Accettabilità e decidibilità: l'Halting problem

Indice

Assodata la tesi di Church-Turing, e dopo la nostra deviazione verso il mondo dell'infinito di Cantor, siamo finalmente pronti a rispondere alla domanda con la quale avevamo iniziato la dispensa precedente: cosa è che può essere calcolato da una macchina di Turing? O meglio, alla luce della tesi di Church-Turing: cosa è che può essere calcolato? O, ancor più precisamente: esiste qualcosa che *non* può essere calcolato?

Esiste un problema che non può essere risolto?

5.1 Quanti sono gli algoritmi?

Certamente, come possiamo immaginare facilmente, esistono un numero infinito di algoritmi. Allora, il senso della domanda "quanti sono gli algoritmi?" va individuato alla luce della teoria cantoriana dei numeri transfiniti, ossia: quale è la cardinalità dell'insieme degli algoritmi? D'altra parte, la tesi di Church-Turing afferma che ad ogni algoritmo corrisponde una macchina di Turing che calcola la stessa funzione e sappiamo, dalla Dispensa 2, che per ogni macchina di Turing definita su un generico alfabeto e dotata di k nastri ne esiste un'altra definita sull'alfabeto $\{0,1\}$ e dotata di un singolo nastro (più l'eventuale nastro di output) ad essa equivalente. Pertanto, quello di cui ci occuperemo in questo paragrafo è calcolare la cardinalità dell'insieme $\mathscr{T}$ delle macchine di Turing definite sull'alfabeto $\{0,1\}$ e dotate di un singolo nastro (più l'eventuale nastro di output).

Il prossimo teorema, seppure interessante di per sé stesso, è funzionale al nostro scopo.

Teorema 5.1: *Sia Σ un insieme finito. Allora l'insieme Σ^* costituito dalle parole (di lunghezza finita) di caratteri di Σ è numerabile.*

Dimostrazione: Sia $|\Sigma| = n$; consideriamo, allora, una qualunque codifica binaria $c : \Sigma \to \{0,1\}^{\lceil \log n \rceil}$ degli elementi di Σ che utilizza lo stesso numero $\lceil \log n \rceil$ di cifre per ciascun elemento di Σ: poiché c è una codifica allora, per costruzione,

$$\forall s,t \in \Sigma : s \neq t \ [c(s) \neq c(t)\ ].$$

Sia $p \in \Sigma^*$ una qualunque parola: poiché p è costituita da un numero finito di caratteri di Σ, indichiamo con $k \in \mathbb{N}$ la lunghezza di p e indichiamo p come $p = s_1 s_2 \ldots s_k$, ossia come concatenazione dei simboli s_1, s_2, ..., s_k, con $s_i \in \Sigma$ per $i = 1, \ldots, k$.

Indichiamo ora con $C(p)$ la concatenazione delle parole (nell'alfabeto binario) $c(s_1)$, $c(s_2)$, ..., $c(s_k)$, e con $\mu(p)$ la parola $C(p)$ preceduta da un carattere '1':

$$\mu(p) = 1\ C(p) = 1\ c(s_1)\ c(s_2)\ \ldots\ c(s_k).$$

Osserviamo che $\mu(p)$ può essere interpretato come numero naturale. Inoltre, è semplice verificare che, se $p, p' \in \Sigma^*$ e $p \neq p'$, allora $\mu(p) \neq \mu(p')$. Infatti, siano $p, p' \in \Sigma^*$ tali che $p = s_1 s_2 \ldots s_k$, $p' = s'_1 s'_2 \ldots s'_{k'}$ e $p \neq p'$:

- se $k > k'$, allora $\mu(p) > \mu(p')$ in quanto $\mu(p)$ contiene $1 + k\lceil \log n \rceil$ cifre e $\mu(p')$ contiene $1 + k'\lceil \log n \rceil < 1 + k\lceil \log n \rceil$ cifre; analogamente se $k < k'$;
- se $k = k'$, allora, poiché $p \neq p'$, esiste almeno un indice i tale che $s_i \neq s'_i$ e, quindi, $\mu(p)$ differisce da $\mu(p')$ per almeno qualche cifra (binaria) in posizione j con $1 + (i-1)\lceil \log n \rceil \leq j \leq 1 + i\lceil \log n \rceil$.

Quindi, μ è una biezione fra Σ^* ed un sottoinsieme (infinito) dei numeri naturali. Allora, per il Teorema 4.1, Σ^* è numerabile. □

Possiamo ora dimostrare il principale risultato di questo paragrafo:

Teorema 5.2: *L'insieme $\mathscr{T}$ delle macchine di Turing definite sull'alfabeto $\{0,1\}$ e dotate di un singolo nastro (più l'eventuale nastro di output) è numerabile.*

Dimostrazione: Ricordiamo che per dimostrare che un insieme è numerabile occorre dimostrare l'esistenza di una biezione fra tale insieme e l'insieme $\mathbb{N}$ dei numeri naturali: tale biezione non è altro che una etichettatura degli elementi dell'insieme con etichette appartenenti ad $\mathbb{N}$, ossia, in ultima analisi, una *numerazione* degli elementi dell'insieme. Vediamo, dunque, come numerare le macchine di Turing.

Sia, dunque, T una macchina di Turing ad un nastro (più il nastro di output se T è di tipo trasduttore) definita sull'alfabeto $\{0,1\}$ e su un insieme finito di stati Q, ove q_0 è lo stato iniziale e q_1 è lo stato di accettazione (ossia, $q_A = q_1$) se T è di tipo riconoscitore, oppure lo stato finale (ossia, $q_F = q_1$) se T è di tipo trasduttore. Analogamente a quanto descritto nel Paragrafo 2.6 della Dispensa 2, sia $b^Q : Q \to \{0,1\}^m$ una codifica binaria degli stati di T che utilizza per ciascuno di essi $m = \lceil \log |Q| \rceil$ cifre, e, per ogni $q \in Q$, indichiamo con $b^Q(q) = b^Q(q)b^Q(q)\ldots b^Q(q)$ la codifica di q. Allora, come abbiamo mostrato nel già citato Paragrafo 6.2 della Dispensa 2, rappresentiamo T mediante la parola $\beta_t \in \Sigma^*$, con $\Sigma = \{0,1,\oplus,\otimes,-,f,s,d\}$ descritta di seguito:

$$\beta_T = b^Q(q_0) - b^Q(q_1) \otimes b^Q(q_{1_1}) - b_{1_1} - b_{1_2} - b^Q(q_{1_2}) - m_1 \oplus \ldots \oplus b^Q(q_{h_1}) - b_{h_1} - b_{h_2} - b^Q(q_{h_2}) - m_h \oplus$$

Si osservi, ora, che T è univocamente caratterizzata una volta che sono definiti l'insieme P delle sue quintuple ed i suoi stati iniziale q_0 e finale q_A o q_F (dipendentemente dall'essere T riconoscitore o trasduttore). Infatti, fissati P, stato iniziale e stato finale, è fissato anche il comportamento della macchina di Turing su qualunque $x \in \{0,1\}^*$. In altri termini, T è univocamente rappresentata dalla parola $\beta_T \in \Sigma^*$ o, ancora più formalmente,

$$\forall\, T, T' \in \mathscr{T} :\ T \neq T' \ \Leftrightarrow\ \beta_T \neq \beta_{T'}.$$

Questo prova che abbiamo costruito una biezione fra $\mathscr{T}$ e un sottoinsieme di Σ^* e, dunque, poiché Σ è un insieme finito, in conseguenza del Teorema 5.1, esso è numerabile. □

Dimostrata la numerabilità di $\mathscr{T}$, osserviamo ora che, per ogni $T \in \mathscr{T}$, possiamo eprimere esplicitamente la biezione fra $\mathscr{T}$ ed un sottoinsieme dei nimeri naturali, ossia, possiamo trasformare la codifica β_T di una macchina di Turing T in un numero naturale $\nu(T)$ in modo tale che, per ogni coppia di macchine di Turing distinte T e T', $\nu(T) \neq \nu(T')$. Questa biezione ci risulterà utile più avanti.

Sia, dunque, $T \in \mathscr{T}$ una macchina di Turing ad un nastro (più, eventualmente, il nastro di output) e sia $\beta_T \in \Sigma^*$ la sua codifica. Iniziamo con il trasformare la parola $\beta_T \in \Sigma^*$ in una parola in $\{0,1,3,4,5,6,7,8,9\}^*$ nel modo seguente:

- sostituendo ogni carattere 's' in β_T con il carattere '5', ogni carattere 'f con il carattere '6', e ogni carattere 'd' con il carattere '7';
- sostituendo ogni carattere '-' in β_T con il carattere '4';
- sostituendo ogni carattere '$\oplus$' in β_T con il carattere '3' e ogni carattere '$\otimes$' con il carattere '2';
- premettendo il carattere '2' alla stringa ottenuta.

La parola in $\{0,1,2,3,4,5,6,7\}^*$ così ottenuta può, ovviamente, essere considerata come un numero espresso in notazione decimale: ecco il numero $\nu(T) \in \mathbb{N}$ associato, univocamente, a T.

5.2 Quanti sono i linguaggi?

Anche in questo caso, analogamente al paragrafo precedente, ci domandiamo quale sia la cardinalità dell'insieme dei linguaggi su un alfabeto finito.

Cominciamo con il valutare la cardinalità dell'insieme delle parole su un alfabeto finito.

Sia Σ un alfabeto e $\mathscr{L}_\Sigma$ l'insieme dei linguaggi su Σ. Vale, allora, il seguente teorema.

Teorema 5.3: *Se Σ è un alfabeto finito, allora l'insieme $\mathscr{L}_\Sigma$ non è numerabile.*

Dimostrazione: I linguaggi su un certo alfabeto Σ sono tutti e soli i sottoinsiemi di Σ^*, ossia, $\mathscr{L}_\Sigma = 2^{\Sigma^*}$. Poiché per il Teorema 5.1 Σ^* è un insieme numerabile, allora, in virtù del Teorema 4.6 l'insieme $\mathscr{L}_\Sigma$ è non numerabile. □

Non rimane che mettere in relazione gli elementi di $\mathscr{L}_\Sigma$ con gli elementi di $\mathscr{T}_\Sigma$, il sottoinsieme di $\mathscr{T}$ delle macchine di Turing di tipo riconoscitore definite sull'alfabeto Σ.

Definizione 5.1: Sia $T \in \mathscr{T}_\Sigma$ una macchina di Turing di tipo riconoscitore definita sull'alfabeto Σ. Il linguaggio $L_A(T) \in \mathscr{L}_\Sigma$ è il sottoinsieme di Σ^* delle parole accettate da T.

Osserviamo che, per definizione, per ogni $T \in \mathscr{T}_\Sigma$ esiste un unico $L \in \mathscr{L}_\Sigma$ tale che $L = L_A(T)$. Dalla non numerabilità di $\mathscr{L}_\Sigma$ e poiché $\mathscr{T}_\Sigma$ è numerabile (in quanto sottoinsieme dell'insieme numerabile di $\mathscr{T}$, in virtù del Teorema 4.1) segue allora il seguente corollario.

Corollario 5.1: *Esiste un linguaggio non accettabile.*

5.3 L'Halting Problem

Il Corollario 5.1 dimostra l'esistenza di un linguaggio non accettabile, ma lascia aperte due questioni. Da un lato, la prova del corollario (e dei teoremi da cui dipende) è puramente esistenziale e, pertanto, inerentemente *non costruttiva*. Questo significa che essa non fornisce alcuna metodologia per *costruire* un linguaggio non accettabile. Inoltre, il corollario non chiarisce il ruolo dei linguaggi decidibili. In altre parole, non chiarisce se i linguaggi non decidibili sono tutti e soli i linguaggi non accettabili oppure se esistono linguaggi accettabili ma non decidibili. Questo è l'obiettivo del presente paragrafo.

Consideriamo la funzione $\nu : \mathscr{T} \to \mathbb{N}$ descritta nel Paragrafo 5.1 e definiamo il linguaggio seguente

$L_H = \{(i,x) : i$ contiene solo le cifre da ' 0' a '7' $\wedge\, i$ è la codifica di una macchina di Turing $\wedge\, T_i(x)$ termina $\} \subseteq \mathbb{N} \times \mathbb{N}$.

Osserviamo che se un numero naturale i non è la codifica di alcuna macchina di Turing in $\mathscr{T}$, allora, comunque si scelga $x \in \{0,1\}^*$, $(i,x) \notin L_H$.

Il resto di questo paragrafo è dedicato a mostrare che L_H è il linguaggio accettabile ma non decidibile cercato.

Teorema 5.4: *L_H è un linguaggio accettabile.*

Dimostrazione: Dobbiamo mostrare che esiste una macchina di Turing T tale che, per ogni $(i,x) \in \mathbb{N} \times \mathbb{N}$, $o_T(i,x) = q_A$ se e soltanto se $(i,x) \in L_H$.

Mostriamo, ora, che la macchina T cercata è, in effetti, una modifica della macchina di Turing universale U definita nella Dispensa 2: la macchina T, a 4 nastri come U, inizia la computazione con il numero i (scritto in notazione decimale) memorizzato sul nastro N_1 e con $x \in \{0,1\}^*$ memorizzato sul nastro N_2. T inizia la sua computazione verificando che i non contenga le cifre '8' e'9' e che inizi con la cifra '2': se non è così, termina nello stato di rigetto altrimenti cancella la cifra '2' iniziale e traduce quello che rimane nell'alfabeto di lavoro Σ di U (in accordo alle regole descritte al termine del Paragrafo 5.1). Poi, T simula la computazione di U e, se U termina (sia nello stato di accettazione che di rigetto) allora T termina nello stato di accettazione. Osserviamo esplicitamente che, se i non è la codifica di alcuna macchina di Turing, allora, poiché l'insieme delle quintuple di una qualsiasi macchina di Turing è totale e le computazioni con input che non rispettano le specifiche non terminano (si ricordi la discussione nel Paragrafo 2.3 della Dispensa 2), allora $U(i,x)$ non termina.

Sia $(i,x) \in L_H$: allora, la computazione $T_i(x)$ termina e quindi, in virtù di quanto appena descritto circa la macchina T, la computazione $T(i,x)$ accetta.
Viceversa, sia $(i,x) \in \mathbb{N} \times \mathbb{N}$ tale che $T(i,x)$ accetta; di nuovo, poiché la computazione $T(i,x)$ simula la computazione $U(i,x)$, allora anche $U(i,x)$ termina e, dunque, i è la codifica di una macchina di Turing e $T_i(x)$ termina: quindi, $(i,x) \in L_H$. □

Prima di procedere, dobbiamo modificare leggermente la notazione che abbiamo utilizzato sino ad ora in queste dispense. Osserviamo che, data una macchina di Turing T ed un suo input x, il valore di $o_T(x)$ è definito soltanto per gli x tali che la computazione $T(x)$ termina. Poiché in quanto segue dovremo pesantemente utilizzare la possibilità che una computazione non termini, con un leggero abuso di notazione indicheremo con $T(x)$ sia la computazione eseguita dalla macchina T sull'input x che il suo esito. Più in particolare, assumeremo

$$T(x) = \begin{cases} q \in Q_F & \text{se la computazione } T(x) \text{ termina} \\ \text{non termina} & \text{se la computazione } T(x) \text{ non termina.} \end{cases}$$

Teorema 5.5: *Il linguaggio L_H non è decidibile.*

Dimostrazione: Supponiamo che L_H sia decidibile. Allora, deve esistere una macchina di Turing T tale che

$$T(i,x) = \begin{cases} q_A & \text{se } (i,x) \in L_H \\ q_R & \text{se } (i,x) \notin L_H. \end{cases}$$

Senza perdità di generalità, possiamo assumere che T sia una macchina ad un singolo nastro.

Da T possiamo, allora, semplicemente complementando gli stati di accettazione e di rigetto di T, derivare una nuova macchina T' (sempre ad un nastro) che, terminando su ogni input (come T), accetta tutte e sole le coppie $(i,x) \in \mathbb{N} \times \mathbb{N} - L_H$, ossia,

$$T'(i,x) = \begin{cases} q_R & \text{se } (i,x) \in L_H \\ q_A & \text{se } (i,x) \notin L_H \end{cases}$$

A partire da T' deriviamo, poi, una terza macchina T^* (ancora ad un nastro) che, invece che su una coppia di interi, opera su un singolo input $i \in \mathbb{N}$. Inoltre, $T^*(i)$ accetta se $T'(i,i)$ accetta, mentre *non termina* se $T'(i,i)$ rigetta. Questo è possibile apportando a T' le seguenti modifiche:

- sostituiamo lo stato q_R con un nuovo stato non finale q'_R in tutte le quintuple di T' che terminano nello stato q_R;
- aggiungiamo alle quintuple di T' la quintupla $\langle q'_R, y, y, q'_R, ferma \rangle$, per ogni $y \in \{0,1\}$.

Allora,

$$T^*(i) = \begin{cases} \text{non termina} & \text{se } T'(i,i) \text{ rigetta} \\ q_A & \text{se } T'(i,i) \text{ accetta.} \end{cases}$$

Poiché $\mathscr{T}$ è un insieme numerabile e $T^* \in \mathscr{T}$, allora deve esistere $k \in \mathbb{N}$ tale che $T^* = T_k$.

Ci chiediamo, ora: quale è l'esito della computazione $T_k(k)$?
Se $T_k(k) = T^*(k)$ accettasse, allora $T'(k,k)$ dovrebbe accettare anch'essa. Ma se $T'(k,k)$ accetta, allora $(k,k) \notin L_H$, ossia, $T_k(k)$ non termina. Allora, $T^*(k)$ non può accettare e, dunque, necessariamente non termina.
Ma, se $T^*(k)$ non termina, allora $T'(k,k)$ rigetta e, quindi, $(k,k) \in L_H$. Dunque, per definizione, $T_k(k)$ termina.

Quindi, entrambe le ipotesi, $T_k(k)$ termina o $T_k(k)$ non termina, portano ad una contraddizione. Allora, la macchina T^* non può esistere. Poiché T^* è ottenuta mediante semplici modifiche della macchina che dovrebbe decidere L_H, ne consegue che L_H non è decidibile. □

Il resto del paragrafo è dedicato a chiarire il significato del Teorema 5.5 nonché del concetto di decidibilità. Cosa significa dimostrare che un dato linguaggio L è decidibile? Significa

dimostrare che esiste un algoritmo, ossia la specifica di un numero finito di passi elementari, che, per ogni possibile parola fornitagli come input, sia in grado di verificare in un numero finito di passi se tale parola appartiene ad L.

Alla luce della precedente osservazione possiamo quindi affermare che il Teorema 5.5 prova quanto segue:

non esiste un algoritmo che, avendo in input la descrizione di un algoritmo A_i ed un possibile input per A_i, è in grado di verificare in un numero finito di passi se A_i con input x terminerà in un numero finito di passi.

Se analizziamo con attenzione questo risultato, osserviamo che ciò che abbiamo dimostrato non esistere è un algoritmo che opera su un altro algoritmo. In particolare, nella dimostrazione del Teorema 5.5 viene utilizzato il fatto che, se un tale algoritmo esistesse, non saprebbe come comportarsi qualora come input ricevesse sé stesso. O meglio: *l'esistenza di un tale algoritmo determinerebbe, qualora dovesse operare su sé stesso, un paradosso.*

5.4 Accettabilità, decidibilità ed operazioni fra linguaggi

In questo paragrafo, analizziamo come particolari relazioni fra linguaggi permettano di dedurre la (non) accettabilità/decidibilità di un linguaggio in funzione di quella dei linguaggi ad esso correlati. Limiteremo la nostra attenzione a linguaggi contenuti in $\{0,1\}^*$; sulla base di quanto osservato nel precedente paragrafo questa assunzione non è restrittiva.

Iniziamo la nostra analisi partendo dalle basilari operazioni fra insiemi.

Definizione 5.2: Il *linguaggio complemento* L^c di un linguaggio $L \subseteq \{0,1\}^*$ è il linguaggio di tutte e sole le parole non contenute in L:

$$L^c = \{0,1\}^* - L.$$

Teorema 5.6: *Un linguaggio $L \subseteq \{0,1\}^*$ è decidibile se e soltanto se L è accettabile e L^c è accettabile.*

Dimostrazione: Se L è decidibile, allora esiste una macchina di Turing T tale che, per ogni $x \in \{0,1\}$, $T(x)$ termina e, inoltre,$o_T(x) = q_A$ se e soltanto se $x \in L$.
Poiché $o_T(x) = q_A$ se e soltanto se $x \in L$, allora L è accettabile. Poiché $T(x)$ termina per ogni $x \in \{0,1\}$, se non termina nello stato di accettazione allora deve necessariamente terminare nello stato di rigetto. Deriviamo, quindi, da T una macchina T' invertendo gli stati di accettazione e di rigetto di T. Poiché $T(x)$ termina per ogni $x \in \{0,1\}$, allora anche $T'(x)$ termina per ogni $x \in \{0,1\}$. Inoltre, per ogni $x \in \{0,1\}$, $o_{T'}(x) = q_A$ se e soltanto se $T(x)$ termina nello stato di rigetto, ossia, se e soltanto se $x \notin L$. Quindi, T' accetta L^c.

La dimostrazione dell'implicazione inversa è più complessa. Se L è accettabile e L^c è accettabile, allora esistono due macchine di Turing T_1 e T_2 tali che, per ogni $x \in \{0,1\}^*$, $o_{T_1}(x) = q_A$ se e soltanto se $x \in L$ e $o_{T_2}(x) = q_A$ se e soltanto se $x \in L^c$. Deriviamo da T_1 e T_2 una nuova macchina T, di seguito descritta.

1) T utilizza due nastri, su ciascuno dei quali, inizialmente, è scritta una copia dell'input $x \in \{0,1\}^*$: il primo nastro viene utilizzato per simulare la computazione $T_1(x)$, il secondo nastro per simulare la computazione $T_2(x)$.

2) Osserviamo che T non può simulare *l'intera* computazione di $T_1(x)$ e poi iniziare la simulazione della computazione $T_2(x)$ (o viceversa), perché $T_1(x)$ potrebbe non terminare; allora, $T(x)$ esegue, alternativamente, una istruzione di $T_1(x)$ sul nastro 1 ed una istruzione di $T_2(x)$ sul nastro 2: non appena una delle due computazioni raggiunge uno stato finale, T decide se accettare o rigettare x:

   - se la prima computazione che termina è $T_1(x)$ allora $T(x)$ termina nello stesso stato in cui termina $T_1(x)$ (accetta se $T_1(x)$ accetta, rigetta altrimenti);
   - se la prima computazione che termina è $T_2(x)$ allora $T(x)$ termina nello stato complementare a quello in cui termina $T_2(x)$ (accetta se $T_1(x)$ rigetta, rigetta se $T_1(x)$ accetta).

È semplice verificare che T decide il linguaggio L, e, quindi, L è decidibile. □

Il corollario che segue è conseguenza diretta del Teorema 5.6.

Corollario 5.2: *Un linguaggio $L \subseteq \{0,1\}^*$ è decidibile se e soltanto se L^c è decidibile.*

I prossimi due teoremi mettono in relazione la accettabilità/decidibilità di due linguaggi con la accettabilità/decidibilità del linguaggio che ne è l'intersezione e del linguaggio che ne è l'unione. La dimostrazione, che ricalca quella del Teorema 5.6, è molto semplice e viene lasciata per esercizio.

Teorema 5.7: *Se L_1 ed L_2 sono due linguaggi in $\{0,1\}^*$ accettabili, allora $L_1 \cap L_2$ è un linguaggio accettabile.*
Se L_1 ed L_2 sono due linguaggi in $\{0,1\}^*$ decidibili, allora $L_1 \cap L_2$ è un linguaggio decidibile.

Teorema 5.8: *Se L_1 ed L_2 sono due linguaggi in $\{0,1\}^*$ accettabili, allora $L_1 \cup L_2$ è un linguaggio accettabile.*
Se L_1 ed L_2 sono due linguaggi in $\{0,1\}^*$ decidibili, allora $L_1 \cup L_2$ è un linguaggio decidibile.

Osserviamo esplicitamente che i teoremi inversi dei due precedenti non sono, ovviamente, veri.

Infine, diversamente dalle operazioni fra linguaggi, la relazione di inclusione fra linguaggi non fornisce informazioni relative circa la loro calcolabilità/decidibilità. In altre parole, l'inclusione di un linguaggio L_1 in un altro L_2 non garantisce che L_1 sia accettabile/decidibile data l'accettabilità/decidibilità di L_2 né, viceversa, garantisce che L_2 sia accettabile/decidibile data l'accettabilità/decidibilità di L_1.

5.5 Riduzioni fra linguaggi

Introduciamo, ora, il concetto di riducibilità funzionale. Questo concetto permette di correlare tra loro linguaggi in modo tale che

- la decidibilità/accettabilità di un linguaggio implichi la decidibilità/accettabilità dei linguaggi ad esso riducibili; e, inoltre,
- la non decidibilità/accettabilità di un linguaggio implichi la non decidibilità/accettabilità dei linguaggi cui esso si riduce.

Definizione 5.3: Siano $L_1 \subseteq \{0,1\}^*$ e $L_2 \subseteq \{0,1\}^*$ due linguaggi; diciamo che L_1 è (many to one) *riducibile* ad L_2 se esiste una funzione totale e calcolabile $f : \{0,1\}^* \to \{0,1\}^*$ tale che

$$\forall x \in \{0,1\}^* \; [\, x \in L_1 \Leftrightarrow f(x) \in L_2 \,].$$

Se L_1 è riducibile ad L_2 scriviamo $L_1 \preceq_m L_2$.

Esempio. Consideriamo il linguaggio

$$L_{H_\square} = \{i \in \mathbb{N} : T_i(\square) \text{ termina } \},$$

dove $T_i(\square)$ è la computazione che la macchina T_i esegue sulla parola vuota (ossia, la computazione che ha inizio con il nastro riempito da $\square$). Riduciamo, ora, ad esso il linguaggio L_H, cioè mostriamo che $L_H \preceq_m L_{H_\square}$. Dobbiamo, cioè, definire una funzione f totale e calcolabile che, ad ogni coppia $(i,x) \in \mathbb{N} \times \{0,1\}^*$ associa un intero $k \in \mathbb{N}$ tale che $(i,x) \in L_H$ se e soltanto se $k \in L_{H_\square}$
Per ogni $i \in \mathbb{N}$ e per ogni $x \in \{0,1\}^*$, costruiamo una nuova macchina di Turing $T_{i,x}$ che, inglobando x nei suoi stati interni esegue una computazione sulla parola vuota che ha lo stesso esito della computazione $T_i(x)$, ossia, $o_{T_{i,x}}(\square) = o_{T_i}(x)$ se $T_i(x)$ termina e $T_{i,x}(\square)$ non termina se $T_i(x)$ non termina. Sia $k = k(i,x) \in \mathbb{N}$ il numero naturale associato alla macchina $T_{i,x}$ mediante il procedimento descritto nella dimostrazione del Teorema 5.2. Allora la funzione f cercata è proprio $f(i,x) = k(i,x)$.
Per costruzione, f è una funzione totale; inoltre, essa è calcolabile perché il procedimento descritto nella dimostrazione del Teorema 5.2 è costruttivo.
Sia, ora, $(i,x) \in \mathbb{N} \times \{0,1\}^*$: per costruzione di $T_k = T_{i,x}$, $T_i(x)$ termina se e soltanto se $T_k(\square)$ termina, ossia, $(i,x) \in L_H$ se e soltanto se $i \in L_{H_\square}$.
Questo conclude la prova che $L_H \preceq_m L_{H_\square}$. □

La relazione $\preceq_m$ gode delle proprietà riflessiva e transitiva, ossia:

$$\forall L \in \{0,1\}^* : \qquad L \preceq_m L \qquad \text{(proprietà riflessiva)}$$

$$\forall L_1, L_2, L_3 \in \{0,1\}^* : \quad L_1 \preceq_m L_2 \wedge L_2 \preceq_m L_3 \Rightarrow L_1 \preceq_m L_3 \qquad \text{(proprietà transitiva).}$$

In particolare, la validità della proprietà transitiva segue osservando che la composizione di riduzioni è ancora una riduzione. La dimostrazione formale di questo fatto viene lasciata come esercizio.

Vediamo ora come utilizzare il concetto di riduzione funzionale per dimostrare la decidibilità o meno di un linguaggio a partire dalla conoscenza della decidibilità o meno di un altro linguaggio.

- *Sia L_1 un linguaggio non decidibile e sia L_2 un secondo linguaggio tale che $L_1 \preceq_m L_2$; allora L_2 non è decidibile.*
  Indichiamo con $f_{1,2}$ la funzione che riduce L_1 ad L_2. Se L_2 fosse decidibile, allora potremmo decidere se $x \in L_1$ nella maniera seguente: innanzi tutto, calcoleremmo $f_{1,2}(x)$ e poi decideremmo se $f_{1,2}(x) \in L_2$. Poiché $x \in L_1$ se e soltanto se $f_{1,2}(x) \in L_2$, l'esito della decisione circa $f_{1,2}(x) \in L_2$ risponderebbe anche al quesito "$x \in L_1$?".
- *Sia L_1 un linguaggio non accettabile e sia L_2 un secondo linguaggio tale che $L_1 \preceq_m L_2$; allora L_2 non è accettabile.*
  Le argomentazioni per provare questo caso sono analoghe a quelle utilizzate nel punto precedente.
- *Sia L_3 un linguaggio decidibile e sia L_4 un secondo linguaggio tale che $L_4 \preceq_m L_3$; allora L_4 è decidibile.*
  Indichiamo con $f_{4,3}$ la funzione che riduce L_4 ad L_3. Sia $x \in \{0,1\}$; decidiamo se $x \in L_4$ nella maniera seguente: innanzi tutto, calcoliamo $f_{4,3}(x)$ e poi decidiamo se $f_{4,3}(x) \in L_3$. Poiché $x \in L_4$ se e soltanto se $f_{4,3}(x) \in L_3$, l'esito della decisione circa $f_{4,3}(x) \in L_3$ risponde anche al quesito "$x \in L_4$?".

- *Sia L_3 un linguaggio accettabile e sia L_4 un secondo linguaggio tale che $L_4 \preceq_m L_3$; allora L_4 è accettabile.* Le argomentazioni per provare questo caso sono analoghe a quelle utilizzate nel punto precedente.

Quanto sopra è riassunto nell'affermazione seguente:

la riducibilità funzionale preserva sia l'accettabilità che la decidibilità.

Sia, ora, L un linguaggio accettabile ma non decidibile: allora, sappiamo che esiste una macchina di Turing che accette tutte e sole le parole di L ma che ha un comportamento indefinito quando il suo input non è una parola di L, ossia, è una parola di L^c. Dunque, intuitivamente, L e L^c hanno un diverso grado di insolubilità. E, in effetti, in virtù di quanto appena affermato, non può accadere che $L^c \preceq_m L$ (altrimenti anche L^c sarebbe accettabile).

Appendice. La gödelizzazione

Con *Gödelizzazione* si intende un procedimento matematico che codifica le parole di un linguaggio composto da elementi di un alfabeto finito in numeri naturali. Più in particolare, il procedimento di gödelizzazione stabilisce una corrispondenza biunivoca tra le parole del linguaggio e un sottoinsieme dell'insieme dei numeri naturali: ad ogni elemento del linguaggio viene associato uno ed un solo numero naturale e viceversa.

Il procedimento di gödelizzazione assegna ad ogni elemento dell'alfabeto del linguaggio un numero *dispari*: esso sarà il *numero godeliano* del simbolo. Il numero naturale corrispondente alla parola $a_1a_2\ldots a_n$ del linguaggio (concatenazione dei caratteri $a_1, a_2, \ldots, a_n$ dell'alfabeto) si calcola come di seguito descritto:

1) siano $g_1, g_2, \ldots, g_n$ i numeri gödeliani assegnati, rispettivamente, ad $a_1, a_2, \ldots, a_n$;
2) si considerano i primi n numeri primi (escluso il numero 1) $p_1 = 2, p_2 = 3, \ldots, p_n$;
3) allora, la codifica di $a_1a_2\ldots a_n$ è il prodotto $p_1^{g_1} \cdot p_2^{g_2} \cdot \ldots \cdot p_n^{g_n}$.

Illustriamo questa tecnica con un esempio. Come alfabeto scegliamo l'alfabeto della lingua italiana e come linguaggio l'insieme delle parole della lingua italiana. Allora:

- iniziamo associando un numero dispari ad ogni lettera dell'alfabeto italiano:

  $$a \to 1,\ b \to 3,\ c \to 5,\ d \to 7,\ e \to 9,\ f \to 11,\ g \to 13,\ h \to 15,\ i \to 17,\ l \to 19,\ \ldots$$

- Proviamo, ora a codificare una parola qualsiasi, ad esempio la parola *cade*; poiché *cade* contiene quattro lettere, allora dobbiamo considerare i primi quattro numeri primi 2, 3, 5, 7. Quindi, indicando con $g(\cdot)$ il numero gödeliano di ciascun carattere, la codifica di *cade* è:

  $$2^{g(c)} \cdot 3^{g(a)} \cdot 5^{g(d)} \cdot 7^{g(e)} \ = \ 2^5 \cdot 3^1 \cdot 5^7 \cdot 7^9 \ = \ 32 \cdot 3 \cdot 78125 \cdot 40353607 \ = \ 302652052500000.$$

L'unicità della fattorizzazione in numeri primi e l'unicità della radice n-esima per ogni n dispari garantisce l'univocità di questo tipo di codifica.