leqslant k \wedge |v| > 0 \\ &\quad \implies \forall i \in \mathbb{N} : uv^i w \in L. \end{aligned} \tag{2.59}$$

Il DFA M possiede $|Q| = n \in \mathbb{N}$ stati. Prendiamo $k = n$ e $z \in L$, senza introdurre ipotesi su z, tale che $|z| \geqslant k$. Si può vedere z come una concatenazione di simboli su Σ:

$$z = a_1 a_2 \cdots a_h, \quad \text{con } h \geqslant k. \tag{2.60}$$

Poiché z è composta da h transizioni, che per ipotesi sono almeno $k = n$, vengono attraversati almeno $n + 1$ stati. Inoltre, poiché M possiede n stati, per il *principio dei cassetti*, almeno uno stato deve essere attraversato due volte.



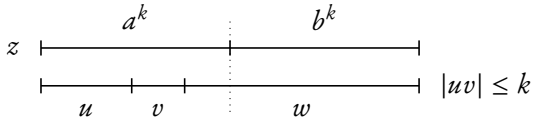
Sia q_i il primo stato visitato due volte. Si pone u pari alla stringa che porta da q_0 alla prima occorrenza di q_i, v la stringa dalla prima occorrenza di q_i alla seconda e w la stringa dalla seconda occorrenza di q_i a q_n. Si dimostra che valgono le seguenti proprietà:

- $|uv| \leqslant k$: se valesse $|uv| > k$, allora anche la stringa uv avrebbe una lunghezza maggiore del numero di stati di M, ma questo significherebbe che almeno un altro stato è stato attraversato due volte prima di q_i. Per ipotesi, invece q_i è il primo stato visitato due volte e quindi non può valere l'ipotesi.
- $|v| > 0$: poiché q_i è stato visitato almeno due volte, esiste un ciclo che parte da q_i e ritorna a q_i. Quindi, poiché v è la parte del ciclo che viene ripetuta, è anche non nulla.
- $\forall i \in \mathbb{N} : uv^i w \in L$: la ripetizione di v nella stringa coincide con l'attraversamento del ciclo. Poiché questa azione è ripetibile un numero indefinito di volte, la stringa ottenuta è ancora riconosciuta dal DFA. □

Esempio. Sia $L = \{ a^n b^n \mid n \in \mathbb{N} \}$ definito su $\Sigma = \{a, b\}$. Fissato $k \in \mathbb{N}$, senza ipotesi, si costruisce $z \in L$ e $|z| \geqslant k$. Si può scrivere z come:

$$z = a^k b^k. \tag{2.61}$$

Questa generica z appartiene al linguaggio dato che $k = k$. Inoltre, la lunghezza di z è $2k \geqslant k$. La suddivisione di z in uvw dovrà avvenire in modo che uv sia il prefisso di a^k, affinché $|uv| \leqslant k$.



(2.62)

La generica stringa z_i ottenuta da $uv^i w$ sarà:[7]

$$\begin{aligned} z_i &= uv^i w \\ &= a^{k+(i-1)|v|} b^k. \end{aligned} \tag{2.63}$$

L'idea per incorrere in una contraddizione è cercare un i tale che $|z_i|_a \neq |z_i|_b$. In particolare, se si pone $i = 2$, si ottiene:

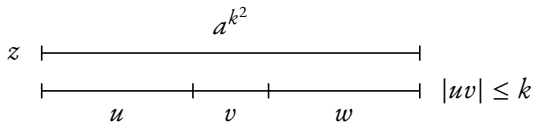
$$z_2 = a^{k+|v|} b^k. \tag{2.64}$$

Si vuole dimostrare che $z_2 \notin L$. Innanzitutto, la condizione per cui la stringa $a^i b^j$ appartenga al linguaggio L è che $i = j$. Fissato questo, si può vedere che:

$$\begin{aligned} z_2 = a^{k+|v|} b^k \in L &\iff k + |v| = k \\ &\iff |v| = 0 \end{aligned} \tag{2.65}$$

e questo viola l'ipotesi che $|v| > 0$. Pertanto, L non è regolare.[8] ◇

Esempio. Sia $L = \{ a^n \mid \exists m \in \mathbb{N}. n = m^2 \} = \{ a^{m^2} \mid m \in \mathbb{N} \}$ un linguaggio definito su $\Sigma = \{a\}$. In questo caso, rispetto all'esempio precedente, non è possibile confrontare porzioni di stringa.



(2.66)

Per non imporre condizioni su k, si scrive z come:

$$z = a^{k^2}. \tag{2.67}$$

Si verifica che valgano le ipotesi di appartenenza al linguaggio e di lunghezza su z:

$$z = a^{k^2} \in L, \qquad \forall k \in \mathbb{N} \colon |z| = k^2 \geqslant k. \tag{2.68}$$

Si scrive la generica stringa z_i come:

$$z_i = uv^i w = a^{k^2+(i-1)|v|} \in L. \tag{2.69}$$

Ancora una volta, per $i = 2$ si ottiene:

$$z_2 = a^{k^2+|v|} \in L \iff \exists j \in \mathbb{N}. k^2 + |v| = j^2. \tag{2.70}$$

Poiché la funzione parabolica è monotona crescente su $\mathbb{N}$, si può sempre scrivere:

$$\forall k, j \in \mathbb{N} \colon k^2 < j^2 \iff k < j. \tag{2.71}$$

Quindi, si può riscrivere j come $k + m$, per un certo $m \in \mathbb{N}$:

$$\begin{aligned} k^2 + |v| &= (k + m)^2 \\ &= k^2 + 2km + m^2 \\ |v| &= 2km + m^2 > k. \end{aligned} \tag{2.72}$$

Dalle condizioni del teorema, tuttavia, è noto che nella separazione della stringa vale la relazione $|uv| \leqslant k$. Per questo, si deve avere $|v| \leqslant k$, in contraddizione con il risultato ottenuto precedentemente.

Si può concludere che L non è regolare. ◇

[7] Nel contare le occorrenze di v si deve escludere la prima perché è già inclusa in k.

[8] Il livello di memoria che un DFA fornisce non è sufficiente, in questo caso, per far sì che le iterazioni di a e b siano in egual numero. Infatti, la possibilità di percorrere il ciclo su v infinite volte induce la perdita del conteggio degli attraversamenti.

3 Linguaggi liberi dal contesto

Definizione (Grammatica libera dal contesto). *Una grammatica Context Free* (CF) *è una quadrupla:*

$$G = \langle V, T, P, S \rangle \tag{3.1}$$

dove:

- V *è l'insieme finito dei simboli non terminali.*
- T *è l'insieme finito dei simboli terminali.*
- P *è l'insieme finito delle produzioni.*
- $S \in V$ *è il simbolo iniziale.*

Il significato di S dopo l'espansione è il linguaggio generato da G.

L'insieme delle produzioni contiene elementi della forma $A \to \alpha$, dove $A \in V$ e $\alpha \in (V \cup T)^*$.[1] In altri termini, A è un simbolo non terminale che viene sostituito con α, il quale a sua volta è una sequenza di simboli terminali o non terminali. Secondo la definizione, dunque, ogni volta che si trova il simbolo A nella stringa, esso può essere sostituito con α, indipendentemente dai simboli che precedono e seguono A, ovvero il suo *contesto*.

Esempio. La grammatica:

$$G = \langle \{E\}, \{ \text{or, not, and, (,), 0, 1} \}, P, E \rangle \tag{3.2}$$

è la grammatica che genera il linguaggio delle espressioni booleane. Le produzioni sono:

- $E \to 0$,
- $E \to 1$,
- $E \to (E \text{ or } E)$,
- $E \to (E \text{ and } E)$,
- $E \to (\text{not } E)$.

Una scrittura alternativa prevede di scrivere le diverse produzioni separandole con una barra verticale (OR):

$$E \to 0 \mid 1 \mid (E \text{ or } E) \mid (E \text{ and } E) \mid (\text{not } E). \tag{3.3}$$

Ad esempio, la stringa:

$$(\text{not } ((0 \text{ and } 1) \text{ or } 0)) \tag{3.4}$$

può essere ottenuta nel seguente modo:

$$\begin{aligned} E &\to (\text{not } E) \to (\text{not } (E \text{ or } E)) \to (\text{not } (E \text{ or } 0)) \to (\text{not } ((E \text{ and } E) \text{ or } 0)) \\ &\to (\text{not } ((0 \text{ and } E) \text{ or } 0)) \to (\text{not } ((0 \text{ and } 1) \text{ or } 0)). \end{aligned} \tag{3.5}$$

◇

[1] Le produzioni sono l'equivalente della funzione di transizione ∂ per i linguaggi regolari.

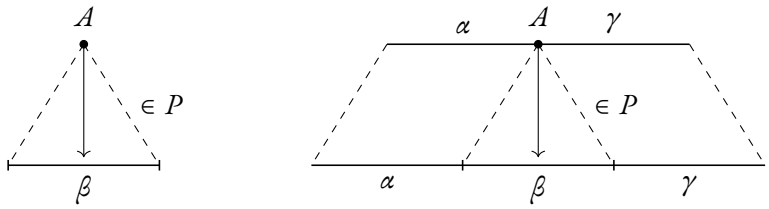
Simbolo	Significato
$\Rightarrow / \rightarrow$	Derivazione immediata
$\Rightarrow_n$	Derivazione applicando esattamente n produzioni in sequenza
$\Rightarrow_*$	Derivazione applicando un certo numero di produzioni in sequenza

Tabella 3.1 *Tipologie di derivazione.*

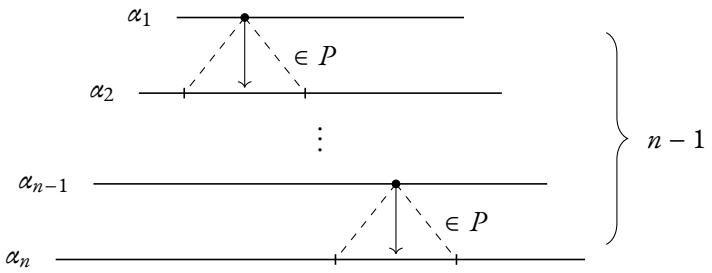
Definizione (Derivazione immediata). *Sia $A \rightarrow \beta \in P$ una produzione. Allora si dice che in:*

$$\alpha A \gamma \overset{G}{\Rightarrow} \alpha \beta \gamma \quad \text{con } \alpha, \gamma \in (V \cup T)^* \tag{3.6}$$

$\alpha\beta\gamma$ è immediatamente derivato da $\alpha A \gamma$.



(a) *Derivazione immediata.*



(b) *Derivazione in più passi.*

Figura 3.1 *Derivazioni in una grammatica* CF.

Definizione (Derivazione in più passi). *Siano $\alpha_1, \alpha_2, \ldots, \alpha_n \in (V \cup T)^*$ tali che:*

$$\forall j \in [1, n-1] : \alpha_j \rightarrow \alpha_{j+1}. \tag{3.7}$$

Allora si dice che α_1 deriva α_n in n passi.

Definizione (Linguaggio generato da G). *Si definisce il linguaggio generato da G come:*

$$L(G) = \{ w \in T^* \mid S \Rightarrow_* w \}. \tag{3.8}$$

Le stringhe contenute in $L(G)$ sono tutte e sole le sequenze di simboli terminali, ovvero non ulteriormente espandibili, generate a partire da S e applicando transitivamente produzioni di G.

Definizione (Linguaggio CF). *Un linguaggio $L \subseteq T^*$ è un linguaggio* CF *se esiste una grammatica* CF *G tale che $L = L(G)$.*

Esempio. Si consideri il linguaggio:

$$L = \{\, 0^n 1^n \mid n \in \mathbb{N} \,\}. \tag{3.9}$$

La grammatica che lo genera è:

$$S \to 0S1 \mid \varepsilon. \tag{3.10}$$

Infatti, le stringhe appartenenti a L sono della forma:

$$\begin{aligned} &S \to \varepsilon \\ &S \to 0S1 \to 01 \\ &S \to 0S1 \to 00S11 \to 0011 \\ &\quad\vdots \\ &S \Rightarrow_n 0^n S 1^n. \end{aligned} \tag{3.11}$$

Come per i linguaggi regolari, è necessario dimostrare che il linguaggio generato dalla grammatica è effettivamente L. Questa operazione viene svolta in due passaggi:

a.
$$\begin{aligned} & L \subseteq L(G) \\ \iff\ & x \in L \implies x \in L(G) \\ \iff\ & x \in L \implies S \Rightarrow_* x. \end{aligned}$$
Per induzione sulla lunghezza di x.

b.
$$\begin{aligned} & L(G) \subseteq L \\ \iff\ & x \in L(G) \implies x \in L \\ \iff\ & S \Rightarrow_n x \implies x \in L. \end{aligned}$$
Per induzione sui passi della derivazione.

a. Si vuole dimostrare la seguente tesi:

$$\forall x \in T^*. x \in L \implies S \Rightarrow_* x. \tag{3.12}$$

Base Si deve cercare la lunghezza minima delle stringhe appartenenti a L.[2] In questo caso, la lunghezza da considerare è $|x| = 0$, ovvero $x = \varepsilon$. Pertanto:

$$|x| = 0 \colon x = \varepsilon \in L \quad \text{e} \quad S \to \varepsilon. \tag{3.13}$$

Poiché esiste una derivazione, la base è verificata.

Passi induttivo L'ipotesi induttiva è analoga alla tesi, ma limita l'insieme di definizione attraverso n:

$$\text{Hp Ind}\colon \forall x \in T^*. |x| \leqslant n \colon x \in L \implies S \Rightarrow_* x. \tag{3.14}$$

Dimostriamo quindi che per ogni $x \in T^*. |x| = n + 1$ vale la tesi.

Sia, appunto, $x \in T^*$ tale che $|x| = n + 1$. La stringa è della forma $x = 0^i 1^i \in L$. Si prende, in particolare, i in modo che $|x| = 2i = n + 1$, ovvero la lunghezza della stringa sia $n + 1$. Si riscrive ora x in una forma alternativa:

$$x = 00^{i-1}1^{i-1}1. \tag{3.15}$$

[2] A differenza dei linguaggi regolari, non è necessario che vi siano anche stringhe non appartenenti a L per tale lunghezza.

Se si pone $x' = 0^{i-1}1^{i-1} \in L$, si ha che $x = 0x'1$. Per l'ipotesi induttiva, $S \Rightarrow_* x'$, dato che:

$$|x'| = |x| - 2 = n + 1 - 2 = n - 1 < n. \tag{3.16}$$

Per come è fatta la grammatica, l'ultima derivazione deve essere $S \to \varepsilon$. Inoltre, l'ultima S separa gli 0 dagli 1. Quindi:

$$S \Rightarrow_* 0^{i-1}1^{i-1} \equiv S \Rightarrow_* 0^{i-1}S1^{i-1} \to 0^{i-1}\varepsilon 1^{i-1} = 0^{i-1}1^{i-1}. \tag{3.17}$$

Si modifica quest'ultima derivazione per generare x, ovvero, invece di sostituire ε si sostituisce $0S1$:

$$S \Rightarrow_* 0^{i-1}S1^{i-1} \to 0^{i-1}0S11^{i-1} = 0^iS1^i \to 0^i\varepsilon 1^i = 0^i1^i. \tag{3.18}$$

Si è riusciti a derivare x partendo da S, quindi la tesi è verificata.

b. Si vuole dimostrare la seguente tesi:

$$\forall x \in T^*\colon S \Rightarrow_i x \implies x \in L. \tag{3.19}$$

Base Si considera la lunghezza di derivazione più corta che porta da S a soli simboli terminali:

$$n = 1\colon S \Rightarrow_1 \varepsilon \quad \text{e} \quad \varepsilon \in L. \tag{3.20}$$

Passo induttivo Si considera l'ipotesi induttiva:

$$\forall x \in T^*\colon S \Rightarrow_i x \implies x \in L \quad \text{con } i \leqslant n \in \mathbb{N}. \tag{3.21}$$

Ovvero, tutte le derivazioni più corte (o di uguale lunghezza) di n passi generano stringhe appartenenti a L. Bisogna dimostrare che $S \Rightarrow_{n+1} x \implies x \in L$.

Sia $S \Rightarrow_{n+1} x$ una derivazione in $n+1$ passi che genera x. Per come è definita la grammatica, questa derivazione termina con $S \to \varepsilon$. Esistono dunque x_1 e x_2 tali che:

$$S \Rightarrow_{n+1} x \equiv S \Rightarrow_n x_1Sx_2 \to x_1\varepsilon x_2 = x_1x_2. \tag{3.22}$$

Sempre secondo la definizione della grammatica, la variabile S compresa tra x_1 e x_2 in (3.22) può essere generata solo da $0S1$. Quindi:

$$S \Rightarrow_{n+1} x \equiv \underbrace{\underbrace{S \Rightarrow_{n-1} x_1'Sx_2' \to x_1'0S1x_2'}_{n} \to x_1\varepsilon x_2 = x.}_{n+1} \tag{3.23}$$

A questo punto sappiamo che:

- $x = x_1x_2 = x_1'01x_2'$,
- $S \Rightarrow_{n-1} x_1'Sx_2' \to x_1'x_2'$, ovvero che si genera $x_1'x_2'$ in n passi di derivazione.

Si applica ora l'ipotesi induttiva: siano $x_1', x_2' \in L$. Per come la grammatica genera le stringhe, S separa gli 0 dagli 1, ovvero x_1' è composto da soli 0 e x_2' da soli 1. Poiché gli 0 e gli 1 vengono sempre aggiunti a coppie, esiste $j \in \mathbb{N}$ tale che $x_1' = 0^j$ e $x_2' = 1^j$. A questo punto, per costruzione vale:

$$x = x_1'01x_2' = 0^j011^j = 0^{j+1}1^{j+1} \in L. \tag{3.24}$$

$\diamond$

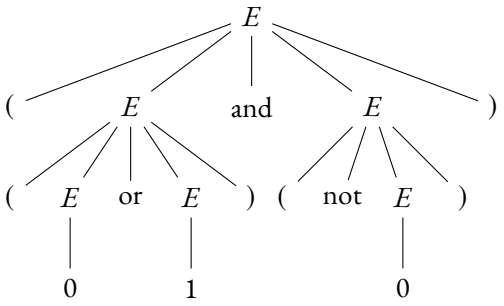
3.1 Alberi di derivazione

Definizione (Albero di derivazione). *Un albero di derivazione per G, chiamato anche* parse tree*, è un albero i cui nodi sono etichettati con simboli in $T \cup V \cup \{\varepsilon\}$ nel seguente modo:*

- *La radice è etichettata con un simbolo non terminale in V.*
- *Ogni vertice interno è etichettato con un simbolo non terminale in V.*
- *Se un vertice è etichettato con $A \in V$ e $n_1, \ldots, n_k$ sono i figli (da sinistra a destra), allora, per ogni i, n_i è etichettato con x_i e $A \to x_1 \cdots x_k \in P$.*
- *Se un vertice ha etichetta ε, allora è una foglia ed è l'unico figlio del padre.*

Se l'albero possiede solo foglie etichettate con simboli terminali, allora la stringa generata è esattamente la sequenza delle foglie letta da sinistra verso destra.

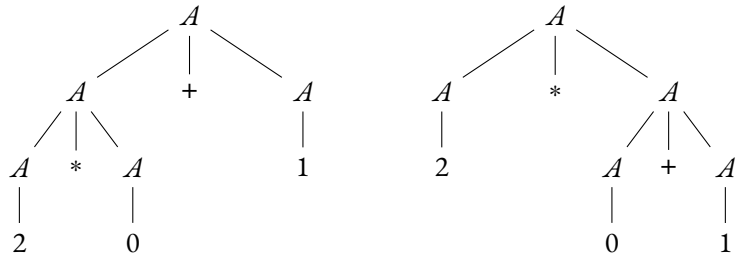
Esempio. Se si considera la stringa (3.3), si può costruire il seguente albero di derivazione:



La stringa associata è data dalla concatenazione delle foglie lette da sinistra a destra. ◇

Definizione (Grammatica ambigua). *Se una derivazione, o in particolare una stringa generata, corrisponde a più alberi di derivazione, la grammatica è detta* ambigua*.*

Esempio. Si consideri la grammatica $A \to A * A \mid A + A \mid 0 \mid 1 \mid 2$. La stringa $s = 2 * 0 + 1$ può essere generata dai seguenti alberi di derivazione:



(A) (B)

Essendoci più di una possibile derivazione che produce s, la grammatica è ambigua. ◇

Teorema 6. *Se $G = \langle V, T, P, S\rangle$ è una grammatica* CF*, allora $S \Rightarrow_* x$, $x \in T^*$ se e solo se esiste un albero di derivazione avente radice etichettata con S e foglie $x = x_1 x_2 \cdots x_n$, lette da sinistra verso destra.*

Possiamo descrivere proprietà della grammatica in funzione degli alberi di derivazione che si costruiscono a partire da essa. L'albero, dunque, rappresenta un insieme di derivazioni, che può variare nell'ordine di applicazione delle produzioni (strategia di derivazione).

Definizione (Linguaggio intrinsecamente ambiguo). *Un linguaggio L è detto* intrinsecamente ambiguo *se è generabile solo da grammatiche ambigue.*

3.2 Semplificazione di grammatiche CF

A differenza dei linguaggi regolari, per cui l'esistenza di un automa minimo che li riconoscesse era garantita dal Teorema 5, nel caso di linguaggi CF non esiste il concetto di grammatica minima. Tuttavia, è possibile eseguire manipolazioni delle grammatiche per ottenere le cosiddette *forme normali*. In particolare, vi sono due forme normali rilevanti:

Forma normale di Chomsky È necessaria per utilizzare e dimostrare il Pumping Lemma per linguaggi CF. Prevede che le produzioni siano di due tipi:

$$A \to BC \quad \text{oppure} \quad A \to a, \qquad \text{con } A, B, C \in V,\ a \in T. \tag{3.25}$$

Forma normale di Greibach È necessaria per la dimostrazione dell'equivalenza tra grammatiche CF e automi a pila (ovvero DFA, NFA e ε-NFA con stack). Prevede che le produzioni siano di un tipo:

$$A \to a\alpha, \qquad \text{con } a \in T,\ \alpha \in V^*. \tag{3.26}$$

In questo caso, α rappresenta la pila.

Portare una grammatica in forma normale significa dunque rimuovere da essa produzioni che sono *inutili*, nel senso che non contribuiscono a generare il linguaggio, e *ridondanti*, nel senso che possono essere ottenute da altre produzioni. Le operazioni di normalizzazione sono di tre tipologie:

Eliminazione dei simboli inutili Innanzitutto, cosa rende inutile un simbolo?

Definizione (Simbolo utile). *Il simbolo non terminale $X \in V$ è utile se $S \Rightarrow_* \alpha X \beta$, ovvero se X può essere raggiunto da S, e se $X \Rightarrow_* w \in T^*$, ovvero se da X si può produrre una stringa di soli terminali.*

L'eliminazione dei simboli inutili consta di due singole operazioni di eliminazione, coincidenti con le due condizioni di cui sopra.

Prima eliminazione Si vogliono eliminare tutti i simboli che non raggiungono simboli terminali.

Teorema 7. *Sia $G = \langle V, T, P, S\rangle$ una grammatica* CF *che non genera il linguaggio vuoto $\emptyset$. Allora esiste un algoritmo per costruire $G' = \langle V', T, P', S\rangle$ tale che $L(G) = L(G')$ e:*

$$\forall x \in V' \colon \exists w \in T^* . x \Rightarrow_* w. \tag{3.27}$$

Si definisce un operatore Γ induttivamente:

$$\Gamma(W) = \{\, A \in V \mid \exists \alpha \in (T \cup W)^* \colon A \to \alpha \in P \,\}, \tag{3.28}$$

$$\begin{cases} \Gamma^0(W) = W \\ \Gamma^{n+1}(W) = \Gamma(\Gamma^n(W)). \end{cases} \tag{3.29}$$

L'eliminazione dei simboli inutili avviene costruendo $\Gamma^i(\emptyset)$ e ponendo $V' = \Gamma^{|V|}(\emptyset)$. Considerando, infatti, la grammatica:

$$\begin{aligned} G' &= \{ \Gamma^{|V|}, T, P', S \}, \\ P' &= \{ A \to \alpha \in P \mid A \in V' \wedge \alpha \in V \cup T \}, \end{aligned} \tag{3.30}$$

si ottiene l'equivalente di G, ma con i simboli inutili eliminati.

L'algoritmo termina sicuramente entro $|V|$ passi, dato che nel caso peggiore si inserisce un simbolo di V alla volta in V' a ogni passo.

Seconda eliminazione L'obiettivo di questa eliminazione è rimuovere tutti i simboli presenti in $V \cup T$ e non raggiunti da S.

Teorema 8. *Sia $G = \langle V, T, P, S \rangle$ una grammatica* CF *che non genera il linguaggio vuoto $\emptyset$. Allora esiste un algoritmo per costruire $G' = \langle V', T', P', S' \rangle$ tale che $L(G) = L(G')$ e:*

$$\forall X \in (V' \cup T') \colon \exists \alpha, \beta \in (V' \cup T')^* . S' \Rightarrow_* \alpha X \beta. \tag{3.31}$$

Anche in questo caso si definisce un operatore Γ:

$$\Gamma(W) = \{ X \in (V \cup T) \mid \exists A \in W . A \to \alpha X \beta \in P,\ \alpha, \beta \in (V \cup T)^* \} \cup \{S\}, \tag{3.32}$$

$$\begin{cases} \Gamma^0(W) = W \\ \Gamma^{n+1}(W) = \Gamma(\Gamma^n(W)). \end{cases} \tag{3.33}$$

Infine, si considera la grammatica:

$$G' = \begin{cases} V' = \Gamma^{|V|+|T|}(\emptyset) \cap V \\ T' = \Gamma^{|V|+|T|}(\emptyset) \cap T \\ P' = \{ A \to \alpha \in P \mid A \in V', \alpha \in (V' \cup T')^* \} \\ S' = S. \end{cases} \tag{3.34}$$

Come per il caso precedente, l'algoritmo termina sicuramente in $|V|+|T|$ passi, corrispondenti al massimo numero di simboli inseribili in V' e T'.

Eliminazione delle ε-produzioni Con questo passaggio si vogliono eliminare tutte le produzioni e simboli non terminali annullabili, cioè che portano a ε. In particolare, se $\varepsilon \in L$, l'unica produzione ammessa è $S \to \varepsilon$.

Teorema 9. *Data $G = \langle V, T, P, S \rangle$ e chiamato $L = L(G)$, allora $L \setminus \{\varepsilon\}$ può essere generato da $G' = \langle V', T, P', S \rangle$ tale che G' è senza simboli inutili e senza ε-produzioni.*

Ancora una volta si definisce l'operatore Γ come:

$$\Gamma(W) = \{ A \in V \mid \exists A \to \alpha \in P, \text{“}\alpha \text{ privata delle variabili in } W \text{ è } \varepsilon\text{”} \}, \tag{3.35}$$

$$\begin{cases} \Gamma^0(W) = W \\ \Gamma^{n+1}(W) = \Gamma(\Gamma^n(W)). \end{cases} \tag{3.36}$$

Quindi, $\alpha \in V^*$ contiene solo variabili in W. Nel caso peggiore si inseriscono tutti i simboli non terminali in Γ, uno alla volta, e quindi l'algoritmo termina dopo $|V|$ passi.

La nuova grammatica sarà quindi composta da:

$$G' = \begin{cases} V' = \Gamma^{|V|}(\emptyset) \\ T = T \\ P' = \left\{ A \to \alpha_1 \cdots \alpha_n \;\middle|\; A \to x_1 \cdots x_n \in P, \begin{array}{l} \text{se } x_i \in \Gamma^{|V|}(\emptyset) \text{ allora } \alpha_i = x_i \text{ e } \alpha_i = \varepsilon \\ \text{se } x_i \notin \Gamma^{|V|}(\emptyset) \text{ allora } \alpha_i = x_i \end{array} \right\} \\ S = S. \end{cases} \tag{3.37}$$

Eliminazione delle produzioni unitarie Con questa semplificazione si intende eliminare tutte le produzioni che da un simbolo non terminale vanno in un altro non terminale.

Teorema 10. *Ogni grammatica* CF $G = \langle V, T, P, S \rangle$ *che non genera il linguaggio vuoto* $\emptyset$ *può essere trasformata in una grammatica* $G' = \langle V', T, P', S \rangle$ *tale che* $L(G) = L(G')$ *e* P' *non contiene simboli inutili,* ε*-produzioni e produzioni unitarie.*

L'algoritmo per l'eliminazione delle produzioni unitarie è il seguente:

```
1  for A → A ∈ P
2      P' := P \ { A → A }
3      while ∃A → B ∈ P
4          P := P \ { A → B }
5          for B → α ∈ P
6              if α ≠ A
7                  P' := P' ∪ { A → α }
```

Esempio. Si consideri la grammatica da normalizzare:

$$G = \langle \{ A, B, C, D, E, S \}, \{ 0, 1, 2, 3 \}, P, S \rangle, \tag{3.38}$$

$$P = \begin{cases} S \to \varepsilon \mid A \mid B \mid D \\ A \to 0 \mid 0A0 \mid B \mid E \\ B \to 1 \mid 1B1 \mid A \mid \varepsilon \\ C \to A \mid B \mid 2 \\ D \to \varepsilon \mid D \\ E \to \varepsilon. \end{cases} \tag{3.39}$$

Si eseguono tutte le semplificazioni possibili:

Eliminazione dei simboli inutili

1. Si esegue la prima eliminazione costruendo $\Gamma^i(\emptyset)$:

$$\Gamma^0(\emptyset) = \emptyset, \tag{3.40}$$

$$\begin{aligned} \Gamma^1(\emptyset) &= \{ X \in (V \cup \emptyset) \mid \text{“da } X \text{ si raggiungono simboli terminali”} \} \\ &= \{ A, B, C \}, \end{aligned} \tag{3.41}$$

$$\Gamma^2(\emptyset) = \Gamma(\Gamma(\emptyset)) = \Gamma(\{ A, B, C \}) = \{ A, B, C, S \}, \tag{3.42}$$

$$\Gamma^3(\emptyset) = \Gamma(\Gamma^2(\emptyset)) = \Gamma(\{ A, B, C, S \}) = \{ A, B, C, S \}. \tag{3.43}$$

Poiché $\Gamma^2(\emptyset) = \Gamma^3(\emptyset)$, si può terminare. Si evince che i simboli non terminali da rimuovere sono D ed E, sia come produzioni a sé stanti che come simboli raggiungibili. Si ottiene quindi la grammatica:

$$G' = \langle \{ A, B, C, S \}, \{ 0, 1, 2, 3 \}, P', S \rangle, \tag{3.44}$$

$$P' = \begin{cases} S \to \varepsilon \mid A \mid B \\ A \to 0 \mid 0A0 \mid B \\ B \to 1 \mid 1B1 \mid A \mid \varepsilon \\ C \to A \mid B \mid 2. \end{cases} \tag{3.45}$$

2. Ora si esegue la seconda eliminazione:

$$\Gamma^0(\emptyset) = \emptyset, \tag{3.46}$$
$$\Gamma^1(\emptyset) = \Gamma^0(\emptyset) \cup \{S\} = \{S\}, \tag{3.47}$$
$$\Gamma^2(\emptyset) = \{ X \in (V \cup T)^* \mid \text{“da } S \text{ si raggiunge } X\text{”} \} \cup \{S\} = \{ A, B, S \}, \tag{3.48}$$
$$\Gamma^3(\emptyset) = \Gamma(\Gamma^2(\emptyset)) = \Gamma(\{ A, B, S \}) = \{ A, B, S, 0, 1 \}. \tag{3.49}$$

Poiché in $\Gamma^3(\emptyset)$ non sono stati aggiunti simboli non terminali rispetto a $\Gamma^2(\emptyset)$, si può terminare. La nuova grammatica è:

$$G'' = \langle \{ A, B, S \}, \{0, 1\}, P'', S \rangle, \tag{3.50}$$

$$P'' = \begin{cases} S \to \varepsilon \mid A \mid B \\ A \to 0 \mid 0A0 \mid B \\ B \to 1 \mid 1B1 \mid A \mid \varepsilon. \end{cases} \tag{3.51}$$

Eliminazione delle ε-produzioni Come per il caso precedente, si costruisce Γ:

$$\Gamma^0(\emptyset) = \emptyset, \tag{3.52}$$
$$\Gamma^1(\emptyset) = \{S, B\}, \tag{3.53}$$
$$\begin{aligned} \Gamma^2(\emptyset) &= \Gamma(\Gamma^1(\emptyset)) = \{ \text{“Non terminali che vanno in } \varepsilon \text{ o } (S \vee B)\text{”} \} \\ &= \{ S, B, A \}. \end{aligned} \tag{3.54}$$

Poiché non rimangono altri simboli non terminali da aggiungere si può terminare. La nuova grammatica è, dunque

$$G''' = \langle \{ A, B, S \}, \{0, 1\}, P''', S \rangle, \tag{3.55}$$

$$P''' = \begin{cases} S \to \varepsilon \mid A \mid B \\ A \to 0 \mid 0A0 \mid 00 \mid B \\ B \to 1 \mid 1B1 \mid 11 \mid A \end{cases} \tag{3.56}$$

dove, in P''', sono state inserite sia le produzioni originali che con la sostituzione di ε, nel caso in cui il simbolo di partenza fosse contenuto in $\Gamma^2(\emptyset)$. In questo caso, poiché $\Gamma^2(\emptyset) = P''$, la trasformazione è stata applicata a tutti i simboli non terminali.

Eliminazione delle produzioni unitarie Si esegue l'algoritmo per l'eliminazione delle produzioni unitarie. Nella pratica è sufficiente espandere le produzioni unitarie in P''', osservando a cosa corrispondono i simboli non terminali nelle altre derivazioni:

$$S \to \varepsilon \mid A \mid B \tag{3.57}$$

$$\downarrow$$

$$S \to \varepsilon \mid 0 \mid 00 \mid 0A0 \mid 1 \mid 11 \mid 1B1. \tag{3.58}$$

In questo passaggio è stato espanso B, andando a sostituire tutte le sue produzioni in S. Ora si modificano A e B:

$$\begin{aligned} A &\to 0 \mid 00 \mid 0A0 \mid 1 \mid 11 \mid 1B1, \\ B &\to 1 \mid 11 \mid 1B1 \mid 0 \mid 00 \mid 0A0. \end{aligned} \tag{3.59}$$

Si osserva che, al netto della produzione $S \to \varepsilon$, la (3.58) e la (3.59) sono equivalenti.

La grammatica finale può quindi essere espressa in funzione della sola S:

$$G = \langle \{ A, B, S \}, \{0, 1\}, P^{(4)}, S \rangle \tag{3.60}$$

$$P^{(4)} = \begin{cases} S \to \varepsilon \\ S \to 0 \mid 00 \mid 0S0 \mid 1 \mid 11 \mid 1S1. \end{cases} \tag{3.61}$$

◇

3.3 Forma normale di Chomsky

Definizione (Forma normale di Chomsky). *Ogni linguaggio* CF L *tale che* $\varepsilon \notin L$ *è generabile da una grammatica in cui tutte le produzioni sono nella forma:*

- $A \to BC$ *con* $A, B, C \in V$.
- $A \to a$ *con* $A \in V$ *e* $a \in T$.

Una grammatica contenente solo produzioni in queste forme è detta in forma normale di Chomsky.

Esempio. Proseguendo l'esempio precedente, le seguenti produzioni sono in forma normale di Chomsky:

$$\begin{aligned} S &\to \varepsilon, \\ S &\to 0 \mid 1. \end{aligned} \tag{3.62}$$

Le seguenti produzioni non sono in forma normale di Chomsky:

$$\begin{aligned} S &\to 00, \\ S &\to 11. \end{aligned} \tag{3.63}$$

Per renderle tali, si devono definire nuovi simboli non terminali:

$$\begin{aligned} V_1 &\to 0, \\ S &\to V_1 V_1, \end{aligned} \tag{3.64}$$

$$\begin{aligned} V_2 &\to 1, \\ S &\to V_2 V_2. \end{aligned} \tag{3.65}$$

Anche le seguenti produzioni non sono in forma normale di Chomsky:

$$\begin{aligned} S &\to 0S0, \\ S &\to 1S1. \end{aligned} \tag{3.66}$$

Si possono combinare le precedenti produzioni per ottenere:

$$\begin{aligned} V_3 &\to V_1 S, \\ S &\to V_3 V_1, \end{aligned} \tag{3.67}$$

$$\begin{aligned} V_4 &\to V_2 S, \\ S &\to V_4 V_2. \end{aligned} \tag{3.68}$$

◇

Proposizione. *Ogni albero di derivazione di una grammatica* CF *in forma normale di Chomsky è binario.*

3.4 Pumping Lemma per linguaggi CF

Lemma 3 (Pumping Lemma per linguaggi CF). *Se L è un linguaggio* CF*, allora esiste un intero $k \in \mathbb{N}$ tale che, per ogni $z \in L$ tale che $|z| \geqslant k$, esiste una suddivisione di z in cinque stringhe $uvwxy$ tali che:*

1. $|vwx| \leqslant k$,
2. $|vx| > 0$,
3. $\forall i \in \mathbb{N}\colon uv^i wx^i y \in L$.

Dimostrazione (Lemma 3). Sia L un linguaggio CF. Allora esiste una grammatica $G = \langle V, T, P, S\rangle$ tale che $L = L(G)$ e G è in forma normale di Chomsky. Sia $n = |V|$ il numero di simboli non terminali.

Poiché la grammatica è in forma di Chomsky, l'albero di derivazione di una stringa è sempre binario (Figura 3.3a). Prendiamo, in particolare, $k = 2^n$, e una stringa z nel linguaggio tale che $|z| \geqslant k = 2^n$. Dato che $|z| \geqslant 2^n$, l'altezza h dell'albero corrispondente è almeno $n + 1$.[3] Con altezza si intende il numero massimo di archi che collegano la radice alle foglie.

Ma se l'altezza h è almeno $n+1$, significa che nel percorso più lungo si attraversano $n+2$ nodi, di cui $n+1$ non terminali. Poiché per costruzione si hanno n nodi non terminali, per il principio dei cassetti, almeno un simbolo non terminale è ripetuto due volte. Sia A il primo simbolo non terminale ripetuto partendo da S. Questo può trovarsi, al più, come primo simbolo, e perciò l'altezza del sottoalbero associato non può essere superiore a h. Le foglie di tale sottoalbero compongono la sottostringa vwx, che avrà quindi lunghezza inferiore o uguale a 2^n, ovvero k. Inoltre, la seconda ripetizione di A deve trovarsi ad almeno una produzione di distanza dalla prima. Da uno dei due non terminali si genererà il secondo A, mentre dall'altro (o da entrambi, prima di A), si genereranno v e/o x, cioè $|vx| > 0$.

Infine, poiché A si ripete più in basso nell'albero è possibile derivare nuovamente un numero illimitato di volte la stringa vwx, ripetendo di fatto v e x (Figura 3.4). □

Esempio. Si consideri il linguaggio:

$$L = \{\, a^n b^n c^n \mid n \in \mathbb{N} \,\}. \tag{3.69}$$

Intuitivamente, non è possibile scrivere una grammatica che generi tale linguaggio, in quanto vi sono

[3] L'incremento di un'unità è dovuto all'inserimento dei nodi terminali nella stringa. Infatti, i primi n livelli sono ottenuti dall'espansione binaria dei nodi non terminali.

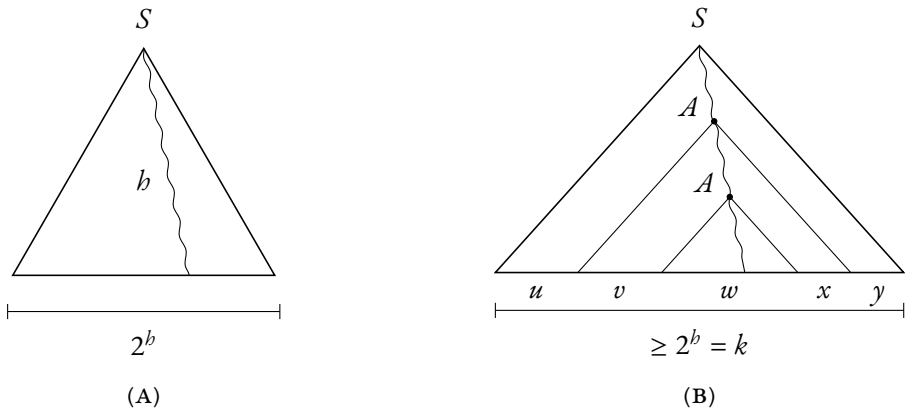


Figura 3.3 *Alberi di derivazione in forma normale di Chomsky nella dimostrazione del* Lemma 3.

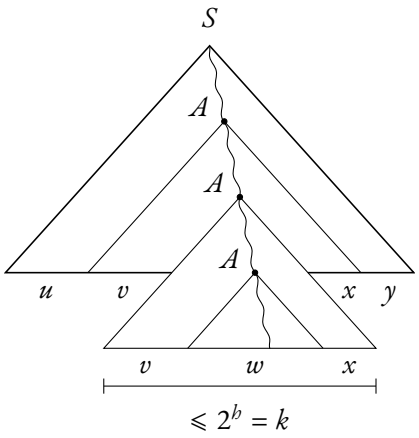
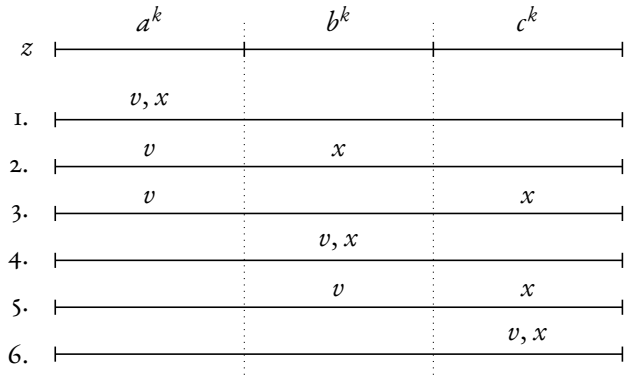


Figura 3.4 *Iterazione illimitata di v e x in un albero di derivazione.*

tre vincoli sugli esponenti. Si vuole quindi dimostrare che il linguaggio è non CF.

Per prima cosa, si sceglie una $k \in \mathbb{N}$ senza introdurre ipotesi su di essa. Si sceglie poi una stringa nel linguaggio tale che $|z| \geqslant k$. Si può prendere $z = a^k b^k c^k$, dato che $|z| = 3k \geqslant k$.

Una generica stringa del tipo $a^i b^j c^k$ appartiene al linguaggio se e solo se $i = j = h$. Poiché è richiesto che si mostri un pompaggio di z che non appartiene a L per ogni possibile suddivisione di z in cinque parti $uvwxy$, si considerano tutte:



(3.70)

Se v e/o x sono a cavallo di gruppi di simboli, per ogni $i \geqslant 2$ la stringa pompata non appartiene a L. Ad esempio, se $v = ab$ e $x = bc$ in z:

$$\begin{aligned} z &= aaaabbbbcccc \in L, \\ i = 2\colon z_2 &= aaaababbbbcbcccc \notin L. \end{aligned} \tag{3.71}$$

Di conseguenza, non si considerano mai v e x in queste posizioni perché sarebbe banale.

Si analizzano, dunque, i singoli casi:

1. $v, x \in a^k$. Se si prende $i = 2$:

$$z_2 = uv^2wx^2y = a^{k+|vx|}b^kc^k \tag{3.72}$$

   appartiene a L se e solo se $k + |vx| = k = k$, ovvero se $|vx| = 0$. $\Rightarrow\Leftarrow$

2. $v \in a^k, x \in b^k$. Se si prende $i = 2$:

$$z_2 = uv^2wx^2y = a^{k+|v|}b^{k+|x|}c^k \tag{3.73}$$

   appartiene a L se e solo se $k + |v| = k + |x| = k$, ovvero se:

$$\begin{cases} k + |v| = k \\ k + |x| = k. \end{cases} \tag{3.74}$$

   Questo sistema può essere risolto come $|v| = |x| = 0$, oppure $k + |v| = k + |x| = k$, ovvero $|v| = |x| = k$ e $|vwx| \geqslant k$. In entrambi i casi si violano vincoli sulle sottoparti di z. $\Rightarrow\Leftarrow$

3. $v \in a^k, v \in c^k$. Questo caso non soddisfa il vincolo $|vwx| \geqslant k$, quindi è da scartare.
4. $v, x \in b^k$. Analogo al caso 1.
5. $v \in b^k, x \in c^k$. Analogo al caso 2.
6. $v, x \in c^k$. Analogo al caso 1. $\diamond$

I linguaggi CF sono chiusi rispetto all'*unione*, alla *concatenazione* e alla *stella di Kleene*. Intuitivamente, è possibile considerare le seguenti produzioni per combinare i linguaggi S_1 e S_2 in questi tre modi:

$$\begin{aligned} \cup &\colon S \to S_1 \mid S_2, \\ \cdot &\colon S \to S_1S_2, \\ * &\colon S \to S_1S. \end{aligned} \tag{3.75}$$

A differenza dei linguaggi regolari, non sono chiusi rispetto all'intersezione.

Esempio. Si considerino i seguenti linguaggi CF:

$$L_1 = \{\, a^nb^nc^i \mid n \in \mathbb{N} \,\}, \qquad L_2 = \{\, a^ib^nc^n \mid n \in \mathbb{N} \,\}. \tag{3.76}$$

La loro intersezione è data dal linguaggio:

$$L_1 \cap L_2 = \{\, a^nb^nc^n \mid n \in \mathbb{N} \,\}. \tag{3.77}$$

che è chiaramente non CF. $\diamond$

3.5 Algoritmi di decisione

Sia G una grammatica CF e $L(G)$ il linguaggio da essa generato. I seguenti problemi sono decidibili, ovvero si riesce a fornire una risposta in modo algoritmico e in tempo finito:

- $L(G) = \emptyset$?
- $L(G)$ è finito?
- $L(G)$ è infinito?
- $z \in L(G)$?

L'ultimo problema, nel contesto dei linguaggi di programmazione, è risolto dai *parser*.

3.6 Automi a pila

Per poter riconoscere linguaggi del tipo $a^n b^n$, si è visto, è necessario aggiungere al DFA una memoria di tipo stack (a pila).

Definizione (Automa a pila). *Un automa a pila è una tupla $M = \langle Q, \Sigma, R, \delta, q_0, Z_0, F \rangle$, dove Q, Σ, q_0 e F sono definiti come per un* DFA*, mentre:*

- *δ è la funzione di transizione definita come:*

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times R \to \wp(Q \times R^*). \tag{3.78}$$

- *R è l'alfabeto di simboli della pila.*
- *Z_0 è il simbolo iniziale della pila.*

La descrizione istantanea dell'automa è $(q, x \in \Sigma^*, \gamma \in R^*)$, corrispondente allo stato in cui si trova, ai simboli letti e alla stringa di simboli sulla pila. Se $(p, \gamma) \in \delta(q, a, Z)$, allora la descrizione istantanea commuta in:

$$(q, aw, Z\alpha) \to (p, w, \gamma\alpha). \tag{3.79}$$

L'automa a pila può accettare per:

Stato finale

$$L_F(M) = \{\, x \in \Sigma^* \mid (q_0 \in Q, x, Z_0) \Rightarrow_* (q \in F, \varepsilon, \gamma \in R^*) \,\}. \tag{3.80}$$

Pila vuota

$$L_P(M) = \{\, x \in \Sigma^* \mid (q_0 \in Q, x, Z_0) \Rightarrow_* (q \in Q, \varepsilon, \varepsilon) \,\}. \tag{3.81}$$

Se in almeno una $\delta(q, a, Z)$ esiste più di una regola di avanzamento, l'automa è non deterministico. Si può dimostrare che:

$$\underbrace{L(\text{Automi a pila deterministici})}_{\text{Parser}} \subsetneq \underbrace{L(\text{Automi a pila non deterministici})}_{\text{CF}}. \tag{3.82}$$

Da questo si evince che il potere espressivo degli automi a pila non deterministici è maggiore rispetto a quello degli automi a pila deterministici, ovvero *non sono equivalenti*.

Lemma 4. *Per ogni automa a pila non deterministico M, esiste un altro automa a pila non deterministico M' tale che $L(M) = L(M')$ e M' possiede un solo stato.*

Teorema ii. *Se M è un automa a pila non deterministico con uno stato tale che $L = L(M)$, allora L è un linguaggio* CF.

Per poter rappresentare l'automa a pila equivalente a una grammatica CF, è necessaria la forma normale di Greibach.

Definizione (Forma normale di Greibach). *Una grammatica* CF *è in forma normale di Greibach se, per ogni produzione $p \in P$:*

$$p = A \to a\alpha \quad \textit{con } a \in T,\ \alpha \in V^*. \tag{3.83}$$

Il simbolo a è il simbolo letto sul nastro, mentre α è l'insieme dei simboli sulla pila. Presa una grammatica $G = \langle V, T, P, S \rangle$ in forma di Greibach, dunque, si può costruire un automa a pila equivalente nel seguente modo:

$$Q = \{q\}, \quad \Sigma = T, \quad R = V, \quad F = \emptyset, \quad q_0 = q, \quad Z_0 = S \tag{3.84}$$

e l'insieme delle transizioni è definito come:

$$A \to a\alpha \in P \implies (q, \alpha) \in \delta(q, a, A). \tag{3.85}$$

II

Teoria della calcolabilità

4 Macchine di Turing

4.1 Introduzione

Dopo aver descritto e stabilito le relazioni tra linguaggi regolari, CF e non CF, l'obiettivo di questa parte è comprendere quali siano i limiti della calcolabilità.

Esempio. Si consideri la sequenza di Fibonacci. Può essere definita in due modi:

$$f(n) = \begin{cases} f(0) = 1 \\ f(1) = 1 \\ f(n+2) = f(n+1) + f(n). \end{cases} \tag{4.1}$$

$$f(n) = \frac{\sqrt{5}}{5}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{\sqrt{5}}{5}\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}. \tag{4.2}$$

La differenza tra queste due definizioni è che la prima è costruttiva, mentre la seconda è analitica. ◇

Definizione (Funzione ricorsiva). *Una funzione ricorsiva è un modello di calcolo definito in modo induttivo.*

Definizione (Insieme di funzioni primitive ricorsive). *L'insieme* PR *è il più piccolo insieme di funzioni contenente le funzioni ricorsive di base* $(0, \mathrm{S}, \Pi_i)$ *e chiuso rispetto a composizione e primitiva ricorsione.*

Le funzioni primitive ricorsive sono quindi costituite da funzioni primitive e operazioni applicabili a funzioni ricorsive. Le funzioni ricorsive di base sono:

Costante 0

$$0 = \lambda x.0. \tag{4.3}$$

Successore

$$\mathrm{S} = \lambda x.x + 1. \tag{4.4}$$

Proiezione i-esima

$$\Pi_i = \lambda x_1, \ldots, x_n.x_i. \tag{4.5}$$

Le operazioni sulle funzioni sono:

Composizione Se g e h sono funzioni definite da $\mathbb{N}$ a $\mathbb{N}$, la composizione di g e h è definita come:

$$f(x) = (h \circ g)(x) = h(g(x)). \tag{4.6}$$

Il simbolo $\circ$ si legge "dopo".

Primitiva ricorsione La funzione f definita per primitiva ricorsione dipende da due parametri: l'input del calcolo e un contatore n.

$$\begin{cases} f(0, \bar{x}) = g(\bar{x}) \\ f(n+1, \bar{x}) = h(n, \bar{x}, f(n, \bar{x})). \end{cases} \tag{4.7}$$

Nota. La scrittura $\bar{x}$ raccoglie un numero finito di parametri $x_1, \ldots, x_m$. ○

In questo caso, la base è determinata solo da g, mentre il passo induttivo coinvolge il contatore, l'input e la funzione calcolata al passo precedente.

Esempi. Seguono ora alcuni definizioni di funzioni ricorsive d'esempio:

Funzione costante n

$$\lambda x.n = \underbrace{\mathrm{S}(\mathrm{S}(\mathrm{S}\ldots\mathrm{S}}_{n \text{ volte}}(0)\ldots)). \tag{4.8}$$

Se gli input sono m:

$$\lambda x_1 \ldots x_m.n = \underbrace{\mathrm{S}(\mathrm{S}(\mathrm{S}\ldots\mathrm{S}}_{n \text{ volte}}(0(\Pi_i(x_1, \ldots, x_m)))\ldots)). \tag{4.9}$$

Somma

$$\lambda x y.x + y = \begin{cases} +(x, 0) = x \\ +(x, y+1) = \mathrm{S}(+(x, y)). \end{cases} \tag{4.10}$$

Moltiplicazione

$$\lambda x y.x * y = \begin{cases} *(x, 0) = 0(\Pi_1(x, 0)) = \Pi_2(x, 0) \\ *(x, y+1) = +(x, *(x, y)). \end{cases} \tag{4.11}$$

Segno

$$\begin{cases} \mathrm{sgn}(0) = 0 \\ \mathrm{sgn}(x+1) = \mathrm{S}(0). \end{cases} \tag{4.12}$$

Modulo

$$\begin{cases} \mathrm{mod}(y, 0) = 0 \\ \mathrm{mod}(0, x) = 0 \\ \mathrm{mod}(y+1, x) = *(\mathrm{S}(\mathrm{mod}(y, x)), \mathrm{sgn}(-(x, \mathrm{S}(\mathrm{mod}(y, x))))). \end{cases} \tag{4.13}$$

Queste funzioni sono *totali*, cioè definite su ogni input. ◇

Una funzione $f \in \mathrm{PR}$ è ottenuta con una sequenza finita di funzioni primitive ricorsive e composizioni. Per questo, ogni $f \in \mathrm{PR}$ è rappresentabile con un insieme di simboli finiti presi da un alfabeto finito. PR è quindi numerabile, ovvero $|\mathrm{PR}| = |\mathbb{N}|$.

Segue la domanda: *tutte le funzioni calcolabili sono primitive ricorsive?* La risposta è no.

Dimostrazione. Supponiamo che sia vera l'ipotesi, cioè che l'insieme delle funzioni calcolabili coincida con quello delle funzioni primitive ricorsive. Poiché $|\mathrm{PR}| = |\mathbb{N}|$, possiamo enumerarle associando a ognuna di esse un numero naturale. Riscriviamo dunque PR come:

$$\mathrm{PR} = \{ f_0, f_1, f_2, \ldots, f_m, \ldots \}. \tag{4.14}$$

Definiamo la funzione $h(x) = f_x(x) + 1$: poiché $f_x \in \text{PR}$, essa è calcolabile per ipotesi, dunque esiste una procedura che la calcola. Inoltre, $h(x) \in \text{PR}$ perché $h(x) = \text{S}(f_x(x))$. Pertanto, esiste un indice anche per h: sia n tale che $h = f_n$. Allora:

$$h(n) = f_n(n) = f_n(n) + 1, \tag{4.15}$$

che è chiaramente un assurdo, perché nessun numero naturale è uguale al suo successore. Si può concludere che non tutte le funzioni calcolabili sono primitive ricorsive. □

Nota. L'equazione:

$$f_n(n) = f_n(n) + 1 \tag{4.16}$$

non è mai verificata per nessun naturale solo se f_n non diverge (cioè se termina). Se così non fosse, l'esecuzione di $f_n(n)$ diverrebbe infinita e il successore non verrebbe mai computato. Quindi, il comportamento di $f_n(n)$ sarebbe analogo a quello di $f_n(n) + 1$. ○

Si può cercare un limite più stretto per le funzioni primitive ricorsive. Si ricerca, in particolare la loro relazione con le funzioni totali. Esistono funzioni totali, definite su ogni input, che non sono primitive ricorsive? La risposta è sì e la seguente funzione ne è un esempio:

$$\begin{cases} \text{Ack}(0, y) = y + 1 \\ \text{Ack}(x + 1, 0) = \text{Ack}(x, 1) \\ \text{Ack}(x + 1, y + 1) = \text{Ack}(x, \text{Ack}(x + 1, y)). \end{cases} \tag{4.17}$$

Questa funzione è chiamata *funzione di Ackermann*. L'idea per cui non è primitiva ricorsiva pur essendo totale è che nella sua definizione non compaiono funzioni primitive ricorsive precedentemente definite, ma la funzione stessa.

Le funzioni $f \in \text{PR}$ sono funzioni calcolabili mediante linguaggi di tipo `for` (Pascal-like), ovvero tramite cicli determinati. La funzione Ack è calcolabile con un linguaggio di tipo `while`, che permette di eseguire cicli indeterminati. Questa operazione viene detta *di ricerca incerta* ed è necessaria per aumentare il potere espressivo dei modelli di calcolo presentati.

Definizione (Algoritmo). *Un* algoritmo *è definito come:*

a. Una sequenza finita di passi elementari.

b. Esiste un agente che porta avanti il calcolo.

c. L'agente di calcolo ha a disposizione una memoria per i calcoli intermedi.

d. I passi sono discreti e non probabilistici.

e. La memoria disponibile è illimitata (non infinita).

f. È presente un limite finito alla complessità delle istruzioni da eseguire.

g. In ogni istante, il passo successivo di calcolo dipende da una porzione finita di memoria.

4.2 Macchina di Turing

Definizione (Macchina di Turing). *Una* Macchina di Turing (MdT) *è definita da una quadrupla:*

$$M = \langle Q, \Sigma, P, q_0 \rangle \tag{4.18}$$

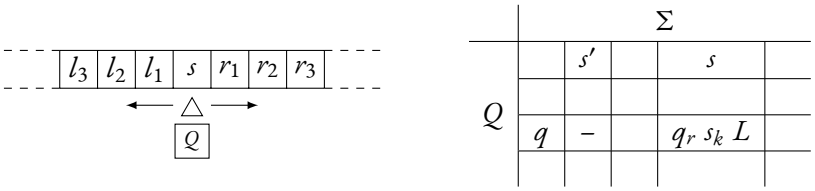


Figura 4.1 *Modello di macchina di Turing.*

dove:

1. *Q è un insieme finito di stati, di cui q_0 è lo stato iniziale.*[1]
2. *Σ è un alfabeto finito con almeno due simboli speciali: $S_0 =$ \$ e $S_1 = 0$. Il simbolo* \$ *rappresenta una cella vuota, mentre* 0 *è il simbolo di lavoro.*
3. *P è un insieme non vuoto di istruzioni aventi la forma:*

$$q\, s\, q'\, s'\, L, \qquad q\, s\, q'\, s'\, R \tag{4.19}$$

dove:

- *q è lo stato in cui si trova la macchina,*
- *s è il simbolo letto sul nastro,*
- *q' è lo stato in cui la macchina transita,*
- *s' è il simbolo scritto sul nastro,*
- *L e R sono le direzioni di movimento della testina.*

La funzione di transizione di una MdT è quindi definita come:

$$\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\} \tag{4.20}$$

ed è deterministica, ovvero associa a ogni coppia (q, s) un'unica tripla $(q', s', d \in \{L, R\})$.

Se una cella della tabella in Figura 4.1 è vuota, la macchina termina. Come per gli automi a pila, si può esprimere la descrizione istantanea di una MdT come:

$$(q, v, s, w) \tag{4.21}$$

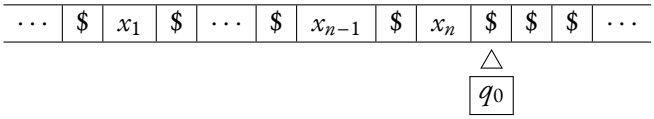
dove:

- q è lo stato corrente,
- v è la parte sinistra del nastro, contenente solo simboli significativi (non vuoti),
- s è il simbolo corrente,
- w è la parte destra del nastro, contenente solo simboli significativi (non vuoti).

L'esecuzione di una MdT è data dalla sequenza di descrizioni istantanee eseguite (semantica). La tabella di transizione rappresenta invece la sintassi della macchina.

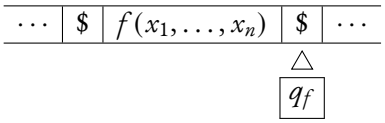
[1] Questo implica che $Q \neq \emptyset$.

Definizione (Funzione calcolata da una MdT). *Una funzione* $f\colon \mathbb{N}^m \to \mathbb{N}$ *è detta Turing-calcolabile se esiste una* MdT M *che, partendo dalla configurazione iniziale:*



(4.22)

termina nella configurazione finale:



(4.23)

quando $f(x_1, \ldots, x_n)$ *è definita, non termina (diverge) altrimenti.*

Sia i parametri di input che i valori risultanti dalla valutazione di f possono essere rappresentati sul nastro in diverse forme: unaria, binaria, decimale, …

4.3 Macchine di Turing generalizzate

Definizione (Macchina di Turing generalizzata). *Una* MdT generalizzata *è una* MdT M *con n nastri e m testine, dove* $m \geqslant n$.

Ogni MdT generalizzata può essere simulata da una MdT M' con un solo nastro e una sola testina. Inoltre, ogni MdT M avente n simboli e m stati può essere simulata da una MdT M' composta da due stati o da un'altra MdT M'' con due simboli.

Osservazione. Gli stati rappresentano le risorse temporali della macchina. I simboli rappresentano le risorse spaziali. △

Tesi di Church. *La tesi di Church afferma che l'insieme delle funzioni intuitivamente calcolabili coincide con quello delle funzioni Turing-calcolabili, ovvero calcolabili mediante una* MdT.

Estendiamo ora le funzioni PR alle funzioni parziali ricorsive.

Definizione (Insieme delle funzioni parziali ricorsive). *L'insieme* KR *è la più piccola classe di funzioni che contiene le* PR *ed è chiusa per minimizzazione (μ-ricorsione o ricerca incerta).*

Definizione (Minimizzazione di una funzione). *Sia* f *una funzione totale definita da* $\mathbb{N}^{n+1}$ *a* $\mathbb{N}$. *Si definisce una nuova funzione* φ *come:*

$$\varphi(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \mu z.\big(f(x_1, \ldots, x_n, z) = 0\big). \tag{4.24}$$

La funzione φ potrebbe non terminare su tutti gli input: in tal caso di dice *parziale*. Una definizione alternativa di φ è:

$$\varphi(x_1, \ldots, x_n) = \begin{cases} \arg\min_z f(x_1, \ldots, x_n, z) = 0 & \text{se } z \text{ esiste} \\ \uparrow & \text{altrimenti.} \end{cases} \tag{4.25}$$

L'equivalente in pseudocodice di φ è:

```
1  input(x_1, ..., x_n)
2  z := 0
3  while f(x_1, ..., x_n, z) ≠ 0
4      z := z + 1
5  return z
```

Esempi. Alcune funzioni parziali ricorsive sono:

n-esimo numero primo

$$\begin{cases} p(0) = 0 \\ p(1) = 2 \\ p(n+1) = f(p(n)) \end{cases} \tag{4.26}$$

dove:

$$f(y) = \mu z.\text{and}(\text{sgn}(-(z, y)), \text{primo}(z)) \tag{4.27}$$

prende in input l'$(n-1)$-esimo numero primo e $\text{and}(x, y)$ restituisce 1 se entrambi gli input valgono 1, 0 altrimenti.

Logaritmo in base a

$$\left\lfloor \log_a(x) \right\rfloor = \mu y.\big(\text{leq}(x, a^{y+1}) = 0\big). \tag{4.28}$$

Radice n-esima

$$\left\lfloor \sqrt[n]{x} \right\rfloor = \mu y.\big(\text{leq}(x, (y+1)^n) = 0\big). \tag{4.29}$$

◇

Teorema 12. *Per ogni funzione ricorsiva f, esiste una* MdT M *che la calcola. Analogamente, per ogni* MdT M*, esiste una funzione ricorsiva f equivalente alla funzione calcolata da M.*

Per questo, si può dire che una funzione f è ricorsiva se e solo se è Turing-calcolabile.

4.4 Aritmetizzazione delle macchine di Turing

Definizione (Aritmetizzazione). *L'aritmetizzazione consiste nell'associazione biunivoca tra numeri naturali e oggetti matematici.*

Nel contesto delle MdT, l'aritmetizzazione consente di enumerare le relative configurazioni e transizioni. Poiché le macchine di Turing sono descritte dall'insieme delle loro descrizioni istantanee, è possibile ordinarle rispetto a esse. Pertanto, per ogni MdT M, esiste un numero naturale n tale che M è la n-esima MdT. In questo senso, $n \in \mathbb{N}$ identifica il programma o algoritmo che definisce M. In altri termini, si vuole far sì che da M si possa ottenere n, e da n si possa ricostruire la MdT corrispondente.

Esempio. Una modalità per aritmetizzare le MdT è detta *gödelizzazione*. Si consideri una MdT M con Q come insieme finito di stati, $\Sigma = \{\$, 0\}$ come alfabeto di input e $\{L, R\}$ come possibili direzioni della testina. Si eseguono le seguenti associazioni:

$$\begin{array}{lll} q_0 \to 3, & s_0 \to 2n+5, & m_0 = L = 2n + 2k + 7, \\ q_1 \to 5, & s_1 \to 2n+7, & m_1 = R = 2n + 2k + 9. \\ \quad\vdots & \quad\vdots & \\ q_n \to 2n+3, & s_k \to 2n + 2k + 5, & \end{array} \tag{4.30}$$

Se la descrizione istantanea di M è rappresentata attraverso la quintupla $q\ s\ q'\ s'\ m$, si definisce la funzione di gödelizzazione come segue:

$$E = (q\ s\ q'\ s'\ m), \qquad \mathrm{gn}(E) = \prod_{i=1}^{k} p_i^{a(x_i)} \tag{4.31}$$

dove p_i è l'i-esimo numero primo, e $a(x_i)$ è il valore associato all'i-esimo simbolo x_i della quintupla. Poiché la fattorizzazione di un numero è unica, si riescono sempre a ricostruire i singoli termini $a(x_i)$.

Se si considera, poi, un programma P come insieme ordinato di evoluzioni della macchina:

$$P = (E_1, E_2, \ldots, E_n), \tag{4.32}$$

allora si definisce la funzione di gödelizzazione come:

$$\mathrm{gn}(P) = \prod_{i=1}^{n} p_i^{\mathrm{gn}(E_i)}. \tag{4.33}$$

◇

Chiamiamo, dunque, φ_x la funzione calcolata dal programma $x \in \mathbb{N}$.

4.5 Macchina di Turing universale

Definizione (Macchina di Turing universale). *Una* macchina di Turing universale *è una* MdT *che può eseguire altre* MdT.

Formalmente si definisce M_{I} come la MdT universale, tale che:

$$M_{\mathrm{I}}(x_1, x_2) = M_{x_1}(x_2), \tag{4.34}$$

con $x_1, x_2 \in \mathbb{N}$ e M_{x_1} la MdT x_1-esima. Alla luce di quanto detto, il valore x_1 è l'interpretazione del programma. L'equivalente in pseudocodice di M_{I} è il seguente:

```
1  input(x_1)
2  input(x_2)
3  M_{x_1} := Aritmetizzazione^{-1}(x_1)
4  if M_{x_1} è un programma
5      esegui M_{x_1}(x_2)
6  else
7      while TRUE
```

In questo caso, la procedura può non terminare per due motivi: perché M_{x_1} non è un programma, oppure perché $M_{x_1}(x_2)$ non termina. Una scrittura più compatta per M_{I} è:

$$M_{\mathrm{I}}(x_1, x_2) = \begin{cases} M_{x_1}(x_2) & \text{se } M_{x_1}(x_2)\downarrow \\ \uparrow & \text{altrimenti.} \end{cases} \tag{4.35}$$

Nota. Con la scrittura:

$$\varphi(x)\downarrow \tag{4.36}$$

si intende che la funzione parziale φ a cui viene applicato l'input x è definita e termina con risultato $\varphi(x) \in \mathbb{N}$. Invece, con la scrittura:

$$\varphi(x)\uparrow \tag{4.37}$$

si intende che la funzione parziale φ a cui viene applicato l'input x non è definita o non termina. ○

4.6 Teorema SMN

Il teorema SMN garantisce l'esistenza di una funzione Spec. Questa funzione permette di eseguire parzialmente un programma, nel caso in cui si conoscano i valori di alcuni parametri di input.

Esempio. Se si considera la funzione:

$$\lambda x\, y.x + y \tag{4.38}$$

e si suppone che $x = 5$, allora essa può essere specializzata come:

$$\lambda y.5 + y. \tag{4.39}$$

In quest'altro caso, invece, la specializzazione per $x = 5$ è rilevante perché elimina un intero ramo:

$$\lambda x\, y.f(x, y) \qquad f\colon \ldots \textbf{ if } x > 0 \textbf{ then } \ldots \textbf{ else } \ldots \tag{4.40}$$

$$\lambda y.f(5, y) \qquad f\colon \ldots \textbf{ then } \ldots \tag{4.41}$$

◇

Teorema 13. *Per ogni $n, m \geqslant 1$, $n, m \in \mathbb{N}$, esiste una funzione totale Spec_m^n con $m + 1$ input tale che:*

$$\lambda z_1, \ldots, z_n.\varphi_x(\underbrace{y_1 \ldots y_m}_{\text{noti}}, z_1 \ldots z_n) = \lambda z_1, \ldots z_n.\varphi_{\text{Spec}(x, y_1 \ldots y_m)}(z_1 \ldots z_n). \tag{4.42}$$

La funzione φ_x possiede $m + n$ input e viene specializzata su $y_1 \ldots y_m$. Il teorema dice che il programma specializzato, ovvero in cui l'input noto è integrato nel codice, possiede a sua volta un codice che dipende in modo ricorsivo dal programma originale (x) e dall'input specializzato ($y_1 \ldots y_m$).

Esempio. Si consideri di nuovo la funzione che calcola la somma tra due numeri:

$$\lambda yz.y + z. \tag{4.43}$$

Lo pseudocodice equivalente è:

```
1  input(y)
2  input(z)
3  w := 1
4  while w ⩽ y
5      z := z + 1
6      w := w + 1
7  return z
```

La specializzazione della funzione per $y = 5$ è:

$$\lambda z.5 + z, \tag{4.44}$$

che corrisponde allo pseudocodice:

```
1  input(z)
2  w := 1
3  y := 5
4  while w ⩽ z
5      z := z + 1
6      w := w + 1
7  return z
```

Come si può vedere, il programma specializzato viene modificato a partire dal precedente (x) e aggiungendo il parametro noto ($y = 5$). Pertanto, la dipendenza di Spec da x e y è evidente. ◇

Il modello dello specializzatore ha la seguente firma:

$$\textsc{Spec}\colon \mathbb{N} \times \mathbb{N}^m \to \mathbb{N} \tag{4.45}$$

dove il primo $\mathbb{N}$ corrisponde al programma in input, $\mathbb{N}^m$ agli m input e l'ultimo $\mathbb{N}$ al programma specializzato. Può essere evidenziato un parallelismo tra Spec e $\mathcal{M}_{\text{I}}$, poiché:

$$\mathcal{M}_{\text{I}} \equiv \textsc{Int}\colon \mathbb{N} \times \mathbb{N}^m \to \mathbb{N} \tag{4.46}$$

dove Int è l'*interprete* e l'unica differenza rispetto a Spec è che l'ultimo $\mathbb{N}$ si riferisce a un risultato numerico piuttosto che a un programma.

4.7 Prima proiezione di Futamura

Definizione (Prima proiezione di Futamura). *La* prima proiezione di Futamura *permette di definire un compilatore utilizzando un specializzatore e un interprete.*

Dato un programma Source $\in L$, dove L è un linguaggio Turing-completo, esiste un altro programma Comp che, applicato a Source, restituisce un programma Target equivalente a Source:

$$\varphi_{\textsc{Source}} = \varphi_{\textsc{Target}} \iff \forall x \in \mathbb{N}\colon \big(\varphi_{\textsc{Source}}(x) = \varphi_{\textsc{Target}}(x)\big), \tag{4.47}$$

$$\varphi_{\textsc{Source}}(x) = \varphi_{\textsc{Int}}(\textsc{Source}, x) = \varphi_{\textsc{Spec}(\textsc{Int}, \textsc{Source})}(x). \tag{4.48}$$

Nota. L'esistenza dello specializzatore e dell'inteprete è garantita dal fatto che L, per ipotesi, è Turing-completo. ○

Si può dunque costruire la seguente equivalenza:

$$\textsc{Target} \overset{\text{def}}{=} \textsc{Spec}(\textsc{Int}, \textsc{Source}). \tag{4.49}$$

Il programma Target è scritto nel linguaggio in cui è scritto Int, che può non essere L.

4.8 Problemi irrisolvibili

Definizione (Problema irrisolvibile). *Un* problema irrisolvibile *è un problema che non può essere risolto da un algoritmo.*

Alcuni esempi di problemi di questa natura sono:

Problema della terminazione Esiste una funzione totale $g\colon \mathbb{N} \to \mathbb{N}$ tale che:

$$g(x) = \begin{cases} 1 & \text{se } \varphi_x(x)\downarrow \\ 0 & \text{se } \varphi_x(x)\uparrow \end{cases} \; ? \tag{4.50}$$

cioè che restituisca 1 se la funzione φ_x termina sul proprio codice, e 0 altrimenti.

La risposta è no.

Dimostrazione. Supponiamo che g, totale, esista. Allora è anche calcolabile, ovvero esiste una MdT M che la calcola. In altri termini, esiste i_0 tale che $g = \varphi_{i_0}$. Si consideri ora la funzione g' definita come segue:

$$g'(x) = \begin{cases} \uparrow & \text{se } g(x) = 1 \\ 0 & \text{se } g(x) = 0. \end{cases} \tag{4.51}$$

Lo pseudocodice equivalente di g' è:

```
1  input(x)
2  y := φ_{i_0}(x)
3  if y == 1
4      while TRUE
5  else
6      return 0
```

Per questo, g' è calcolabile e, quindi, esiste un indice i_1 tale che $g' = \varphi_{i_1}$. Controlliamo i casi di terminazione di $\varphi_x(x)$ su i_1:

- Se $\varphi_{i_1}(i_1)\downarrow$, allora:

$$\begin{aligned} g'(i_1)\downarrow &\implies g'(i_1) = 0 \\ &\implies g(i_1) = 0 \\ &\implies \varphi_{i_1}(i_1)\uparrow. \quad \Rightarrow\!\Leftarrow \end{aligned} \tag{4.52}$$

- Se $\varphi_{i_1}(i_1)\uparrow$, allora:

$$\begin{aligned} g'(i_1)\uparrow &\implies g(i_1) = 1 \\ &\implies \varphi_{i_1}(i_1)\downarrow. \quad \Rightarrow\!\Leftarrow \end{aligned} \tag{4.53}$$

Pertanto, g con queste caratteristiche non può esistere. □

Il problema può essere esteso anche su un generico input. Questo equivale a domandarsi se esiste una funzione g totale tale che:

$$g(x, y) = \begin{cases} 1 & \text{se } \varphi_x(y)\downarrow \\ 0 & \text{se } \varphi_x(y)\uparrow. \end{cases} \tag{4.54}$$

Anche in questo caso, g non può esistere poiché si potrebbe costruire il seguente programma:

```
1  input(x)
2  if g(x, x) == 1
3      return 1
4  else
5      return 0
```

che calcola $g(x)$ e, quindi, $\varphi_x(x)$. Dato che si è dimostrato che g non può esistere, ne consegue che non può esistere nemmeno $g(x, y)$.

Problema della totalità Esiste una funzione totale h che decide se una funzione è totale oppure no? Il comportamento della funzione dovrebbe essere il seguente:

$$h(x) = \begin{cases} 1 & \text{se } \varphi_x \text{ è totale: } \forall y \in \mathbb{N}\colon \varphi_x(y)\downarrow \\ 0 & \text{se } \varphi_x \text{ non è totale: } \exists y \in \mathbb{N}.\varphi_x(y)\uparrow. \end{cases} \tag{4.55}$$

Dimostrazione. Supponiamo che h esista. Consideriamo la funzione $h'(n)$, definita per primitiva ricorsione, che restituisce l'indice dell'n-esima funzione totale:

$$\begin{cases} h'(0) = \mu y.(h(y) == 1) \\ h'(x+1) = \mu y.(y > h'(x) \wedge h(y) == 1). \end{cases} \tag{4.56}$$

Anche la funzione h' stessa è totale. Costruiamo ora un'altra funzione f definita come:

$$f(0) = \varphi_{h'(x)}(x) + 1. \tag{4.57}$$

La funzione ottiene tramite h' l'indice della x-esima funzione totale, costruisce la corrispondente φ, la valuta e vi somma 1.

Poiché $h'(x)$ è l'indice di una funzione totale, anche $\varphi_{h'(x)}(x)$ è totale, e lo è a sua volta anche f. Essendo totale, è anche ricorsiva, pertanto esiste una MdT che la calcola:

$$\exists z_0 \in \mathbb{N}.f = \varphi_{z_0}. \tag{4.58}$$

In particolare, l'indice può essere calcolato mediante h':

$$\exists y_0 \in \mathbb{N}.h'(y_0) = z_0. \tag{4.59}$$

Calcoliamo ora $f(y_0)$:

$$f(y_0) = \varphi_{h'(y_0)}(y_0) + 1. \tag{4.60}$$

Dato che $h'(y_0) = z_0$, possiamo scrivere:

$$f(y_0) = \varphi_{z_0}(y_0) + 1 = f(y_0) + 1. \quad \Rightarrow\Leftarrow \tag{4.61}$$

Nuovamente un assurdo. □

5 Teoria della ricorsione

5.1 Linguaggi di Turing

Definizione (Linguaggio di Turing). *Il linguaggio L si dice* linguaggio di Turing *o* a struttura di frase *se esiste una* MdT M *tale che:*

$$L = L(M) = \{ x \in \Sigma^* \mid \varphi_M(x)\downarrow \}. \tag{5.1}$$

Definizione (Dominio di una funzione parziale ricorsiva). *Il dominio di una funzione parziale ricorsiva φ è l'insieme dei valori di input per cui la funzione termina:*

$$\mathrm{dom}(\varphi) = \{ x \in \Sigma^* \mid \varphi(x)\downarrow \}. \tag{5.2}$$

Essendo φ ricorsiva, esiste una MdT di indice x che la calcola:

$$\varphi = \varphi_x. \tag{5.3}$$

Pertanto, il dominio può essere scritto anche come:

$$\mathrm{dom}(\varphi) = \mathrm{dom}(\varphi_x) = \{ y \in \mathbb{N} \mid \varphi_x(y)\downarrow \} \stackrel{\text{def}}{=} W_x. \tag{5.4}$$

In particolare, se φ_x è totale, $W_x = \mathbb{N}$.

5.2 Insiemi ricorsivamente enumerabili e ricorsivi

Definizione (Insieme ricorsivamente enumerabile). *L'insieme $I \subseteq \mathbb{N}$ si dice* ricorsivamente enumerabile (RE) *se esiste una funzione parziale ricorsiva ψ tale che:*

$$I = \mathrm{dom}(\psi). \tag{5.5}$$

In altri termini, B sono tutti e soli gli elementi su cui ψ termina. Essendo ψ ricorsiva esisterà anche per essa una MdT di indice x:

$$\exists x \in \mathbb{N}. \varphi_x = \psi. \tag{5.6}$$

Osservazione. Gli insiemi W_x, per $x \in \mathbb{N}$, sono sempre RE. △

Definizione (Semicaratteristica di I). *Si definisce* semicaratteristica di I *la funzione parziale ricorsiva:*

$$\psi_I(x) = \begin{cases} 1 & \text{se } x \in I \\ \uparrow & \text{altrimenti}. \end{cases} \tag{5.7}$$

L'equivalente pseudocodice di ψ_I è:

```
1  input(x)
2  if ψ_I(x)↓
3      return 1
```

Si noti come non è presente un ramo **else** poiché non è possibile ricadervi nel caso in cui la funzione diverga.

Definizione (Insieme ricorsivo). *L'insieme $I \subseteq \mathbb{N}$ si dice* ricorsivo *se esiste una funzione totale detta* funzione caratteristica *tale che:*

$$f_I(x) = \begin{cases} 1 & \text{se } x \in I \\ 0 & \text{se } x \notin I. \end{cases} \tag{5.8}$$

Esempio. Si consideri il seguente insieme:

$$I = \{\, x \in \mathbb{N} \mid \exists y \in \mathbb{N}. x = 2^y \,\}. \tag{5.9}$$

L'insieme è ricorsivo, in quanto la sua funzione caratteristica è realizzata dal seguente pseudocodice:

```
1  for y := 0 to x
2      if x == 2^y
3          return 1
4  return 0
```

Sarebbe sbagliato non imporre un limite alla ricerca, poiché la funzione caratteristica deve sempre produrre il risultato (non divergere). ◇

Osservazione. Un insieme A è ricorsivo se e solo se anche il suo complementare $\overline{A}$ è ricorsivo. △

Dimostrazione. Partendo da uno dei due insiemi, che per ipotesi sono ricorsivi e dunque esiste la relativa funzione caratteristica, è possibile ottenere quella dell'altro insieme semplicemente invertendo il valore di ritorno. □

Nota. Come si dimostra che A è RE?

1. Si definisce ψ_A.
2. Si dimostra che ψ_A è ricorsiva, dunque calcolabile, fornendo lo pseudocodice che la calcola.

Dunque, $A = \text{dom}\, \psi_A$ e di conseguenza A è RE. Come si dimostra, invece, che A è REC?

1. Si definisce f_A.
2. Si dimostra che f_A è totale ricorsiva fornendo lo pseudocodice, tipicamente senza `while`, che la calcola e si dimostra che termina sempre (è totale). ○

Teorema 14. *Sia A un insieme* REC. *Allora A è anche* RE.

Dimostrazione (Teorema 14). Per ipotesi A è REC, quindi esiste una funzione ricorsiva totale f_A tale che:

$$f_A(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{se } x \notin A. \end{cases} \tag{5.10}$$

Allora si può costruire il seguente codice:

```
1  input(x)
2  if f_A(x) == 1
3      return 1
4  else
5      while TRUE
```

che corrisponde alla semicaratteristica di A. □

Teorema 15 (di Post). *Un insieme A è* REC *se e solo se sia A che il suo complementare $\overline{A}$ sono* RE.

Dimostrazione (Teorema 15).

($\Longrightarrow$) Per ipotesi A è REC, quindi esiste una funzione totale ricorsiva f_A tale che:

$$f_A(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{se } x \notin A. \end{cases} \tag{5.11}$$

Definiamo $g(x)$ come segue:

$$g(x) = \begin{cases} 0 & \text{se } f_A(x) = 1 \\ 1 & \text{se } f_A(x) = 0. \end{cases} \tag{5.12}$$

Le due condizioni di $g(x)$ sono equivalenti a:

$$\begin{aligned} f_A(x) = 1 &\iff x \in A \iff x \notin \overline{A}, \\ f_A(x) = 0 &\iff x \notin A \iff x \in \overline{A}. \end{aligned} \tag{5.13}$$

La funzione $g(x)$ è quindi la funzione caratteristica totale e ricorsiva corrispondente al complementare di A, $\overline{A}$. Si può concludere che $\overline{A}$ è REC. Per il Teorema 14, $\overline{A}$ è anche RE.

($\Longleftarrow$) Per ipotesi sia A che $\overline{A}$ sono RE. Quindi esistono due funzioni ψ_A e $\psi_{\overline{A}}$ tali che:

$$\psi_A(x) = \begin{cases} 1 & \text{se } x \in A \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.14}$$

$$\psi_{\overline{A}}(x) = \begin{cases} 1 & \text{se } x \in \overline{A} \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.15}$$

Si può costruire il seguente codice:

```
1  input(x)
2  if ψ_A(x) == 1 || ψ_Ā(x) == 1
3      if ha terminato ψ_A
4          return 1
5      else if ha terminato ψ_Ā
6          return 0
```

dove il simbolo || rappresenta l'esecuzione parallela delle due funzioni. Infatti, se ψ_A e $\psi_{\overline{A}}$ venissero eseguite in sequenza, una delle due potrebbe bloccare l'altra, nel caso l'input la facesse divergere. L'esecuzione parallela non bloccante può essere implementata eseguendo alternativamente un passo di ciascuna funzione, finché una delle due non termina (Figura 5.1).

Questa funzione opera dunque come la caratteristica di A, restituendo 1 se $x \in A$ e 0 se $x \notin A$. Essendo sicuri che termina sempre (infatti, $x \in A$ o $x \notin A$), possiamo concludere che A è REC. □

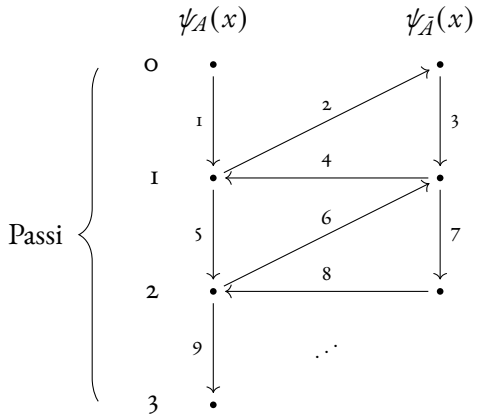


Figura 5.1 *Esecuzione parallela non bloccante delle funzioni semicaratteristiche di A e $\overline{A}$.*

Definizione. *Si definisce $K = \{ x \mid \varphi_x(x)\downarrow \} = \{ x \mid x \in W_x \} \subseteq \mathbb{N}$ come l'insieme di tutti gli indici (codici) di funzioni che terminano sul proprio indice.*

L'insieme K è un insieme RE perché il codice:

```
1  input(x)
2  if φ_x(x)↓
3      return 1
```

costituisce la sua funzione semicaratteristica. Non è tuttavia un insieme REC.

Dimostrazione. Supponiamo che K sia ricorsivo. Allora, per il Teorema 15, sia K sia $\overline{K}$ sono RE. Se $\overline{K}$ è RE, allora esiste un indice $y_0 \in \mathbb{N}$ tale che $\overline{K} = \mathrm{dom}(\varphi_{y_0})$.

Ci si chiede ora se $y_0 \in W_{y_0} = \mathrm{dom}(\varphi_{y_0})$, ovvero se $\varphi_{y_0}(y_0)$ converga o no. Si analizzano i due casi:

- Se $\varphi_{y_0}(y_0)\downarrow$, allora:

$$\begin{aligned} & y_0 \in W_{y_0} = \overline{K} = \{ x \mid \varphi_x(x)\uparrow \} \\ \implies & \varphi_{y_0}(y_0)\uparrow. \quad \Rightarrow\Leftarrow \end{aligned} \tag{5.16}$$

- Se $\varphi_{y_0}(y_0)\uparrow$, allora:

$$\begin{aligned} & y_0 \notin W_{y_0} = \overline{K} \\ \implies & y_0 \in K = \{ x \mid \varphi_x(x)\downarrow \} \\ \implies & \varphi_{y_0}(y_0)\downarrow. \quad \Rightarrow\Leftarrow \end{aligned} \tag{5.17}$$

In entrambi i casi si è ottenuto un assurdo, dunque $\overline{K}$ non è RE e K non è REC. □

Teorema 16. *Le seguenti affermazioni sono equivalenti:*

1. *$A \subseteq \mathbb{N}$ è* RE.
2. *$\exists \psi$ parziale ricorsiva tale che $A = \mathrm{Range}(\psi) = \{ \psi(x) \mid \psi(x)\downarrow \}$.*
3. *$A = \emptyset$ oppure esiste f totale tale che $A = \mathrm{Range}(f)$.*

La funzione Range determina tutti i possibili valori che una funzione può assumere, quando essa termina.

Esempi.

Insieme dei numeri pari Questo insieme è RE. Possiamo definire la sua semicaratteristica:

$$\varphi(x) = \begin{cases} 1 & \text{se } x \bmod 2 = 0 \\ \uparrow & \text{altrimenti,} \end{cases} \tag{5.18}$$

oppure la funzione Range che restituisce l'$(x-1)$-esimo elemento dell'insieme:

$$\psi(x) = \begin{cases} 2x & x \geqslant 0 \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.19}$$

Questa funzione calcola l'insieme:

$$\{\, 2x \mid x \in \mathbb{N} \,\}. \tag{5.20}$$

Insieme delle radici quadrate Anche questo insieme è RE. La sua semicaratteristica è:

$$\varphi(x) = \begin{cases} 1 & \text{se } \exists y \in \mathbb{N}.x = y^2 \\ \uparrow & \text{altrimenti,} \end{cases} \tag{5.21}$$

oppure la funzione Range:

$$\psi(x) = \begin{cases} x & \text{se } \exists y \in \mathbb{N}.x = y^2 \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.22}$$

Questa funzione calcola l'insieme:

$$\{\, x \mid \exists y \in \mathbb{N}.x = y^2 \,\} = \{\, y^2 \mid y \in \mathbb{N} \,\}. \tag{5.23}$$

Insieme delle funzioni con dominio non vuoto Si consideri l'insieme:

$$A = \{\, x \mid W_x \neq \emptyset \,\} = \{\, x \mid \exists y \in \mathbb{N}.\varphi_x(y)\downarrow \,\}. \tag{5.24}$$

La funzione che determina se un indice x appartiene a A non può iterare su tutti i possibili valori di y per determinare se $\varphi_x(y)$ termina. Infatti, potrebbe divergere non appena ne trovi uno che non fa terminare φ_x. Pertanto è necessario applicare la tecnica di *dovetailing*:

```
1  input(x)
2  y := 0
3  n := 0
4  while n < 1
5      for i := 0 to y
6          esegui il passo successivo di φ_x(i)
7          if φ_x(i) ha terminato in questo passo
8              n := n + 1
9      y := y + 1
10 return 1
```

◇

Teorema 17. *Le seguenti affermazioni sono equivalenti:*

1. $A \subseteq \mathbb{N}$ *è* REC.
2. $A = \emptyset$ *oppure esiste una funzione totale* f *non decrescente tale che* $A = \text{Range}(f)$.

Esempi.

- Si consideri l'insieme:
$$A = \{\, x \mid \exists y \in \mathbb{N}. x = y^2 + y + 1 \,\}. \tag{5.25}$$
Questo insieme è REC. Infatti, coincide con l'insieme $\text{Range}(f)$, dove f è la funzione:
$$f(y) = y^2 + y + 1. \tag{5.26}$$
La dimostrazione del fatto che f generi tutti e soli gli elementi di A può avvenire in due modalità:
   1. Mostrando che f è non decrescente (per Teorema 17):
   $$\forall y_1, y_2 \in \mathbb{N}\colon\ y_1 \leqslant y_2 \implies f(y_1) \leqslant f(y_2) \iff f(y_2) - f(y_1) \geqslant 0. \tag{5.27}$$
   $$\begin{aligned} f(y+1) &= (y+1)^2 + (y+1) + 1 \\ &= y^2 + 2y + 1 + y + 1 + 1 \\ &= y^2 + 3y + 3, \end{aligned} \tag{5.28}$$
   $$f(y) = y^2 + y + 1, \tag{5.29}$$
   $$f(y+1) - f(y) = 2y + 2 > 0 \geqslant 0. \tag{5.30}$$
   2. Fornendo lo pseudocodice della funzione caratteristica f:
   ```
   1  input(y)
   2  if x ⩽ 1
   3      return x
   4  for y ≔ 1 to x
   5      z ≔ y² + y + 1
   6      if x == z
   7          return 1
   8  return 0
   ```
- Si consideri l'insieme:
$$A_m = \{\, x \mid \varphi_x(x^3 + x^2 + x) \text{ non } \downarrow \text{ in meno di } m \text{ passi} \,\}. \tag{5.31}$$
dove m è fissato. Questo insieme è REC. Per dimostrarlo, si fornisce la funzione caratteristica:
   ```
   1  input(x)
   2  input(m)
   3  y ≔ x³ + x² + x
   4  for i ≔ 1 to m
   5      esegui il passo i-esimo di φ_x(y)
   6      if φ_x(y) ha terminato al passo i-esimo
   7          return 0
   8  return 1
   ```

◇

Nota. Con la scrittura $[\![P]\!]$ si intende la semantica del programma P corrispondente alla funzione φ_x, ovvero si svincola la sintassi specifica di P da ciò che calcola. ○

Teorema 18 (del padding). *Per ogni $i \in \mathbb{N}$ e $k \in \mathbb{N}$, esiste $j \in \mathbb{N}$ tale che:*

$$\varphi_j = \varphi_i \wedge j > k. \tag{5.32}$$

In altri termini, esiste sempre un programma j corrispondente a un indice maggiore di k tale che la funzione calcolata da j equivale a quella calcolata da i. Questo indica che ci sono infiniti codici che calcolano la stessa funzione.[1]

Esempio. Si considerino i seguenti programmi:

$$P = \text{input}(x);\ \text{proc}(x);\ \text{output}(x); \tag{5.33}$$

$$P' = \text{input}(x);\ \text{proc}(x);\ \text{skip};\ \text{output}(x); \tag{5.34}$$

In questo caso: $P \neq P'$, ma $[\![P]\!] = [\![P']\!]$. ◇

Definizione (Slicing). *Si definisce* slicing *l'operazione di trasformazione del programma in uno equivalente per un determinato insieme di parametri di input e/o risultati mediante una funzione di trasformazione t.*

Esempio. Se si considera l'indice x applicato alla funzione t si ottiene un nuovo indice $y = t(x)$ tale che:

$$\forall z \in I\colon \varphi_x(z) = \varphi_{t(x)}(z) = \varphi_y(z). \tag{5.35}$$

◇

La compilazione di un programma è un processo di slicing che preserva il comportamento della funzione originale per tutti gli input:

$$\forall z \in \mathbb{N}\colon \varphi_x(z) = \varphi_{t(x)}(z) = \varphi_y(z). \tag{5.36}$$

5.3 Teoremi di ricorsione

Teorema 19 (Primo teorema di ricorsione di Kleene). *Sia t una funzione ricorsiva totale. Allora esiste un indice $n \in \mathbb{N}$ tale che:*

$$\varphi_n = \varphi_{t(n)}. \tag{5.37}$$

L'indice n viene detto *punto fisso* di t. Il teorema afferma quindi che esiste un programma (di indice n) su cui la trasformazione t non cambia la semantica della funzione calcolata.

Definizione (Proprietà). *L'insieme $\pi \subseteq \mathbb{N}$ è detto* proprietà *se contiene gli oggetti dell'universo che verificano una certa condizione, detta appunto* proprietà.

Esempi. I seguenti insiemi sono esempi di proprietà:

$$\{\, x \mid x \text{ è pari} \,\} = \{\, 0, 2, 4, \ldots, 2n \,\}. \tag{5.38}$$

$$\{\, x \mid x \text{ è multiplo di } 3 \,\} = \{\, 0, 3, 6, \ldots, 3n \,\}. \tag{5.39}$$

$$\{\, x \mid x \text{ è quadrato perfetto} \,\} = \{\, 0, 1, 4, \ldots, n^2 \,\}. \tag{5.40}$$

◇

[1] Banalmente, è sufficiente aggiungere istruzioni come `skip` oppure autoassegnamenti.

Definizione (Proprietà estensionale). *Una proprietà $\pi \subseteq \mathbb{N}$ è detta* estensionale *se:*

$$\forall x, y \in \mathbb{N}\colon\ x \in \pi \wedge \varphi_x = \varphi_y \implies y \in \pi. \tag{5.41}$$

Una proprietà estensionale, dunque, descrive solo la funzione calcolata e non il programma che la calcola.

Esempi.

- $\{\, x \mid \varphi_x \text{ è totale} \,\}$ è una proprietà estensionale. Infatti, se $x \in \pi$ e $\varphi_x = \varphi_y$, e, per ipotesi:
  - φ_x è totale, cioè $\forall z \in \mathbb{N}\colon\ \varphi_x(z)\downarrow$,
  - $\forall z \in \mathbb{N}\colon\ \varphi_x(z) = \varphi_y(z)$,

  allora anche φ_y è totale, ovvero $y \in \pi$.
- $\{\, x \mid \varphi_x(5) = 0 \,\}$ è una proprietà estensionale. Infatti, se $x \in \pi$ e $\varphi_x = \varphi_y$:
$$\varphi_x(5) = 0 \implies \varphi_y(5) = 0 \implies y \in \pi. \tag{5.42}$$
- $\{\, x \mid \varphi_x(x) = 0 \,\}$ non è una proprietà estensionale. Infatti $x \in \pi \implies \varphi_x(x) = 0$ e per ogni y tale che $\varphi_y = \varphi_x$ vale:
$$\varphi_y(x) = \varphi_x(x) = 0, \tag{5.43}$$
ma non è detto che $\varphi_y(y) = 0$:
$$\varphi_y(y) = \varphi_x(y) = ? \tag{5.44}$$

◇

Teorema 20 (di Rice). *Sia $\pi \subseteq \mathbb{N}$ una proprietà estensionale. Allora π è ricorsivo se e solo se è banale.*

La banalità di π consiste nell'essere vuoto (non contiene nessun programma) oppure nel coincidere con $\mathbb{N}$ (contiene tutti i programmi). Il teorema di Rice dice quindi che se π è un insieme estensionale non banale, allora non può essere ricorsivo, ovvero l'appartenenza o meno di elemento non è decidibile.

Dimostrazione (Teorema 20).

($\Longleftarrow$) Se π è banale, ovvero è vuoto o coincide con $\mathbb{N}$, allora è ricorsivo, poiché è possibile prendere come funzione caratteristica $f(x) = 0$ nel primo caso e $f(x) = 1$ nel secondo.

($\Longrightarrow$) Per ipotesi π è estensionale, ovvero se $x \in \pi$ e $\varphi_x = \varphi_y$, allora $y \in \pi$. Supponiamo ora che $\pi \neq \emptyset$ e $\pi \neq \mathbb{N}$. Le due condizioni corrispondono a:

$$\pi \neq \emptyset \iff \exists a \in \mathbb{N}.a \in \pi, \tag{5.45}$$
$$\pi \neq \mathbb{N} \iff \exists b \in \mathbb{N}.b \notin \pi. \tag{5.46}$$

Poiché π verifica entrambe le condizioni, esistono $a, b \in \mathbb{N}$ tali che $a \in \pi$ e $b \notin \pi$. Inoltre, π è ricorsivo, quindi esiste una funzione calcolabile $f_\pi\colon \mathbb{N} \to \mathbb{N}$ tale che:

$$\forall x \in \mathbb{N}\colon\ f_\pi(x) = \begin{cases} 1 & \text{se } x \in \pi \\ 0 & \text{se } x \notin \pi. \end{cases} \tag{5.47}$$

Costruiamo una nuova funzione $h\colon \mathbb{N} \to \mathbb{N}$ tale che:

$$\forall x \in \mathbb{N}\colon\ h(x) = \begin{cases} b & \text{se } f_\pi(x) = 1 \\ a & \text{se } f_\pi(x) = 0. \end{cases} \tag{5.48}$$

Lo pseudocodice associato è:

```
1  input(x)
2  if f_π(x) == 1
3      return b
4  else
5      return a
```

Anche h è totale ricorsiva, dunque calcolabile e per il Teorema 19:

$$\exists n_0 \in \mathbb{N}.\varphi_{n_0} = \varphi_{h(n_0)}. \tag{5.49}$$

Verifichiamo ora cosa accade se n_0 sta in π o meno:

- Se $n_0 \in \pi$: per ipotesi π è estensionale, quindi poiché $\varphi_{n_0} = \varphi_{h(n_0)}$, allora $h(n_0) \in \pi$. Tuttavia, $h(n_0) = b$ perché $f_\pi(n_0) = 1$, quindi $b \in \pi$, mentre $b \notin \pi$ per costruzione.
- Se $n_0 \notin \pi$: per ipotesi π è estensionale, quindi poiché $\varphi_{n_0} = \varphi_{h(n_0)}$, allora $h(n_0) \notin \pi$. Tuttavia, $h(n_0) = a$ perché $f_\pi(n_0) = 0$, quindi $a \notin \pi$, mentre $a \in \pi$ per costruzione.

In entrambi i casi si è ottenuto un assurdo, dunque π – estensionale – non può essere ricorsivo se non è banale. □

Esempi.

- L'insieme:
$$I = \{\, P \in \text{PL} \mid \forall n \in \mathbb{N}\colon [\![P]\!](n)\downarrow \,\}, \tag{5.50}$$
dove l'insieme PL corrisponde all'insieme dei linguaggi di programmazione, è:
  - Estensionale, poiché dipende dalla semantica e non dal codice del programma.
  - Ricorsivo? Non essendo banale (si riescono a trovare programmi che terminano per ogni input e programmi che non terminano per nessun input), non è ricorsivo per il Teorema 20.
- L'insieme:
$$L = \{\, P \in \text{PL} \mid P \text{ contiene un ciclo } \texttt{while} \,\} \tag{5.51}$$
è:
  - Non estensionale, perché nella sua definizione compare un vincolo sul codice del programma.
  - Ricorsivo, perché si può sempre verificare algoritmicamente se un programma contiene un ciclo `while`.
- L'insieme:
$$J = \{\, P \in \text{PL} \mid |P| < 1000 \,\}, \tag{5.52}$$
dove $|P|$ indica il numero di righe di codice del programma P, è:
  - Non estensionale, perché preso un programma $P' \in J$, attraverso il Teorema 18 è possibile costruire un programma P tale che $|P| > 1000$ e $[\![P]\!] = [\![P']\!]$, ma $P \notin J$.
  - Ricorsivo, perché si può sempre contare algoritmicamente il numero di righe (finito) di un programma.

- L'insieme:

$$R = \{\, P \in \text{PL} \mid P \text{ soddisfa le specifiche di I/O} \,\} \tag{5.53}$$

è estensionale, perché si vincola il comportamento di P ma non il suo codice. Tuttavia, non è ricorsivo perché non è banale. ◇

5.4 Riducibilità funzionale

La riducibilità funzionale è una tecnica che permette di sfruttare informazioni riguardanti un problema noto per risolverne un'altro.

Si è dimostrato che l'insieme K non è REC e che $\overline{K}$ non è RE, pertanto, poiché valgono le seguenti relazioni:

$$\underset{\text{REC}}{\emptyset} \subseteq \underset{\neg\text{REC}}{K} \subseteq \underset{\text{REC}}{\mathbb{N}}. \tag{5.54}$$

$$\underset{\text{REC}}{\emptyset} \subseteq \underset{\neg\text{RE}}{\overline{K}} \subseteq \underset{\text{REC}}{\mathbb{N}}. \tag{5.55}$$

l'inclusione non propaga la classe di ricorsività, essendo $\emptyset, \mathbb{N} \in \text{REC}$.

Definizione (Riduzione funzionale). *Siano $A, B \subseteq \mathbb{N}$. Si dice che A è* riducibile funzionalmente *a B mediante f:*

$$A \preccurlyeq_f B \tag{5.56}$$

se esiste una funzione totale $f\colon \mathbb{N} \to \mathbb{N}$ tale che:

$$\forall x \in \mathbb{N}\colon x \in A \iff f(x) \in B. \tag{5.57}$$

Si definisce ora un nuovo insieme, K_2:

$$K_2 = \{\, \langle x, y\rangle \mid \varphi_x(y)\downarrow \,\}. \tag{5.58}$$

Si vuole dimostrare che $K \preccurlyeq K_2$.

Dimostrazione. Dobbiamo mostrare che esiste una funzione f tale he $x \in K \iff f(x) \in K_2$. Si definisce $f\colon \mathbb{N} \to \mathbb{N}^2$ come:

$$f(x) = \langle x, x\rangle. \tag{5.59}$$

Infatti:

$$x \in K \overset{\text{def. } K}{\implies} \varphi_x(x)\downarrow \overset{\text{def. } K_2}{\implies} \langle x, x\rangle \in K_2. \tag{5.60}$$

$$\langle x, x\rangle \in K_2 \overset{\text{def. } K_2}{\implies} \varphi_x(x)\downarrow \overset{\text{def. } K}{\implies} x \in K. \tag{5.61}$$

□

La riduzione funzionale è una relazione:

Transitiva Se $A \preccurlyeq_{f_1} B$ e $B \preccurlyeq_{f_2} C$ allora $A \preccurlyeq_f C$, prendendo $f = f_2 \circ f_1$.

Riflessiva $A \preccurlyeq_f A$ con $f(x) = x$.

Definizione (Insieme RE completo). *Un insieme $A \in \text{RE}$ si dice completo se, per ogni $B \in \text{RE}$, vale $B \preccurlyeq A$.*

Osservazione. Non esistono insiemi completi ricorsivi: se ne esistesse uno sia ricorsivo che completo, ad esempio A, poiché K è RE, lo si potrebbe ridurre ad A ($A \in \text{REC} \implies A \in \text{RE}$). Così facendo, però, la ricorsività di A si propagherebbe anche a K. Si è invece dimostrato che K non è ricorsivo. △

Si vuole dimostrare che K_2 è completo. In questo modo, per dimostrare che un insieme è RE, è sufficiente ridurlo a K_2.

Teorema 21. *K_2 è completo.*

Dimostrazione (Teorema 21). Se $A \in \text{RE}$ per ipotesi, allora esiste un $y_0 \in \mathbb{N}$ tale che:

$$A = \text{dom}(\varphi_{y_0}) = W_{y_0}. \tag{5.62}$$

Per ogni $x \in \mathbb{N}$, $x \in A = W_{y_0}$, se e solo se $\varphi_{y_0}(x)\downarrow$. In altri termini, se e solo se $\langle y_0, x\rangle \in K_2$. Se $f_{y_0}(x) = \langle y_0, x\rangle$, allora $x \in A \iff f_{y_0}(x) \in K_2$. Chiaramente y_0 dipende da A. □

Teorema 22. *K è completo.*

Se anche K è completo, allora $K_2 \preccurlyeq K$. Dunque è possibile utilizzare K al posto di K_2 per la riduzione funzionale, dal momento che è più semplice da trattare.

Dimostrazione (Teorema 22). Mostriamo ancora una volta l'esistenza della funzione totale di riduzione:

$$\begin{gathered} f\colon \mathbb{N}^2 \to \mathbb{N}, \\ \langle x, y\rangle \in K_2 \iff f(x, y) \in K. \end{gathered} \tag{5.63}$$

Costruiamo la funzione parziale ricorsiva[2] $\psi\colon \mathbb{N}^3 \to \mathbb{N}$ tale che:

$$\psi(x, y, z) = \begin{cases} 1 & \text{se } \varphi_x(y)\downarrow \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.64}$$

Lo pseudocodice corrispondente è:

```
1  input(x, y, z)
2  costruisci φ_x
3  flag := FALSE
4  while ¬flag
5      esegui next step di φ_x(y)
6      if φ_x(y)↓
7          flag := TRUE
8  return 1
```

Per il teorema SMN, esiste una funzione totale ricorsiva $g(x, y)$ tale che:

$$\forall z \in \mathbb{N}\colon \psi(x, y, z) = \varphi_{g(x,y)}(z). \tag{5.65}$$

Bisogna dimostrare:

$$\langle x, y\rangle \in K_2 \iff g(x, y) \in K. \tag{5.66}$$

[2] La parzialità è necessaria per poter applicare il teorema SMN, la ricorsività per non aggiungere complessità.

in entrambi i versi:

($\Longrightarrow$)

$$\begin{aligned}
\langle x, y\rangle \in K_2 &\implies \varphi_x(y)\downarrow && \text{definizione di } K_2 \\
&\implies \forall z \in \mathbb{N}\colon \psi(x, y, z)\downarrow && \text{definizione di } \psi \\
&\implies \forall z \in \mathbb{N}\colon \varphi_{g(x,y)}(z)\downarrow && \text{teorema SMN} \\
&\implies \varphi_{g(x,y)}(g(x, y))\downarrow && z = g(x, y) \\
&\implies g(x, y) \in K. && \text{definizione di } K
\end{aligned} \tag{5.67}$$

($\Longleftarrow$)

$$\begin{aligned}
\langle x, y\rangle \notin K_2 &\implies \varphi_x(y)\uparrow && \text{definizione di } K_2 \\
&\implies \forall z \in \mathbb{N}\colon \psi(x, y, z)\uparrow && \text{definizione di } \psi \\
&\implies \forall z \in \mathbb{N}\colon \varphi_{g(x,y)}(z)\uparrow && \text{teorema SMN} \\
&\implies \varphi_{g(x,y)}(g(x, y))\uparrow && z = g(x, y) \\
&\implies g(x, y) \notin K. && \text{definizione di } K
\end{aligned} \tag{5.68}$$

Pertanto, si può concludere che $K_2 \preccurlyeq K$. □

Altre proprietà della riduzione funzionale sono:

$$A \preccurlyeq B \implies \bar{A} \preccurlyeq \bar{B}. \tag{5.69}$$

$$A \preccurlyeq B \wedge B \in \text{RE} \implies A \in \text{RE}. \tag{5.70}$$

$$A \preccurlyeq B \wedge B \in \text{REC} \implies A \in \text{REC}. \tag{5.71}$$

Le ultime due implicazioni affermano che la ricorsività e la ricorsiva enumerabilità si propagano da destra verso sinistra per la riduzione funzionale.

5.5 Insiemi creativi e produttivi

Definizione (Insieme produttivo). *Un insieme A è produttivo se e solo se esiste una funzione f_A totale ricorsiva tale che:*

$$\forall x \in \mathbb{N}\colon \left(W_x \subseteq A \implies f_A(x) \in A \setminus W_x\right). \tag{5.72}$$

Osservazione. Un insieme produttivo non può essere RE: se lo fosse, infatti, esisterebbe un $n_0 \in \mathbb{N}$ tale che $A = W_{n_0}$. Ma allora, se si considera $x = n_0$: $W_{n_0} \subseteq A \implies f_A(x) \in A \setminus W_{n_0} = \emptyset$, il che è assurdo perché f_A è totale per definizione. △

Definizione (Insieme creativo). *Un insieme A è creativo se è* RE *e il suo complemento $\bar{A}$ è produttivo.*

Osservazione. Se un insieme è creativo si può certamente concludere che il suo complemento è produttivo; se l'insieme è, invece, produttivo, non si può dire nulla a priori sul suo complemento. △

Se un insieme è creativo, dire se un elemento vi appartiene è un'operazione calcolabile; viceversa, se è produttivo, non lo è. Valgono inoltre le seguenti implicazioni sulla produttività e la creatività degli insiemi:

$$A \preccurlyeq B \wedge A \in \text{PROD} \implies B \in \text{PROD}. \tag{5.73}$$

$$A \preccurlyeq B \wedge A \in \text{CREAT} \wedge B \in \text{RE} \implies B \in \text{CREAT}. \tag{5.74}$$

Al contrario della ricorsività e della ricorsiva enumerabilità, la produttività e la creatività di insiemi si propaga da sinistra verso destra.

Teorema 23 (Creatività e completezza). *Un insieme A è creativo se e solo se è completo.*

Dimostrazione (Teorema 23). Si dimostra qui solo un lato della doppia implicazione:

($\Longrightarrow$) Sia $A \in \text{RE}$ completo, ovvero $\forall B \in \text{RE}\colon B \preccurlyeq A$. Anche $K \in \text{RE}$, dunque $K \preccurlyeq A$; questo equivale a $\overline{K} \preccurlyeq \overline{A}$. Poiché $\overline{K}$ è produttivo, anche $\overline{A}$ è produttivo per (5.73) e A è creativo perché ne è il complemento. □

Si sintetizzano ora le condizioni di appartenenza alle diverse classi di ricorsività:

- Se $x \in A$ e $x \notin A$ sono problemi decidibili, allora A e $\overline{A} \in \text{REC}$.
- Se $x \in A$ è un problema decidibile, ma $x \notin A$ non lo è, allora $\overline{A} \in \text{RE}$ ed è creativo. Il complemento $\overline{A}$ è invece produttivo.
- Se $x \in A$ non è un problema decidibile, allora A è produttivo. Il suo complemento può essere:
  - Creativo, se si dimostra che $\overline{A} \in \text{RE}$.
  - Produttivo, se si dimostra che $\overline{K} \preccurlyeq \overline{A}$.

Esempi. Gli esempi che seguono dimostrano la creatività degli insiemi A.

- Si consideri l'insieme:

$$A = \{\, x \mid \varphi_x(2^x)\downarrow \,\}. \tag{5.75}$$

1. *Si dimostra che $A \in \text{RE}$.* La semicaratteristica di A è:

$$\varphi_A(x) = \begin{cases} 1 & \text{se } x \in A \iff \varphi_x(2^x)\downarrow \\ \uparrow & \text{altrimenti} \end{cases} \tag{5.76}$$

il cui pseudocodice equivalente è:

```
1  input(x)
2  z := 2^x
3  costruisci φ_x
4  stop := FALSE
5  while ¬stop
6      esegui next step di φ_x(z)
7      if φ_x(z)↓
8          stop := TRUE
9  return 1
```

2. *Si costruisce la funzione di riduzione ψ.* Si consideri la funzione $\psi(x, y)$ parziale ricorsiva definita come:

$$\psi(x, y) = \begin{cases} 1 & \text{se } x \in K \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.77}$$

Questa funzione è parziale ricorsiva dato che l'appartenenza a un insieme RE è decidibile. Pertanto, il teorema SMN garantisce l'esistenza di una funzione totale ricorsiva $g(x)$ tale che $\varphi_{g(x)} = \psi(x, y)$.

In ogni caso, lo pseudocodice equivalente è:

```
1  input(x)
2  costruisci φ_x
3  stop ≔ FALSE
4  while ¬stop
5      esegui next step di φ_x(z)
6      if φ_x(x)↓
7          stop ≔ TRUE
8  return 1
```

3. *Si dimostra che* $K \preccurlyeq A$.

$$\begin{aligned} x \in K &\implies \forall y \in \mathbb{N}\colon \psi(x,y)\downarrow && \text{definizione di } \psi \\ &\implies \forall y \in \mathbb{N}\colon \varphi_{g(x)}(y)\downarrow && \text{teorema SMN} \\ &\implies \varphi_{g(x)}\left(2^{g(x)}\right)\downarrow && y = 2^{g(x)} \\ &\implies g(x) \in A. && \text{definizione di } A \end{aligned} \tag{5.78}$$

$$\begin{aligned} x \notin K &\implies \forall y \in \mathbb{N}\colon \psi(x,y)\uparrow && \text{definizione di } \psi \\ &\implies \forall y \in \mathbb{N}\colon \varphi_{g(x)}(y)\uparrow && \text{teorema SMN} \\ &\implies \varphi_{g(x)}\left(2^{g(x)}\right)\uparrow && y = 2^{g(x)} \\ &\implies g(x) \notin A. && \text{definizione di } A \end{aligned} \tag{5.79}$$

- Si consideri l'insieme:

$$A = \{\, x \mid \varphi_x(2^x) = 2 \,\}. \tag{5.80}$$

La principale differenza rispetto al precedente esempio è il vincolo sul valore di terminazione.

1. *Si dimostra che* $A \in \text{RE}$. La semicaratteristica di A è:

$$\varphi_A(x) = \begin{cases} 1 & \text{se } x \in A \iff \varphi_x(2^x) = 2 \\ \uparrow & \text{altrimenti} \end{cases} \tag{5.81}$$

il cui pseudocodice equivalente è:

```
1  input(x)
2  z ≔ 2^x
3  costruisci φ_x
4  stop ≔ FALSE
5  while ¬stop
6      esegui next step di φ_x(z)
7      if φ_x(z) == 2
8          stop ≔ TRUE
9  return 1
```

2. *Si costruisce la funzione di riduzione* ψ. Si consideri la funzione $\psi(x,y)$ parziale ricorsiva definita come:

$$\psi(x,y) = \begin{cases} 2 & \text{se } x \in K \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.82}$$

Questa funzione è parziale ricorsiva, pertanto è applicabile il teorema SMN.

3. *Si dimostra che* $K \preccurlyeq A$.

$$\begin{aligned} x \in K &\implies \forall y \in \mathbb{N}\colon \psi(x, y) = 2 && \text{definizione di } \psi \\ &\implies \forall y \in \mathbb{N}\colon \varphi_{g(x)}(y) = 2 && \text{teorema SMN} \\ &\implies \varphi_{g(x)}\left(2^{g(x)}\right) = 2 && y = 2^{g(x)} \\ &\implies g(x) \in A. && \text{definizione di } A \end{aligned} \tag{5.83}$$

$$\begin{aligned} x \notin K &\implies \forall y \in \mathbb{N}\colon \psi(x, y)\uparrow \neq 2 && \text{definizione di } \psi \\ &\implies \forall y \in \mathbb{N}\colon \varphi_{g(x)}(y)\uparrow \neq 2 && \text{teorema SMN} \\ &\implies \varphi_{g(x)}\left(2^{g(x)}\right)\uparrow \neq 2 && y = 2^{g(x)} \\ &\implies g(x) \notin A. && \text{definizione di } A \end{aligned} \tag{5.84}$$

- Si consideri l'insieme:

$$A = \{\, x \mid \varphi_x(2^x) = x^2 \,\}. \tag{5.85}$$

In questo caso, vogliamo che il valore restituito da φ sia legato al valore di x.

1. *Si dimostra che* $A \in \text{RE}$. La semicaratteristica di A è:

```
1  input(x)
2  z := 2^x
3  costruisci φ_x
4  stop := FALSE
5  while ¬stop
6      esegui next step di φ_x(z)
7      if φ_x(z) = x^2
8          stop := TRUE
9  return 1
```

$$\varphi_A(x) = \begin{cases} 1 & \text{se } \varphi_x(2^x) = x^2 \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.86}$$

2. *Si costruisce la funzione di riduzione* ψ. Si consideri la funzione $\psi(x, y)$ parziale ricorsiva definita come:

$$\psi(x, y) = \begin{cases} z^2 & \text{se } x \in K \land \exists z \in \mathbb{N}.\, y = 2^z \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.87}$$

Lo pseudocodice associato a ψ è:

```
1  input(x, y)
2  z := 0
3  w := 1
4  while y ≠ w
5      z := z + 1
6      w := 2^z
7  costruisci φ_x
8  stop := FALSE
9  while ¬stop
10     esegui next step di φ_x(x)
11     if φ_x(x)↓
12         stop := TRUE
13 return z^2
```

3. *Si dimostra che $K \preccurlyeq A$.*

$$\begin{aligned}
x \in K &\implies (\psi(x,y) = z^2 \iff \exists z \in \mathbb{N}.\, y = 2^z) && \text{definizione di } \psi \\
&\implies (\varphi_{g(x)}(y) = z^2 \iff \exists z \in \mathbb{N}.\, y = 2^z) && \text{teorema SMN} \\
&\implies \varphi_{g(x)}\left(2^{g(x)}\right) = g^2(x) && y = 2^{g(x)}, z = g(x) \\
&\implies g(x) \in A. && \text{definizione di } A
\end{aligned} \tag{5.88}$$

$$\begin{aligned}
x \notin K &\implies \forall y \in \mathbb{N}\colon \psi(x,y)\uparrow \neq z^2 && \text{definizione di } \psi \\
&\implies \forall y \in \mathbb{N}\colon \varphi_{g(x)}(y)\uparrow \neq z^2 && \text{teorema SMN} \\
&\implies \varphi_{g(x)}\left(2^{g(x)}\right)\uparrow \neq g^2(x) && y = 2^{g(x)}, z = g(x) \\
&\implies g(x) \notin A. && \text{definizione di } A
\end{aligned} \tag{5.89}$$

- Si consideri l'insieme:

$$A = \{\, x^4 \mid \varphi_x(2^x) = x^2 \,\}. \tag{5.90}$$

Questa volta, gli elementi dell'insieme sono di una particolare forma.

1. *Si dimostra che $A \in$ RE.* La semicaratteristica di A è:

$$\varphi_A(x) = \begin{cases} 1 & \text{se } \exists y \in \mathbb{N}. x = y^4 \wedge \varphi_y(2^y) = y^2 \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.91}$$

e il relativo pseudocodice è:

```
1  input(x)
2  y := 0
3  z := 0
4  while x ≠ z
5      y := y + 1
6      z := y^4
7  z := 2^y
8  w := y^2
9  costruisci φ_y
10 stop := FALSE
11 while ¬stop
12     esegui next step di φ_y(z)
13     if φ_y(z) == w
14         stop := TRUE
15 return 1
```

2. *Si costruisce la funzione di riduzione ψ.* Si consideri la funzione $\psi(x,y)$ parziale ricorsiva definita come:

$$\psi(x,y) = \begin{cases} z^2 & \text{se } x \in K \wedge \exists z \in \mathbb{N}.\, y = 2^z \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.92}$$

Lo pseudocodice associato a ψ è:

```
1  input(x, y)
2  z := 0
3  w := 1
4  while y ≠ w
5      z := z + 1
6      w := 2^z
7  costruisci φ_x
8  stop := FALSE
9  while ¬stop
10     esegui next step di φ_x(x)
11     if φ_x(x)↓
12         stop := TRUE
13 return z^2
```

3. *Si dimostra che* $K \preccurlyeq A$.

$$\begin{aligned}
x \in K &\implies (\psi(x,y) = z^2 \iff \exists z \in \mathbb{N}.\, y = 2^z) && \text{definizione di } \psi \\
&\implies (\varphi_{g(x)}(y) = z^2 \iff \exists z \in \mathbb{N}.\, y = 2^z) && \text{teorema SMN} \\
&\implies \varphi_{g(x)}\left(2^{g(x)}\right) = g^2(x) && y = 2^{g(x)},\ z = g(x) \\
&\implies f(x) = g^4(x) \in A. && \text{definizione di } A
\end{aligned} \tag{5.93}$$

$$\begin{aligned}
x \notin K &\implies \forall y \in \mathbb{N}\colon \psi(x,y)\uparrow \neq z^2 && \text{definizione di } \psi \\
&\implies \forall y \in \mathbb{N}\colon \varphi_{g(x)}(y)\uparrow \neq z^2 && \text{teorema SMN} \\
&\implies \varphi_{g(x)}\left(2^{g(x)}\right)\uparrow \neq g^2(x) && y = 2^{g(x)},\ z = g(x) \\
&\implies f(x) = g^4(x) \notin A. && \text{definizione di } A
\end{aligned} \tag{5.94}$$

Nota. Questo insieme può essere anche rappresentato in modo equivalente come:

$$A = \{\, x \mid \exists y \in \mathbb{N}.x = y^4 \wedge \varphi_y(2^y) = y^2 \,\}. \tag{5.95}$$

○

- Si consideri l'insieme:

$$A = \{\, x \mid \exists y \in \mathbb{N}.x = y^4 \implies \varphi_y(2^y) = y^2 \,\}. \tag{5.96}$$

Questa scrittura è molto simile a quella alternativa del precedente esempio, tuttavia è necessario prestare attenzione: poiché l'implicazione è verificata anche quando l'antecedente è falso, in A sono contenuti anche tutti gli indici che non sono della forma y^4.

1. *Si dimostra che* $A \in \mathrm{RE}$. La semicaratteristica di A è:

$$\varphi_A(x) = \begin{cases} 1 & \text{se } \neg\exists y \in \mathbb{N}.x = y^4 \vee \varphi_y(2^y) = y^2 \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.97}$$

e il relativo pseudocodice è:

```
1  input(x)
2  for y = 0 to x
3      z := y^4
4      if z = y^4
5          costruisci φ_y
6          w := 2^y
7          z = y^2
8          stop := FALSE
9          while ¬stop
10             esegui next step di φ_y(w)
11             if φ_y(w) == z
12                 stop := TRUE
13         return 1
14 return 1
```

2. *Si costruisce la funzione di riduzione ψ*, come per l'esempio precedente.
3. *Si dimostra che $K \preccurlyeq A$*, come per l'esempio precedente.

- Si consideri l'insieme:

$$A = \{\, y^4 \mid \varphi_{y-5}(2^y) = y^2 \,\}. \tag{5.98}$$

Per questo insieme si esegue una semplice sostituzione, al fine di portare l'indice della funzione φ a x. Si pone, dunque: $y - 5 = x \iff y = x + 5$ e si riscrive l'insieme come:

$$A = \left\{\, (x+5)^4 \mid \varphi_x\left(2^{x+5}\right) = (x+5)^2 \,\right\}. \tag{5.99}$$

1. *Si dimostra che $A \in$ RE*. La semicaratteristica di A è:

$$\varphi_A(x) = \begin{cases} 1 & \text{se } \varphi_x\left(2^{x+5}\right) = (x+5)^2 \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.100}$$

e il relativo pseudocodice è:

```
1  input(x)
2  z := 2^{x+5}
3  costruisci φ_x
4  stop := FALSE
5  while ¬stop
6      esegui next step di φ_x(z)
7      if φ_x(z) == (x + 5)^2
8          stop := TRUE
9  return 1
```

2. *Si costruisce la funzione di riduzione ψ*. Si consideri la funzione $\psi(x, y)$ parziale ricorsiva definita come:

$$\psi(x, y) = \begin{cases} (z+5)^2 & \text{se } x \in K \wedge \exists z \in \mathbb{N}.\, y = 2^{z+5} \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.101}$$

Lo pseudocodice associato a ψ è:

```
1  input(x)
2  y := 5
3  w := 5^4
4  while x ≠ w
5      y := y + 1
6      w := y^4
7  costruisci φ_y
8  z := 2^(y+5)
9  w := (y + 5)^2
10 stop := FALSE
11 while ¬stop
12     esegui next step di φ_x(z)
13     if φ_x(z) == w
14         stop := TRUE
15 return 1
```

3. *Si dimostra che $K \preccurlyeq A$.*

$$\begin{aligned}
x \in K &\implies (\psi(x,y) = (z+5)^2 \iff \exists z \in \mathbb{N}. y = 2^{z+5}) && \text{definizione di } \psi \\
&\implies (\varphi_{g(x)}(y) = (z+5)^2 \iff \exists z \in \mathbb{N}. y = 2^{z+5}) && \text{teorema SMN} \\
&\implies \varphi_{g(x)}\left(2^{g(x)+5}\right) = (g(x)+5)^2 && y = 2^{g(x)+5} \\
&\implies f(x) = (g(x)+5)^4 \in A. && \text{definizione di } A
\end{aligned} \tag{5.102}$$

$$\begin{aligned}
x \notin K &\implies \forall y \in \mathbb{N}\colon \psi(x,y)\uparrow \neq (z+5)^2 && \text{definizione di } \psi \\
&\implies \forall y \in \mathbb{N}\colon \varphi_{g(x)}(y)\uparrow \neq (z+5)^2 && \text{teorema SMN} \\
&\implies \varphi_{g(x)}\left(2^{g(x)+5}\right)\uparrow \neq (g(x)+5)^2 && y = 2^{g(x)+5} \\
&\implies f(x) = (g(x)+5)^4 \notin A. && \text{definizione di } A
\end{aligned} \tag{5.103}$$

Poiché tutti questi insiemi sono creativi, i relativi complementari sono produttivi. ◇

Esempi. Gli esempi che seguono dimostrano la produttività degli insiemi A.

- Si considerino gli insiemi:

$$\begin{aligned}
A &= \{\, x \mid \forall z \in \mathbb{N}\colon \varphi_x(z) = 4 \,\}, \\
\overline{A} &= \{\, x \mid \exists z \in \mathbb{N}. \varphi_x(z) \neq 4 \,\}.
\end{aligned} \tag{5.104}$$

  Poiché dovremmo controllare che $\varphi_x(z) = 4$ per ogni $z \in \mathbb{N}$, l'insieme A è intuitivamente produttivo. Anche $\overline{A}$ è produttivo, poiché essere diverso da 4 implica la possibilità di divergere.

  1. *Si costruiscono le funzioni di riduzione ψ.* La funzione di riduzione per A è:

$$\psi(x,y) = \begin{cases} 4 & \text{se } x \in K \text{ non deciso in meno di } y \text{ passi} \\ \uparrow & \text{altrimenti} \end{cases} \tag{5.105}$$

il cui pseudocodice equivalente è:

```
1  input(x)
2  costruisci φ_x
3  for z = 0 to y
4      esegui next step di φ_x(x)
5      if φ_x(x)↓
6          while TRUE
7  return 4
```

Quella per $\overline{A}$ è:

$$\psi(x, y) = \begin{cases} 4 & \text{se } x \in K \\ \uparrow & \text{altrimenti,} \end{cases} \tag{5.106}$$

che è banalmente parziale ricorsiva in quanto la terminazione è determinata dall'appartenenza a un insieme RE.

2. *Si dimostra che* $\overline{K} \preccurlyeq A$.

$$\begin{aligned}
x \in K &\implies \exists y_0 \in \mathbb{N}.\varphi_x(x)\downarrow \text{ in } y_0 \text{ passi} && \text{definizione di } K \\
&\implies \forall y \in \mathbb{N}.y > y_0 \colon \varphi_x(x)\downarrow \text{ prima di } y \text{ passi} && \text{definizione di } \varphi_x \\
&\implies \forall y \in \mathbb{N}.y > y_0 \colon \psi(x, y)\uparrow && \text{definizione di } \psi \\
&\implies \forall y \in \mathbb{N}.y > y_0 \colon \varphi_{g(x)}(y)\uparrow \neq 4 && \text{teorema SMN} \\
&\implies \exists y \in \mathbb{N}.y > y_0 \wedge \varphi_{g(x)}(y) \neq 4 && \text{definizione di } \forall \\
&\implies g(x) \in \overline{A}, && \text{definizione di } \overline{A}
\end{aligned} \tag{5.107}$$

$$\begin{aligned}
x \notin K &\implies \forall y \in \mathbb{N} \colon \varphi_x(x)\uparrow \text{ prima di } y \text{ passi} && \text{definizione di } K \\
&\implies \forall y \in \mathbb{N} \colon \psi(x, y) = 4 && \text{definizione di } \psi \\
&\implies \forall y \in \mathbb{N} \colon \varphi_{g(x)}(y) = 4 && \text{teorema SMN} \\
&\implies g(x) \in A. && \text{definizione di } A
\end{aligned} \tag{5.108}$$

3. *Si dimostra che* $\overline{K} \preccurlyeq \overline{A}$.

$$\begin{aligned}
x \in K &\implies \forall y \in \mathbb{N} \colon \psi(x, y) = 4 && \text{definizione di } \psi \\
&\implies \forall y \in \mathbb{N} \colon \varphi_{g(x)}(y) = 4 && \text{teorema SMN} \\
&\implies g(x) \in A, && \text{definizione di } A
\end{aligned} \tag{5.109}$$

$$\begin{aligned}
x \notin K &\implies \forall y \in \mathbb{N} \colon \psi(x, y)\uparrow && \text{definizione di } \psi \\
&\implies \forall y \in \mathbb{N} \colon \varphi_{g(x)}(y)\uparrow && \text{teorema SMN} \\
&\implies \exists y \in \mathbb{N}.\varphi_{g(x)}(y)\uparrow \neq 4 && \text{definizione di } \overline{A} \\
&\implies g(x) \notin A. && \text{definizione di } A
\end{aligned} \tag{5.110}$$

- Si considerino gli insiemi:

$$\begin{aligned}
A &= \{ x \mid W_x = 2\mathbb{N} \}, \\
\overline{A} &= \{ x \mid W_x \neq 2\mathbb{N} \}.
\end{aligned} \tag{5.111}$$

Anche in questo caso dovremmo eseguire per tutti e due gli insiemi un controllo infinito.

1. *Si costruiscono le funzioni di riduzione ψ.* La funzione di riduzione per A è:

$$\psi(x, y) = \begin{cases} 1 & \text{se } x \in K \vee y \in 2\mathbb{N} \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.112}$$

Quella per $\overline{A}$ è:

$$\psi(x, y) = \begin{cases} 1 & \text{se } x \in K \wedge y \in 2\mathbb{N} \\ \uparrow & \text{altrimenti.} \end{cases} \tag{5.113}$$

Entrambe sono banalmente parziali ricorsive in quanto la terminazione è determinata dall'appartenenza a due insiemi RE.

2. *Si dimostra che $\overline{K} \preccurlyeq A$.*

$$\begin{aligned} x \in K &\implies \forall y \in \mathbb{N}\colon \psi(x, y)\downarrow && \text{definizione di } K \\ &\implies \forall y \in \mathbb{N}\colon \varphi_{g(x)}(y)\downarrow && \text{teorema SMN} \\ &\implies W_{g(x)} = \mathbb{N} \neq 2\mathbb{N} && \text{dominio di } g(x) \\ &\implies g(x) \notin A. && \text{definizione di } A \end{aligned} \tag{5.114}$$

$$\begin{aligned} x \notin K &\implies (\varphi_x(x)\downarrow \iff y \in 2\mathbb{N}) && \text{definizione di } \psi \\ &\implies (\varphi_{g(x)}(y)\downarrow \iff y \in 2\mathbb{N}) && \text{teorema SMN} \\ &\implies W_{g(x)} = 2\mathbb{N} && \text{dominio di } g(x) \\ &\implies g(x) \in A. && \text{definizione di } \overline{A} \end{aligned} \tag{5.115}$$

3. *Si dimostra che $\overline{K} \preccurlyeq \overline{A}$.*

$$\begin{aligned} x \in K &\implies (\psi(x, y)\downarrow \iff y \in 2\mathbb{N}) && \text{definizione di } \psi \\ &\implies (\varphi_{g(x)}(y)\downarrow \iff y \in 2\mathbb{N}) && \text{teorema SMN} \\ &\implies W_{g(x)} = 2\mathbb{N} && \text{dominio di } g(x) \\ &\implies g(x) \in A. && \text{definizione di } A \end{aligned} \tag{5.116}$$

$$\begin{aligned} x \notin K &\implies \forall y \in \mathbb{N}\colon \psi(x, y)\uparrow && \text{definizione di } \psi \\ &\implies \forall y \in \mathbb{N}\colon \varphi_{g(x)}(y)\uparrow && \text{teorema SMN} \\ &\implies W_{g(x)} = \emptyset \neq 2\mathbb{N} && \text{dominio di } g(x) \\ &\implies g(x) \notin A. && \text{definizione di } \overline{A} \end{aligned} \tag{5.117}$$

◇

A | Gerarchia di Chomsky

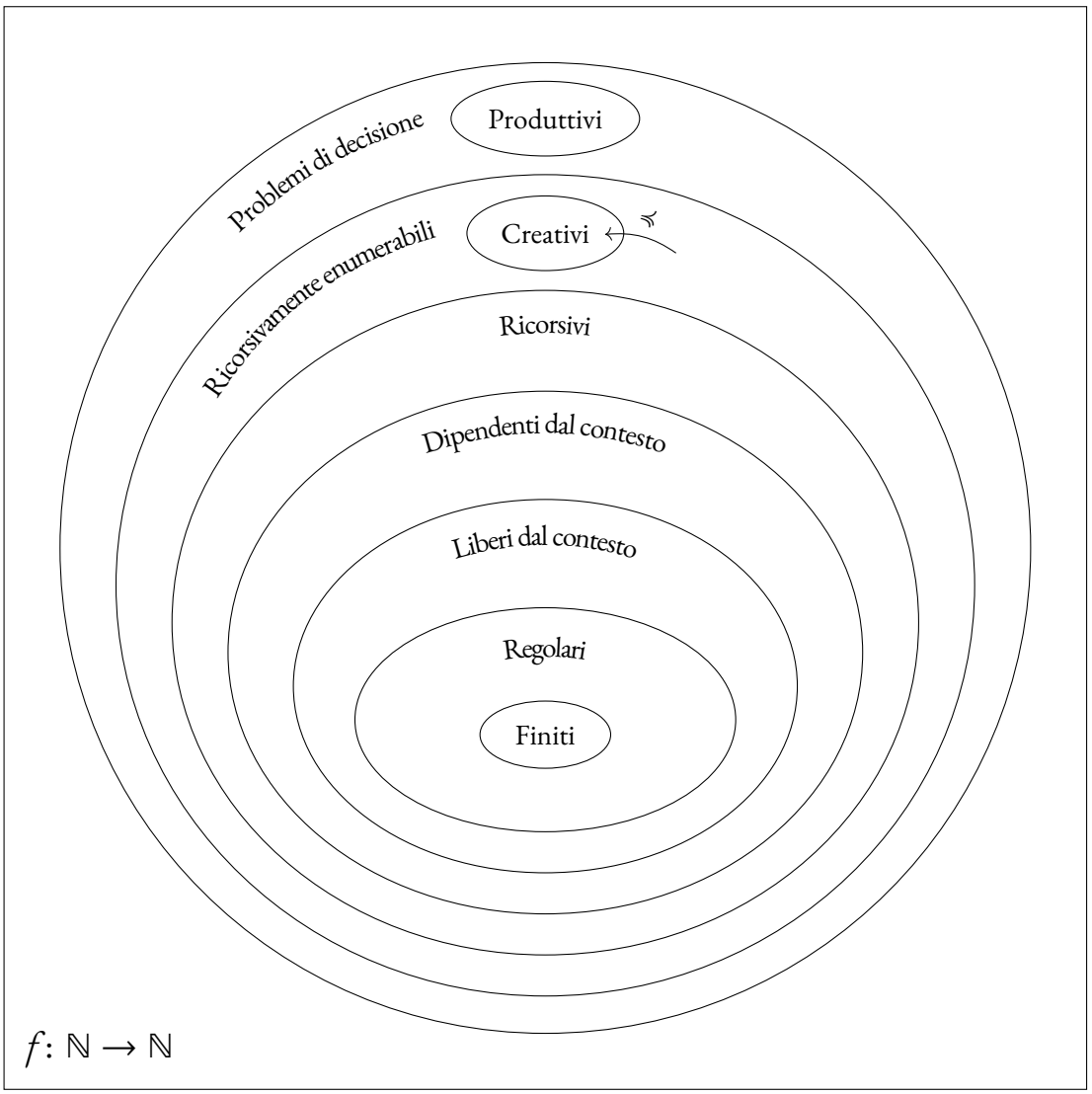


Figura A.1 *I linguaggi e gli insiemi trattati in questo corso.*