Pilu Crescenzi

Macchine di Turing

Firenze, A.A. 2018/2019

Sommario

Prefazione

QUESTE DISPENSE sono dedicate allo studio delle Macchinedi Turing. Queste dispense costituiscono una parte delle dispense sulla teoria della calcolabilità, della teoria dei linguaggi formali e della teoria della complessità scritte per un corso di Informatica Teorica svolto presso l'Università di Firenze. La versione estesa delle dispense è disponibile sul sito dell'autore presso l'Università di Firenze.

Le dispense non presuppongono particolari conoscenze da parte dello studente, a parte quelle relative a nozioni matematiche e logiche di base. A differenza di molti altri volumi, che relegano i necessari requisiti matematici nel capitolo iniziale oppure in un'appendice, in queste dispense si è scelto di includere i requisiti matematici, come quello relativo agli insiemi incluso in questa prefazione, al momento in cui per la prima volta essi si rendano necessari.

Esistono numerosi libri di testo dedicati agli argomenti trattati in queste dispense: in particolare, i seguenti due volumi hanno ispirato in parte la stesura delle dispense stesse e sono fortemente consigliati come letture aggiuntive.

- G. Ausiello, F. D'Amore, G. Gambosi. *Linguaggi, modelli, complessità*. Edizioni Franco Angeli, 2003.
- M. Sipser. *Introduction to the Theory of Computation*. Thomson CT, 2005.
- B. Barak, Introduction to Theoretical Computer Science, libro in preparazione disponibile su HTTPS://INTROTCS.ORG

A chi poi volesse approfondire le nozioni relative alla teoria della complessità, suggeriamo la lettura del libro *Introduction to the Theory of Complexity*, scritto da D. P. Bovet e P. Crescenzi, che è disponibile gratuitamente a partire dal sito web del secondo autore.

Capitolo 1

Macchine di Turing

SOMMARIO

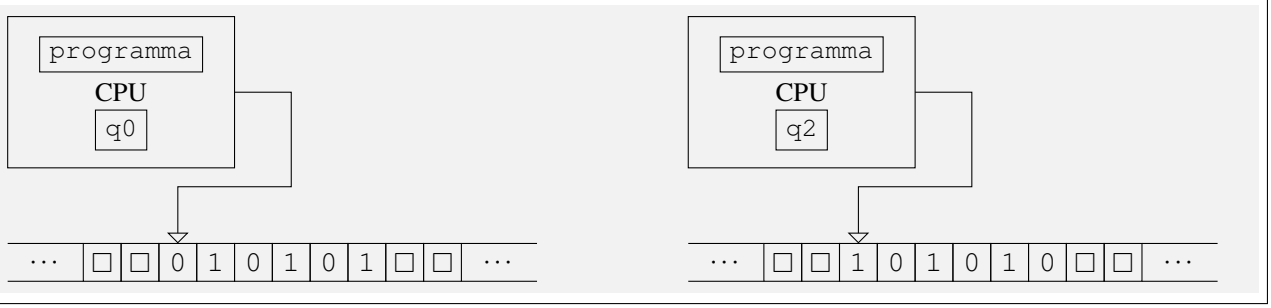
In questo capitolo introdurremo il modello di calcolo proposto dal logico matematico inglese Alan Turing. Una volta definita la macchina di Turing, mostreremo diversi esempi di tali macchine introducendo due tecniche di programmazione comunemente utilizzate per sviluppare macchine di Turing. Definiremo, quindi, le macchine di Turing multi-nastro dimostrando che non sono computazionalmente più potenti di quelle con un singolo nastro e concluderemo il capitolo introducendo il concetto di configurazione di una macchina di Turing e quello di sotto-macchina.

1.1 Definizione di macchina di Turing

UNA MACCHINA di Turing è un modello di calcolo abbastanza simile agli odierni calcolatori. Analogamente a questi ultimi, infatti, essa possiede un'unità di elaborazione centrale (CPU, dall'inglese *Central Processing Unit*) e una memoria su cui poter leggere e scrivere. In particolare, la CPU di una macchina di Turing è composta da un **registro di stato**, contenente lo stato attuale della macchina, e da un **programma** contenente le istruzioni che essa deve eseguire. La memoria di una macchina di Turing è composta da un **nastro** infinito, suddiviso in **celle** e al quale la CPU può accedere attraverso una **testina** di lettura/scrittura (si veda la Figura 1.1).

Inizialmente, il nastro contiene la stringa di **input** preceduta e seguita da una serie infinita di simboli vuoti (in queste dispense, il **simbolo vuoto** è indicato con □), la testina è posizionata sul primo simbolo della stringa di input e la CPU si trova in uno stato speciale, detto **stato iniziale** (si veda la parte sinistra della Figura 1.1). Sulla base dello stato in cui si trova la CPU e del simbolo letto dalla testina, la macchina esegue un'istruzione del programma che può modificare il simbolo attualmente scandito dalla testina, spostare la testina a destra oppure a sinistra e cambiare lo stato della

Figura 1.1: rappresentazione schematica di una macchina di Turing.



CPU. La macchina prosegue nell'esecuzione del programma fino a quando la CPU non viene a trovarsi in uno di un insieme di stati particolari, detti **stati finali**, oppure non esistono istruzioni del programma che sia possibile eseguire. Nel caso in cui il programma termini perché la CPU ha raggiunto uno stato finale, il contenuto della porzione di nastro racchiusa tra la posizione della testina ed il primo simbolo □ alla sua destra rappresenta la stringa di **output** (si veda la parte destra della Figura 1.1).

Esempio 1.1: una macchina di Turing per il complemento bit a bit

Consideriamo una macchina di Turing la cui CPU può assumere tre possibili stati `q0`, `q1` e `q2`, di cui il primo è lo stato iniziale e l'ultimo è l'unico stato finale. L'obiettivo della macchina è quello di calcolare la stringa binaria ottenuta eseguendo il complemento bit a bit della stringa binaria ricevuta in input: il programma di tale macchina è il seguente.

1. Se lo stato è `q0` e il simbolo letto non è □, allora complementa il simbolo letto e sposta la testina a destra.
2. Se lo stato è `q0` e il simbolo letto è □, allora passa allo stato `q1` e sposta la testina a sinistra.
3. Se lo stato è `q1` e il simbolo letto non è □, allora sposta la testina a sinistra.
4. Se lo stato è `q1` e il simbolo letto è □, allora passa allo stato `q2` e sposta la testina a destra.

La prima istruzione fa sì che la macchina scorra l'intera stringa di input, complementando ciascun bit incontrato, mentre le successive tre istruzioni permettono di riportare la testina all'inizio della stringa modificata. Ad esempio, tale macchina, con input la stringa binaria `010101`, inizia la computazione nella configurazione mostrata nella parte sinistra della Figura 1.1 e termina la sua esecuzione nella configurazione mostrata nella parte destra della figura.

Alfabeti, stringhe e linguaggi

Un **alfabeto** è un qualunque insieme finito non vuoto $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$: un **simbolo** è un elemento di un alfabeto. Una **stringa** su un alfabeto Σ è una sequenza finita $x = \sigma_{i_1}, \ldots, \sigma_{i_n}$ di simboli in Σ (per semplicità, la stringa $\sigma_{i_1}, \ldots, \sigma_{i_n}$ sarà indicata con $\sigma_{i_1} \cdots \sigma_{i_n}$): la **stringa vuota** è indicata con λ. L'insieme infinito di tutte le possibili stringhe su un alfabeto Σ è indicato con Σ^*. La **lunghezza** $|x|$ di una stringa $\sigma_{i_1} \cdots \sigma_{i_n}$ è il numero n di simboli contenuti in x: la stringa vuota ha lunghezza 0. Chiaramente, il numero di stringhe di lunghezza n su un alfabeto di k simboli è uguale a k^n. Date due stringhe x e y, la **concatenazione** di x e y (in simboli, xy) è definita come la stringa z formata da tutti i simboli di x seguiti da tutti i simboli di y: quindi, $|z| = |x| + |y|$. In particolare, la concatenazione di una stringa x con se stessa h volte è indicata con x^h. Date due stringhe x e y, diremo che x è un **prefisso** di y se esiste una stringa z per cui $y = xz$. Dato un alfabeto Σ, un **linguaggio** L su Σ è un sottoinsieme di Σ^*: il complemento di L, in simboli L^c, è definito come $L^c = \Sigma^* - L$. Dato un alfabeto Σ, un ordinamento dei simboli di Σ induce un ordinamento delle stringhe su Σ nel modo seguente: (a) per ogni $n \geqslant 0$, le stringhe di lunghezza n precedono quelle di lunghezza $n+1$, e (b) per ogni lunghezza, l'ordine è quello alfabetico. Tale ordinamento è detto **ordinamento lessicografico**. Ad esempio, dato l'alfabeto binario $\Sigma = \{0, 1\}$, le prime dieci stringhe nell'ordinamento lessicografico di Σ^* sono λ, 0, 1, 00, 01, 10, 11, 000, 001 e 010.

Formalmente, una macchina di Turing è definita specificando l'insieme degli stati (indicando quale tra di essi è quello iniziale e quali sono quelli finali) e l'insieme delle transizioni da uno stato a un altro: questo ci conduce alla seguente definizione.

Definizione 1.1: macchina di Turing

Una *macchina di Turing* è costituita da un **alfabeto di lavoro** Σ contenente il simbolo $\square$ e da un **grafo delle transizioni**, ovvero un grafo etichettato $G = (V, E)$ tale che:

- $V = \{q_0\} \cup F \cup Q$ è l'insieme degli **stati** (q_0 è lo stato **iniziale**, F è l'insieme degli stati **finali** e Q è l'insieme degli stati che non sono iniziali né finali);
- E è l'insieme delle **transizioni** e, a ogni transizione, è associata un'etichetta formata da una lista l di triple (σ, τ, m), in cui σ e τ sono simboli appartenenti a Σ e $m \in \{R, L, S\}$, tale che non esistono due triple in l con lo stesso primo elemento.

I simboli σ e τ che appaiono all'interno di una tripla dell'etichetta di una transizione indicano, rispettivamente, il simbolo attualmente letto dalla testina e il simbolo da scrivere, mentre il valore m specifica il movimento della testina: in particolare, R corrisponde allo spostamento a destra, L allo spostamento a sinistra e S a nessun spostamento. Nel seguito di queste dispense, per evitare di dover specificare ogni volta quale sia lo stato iniziale e quali siano gli stati finali di una macchina di Turing e in

Grafi

Un **grafo** G è una coppia di insiemi finiti (V, E) tale che $E \subseteq V \times V$: V è l'insieme dei **nodi**, mentre E è l'insieme degli **archi**. Se $(u, v) \in E$, allora diremo che u è **collegato** a v: il numero di nodi a cui un nodo x è collegato è detto **grado in uscita** di x, mentre il numero di nodi collegati a x è detto **grado in ingresso** di x. Un grafo $G' = (V', E')$ è un **sotto-grafo** di un grafo $G = (V, E)$ se $V' \subseteq V$ e $E' \subseteq E$. Un grafo **etichettato** è un grafo $G = (V, E)$ con associata una funzione $l : E \to L$ che fa corrispondere a ogni arco un elemento dell'insieme L delle etichette: tale insieme può essere, ad esempio, un insieme di valori numerici oppure un linguaggio. Un grafo $G = (V, E)$ è detto **completo** se $E = V \times V$, ovvero se ogni nodo è collegato a ogni altro nodo. Dato un grafo G e due nodi v_0 e v_k, un **cammino** da v_0 a v_k di lunghezza k è una sequenza di archi (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) tale che, per ogni i e j con $0 \leqslant i < j \leqslant k$, $v_i \neq v_j$ se $i \neq 0$ o $j \neq k$: se $v_0 = v_k$ allora la sequenza di archi è detta essere un **ciclo**. Un **albero** è un grafo senza cicli.

accordo con l'applicativo JFLAP, adotteremo la convenzione di specificare lo stato iniziale affiancando a esso una freccia rivolta verso destra e di rappresentare gli stati finali con un doppio cerchio.

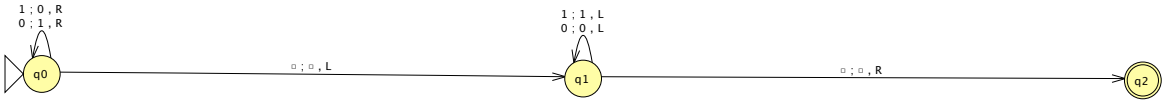
Esempio 1.2: il grafo delle transizioni della macchina per il complemento bit a bit

Il grafo corrispondente alla macchina di Turing introdotta nell'Esempio 1.1 è mostrato nella Figura 1.2. Come si può vedere, ciascuna transizione corrisponde a un'istruzione del programma descritto nell'Esempio 1.1. Ad esempio, l'arco che congiunge lo stato `q0` allo stato `q1` corrisponde alla seconda istruzione: in effetti, nel caso in cui lo stato attuale sia `q0` e il simbolo letto sia □, allora il simbolo non deve essere modificato, la testina si deve spostare a sinistra e la CPU deve passare nello stato `q1`.

1.1.1 Rappresentazione tabellare di una macchina di Turing

Oltre alla rappresentazione basata sul grafo delle transizioni a cui abbiamo fatto riferimento nella Definizione 1.1, una macchina di Turing viene spesso specificata facendo uso di una rappresentazione tabellare. In base a tale rappresentazione, a una macchina di Turing viene associata una tabella di tante righe quante sono le triple contenute nelle etichette degli archi del grafo delle transizioni corrispondente alla macchina. Se (σ, τ, m) è una tripla contenuta nell'etichetta dell'arco che collega lo stato q allo stato p, la corrispondente riga della tabella sarà formata da cinque colonne, contenenti, rispettivamente, q, σ, p, τ e m. In altre parole, ogni riga della tabella corrisponde a un'istruzione del programma della macchina di Turing: la prima e la seconda colonna della riga specificano, rispettivamente, lo stato in cui si deve trovare la CPU e il simbolo che deve essere letto dalla testina affinché l'istruzione sia eseguita, mentre la terza, la quarta e la quinta colonna specificano, rispettivamente,

Figura 1.2: macchina di Turing per il complemento bit a bit.



il nuovo stato in cui si troverà la CPU, il simbolo che sostituirà quello appena letto e il movimento della testina che sarà effettuato eseguendo l'istruzione. Osserviamo che, in generale, la tabella non contiene tante righe quante sono le possibili coppie formate da uno stato e un simbolo dell'alfabeto di lavoro della macchina, in quanto per alcune (generalmente molte) di queste coppie il comportamento della macchina può non essere stato specificato.

Esempio 1.3: rappresentazione tabellare della macchina per il complemento bit a bit

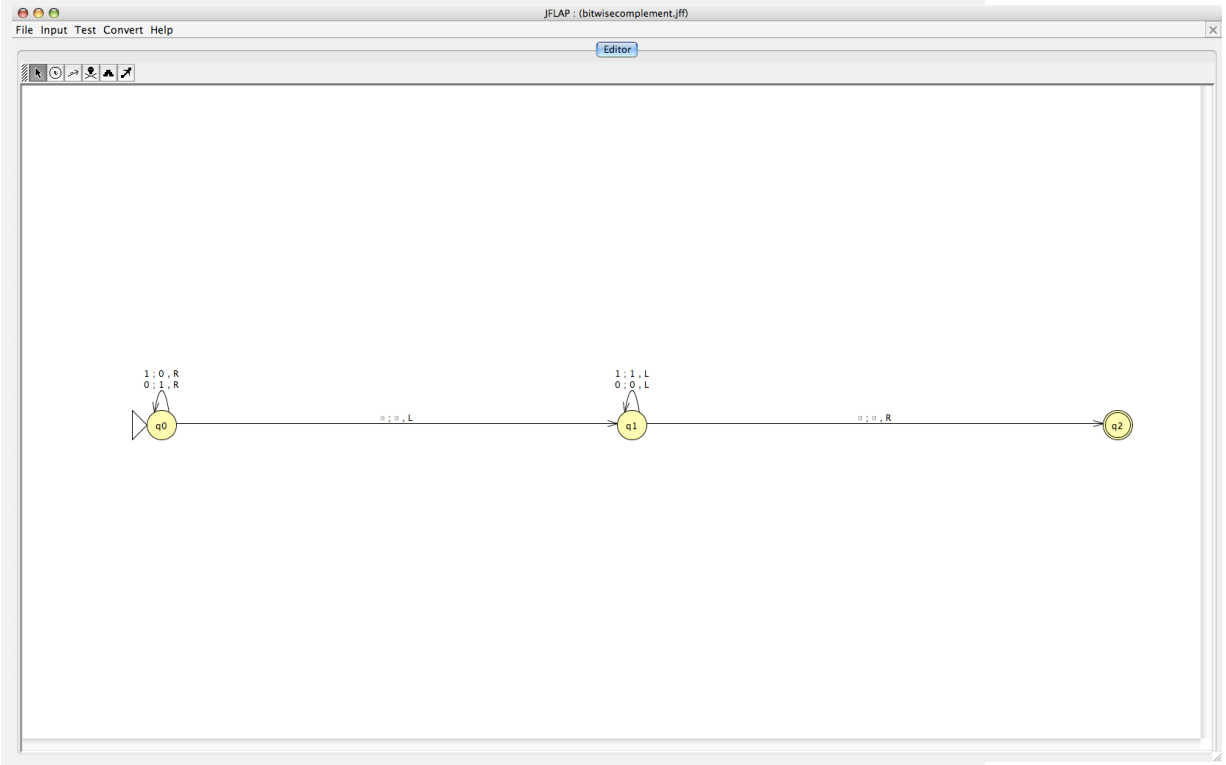
Facendo riferimento alla macchina di Turing introdotta nell'Esempio 1.1 e mostrata nella Figura 1.2, la corrispondente rappresentazione tabellare di tale macchina è la seguente.

stato	simbolo	stato	simbolo	movimento
q0	0	q0	1	R
q0	1	q0	0	R
q0	☐	q1	☐	L
q1	0	q1	0	L
q1	1	q1	1	L
q1	☐	q2	☐	R

1.1.2 Macchine di Turing e JFLAP

Nel seguito faremo spesso riferimento all'applicativo JFLAP, che consente di creare macchine di Turing facendo uso di una semplice interfaccia grafica (la macchina di Turing mostrata nella Figura 1.2 è stata prodotta facendo uso di tale applicativo), e inviteremo il lettore a sperimentare una determinata macchina di Turing attraverso un riquadro come quello seguente, nella cui parte centrale viene specificato il nome del file XML contenente la descrizione della macchina secondo le regole di JFLAP: tale file può essere ottenuto a partire dal sito web del corso *Informatica teorica: linguaggi, computabiltià, complessità.*

macchina per il complemento bit a bit



1.2 Esempi di macchine di Turing

In questo paragrafo mostriamo alcuni esempi di macchine di Turing. Vedremo molti altri esempi nel resto delle dispense, ma già alcuni di quelli che verranno presentati in questo paragrafo evidenziano il tipico comportamento di una macchina di Turing: in generale, infatti, queste macchine implementano una strategia a *zig-zag* che consente loro di attraversare ripetutamente il contenuto del nastro, combinata con una strategia di *piggy-backing* che consiste nel "memorizzare" nello stato il simbolo o i simboli che sono stati letti.

Sebbene ciò sia l'argomento principale della terza parte di queste dispense, nel seguito valuteremo sempre il numero di passi che ciascuna macchina esegue in funzione della lunghezza della stringa ricevuta in input: a tale scopo faremo uso della notazione O, evitando quindi di dover sempre specificare esattamente il numero di

Rappresentazione binaria di numeri interi

I principi alla base del sistema di **numerazione binaria** sono gli stessi di quelli usati dal sistema decimale: infatti, entrambi sono sistemi di numerazione posizionale, nei quali la posizione di una cifra ne indica il valore relativo. In particolare, il sistema binario usa potenze crescenti di 2: per esempio, il numero binario 1001 rappresenta il numero decimale 9, in quanto $1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 1 = 9$. La conversione di un numero decimale n in uno binario è invece leggermente più complicata e può essere realizzata nel modo seguente: si divide ripetutamente n per 2, memorizzando in ordine inverso i resti che vengono ottenuti, fino a ottenere un quoziente minore di 2 che rappresenta la cifra binaria più a sinistra. Ad esempio, per ottenere la rappresentazione binaria del numero decimale 9 vengono effettuate le seguenti divisioni: (a) $9 \div 2 = 4$ con resto $\underline{1}$, (b) $4 \div 2 = 2$ con resto $\underline{0}$, (c) $2 \div 2 = \underline{1}$ con resto $\underline{0}$. Pertanto, al numero decimale 9 corrisponde il numero binario 1001. Osserviamo che, come nel sistema decimale con k cifre possiamo rappresentare al massimo 10^k numeri (da 0 a $10^k - 1$), anche nel sistema binario avendo a disposizione k cifre (binarie) possiamo rappresentare al massimo 2^k numeri (da 0 a $2^k - 1$). La maggior parte dei calcolatori, in realtà, rappresenta i numeri interi facendo uso della rappresentazione **complemento a due**. Dati k bit, un numero intero non negativo viene rappresentato normalmente facendo uso di $k-1$ bit preceduti da uno 0, mentre un numero intero negativo x viene rappresentato mediante la rappresentazione binaria di $2^k - x$ che faccia uso di k bit. Ad esempio, avendo a disposizione 4 bit, la rappresentazione complemento a due di 6 è 0110, mentre quella di -6 è 1010 (ovvero la rappresentazione binaria di $2^4 - 6 = 10$).

passi eseguiti e accontentandoci di valutarne l'ordine di grandezza. In particolare, data una macchina di Turing T, indicheremo con $t_T(n)$ la funzione che rappresenta il massimo numero di passi eseguito da T con input una stringa di lunghezza n.

1.2.1 Una macchina per il complemento a due

Definiamo una macchina di Turing T che, data la rappresentazione complemento a due di un numero intero non negativo x che faccia uso di k bit, produca la rappresentazione complemento a due di $-x$ che faccia anch'essa uso di k bit. Tale rappresentazione può essere calcolata individuando il simbolo `1` più a destra della rappresentazione di x e complementando tutti i simboli alla sua sinistra: quindi, T opera nel modo seguente.

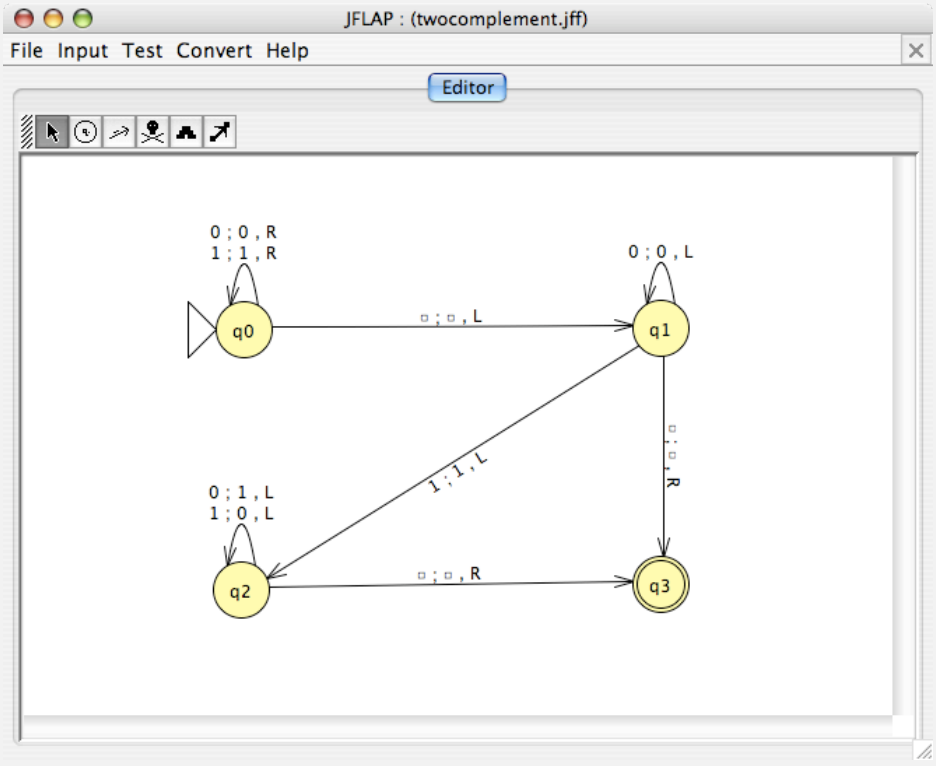
1. Attraversa la stringa in input fino a raggiungere il primo $\square$ alla sua destra.

2. Si muove verso sinistra fino a trovare e superare un `1`: se tale `1` non esiste (ovvero, la stringa in input è costituita da soli simboli `0` oppure è la stringa vuota), la macchina può terminare la sua esecuzione posizionando la testina sul simbolo `0` più a sinistra (se tale simbolo esiste).

3. Si muove verso sinistra, complementando ogni simbolo incontrato, fino a raggiungere il primo □ e posiziona la testina sul simbolo immediatamente a destra.

Chiaramente, $t_T(n) \in O(n)$: infatti, per ogni stringa x di lunghezza n, n passi sono eseguiti per raggiungere il primo □ alla destra di x e altri n passi sono eseguiti per effettuare la ricerca del simbolo `1` e la complementazione dei simboli alla sua sinistra.

macchina per il calcolo del complemento a due



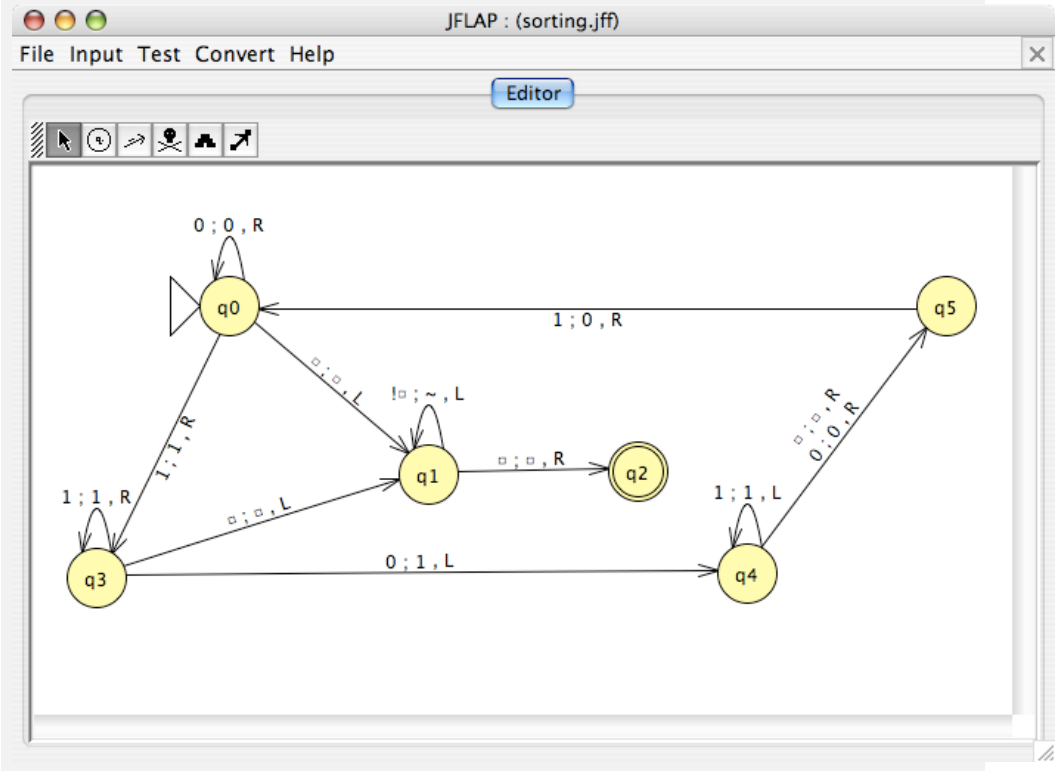
1.2.2 Una macchina per l'ordinamento di stringhe binarie

Data una stringa x sull'alfabeto $\Sigma = \{0, 1\}$, la stringa ordinata corrispondente a x è la stringa su Σ ottenuta a partire da x mettendo tutti i simboli `0` prima dei simboli `1`: ad esempio, la stringa ordinata corrispondente a `0101011` è la stringa `0001111`. La figura seguente illustra la macchina di Turing T che, data in input una stringa binaria x, termina producendo in output la sequenza ordinata corrispondente a x.

Ordini di grandezza

Data una funzione $f : N \to R$, $O(f(n))$ denota l'insieme delle funzioni $g : N \to R$ per le quali esistono due costanti $c > 0$ e $n_0 > 0$ per cui vale $g(n) \leqslant cf(n)$, per ogni $n > n_0$. Inoltre, $\Omega(f(n))$ denota l'insieme delle funzioni $g : N \to R$ per le quali esistono due costanti $c > 0$ e $n_0 > 0$ e infiniti valori $n > n_0$ per cui vale $g(n) \geqslant cf(n)$. Infine, $o(f(n))$ denota l'insieme delle funzioni $g : N \to R$ tali che $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.

macchina per l'ordinamento di stringhe binarie



La macchina di Turing T opera nel modo seguente.

1. Scorre il nastro verso destra alla ricerca di un simbolo `1`. Se non lo trova, vuol dire che la testina è posizionata sul primo simbolo □ successivo alla stringa che è ora ordinata: in tal caso, scorre il nastro verso sinistra fino a posizionare la testina sul primo simbolo diverso da □ e termina l'esecuzione.

2. Scorre il nastro verso destra alla ricerca di un simbolo `0`. Se non lo trova, vuol dire che la testina è posizionata sul primo simbolo $\square$ successivo alla stringa che è ora ordinata: in tal caso, scorre il nastro verso sinistra fino a posizionare la testina sul primo simbolo diverso da $\square$ e termina l'esecuzione.

3. Complementa il simbolo `0` e scorre il nastro verso sinistra alla ricerca di un simbolo `0` oppure di un simbolo $\square$, posizionando la testina immediatamente dopo tale simbolo, ovvero, sul simbolo `1` trovato in precedenza: complementa tale simbolo e ricomincia dal primo passo.

Per ogni stringa x di lunghezza n, la ricerca di una coppia di simboli `1` e `0` posizionati in ordine sbagliato richiede al più n passi: il successivo scambio di questi due simboli richiede anch'esso un numero lineare di passi. Poiché vi possono essere al più $\frac{n}{2}$ coppie di simboli invertiti, abbiamo che $t_T(n) \in O(n^2)$.

1.2.3 Una macchina per un linguaggio non regolare

Consideriamo il seguente linguaggio L sull'alfabeto $\Sigma = \{0, 1\}$.

$$L = \{0^n 1^n : n \geqslant 0\}$$

Ad esempio, la stringa `00001111` appartiene a L, mentre la stringa `0001111` non vi appartiene. Tale linguaggio, come vedremo nella seconda parte di queste dispense, svolge un ruolo molto importante nell'ambito della teoria dei linguaggi formali, in quanto rappresenta il classico esempio di linguaggio generabile da una grammatica contestuale, ma non generabile da una grammatica regolare. Definiamo una macchina di Turing che, data in input una sequenza di simboli `0` seguita da una sequenza di simboli `1`, termina in uno stato finale se il numero di simboli `0` è uguale a quello dei simboli `1`, altrimenti termina in uno stato non finale non avendo istruzioni da poter eseguire. In questo caso, la posizione finale della testina è ininfluente, in quanto l'output prodotto dalla macchina viene codificato attraverso il fatto che la computazione sia terminata o meno in uno stato finale. La macchina opera nel modo seguente.

1. Partendo dall'estremità sinistra della stringa in input, sostituisce un simbolo `0` con il simbolo $\square$ e si muove verso destra, ignorando tutti i simboli `0` e `1`, fino a incontrare il simbolo $\square$.

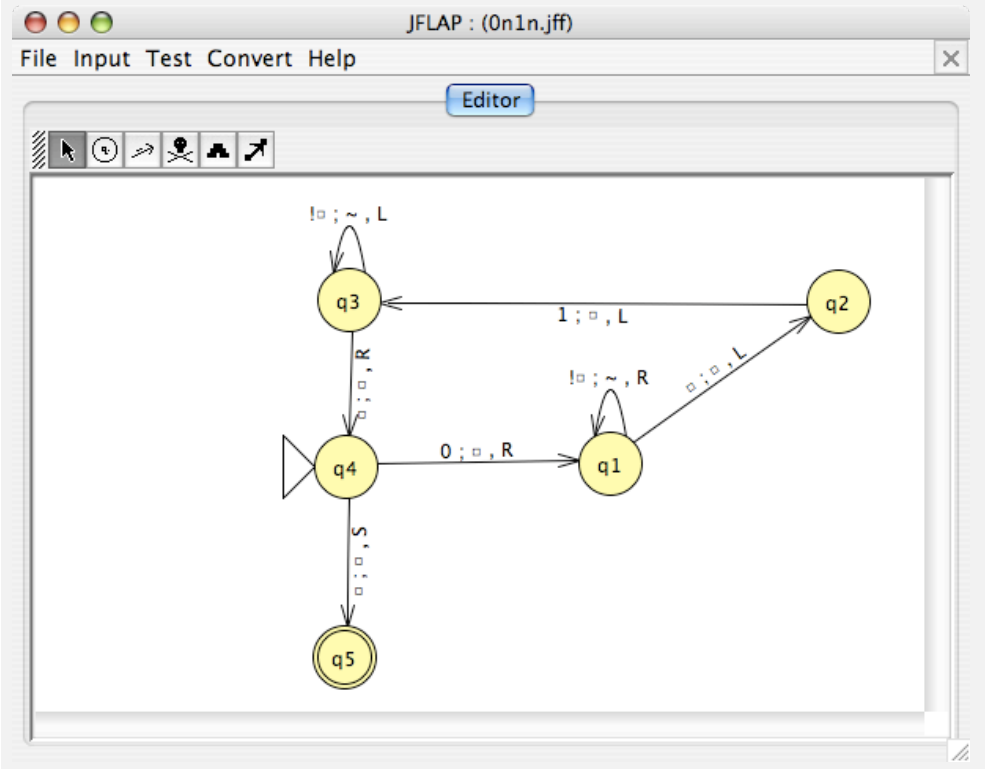
2. Verifica che il simbolo immediatamente a sinistra del simbolo $\square$ sia un `1`: se così non è, allora termina la computazione in uno stato non finale.

3. Sostituisce il simbolo `1` con il simbolo $\square$ e scorre il nastro verso sinistra fino a incontrare un simbolo $\square$.

4. Se il simbolo immediatamente a destra del □ è anch'esso un simbolo □, allora termina in uno stato finale. Se, invece, è un `1`, allora termina in uno stato non finale. Altrimenti, ripete l'intero procedimento a partire dal primo passo.

Anche in questo caso, per ogni stringa x di lunghezza n, la ricerca di una coppia di simboli `0` e `1` posizionati, rispettivamente, all'estremità sinistra e destra della stringa binaria rimanente richiede un numero lineare di passi. Poiché vi possono essere al più $\frac{n}{2}$ di tali coppie, abbiamo che $t_T(n) \in O(n^2)$. Osserviamo che, in base alla descrizione precedente, la stringa in ingresso viene modificata dalla macchina in modo tale da non poter essere successivamente ricostruita: non è difficile, comunque, modificare la macchina in modo che, facendo uso di simboli ausiliari, sia possibile ricostruire in un numero di passi lineare, al termine del procedimento sopra descritto, la stringa ricevuta in ingresso.

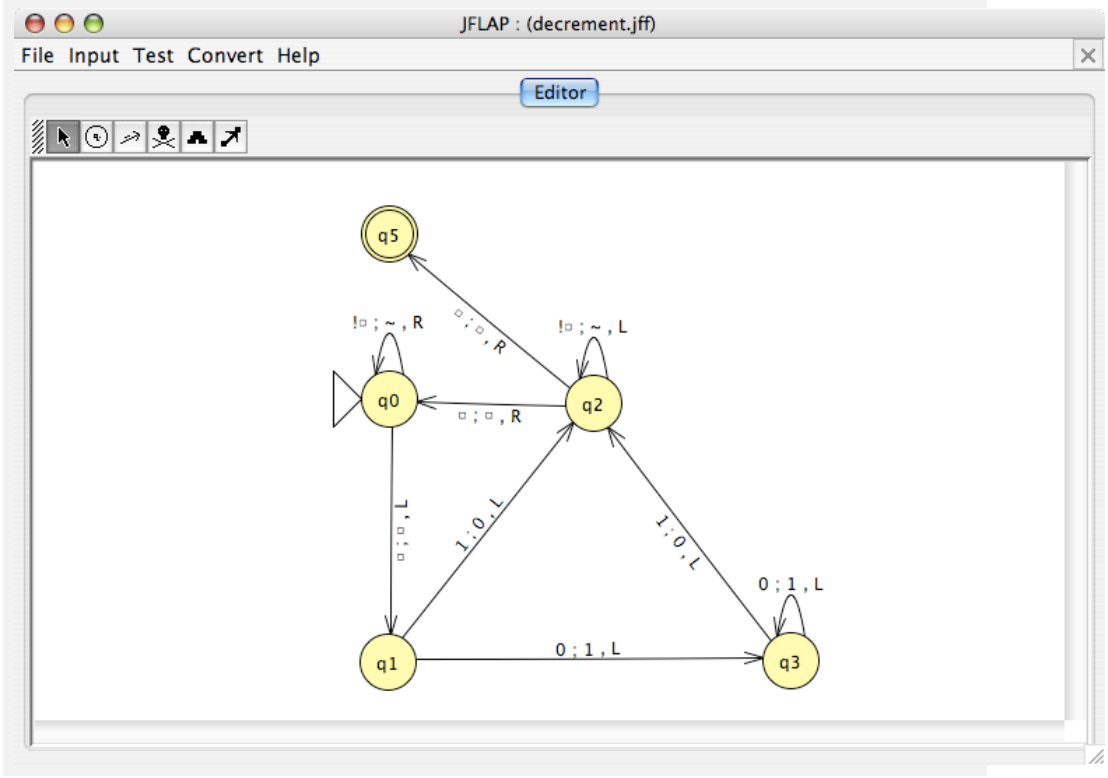
macchina per 0^n1^n



1.2.4 Una macchina per il decremento di un numero intero binario

Nell'esempio precedente abbiamo visto come sia possibile aumentare di un'unità un numero intero rappresentato in forma binaria: in questo paragrafo definiamo, invece, una macchina di Turing T che decrementa di un'unità un numero intero positivo rappresentato in forma binaria.

macchina per il decremento di un numero intero binario



Tale macchina opera nel modo seguente.

1. Si sposta verso destra fino a trovare l'ultimo simbolo della rappresentazione binaria del numero intero: se tale simbolo è un `1`, lo complementa e posiziona la testina sul primo simbolo della stringa binaria.

2. Se l'ultimo simbolo è uno `0`, lo complementa e si sposta verso sinistra fino a trovare un simbolo `1` e complementando ciascun simbolo `0` incontrato: complementa il simbolo `1` e posiziona la testina sul primo simbolo della stringa binaria.

Per ogni stringa binaria x di lunghezza n, la ricerca del suo ultimo simbolo richiede un numero lineare di passi e un analogo numero di passi è richiesto per la ricerca della prima posizione a cui "chiedere in prestito" un simbolo `1` (se tale posizione esiste): pertanto, abbiamo che $t_T(n) \in O(n)$.

Osserviamo che la macchina appena descritta termina in modo anomalo (ovvero, a causa della non esistenza di operazioni da eseguire) nel caso in cui la stringa binaria sia vuota oppure nel caso in cui sia la rappresentazione del numero 0. In generale, in questo e in tutti gli esempi precedenti, abbiamo sempre assunto che l'input sia fornito in modo corretto. Ad esempio, la descrizione della macchina di Turing per la conversione da unario in binario, assume che l'input sia sempre costituito da una sequenza non vuota di simboli `U`. Se così non fosse, potrebbero verificarsi diverse situazioni di errore. Ad esempio, se la stringa in ingresso fosse `UUAUU`, si avrebbe che la macchina, dopo aver contato fino a 2, nello stato iniziale leggerebbe il simbolo `A`: in tal caso, la macchina si arresterebbe in modo anomalo, in quanto non sarebbe definita una transizione in corrispondenza di tale situazione. Se, invece, la stringa in ingresso fosse `XXUUXUX`, allora la macchina, invece di segnalare in qualche modo il fatto che l'input non è una corretta rappresentazione unaria di un un numero intero positivo, si arresterebbe nello stato finale producendo come output la stringa `11` (ovvero, la rappresentazione binaria del numero di simboli `U` presenti nella stringa in ingresso). In tutti gli esempi mostrati sinora, comunque, è sempre facile verificare l'esistenza di una macchina di Turing in grado di controllare che l'input sia stato fornito in modo corretto. Ad esempio, una macchina di Turing che verifichi che l'input sia costituito da una sequenza non vuota di simboli `U`, puó essere definita nel modo seguente.

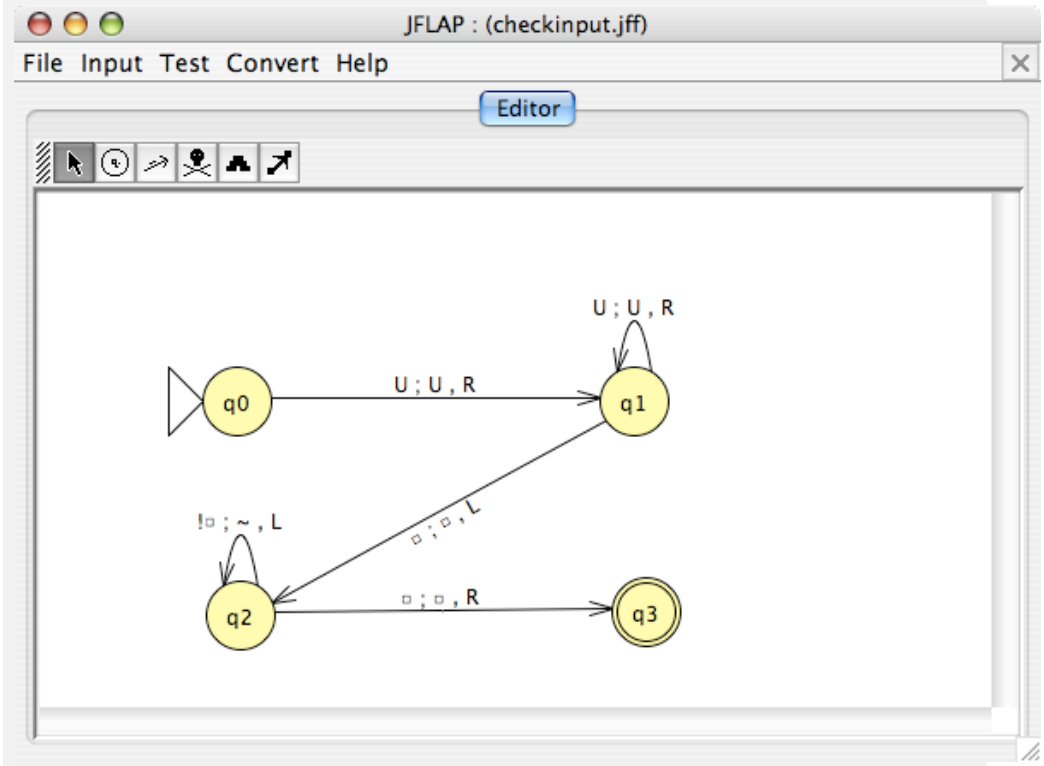
1. Controlla che il simbolo attualmente letto sia `U`: se così non fosse, termina segnalando un errore (ad esempio, non definendo le transizioni corrispondenti a simboli diversi da `U`).

2. Sposta la testina a destra: se il simbolo attualmente letto è $\square$, posiziona nuovamente la testina sul primo simbolo della stringa in input e termina. Altrimenti, torna al passo precedente.

In generale, esplicitare una macchina di Turing che verifichi la correttezza dell'input all'interno di un esempio, complicherebbe solamente l'esposizione senza aggiungere nulla di particolarmente interessante alla comprensione dell'esempio stesso. Per questo motivo, nel seguito continueremo ad assumere che l'input fornito in ingresso a una macchina di Turing sia sempre corretto.

La macchina è data nella seguente figura.

macchina per la verifica della correttezza dell'input



1.3 Macchine di Turing multi-nastro

NEL PRIMO paragrafo di questo capitolo abbiamo definito la macchina di Turing, assumendo che essa possa utilizzare un *solo* nastro di lettura e scrittura: in alcuni degli esempi del secondo paragrafo, questo vincolo ci ha costretto a dover scorrere ripetutamente il contenuto del nastro e, quindi, a eseguire un elevato numero di passi (tipicamente quadratico rispetto alla lunghezza della stringa in ingresso). In questo paragrafo, vedremo come questa restrizione non influenza in alcun modo il potere di calcolo di una macchina di Turing, anche se l'utilizzo di un numero maggiore di nastri può consentire di sviluppare macchine di Turing più veloci.

Una **macchina di Turing multi-nastro** è del tutto simile a una macchina di Turing ordinaria, con l'unica differenza che essa ha a disposizione un numero fissato k di nastri con $k \geqslant 1$. Ciascun nastro è dotato di una testina di lettura e scrittura: ogni

istruzione del programma della macchina di Turing deve, quindi, specificare, oltre al nuovo stato, i k simboli letti dalle k testine, affinché l'istruzione sia eseguita, i k simboli che devono essere scritti al loro posto, e i k movimenti che devono essere eseguiti dalle k testine.

Formalmente, quindi, a ogni transizione è associata un'etichetta formata da una lista di triple

$$((\sigma_1,\ldots,\sigma_k),(\tau_1,\ldots,\tau_k),(m_1,\ldots,m_k))$$

I simboli $\sigma_1,\ldots,\sigma_k$ e $\tau_1,\ldots,\tau_k$ che appaiono all'interno di una tripla dell'etichetta di una transizione indicano, rispettivamente, i k simboli attualmente letti dalle k testine e i k simboli da scrivere, mentre i valori $m_1,\ldots,m_k$ specificano i movimenti che devono essere effettuati dalle k testine.

Inizialmente, una macchina di Turing con k nastri viene avviata con la stringa di input posizionata sul primo nastro, con i rimanenti $k-1$ nastri vuoti (ovvero, contenenti solo simboli $\square$), con la testina del primo nastro posizionata sul primo simbolo della stringa di input e con la CPU configurata nello stato iniziale. Nel caso in cui la macchina produca un output, assumiamo anche che essa termini con la CPU configurata in uno stato finale, con la stringa di output contenuta nel primo nastro e seguita da un simbolo $\square$ e con la testina del primo nastro posizionata sul primo simbolo della stringa di output.

1.3.1 Un esempio di macchina di Turing multi-nastro

Consideriamo nuovamente il linguaggio $L=\{0^n1^n : n \geqslant 0\}$ e definiamo una macchina di Turing T con 2 nastri che, data in input una sequenza di simboli `0` seguita da una sequenza di simboli `1`, termina in uno stato finale se il numero di simboli `0` è uguale a quello dei simboli `1`, altrimenti termina in uno stato non finale non avendo istruzioni da poter eseguire. Tale macchina opera nel modo seguente.

1. Scorre il primo nastro verso destra fino a incontrare il primo simbolo `1`: per ogni simbolo `0` letto viene scritto un simbolo `1` sul secondo nastro, spostando la testina a destra dopo aver eseguito la scrittura.

2. Scorre il primo nastro da sinistra verso destra e il secondo nastro da destra verso sinistra, controllando che i simboli letti sui due nastri siano uguali: se così non fosse, termina senza avere alcuna istruzione da poter eseguire.

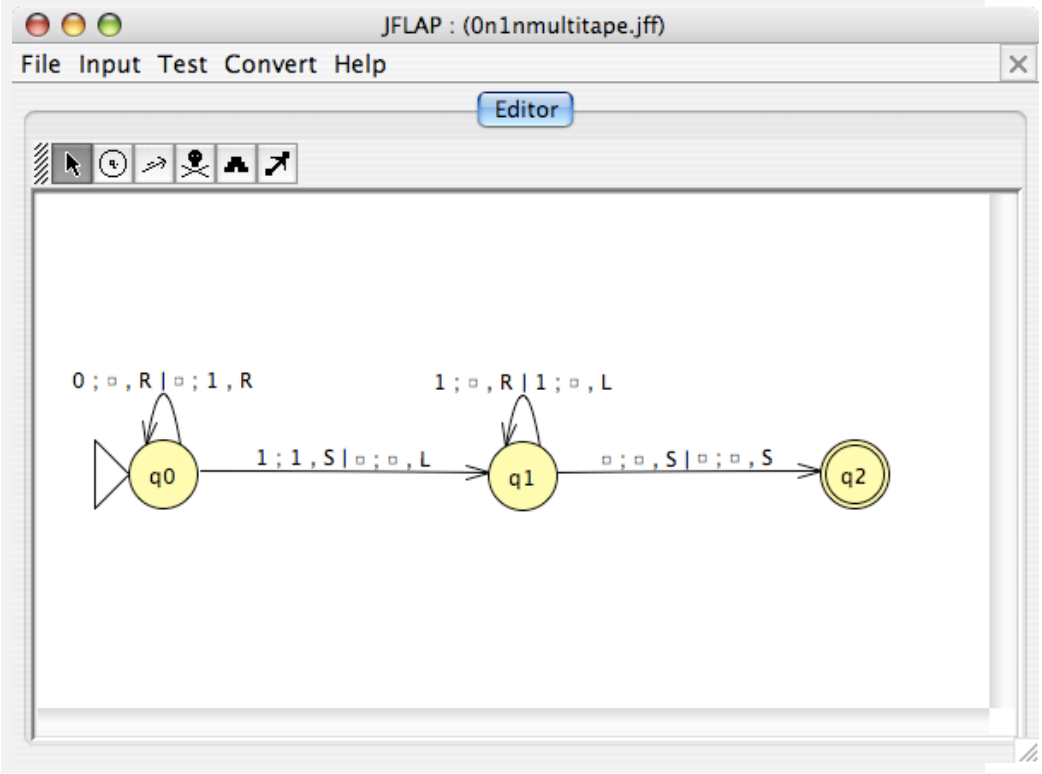
3. Una volta incontrato il simbolo $\square$ su entrambi i nastri, termina nello stato finale.

La rappresentazione tabellare di T è, dunque, la seguente.

stato	simboli	stato	simboli	movimenti
q0	(0,□)	q0	(□,1)	(R,R)
q0	(1,□)	q1	(1,□)	(S,L)
q1	(1,1)	q1	(□,□)	(R,L)
q1	(□,□)	q2	(□,□)	(S,S)

Per ogni stringa x di lunghezza n, il numero di passi eseguito da T prima di terminare è chiaramente lineare in n: in effetti, T legge una e una sola volta ciascun simbolo di x presente sul primo nastro (a eccezione del primo simbolo 1). Pertanto, tale macchina di Turing multi-nastro è significativamente più veloce di quella con un solo nastro, che abbiamo descritto nel secondo paragrafo di questo capitolo e che eseguiva un numero di passi quadratico rispetto alla lunghezza della stringa in ingresso.

macchina multi-nastro per 0^n1^n



1.3.2 Simulazione di macchine di Turing multi-nastro

Chiaramente, tutto ciò che può essere fatto da una macchina di Turing ordinaria risulta essere realizzabile da una macchina di Turing multi-nastro. In effetti, una macchina di Turing ordinaria T corrisponde al caso particolare in cui il numero di nastri k sia uguale a 1: pertanto, una macchina di Turing con $k > 1$ nastri può simulare T sul primo nastro non modificando mai il contenuto dei rimanenti $k-1$ nastri. A prima vista, potrebbe invece sembrare che le macchine multi-nastro siano computazionalmente più potenti di quelle ordinarie. Tuttavia, facendo uso della tecnica di zig-zag combinata con quella di piggy-backing viste nel secondo paragrafo di questo capitolo, è possibile dimostrare che i due modelli di calcolo sono in realtà equivalenti. In particolare, possiamo mostrare che ogni macchina di Turing T con k nastri, dove $k > 1$, può essere simulata da una macchina di Turing T' con un solo nastro.

L'idea della dimostrazione consiste nel concatenare uno dopo l'altro il contenuto dei k nastri, separati da un simbolo speciale, che non faccia parte dell'alfabeto di lavoro della macchina multi-nastro: indichiamo con $\#$ tale simbolo. La macchina T' simulerà ogni istruzione di T scorrendo i k nastri e "raccogliendo" le informazioni relative ai k simboli letti dalle k testine. Terminata questa prima scansione, T' sarà in grado di decidere quale transizione applicare, per cui scorrerà nuovamente i k nastri applicando su ciascuno di essi la corretta operazione di scrittura e di spostamento della testina. Al termine di questa seconda scansione, T' riposizionerà la testina all'inizio del primo nastro e sarà così pronta a simulare la successiva istruzione.

Formalmente, sia Σ l'alfabeto di lavoro di T. Allora, l'alfabeto di lavoro di T' sarà definito nel modo seguente.

$$\Sigma' = \Sigma \cup \{\sigma^{\text{head}} : \sigma \in \Sigma\} \cup \{\#, \top\}$$

Intuitivamente, ciascun simbolo σ^{head} consentirà di identificare, per ogni nastro, il simbolo attualmente scandito dalla sua testina mentre il simbolo $\top$ sarà utilizzato come delimitatore sinistro e destro del nastro di T'. Se q_0 è lo stato iniziale di T, F è l'insieme dei suoi stati finali e Q l'insieme dei rimanenti suoi stati, allora la macchina T' avrà lo stesso stato iniziale di T e gli stessi stati finali e, per ogni stato $q \in \{q_0\} \cup Q$, avrà il seguente insieme di stati.

$$\bigcup_{h=0}^{k} \{q^{\sigma_1,\ldots,\sigma_h}, \bar{q}^{\sigma_1,\ldots,\sigma_h} : \sigma_i \in \Sigma \text{ per ogni } i \text{ con } 1 \leqslant i \leqslant h\}$$

Intuitivamente, tali stati consentiranno di memorizzare i simboli scanditi dalle k testine, man mano che questi vengono identificati. La macchina T', che avrà ulteriori stati ausiliari, opera nel modo seguente.

- "Crea" i k nastri separandoli con il simbolo $\#$. In particolare, ogni nastro (a eccezione del primo) sarà creato con un solo simbolo $\square^{\text{head}}$ al suo interno, mentre sul primo nastro il primo simbolo σ della stringa in ingresso in esso contenuta sarà sostituito con il suo corrispondente simbolo σ^{head}. Inoltre, per indicare l'estremo sinistro del primo nastro e quello destro dell'ultimo nastro, sostituisce il primo simbolo $\square$ alla sinistra del primo nastro e il primo simbolo $\square$ alla destra dell'ultimo nastro con il simbolo speciale $\top$. Infine, lascia la testina posizionata sul simbolo alla destra dell'istanza sinistra di $\top$ e entra nello stato q_0^λ.

- Essendo nello stato q^λ con $q \in \{q_0\} \cup Q$ ed essendo la testina posizionata sul primo simbolo del primo nastro, legge i k simboli attualmente scanditi dalle k testine e memorizza quest'informazione nel corrispondente stato. In particolare, scorre il nastro da sinistra verso destra e, ogni qualvolta incontra un simbolo σ^{head}, passa dallo stato $q^{\sigma_1,\dots,\sigma_h}$ allo stato $q^{\sigma_1,\dots,\sigma_h,\sigma}$, dove $0 \leqslant h < k$. Una volta raggiunto lo stato $q^{\sigma_1,\dots,\sigma_k}$, scorre il nastro verso sinistra fino a incontrare l'istanza sinistra del simbolo $\top$, posiziona la testina sul simbolo immediatamente a destra e entra nello stato $\bar{q}^{\sigma_1,\dots,\sigma_k}$.

- Supponiamo che, essendo nello stato $\bar{q}^{\sigma_1,\dots,\sigma_k}$ con la testina posizionata sul primo simbolo del primo nastro, esista una transizione della macchina di Turing multi-nastro che parte dallo stato q e che termina in uno stato p, la cui etichetta contiene la tripla

$$e = ((\sigma_1,\dots,\sigma_k),(\tau_1,\dots,\tau_k),(m_1,\dots,m_k))$$

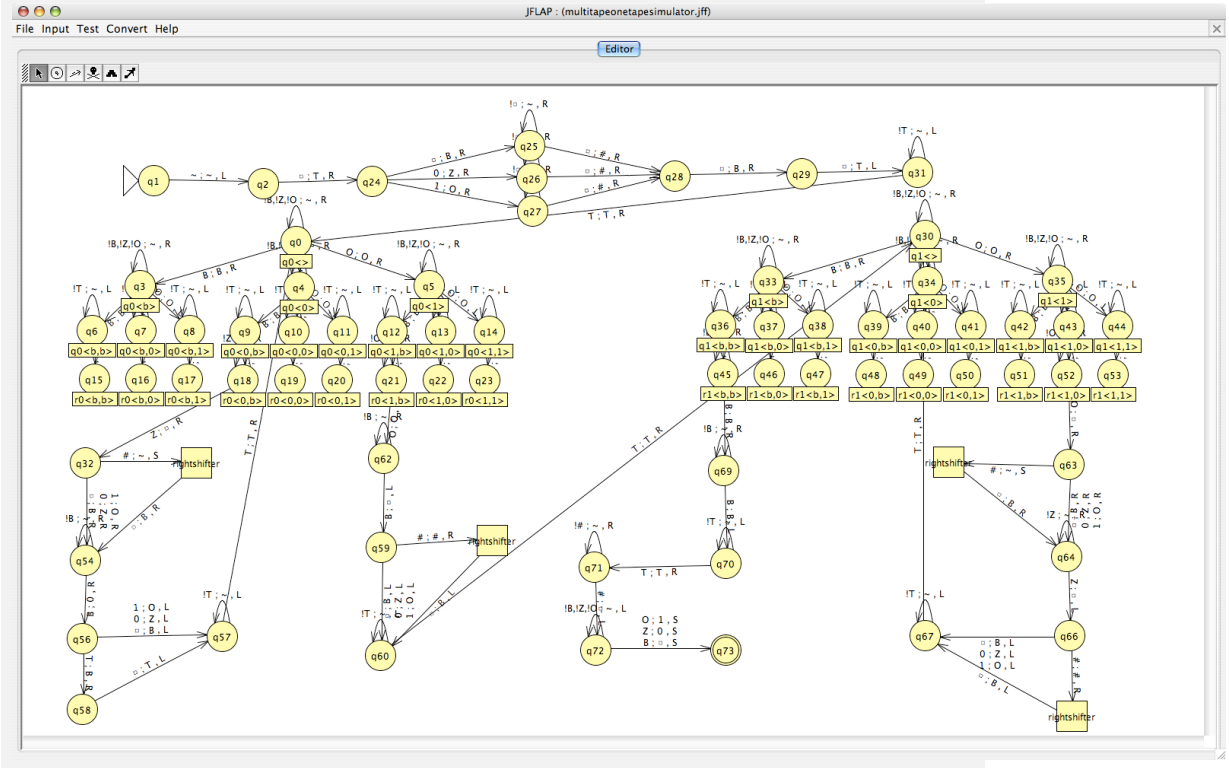
  Allora, T' scandisce i k nastri sostituendo ciascun simbolo σ_i^{head} che incontra con il corrispondente simbolo τ_i e simulando lo spostamento della testina corrispondente al movimento m_i. Per realizzare lo spostamento della testina, potrà essere necessario spostare il contenuto dell'intero nastro di una posizione a destra: ciò accade, per esempio, se la testina di uno dei k nastri si deve spostare a destra, ma il simbolo alla destra della testina della macchina con un solo nastro è il simbolo $\#$. Al termine della simulazione della transizione, posiziona la testina sul simbolo alla destra dell'istanza sinistra di $\top$: se $p \notin F$, allora entra nello stato p^λ e torna al passo precedente. Altrimenti sostituisce il simbolo $\#$ più a sinistra con il simbolo $\square$ sostituisce il simbolo σ^{head} contenuto nel primo nastro con il simbolo σ e termina in p con la testina posizionata su tale simbolo.

- Se, essendo nello stato $\bar{q}^{\sigma_1,\dots,\sigma_k}$ con la testina posizionata sul primo simbolo del primo nastro, non esiste una transizione della macchina di Turing multi-nastro che parte dallo stato q, la cui etichetta contenga una tripla il cui primo

elemento è $(\sigma_1, \ldots, \sigma_k)$, allora termina in modo anomalo, ovvero non avendo alcuna transizione uscente dallo stato $\bar{q}^{\sigma_1,\cdots,\sigma_k}$.

Questa descrizione del comportamento di T' è per forza di cose non del tutto formale, in quanto una costruzione rigorosa di T' richiederebbe la specifica esatta del suo grafo delle transizioni: dovrebbe comunque risultare chiaro che, volendo, ciò può essere fatto.

simulazione di una macchina multi-nastro



In conclusione, abbiamo mostrato come una macchina di Turing con un solo nastro possa simulare in modo corretto il comportamento di una macchina di Turing multi-nastro e, quindi, che le macchine di Turing multi-nastro non sono computazionalmente più potenti di quelle con un singolo nastro. Osserviamo, però, che la simulazione descritta in precedenza causa un numero di passi eseguiti da T' significativamente maggiore di quello relativo alla macchina di Turing T con k nastri. In effetti, per ogni stringa x di lunghezza n, abbiamo che, per ogni passo di T, la

macchina T' esegue anzitutto una doppia scansione del contenuto dei k nastri per determinare i simboli attualmente letti. Successivamente, un'ulteriore doppia scansione viene eseguita per eventualmente simulare la transizione: durante la prima scansione, inoltre, potrebbe essere richiesto uno spostamento a destra del contenuto di tutti i nastri che seguono quello su cui si sta operando la transizione. Poiché la lunghezza di ciascun nastro può essere pari a $t_T(n)$ (nel caso in cui, a ogni passo, una nuova cella di ciascun nastro venga scandita), abbiamo che il numero di passi eseguiti da T' per simulare un passo di T può essere proporzionale a $t_T(n)$. Pertanto, possiamo concludere che $t_{T'}(n) \in O(t_T^2(n))$, ovvero che la simulazione può richiedere un numero di passi quadratico rispetto al numero di passi eseguiti dalla macchina multi-nastro.

1.4 Configurazioni di una macchina di Turing

DURANTE L'ESECUZIONE del suo programma, una macchina di Turing T può cambiare lo stato della CPU, il contenuto del nastro e la posizione della testina. La combinazione di queste tre componenti è detta **configurazione** di T e viene generalmente rappresentata nel seguente modo: date due stringhe x e y, con $x, y \in \Sigma^*$ dove Σ è l'alfabeto di lavoro di T, e dato uno stato q di T, $x^q y$ rappresenta la configurazione in cui lo stato della CPU è q, il nastro contiene la stringa xy e la testina è posizionata sul primo simbolo di y.

Notiamo che sebbene il nastro di una macchina di Turing sia infinito, la rappresentazione di una configurazione, così come è stata appena definita, fa riferimento a sequenze finite di simboli. Ciò non costituisce una contraddizione, in quanto possiamo sempre assumere che i simboli alla sinistra e alla destra di una configurazione siano tutti uguali al simbolo $\square$.

In particolare, in queste dispense faremo sempre riferimento a rappresentazioni minimali che sono definite nel modo seguente: una configurazione $x^q y$ è **minimale** se (1) tutti i simboli che si trovano alla sinistra di x nel nastro sono uguali a $\square$, (2) tutti i simboli che si trovano alla destra di y nel nastro sono uguali a $\square$, (3) il primo simbolo di x è diverso da $\square$, nel caso in cui $|x| \geqslant 1$ e (4) l'ultimo simbolo di y è diverso da $\square$, nel caso in cui $|y| \geqslant 1$.

Data una stringa $x \in \Sigma^* - \{\square\}$, la **configurazione iniziale** della macchina di Turing T con input la stringa x è dunque la configurazione ${}^{q_0}x$, dove q_0 è lo stato iniziale di T. Inoltre, una configurazione $x^q y$ è detta **finale** o **di accettazione** se q è uno stato finale: l'**output** di tale configurazione è il prefisso più lungo di y che non contiene alcun simbolo $\square$. Infine, una configurazione $x^q y$ è detta **di rigetto** se non esiste alcuna transizione a partire dallo stato q la cui etichetta contenga una tripla il cui primo elemento è il primo simbolo di y se $|y| \geqslant 1$, il simbolo $\square$ altrimenti.

Connettivi e quantificatori

Faremo spesso uso di notazioni della logica elementare: in particolare, 'e' sarà abbreviato con $\wedge$, 'o' con $\vee$, 'solo se' con $\rightarrow$ e 'non' con $\neg$. Tutti questi simboli sono chiamati **connettivi**. I simboli $\forall$ e $\exists$ sono invece i **quantificatori esistenziale** e **universale**, rispettivamente: pertanto, $\forall x$ si legge 'per ogni x' mentre $\exists x$ si legge 'esiste x tale che'. I simboli logici appena introdotti consentiranno di rappresentare in modo sintetico espressioni del linguaggio matematico ordinario: ad esempio, $A \subseteq B$ può essere espresso come $\forall x[x \in A \rightarrow x \in B]$ oppure $A \neq B$ può essere espresso come $\exists x[\neg(x \in A \wedge x \in B)]$.

Esempio 1.4: configurazioni della macchina per il complemento bit a bit

Partendo dalla configurazione iniziale $^{q0}0101$, in cui quindi l'input è la stringa 0101, la macchina di Turing rappresentata in Figura 1.2 passerà attraverso le seguenti configurazioni: $1^{q0}101$, $10^{q0}01$, $101^{q0}1$, 1010^{q0}, $101^{q1}0$, $10^{q1}10$, $1^{q1}010$, $^{q1}1010$ e $^{q1}\square 1010$. L'ultima configurazione sarà seguita dalla configurazione $^{q2}1010$, che è finale (in quanto lo stato q2 è uno stato finale) e che corrisponde al termine dell'esecuzione del programma. Come era lecito aspettarsi, la stringa prodotta in output dall'esecuzione, ovvero 1010, è il complemento bit a bit della stringa di input.

1.4.1 Produzioni tra configurazioni

In precedenza, abbiamo ripetutamente parlato di passi eseguiti da una macchina di Turing con input una determinata stringa. In questo paragrafo, specifichiamo formalmente che cosa intendiamo per singolo passo di una macchina di Turing: a tale scopo, faremo uso di alcune funzioni ausiliarie. In particolare, dato un qualunque alfabeto Σ contenente il simbolo $\square$, definiamo le seguenti funzioni con dominio l'insieme Σ^*.

$$\texttt{left}(x) = \begin{cases} x' & \text{se } x = x'\alpha \text{ con } \alpha \in \Sigma, \\ \lambda & \text{altrimenti} \end{cases}$$

$$\texttt{right}(x) = \begin{cases} x' & \text{se } x = \alpha x' \text{ con } \alpha \in \Sigma, \\ \lambda & \text{altrimenti} \end{cases}$$

$$\texttt{first}(x) = \begin{cases} \alpha & \text{se } x = \alpha x' \text{ con } \alpha \in \Sigma, \\ \square & \text{altrimenti} \end{cases}$$

$$\texttt{last}(x) = \begin{cases} \alpha & \text{se } x = x'\alpha \text{ con } \alpha \in \Sigma, \\ \square & \text{altrimenti} \end{cases}$$

$$\texttt{reduceLR}(x) = \begin{cases} \texttt{reduceLR}(x') & \text{se } x = \square x', \\ x & \text{altrimenti} \end{cases}$$

$$\texttt{reduceRL}(x) = \begin{cases} \texttt{reduceRL}(x') & \text{se } x = x'\square, \\ x & \text{altrimenti} \end{cases}$$

Data una macchina di Turing T con alfabeto di lavoro Σ, dati due stati q e p di T di cui q non è finale e date quattro stringhe x, y, u e v in Σ^*, diremo che la configurazione $C_1 = x^q y$ produce la configurazione $C_2 = u^p v$ se T può passare da C_1 a C_2 eseguendo un'istruzione del suo programma. Formalmente, diremo che C_1 **produce** C_2 se esiste un arco del grafo delle transizioni di T da q a p la cui etichetta contiene una tripla (σ, τ, m) tale che

$$(y = \sigma y') \vee (y = \lambda \wedge \sigma = \square)$$

e una delle seguenti tre condizioni sia soddisfatta.

Mossa a sinistra $m = \texttt{L} \wedge u = \texttt{left}(x) \wedge v = \texttt{reduceRL}(\texttt{last}(x)\tau\texttt{right}(y))$.

Nessuna mossa $m = \texttt{S} \wedge u = x \wedge v = \texttt{reduceRL}(\tau\texttt{right}(y))$.

Mossa a destra $m = \texttt{R} \wedge u = \texttt{reduceLR}(x\tau) \wedge v = \texttt{right}(y)$.

Esempio 1.5: produzioni della macchina per il complemento bit a bit

In base a quanto visto nell'Esempio 1.4, possiamo dire che, facendo riferimento alla macchina di Turing rappresentata in Figura 1.2, la configurazione $\texttt{101}^{q0}\texttt{1}$ produce la configurazione $\texttt{1010}^{q0}$. In questo caso, infatti, $x = \texttt{101}$, $y = \texttt{1}$ e l'etichetta della transizione da $q0$ a $q0$ contiene la tripla $(1,0,R)$: quindi, $y = 1 = \sigma\lambda$, $m = R$, $u = \texttt{1010} = \texttt{reduceLR}(x\tau)$ e $v = \lambda = \texttt{right}(y)$. Al contrario, non è vero che la configurazione $\texttt{1}^{q0}\texttt{101}$ produce la configurazione $\texttt{101}^{q0}\texttt{1}$. In questo caso, infatti, $x = 1$, $y = \texttt{101}$ e l'etichetta della transizione da $q0$ a $q0$ contiene le due triple $(0,1,R)$ e $(1,0,R)$. Nel caso della prima tripla, non è soddisfatta la condizione $(y = \sigma y') \vee (y = \lambda \wedge \sigma = \square)$, mentre nel caso della seconda tripla non sono soddisfatte le due condizioni $u = \texttt{reduceLR}(x\tau)$ (in quanto $u = \texttt{101} \neq \texttt{10} = \texttt{reduceLR}(x\tau)$) e $v = \texttt{right}(y)$ (in quanto $v = \texttt{1} \neq \texttt{01} = \texttt{right}(y)$).

1.5 Sotto-macchine

LA STRUTTURAZIONE di un programma, per la risoluzione di un dato problema, in sotto-programmi, ciascuno di essi dedicato alla risoluzione di un problema più semplice di quello originale, è una pratica molto comune nel campo dello sviluppo di software: ci si riferisce spesso a tale pratica con il termine di **programmazione procedurale**. Tale strategia di programmazione può essere applicata anche nel campo dello sviluppo di macchine di Turing, immaginando che una macchina di Turing possa al suo interno "invocare" delle sotto-macchine, ciascuna di esse dedicata al calcolo di una specifica funzione o, più in generale, alla realizzazione di una specifica operazione.[1]

Una **sotto-macchina** di una macchina di Turing T è un nodo s del grafo delle transizioni di T a cui è associata una macchina di Turing T_s e il cui comportamento in fase di esecuzione si differenzia da quello di un normale nodo (ovvero, di un nodo corrispondente a uno stato di T) nel modo seguente. Nel momento in cui la CPU di T "entra" in s, il controllo passa alla CPU di T_s, la quale inizia l'esecuzione del programma di T_s a partire dal suo stato iniziale e con il contenuto del nastro e la posizione della testina così come lo erano in T. Nel momento in cui la CPU di T_s raggiunge uno stato finale, il controllo ritorna alla CPU di T la quale eseguirà, se esiste, la transizione da s allo stato successivo in base al simbolo lasciato sotto la testina da T_s.

In altre parole, se T si trova nella configurazione $u^q v$ e il suo grafo delle transizioni include un arco da q a s la cui etichetta contiene la tripla (σ, τ, m) e $(v = \sigma v') \vee (v = \lambda \wedge \sigma = \square)$, allora la macchina T, invece di passare nella configurazione $w^s x$ prodotta da $u^q v$, passa nella configurazione $w^{q_0^s} x$ di T_s, dove q_0^s è lo stato iniziale di T_s. A questo punto, l'esecuzione procede con il programma di T_s: nel momento in cui tale macchina raggiunge una configurazione finale $y^{q_f^s} z$, allora il controllo passa nuovamente a T che si troverà nella configurazione $y^s z$ e che proseguirà nella configurazione da essa prodotta (se è definita).

Esercizi

Esercizio 1.1. Si definisca una macchina di Turing T con un solo nastro che, dato in input un numero intero non negativo rappresentato in forma binaria, produca in output la stringa 0. Si determini, inoltre, la funzione $t_T(n)$.

[1] Abbiamo già fatto uso di una sotto-macchina per lo slittamento del contenuto del nastro di una posizione a destra all'interno della macchina con un solo nastro che simulava una macchina multi-nastro.

Esercizio 1.2. Si definisca una macchina di Turing T con un solo nastro che, data in input una sequenza binaria x, produca in output due copie di x separate da un simbolo $\mathtt{x}$. Si determini, inoltre, la funzione $t_T(n)$.

Esercizio 1.3. Si definisca una macchina di Turing T con un solo nastro che, data in input una sequenza binaria, termini in uno stato finale se e solo se tale sequenza contiene un ugual numero di $\mathtt{0}$ e di $\mathtt{1}$. Si determini, inoltre, la funzione $t_T(n)$.

Esercizio 1.4. Si definisca una macchina di Turing T con un solo nastro che, data in input una sequenza binaria x, termini in uno stato finale se e solo se tale sequenza è palindroma, ovvero se e solo se $x_1 \cdots x_n = x_n \cdots x_1$. Si determini, inoltre, la funzione $t_T(n)$.

Esercizio 1.5. Si definisca una macchina di Turing T con un solo nastro che, date in input due sequenze binarie x e y separate dal simbolo $\square$, produca in output la sequenza binaria corrispondente alla disgiunzione bit a bit di x e y. Si determini, inoltre, la funzione $t_T(n)$.

Esercizio 1.6. Si definisca una macchina di Turing T con un solo nastro che, data in input la rappresentazione binaria di un numero intero non negativo x, produca in output la sequenza di $x+1$ simboli $\mathtt{1}$. Si determini, inoltre, la funzione $t_T(n)$.

Esercizio 1.7. Data una stringa x sull'alfabeto $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$, la stringa ordinata corrispondente a x è la stringa su Σ ottenuta a partire da x mettendo tutti i simboli $\mathtt{0}$ prima dei simboli $\mathtt{1}$ e tutti questi ultimi prima dei simboli $\mathtt{2}$: ad esempio, la stringa ordinata corrispondente a $\mathtt{012021102120201210}$ è la stringa $\mathtt{000000111111222222}$. Si definisca una macchina di Turing T con un solo nastro che, data in input una stringa x sull'alfabeto Σ, termini producendo in output la sequenza ordinata corrispondente a x. Si determini, inoltre, la funzione $t_T(n)$.

Esercizio 1.8. Definire una macchina di Turing T con 2 nastri che, data in input una stringa binaria x, termini producendo in output la sequenza ordinata corrispondente a x e tale che $t_T(n) \in O(n)$.

Esercizio 1.9. Definire una macchina di Turing T con 3 nastri che calcoli la somma di due numeri interi non negativi codificati in binario e separati dal simbolo $+$ e tale che $t_T(n) \in O(n)$.

Esercizio 1.10. Per ciascuna macchina di Turing T definita in uno degli Esercizi precedenti su una singola macchina, si definisca un'equivalente macchina di Turing T' multi-nastro e si confronti $t_T(n)$ con $t_{T'}(n)$.

Esercizio 1.11. Si definisca una macchina di Turing T con due nastri che, data in input una sequenza di simboli dell'alfabeto $\{(,),[,],\{,\}\}$, termini nello stato finale se e solo se le parentesi sono bilanciate e tale che $t_T(n) \in O(n)$.

Capitolo 2

La macchina di Turing universale

SOMMARIO

In questo capitolo mostriamo come il modello delle macchine di Turing, pur nella sua semplicità, consenta di definire una macchina di Turing universale, ovvero quanto di più simile vi possa essere agli odierni calcolatori: tale macchina, infatti, è in grado di simulare il comportamento di una qualunque altra macchina con input una qualunque stringa. Allo scopo di definire la macchina di Turing universale, introdurremo inoltre il concetto di codifica di una macchina di Turing (dopo aver mostrato come non sia restrittivo considerare solo macchine di Turing con un alfabeto di lavoro costituito da tre simboli). Concluderemo, quindi, il capitolo dimostrando che diverse ulteriori varianti del modello di base delle macchine di Turing sono a esso computazionalmente equivalenti.

2.1 Macchine di Turing con alfabeto limitato

NELLA MAGGIOR parte degli esempi di macchine di Turing visti fino a ora, l'alfabeto di lavoro era costituito da un numero molto limitato di simboli (sempre al di sotto della decina). La domanda che possiamo porci è se limitare ulteriormente tale alfabeto riduca le potenzialità di calcolo del modello. In particolare, consideriamo il caso in cui l'alfabeto di lavoro sia costituito, oltre che dal simbolo $\square$, dai soli simboli `0` e `1`. Chi ha già familiarità con l'informatica, sa bene che questo è l'alfabeto utilizzato dagli odierni calcolatori: ogni carattere di un diverso alfabeto viene, infatti, associato a un codice binario, che può essere costituito da 8 bit (nel caso del codice ASCII) oppure da 16 bit (nel caso del codice Unicode).

Pertanto, è ragionevole supporre che la restrizione a un alfabeto binario non sia assolutamente limitativa rispetto al potere computazionale di una macchina di Turing. L'unica attenzione che dobbiamo prestare, risiede nel fatto di garantire che la

codifica di una sequenza di simboli non binari sia fatta in modo da consentirne in modo univoco la successiva decodifica.

Se, ad esempio, decidessimo di codificare i tre simboli `A`, `B` e `C` associando a essi le sequenze binarie `0`, `1` e `00`, rispettivamente, allora di fronte alla sequenza `000` non saremmo in grado di dire se tale sequenza sia la codifica di `AAA`, `AC` oppure `CA`. Il problema sta nel fatto che la codifica di `A` è un **prefisso** della codifica di `C`, per cui una volta letta tale codifica non sappiamo dire se ciò che abbiamo letto rappresenta la codifica di `A` oppure l'inizio della codifica di `C`.

Esistono molti modi per definire codici che evitano questo problema: in questo contesto, ci accontentiamo di definirne uno molto semplice e del tutto analogo a quello utilizzato dagli odierni calcolatori. Dato un alfabeto $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ con $n \geqslant 2$, sia $k = \lceil \log_2 n \rceil$. Per ogni i con $1 \leqslant i \leqslant n$, associamo al simbolo σ_i la stringa binaria $c_\Sigma(\sigma_i)$ di k simboli `0` e `1`, ottenuta calcolando la rappresentazione binaria di $i-1$ che fa uso di k bit. Data una stringa $x = x_1 \cdots x_m$ su Σ, indichiamo con $c_\Sigma(x)$ la stringa $c_\Sigma(x_1) \cdots c_\Sigma(x_m)$ di lunghezza km.

Esempio 2.1: codifica binaria di un alfabeto

Consideriamo l'alfabeto

$$\Sigma = \{\sigma_1 = a, \sigma_2 = b, \sigma_3 = c, \sigma_4 = d, \sigma_5 = e\}$$

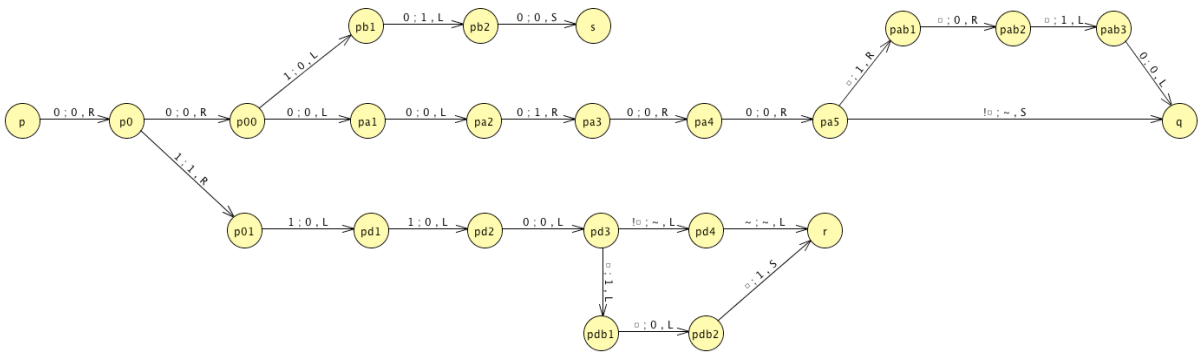
In questo caso, $k = \lceil \log_2 5 \rceil = 3$. Pertanto, $c(a) = c(\sigma_1) = 000$, $c(b) = c(\sigma_2) = 001$, $c(c) = c(\sigma_3) = 010$, $c(d) = c(\sigma_4) = 011$ e $c(e) = c(\sigma_5) = 100$. La codifica binaria della stringa `cade` è uguale a `010000011100`, mentre quella della stringa `debba` è uguale a `011100001001000`.

Data una macchina di Turing T con alfabeto di lavoro $\Gamma = \{\square\} \cup \Sigma$, dove $|\Sigma| \geqslant 2$, possiamo definire una macchina di Turing T' con alfabeto di lavoro $\{\square, 0, 1\}$ tale che valgono le seguenti affermazioni.

- Per ogni $x \in \Sigma^*$, se T con input x non termina, allora T' con input $c_\Sigma(x) \in \{0,1\}^*$ non termina.
- Per ogni $x \in \Sigma^*$, se T con input x termina in una configurazione di rigetto, allora T' con input $c_\Sigma(x) \in \{0,1\}^*$ termina in una configurazione di rigetto.
- Per ogni $x \in \Sigma^*$, se T con input x termina in uno stato finale e produce in output la stringa $y \in \Sigma^*$, allora T' con input $c_\Sigma(x) \in \{0,1\}^*$ termina in uno stato finale e produce in output la stringa $c_\Sigma(y) \in \{0,1\}^*$.

Il comportamento di T' è identico a quello di T con l'unica differenza che ogni singola transizione di T viene simulata mediante $O(k)$ transizioni di T' (dove $k = \lceil \log_2 |\Gamma| \rceil$),

Figura 2.1: le transizioni di una macchina con alfabeto binario.



che consentano di leggere la codifica di un simbolo di Γ, sostituire tale codifica con quella di un altro simbolo di Γ e posizionare in modo corretto la testina. Pertanto, $t_{T'}(n) \in O(t_T(n))$.

Esempio 2.2: macchine di Turing con alfabeto limitato

Consideriamo una macchina di Turing T il cui alfabeto di lavoro sia $\Gamma = \{\square\} \cup \Sigma$ dove Σ è l'alfabeto definito nell'Esempio 2.1, ovvero $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}\}$: supponiamo, inoltre, che la codifica del simbolo $\square$ sia `101`. Se T include (1) una transizione dallo stato `p` allo stato `s` che, leggendo il simbolo `b`, lo sostituisce con il simbolo `c` e non muove la testina, (2) una transizione dallo stato `p` allo stato `q` che, leggendo il simbolo `a`, lo sostituisce con il simbolo `e` e sposta la testina a destra e (3) una transizione dallo stato `p` allo stato `r` che, leggendo il simbolo `d`, lo sostituisce con il simbolo `a` e sposta la testina a sinistra, la macchina T' simula ciascuna di tali transizioni mediante una sequenza di al più 11 transizioni come mostrato nella Figura 2.1 (notiamo come sia necessario, nel caso di uno spostamento a destra o a sinistra, gestire anche la trasformazione di un eventuale simbolo $\square$ nella sua corrispondente codifica binaria).

Possiamo, dunque, concludere che restringendo l'alfabeto di lavoro di una macchina di Turing a tre soli simboli, incluso il simbolo $\square$, non si riduce il potere computazionale delle macchine di Turing. Ovviamente, l'utilizzo di siffatte macchine di Turing richiede che i valori di input e quelli di output siano stringhe binarie. Una macchina di Turing il cui alfabeto di lavoro sia $\{\square, \mathtt{0}, \mathtt{1}\}$ non potrà mai ricevere come stringa di input una sequenza che contenga simboli diversi da `0` e da `1` né potrà mai produrre un output che non sia una sequenza binaria: in altre parole, per poter usare

una tale macchina sarà necessario eseguire una fase di *pre-processing*, in cui l'input originale venga eventualmente codificato in una sequenza binaria, e una fase di *post-processing*, in cui l'output prodotto dalla macchina venga eventualmente decodificato in modo da ottenere l'output desiderato.

2.2 Codifica delle macchine di Turing

FINO A ora abbiamo parlato di macchine di Turing intendendo formalizzare con tale modello di calcolo la definizione del concetto di algoritmo. In questo capitolo e in quello successivo, il nostro scopo sarà quello di caratterizzare il potere computazionale delle macchine di Turing mostrandone sia le capacità che i limiti. In particolare, in questo capitolo mostreremo l'esistenza di una macchina di Turing universale in grado di simulare una qualunque altra macchina di Turing a partire da una descrizione di quest'ultima.

Per poter definire una macchina di Turing universale è dunque necessario stabilire anzitutto in che modo una macchina di Turing da simulare possa essere codificata mediante una stringa. Tale codifica può essere fatta in tanti modi diversi e, in questo paragrafo, ne introdurremo uno che assume di dover operare solo su macchine di Turing con alfabeto di lavoro costituito dai simboli $\square$, `0` e `1` (abbiamo visto, nel paragrafo precedente, come tale restrizione non costituisca alcuna perdita di generalità).

Per codificare una macchina di Turing, dobbiamo decidere come codificare i simboli del suo alfabeto di lavoro, i possibili stati della sua CPU e i possibili movimenti della sua testina. Nel fare ciò, possiamo fare uso di un qualunque insieme di simboli, in quanto la trasformazione di una macchina di Turing universale che abbia un alfabeto di lavoro con più di tre simboli in una macchina con alfabeto di lavoro uguale a $\{\square, \texttt{0}, \texttt{1}\}$ può essere successivamente realizzata in modo simile a quanto visto nel paragrafo precedente.

Codifichiamo, pertanto, i simboli dell'alfabeto di lavoro di una macchina di Turing T nel modo seguente: la codifica di `0` è `Z`, quella di `1` è `U` e quella di $\square$ è `B`. I movimenti della testina di T sono codificati mediante i simboli `L` (movimento a sinistra), `S` (nessun movimento) e `R` (movimento a destra). Infine, se l'insieme degli stati di T (inclusi quello iniziale e quelli finali) ha cardinalità $\mathtt{k}$, allora, per ogni $\mathtt{i}$ con $0 \leqslant \mathtt{i} \leqslant \mathtt{k}-1$, l'$(\mathtt{i}+1)$-esimo stato sarà codificato dalla rappresentazione binaria di $\mathtt{i}$: nel seguito, senza perdita di generalità, assumeremo sempre che il primo stato (ovvero quello con codifica `0`) sia lo stato iniziale e che il secondo stato (ovvero quello con codifica `1`) sia l'*unico* stato finale.[1]

[1] Se la macchina di Turing non ha stati finali, possiamo sempre aggiungerne uno che non sia raggiungibile da nessun altro stato, mentre nel caso in cui la macchina di Turing abbia più di uno stato finale possiamo sempre farli "convergere" in un unico stato finale attraverso delle transizioni fittizie.

Ciascuna transizione di T viene codificata giustapponendo le codifiche dei vari elementi che la compongono, ovvero dello stato di partenza, del simbolo letto, dello stato di arrivo, del simbolo scritto e del movimento della testina. L'intero programma della macchina T viene quindi codificato elencando tutte le codifiche delle possibili transizioni una di seguito all'altra. Osserviamo che tale codifica non è ambigua, in quanto utilizza alfabeti diversi per la codifica degli stati (che fa uso dei simboli `0` e `1`), per quella dei movimenti (che fa uso dei simboli `L`, `R` e `S`) e per quella dei simboli (che fa uso dei simboli `B`, `U` e `Z`). Osserviamo inoltre che alla stessa macchina possono essere associate codifiche diverse, in quanto gli stati possono essere numerati in modo diverso (a parte quello iniziale e quello finale) e le transizioni possono essere elencate in un diverso ordine.

Esempio 2.3: codifica di una macchina di Turing

Consideriamo la macchina di Turing mostrata nella Figura 1.2, che calcola il complemento bit a bit di una sequenza binaria e la cui rappresentazione tabellare è la seguente (una volta rinominati gli stati in modo da soddisfare le assunzioni fatte in precedenza relativamente allo stato iniziale e a quello finale).

stato	simbolo	stato	simbolo	movimento	codifica
q0	0	q0	1	R	0Z0UR
q0	1	q0	0	R	0U0ZR
q0	□	q2	□	R	0B10BR
q2	0	q2	0	L	10Z10ZL
q2	1	q2	1	L	10U10UL
q2	□	q1	□	R	10B1BR

La codifica di una transizione è indicata nell'ultima colonna della tabella, per cui una codifica dell'intera macchina di Turing è `0Z0UR0U0ZR0B10BL10Z10ZL10U10UL10B1BR`. La stessa macchina di Turing potrebbe essere rappresentata dalla seguente tabella.

stato	simbolo	stato	simbolo	movimento	codifica
q0	□	q2	□	R	0B10BR
q0	1	q0	0	R	0U0ZR
q0	0	q0	1	R	0Z0UR
q2	□	q1	□	R	10B1BR
q2	1	q2	1	L	10U10UL
q2	0	q2	0	L	10Z10ZL

In tal caso, la codifica sarebbe `0B10BL0U0ZR0Z0UR10B1BR10U10UL10Z10ZL` che è diversa da quella precedente ma che, comunque, rappresenta la stessa macchina.

2.3 La macchina di Turing universale

LA MACCHINA di Turing universale fu proposta la prima volta da Turing stesso e ha svolto un ruolo importante nello stimolare lo sviluppo di calcolatori basati sulla cosiddetta architettura di von Neumann, ovvero calcolatori la cui memoria conserva sia i dati che i programmi da eseguire sui dati stessi. Esistono diverse definizioni di tale macchina, che differiscono per il numero di nastri, per il numero di simboli e per il numero di stati utilizzati: in questo paragrafo, ne definiamo una basata sulla codifica delle macchine di Turing introdotta nel paragrafo precedente. In particolare, la nostra **macchina di Turing universale** U farà uso di tre nastri: il primo nastro inizialmente conterrà la codifica c_{T} di una macchina di Turing T seguita da un simbolo `;` e dalla stringa binaria x in input, il secondo nastro a regime memorizzerà la codifica dello stato corrente di T, mentre il terzo e ultimo nastro servirà a simulare il nastro di lavoro di T. La macchina U opera nel modo seguente.

1. Copia sul terzo nastro l'input x codificato mediante i simboli `U` e `Z`.
2. Inizializza il contenuto del secondo nastro con la codifica dello stato iniziale di T.
3. In base allo stato contenuto nel secondo nastro e il simbolo letto sul terzo nastro, cerca sul primo nastro una transizione che possa essere applicata. Se tale transizione non viene trovata, termina in una configurazione di rigetto.
4. Altrimenti, applica la transizione modificando il contenuto del terzo nastro e aggiornando il secondo nastro in base al nuovo stato di T.
5. Se il nuovo stato è uno stato finale di T, termina nell'unico stato finale di U. Altrimenti, torna al Passo 3.

La descrizione formale di U sarà data ricorrendo alla programmazione procedurale, ovvero all'uso di sotto-macchine: in particolare, definiremo una sotto-macchina per ciascuno dei cinque passi sopra descritti.

2.3.1 Il primo passo della macchina universale

La sotto-macchina che realizza il primo passo (ovvero la copia sul terzo nastro dell'input x codificato mediante i simboli `U` e `Z`) è definita dal grafo delle transizioni mostrato nella Figura 2.2 e opera nel modo seguente.

1. Posiziona la testina sul primo simbolo alla destra del simbolo `;`, il quale viene cancellato.

Figura 2.2: il primo passo della macchina di Turing universale.

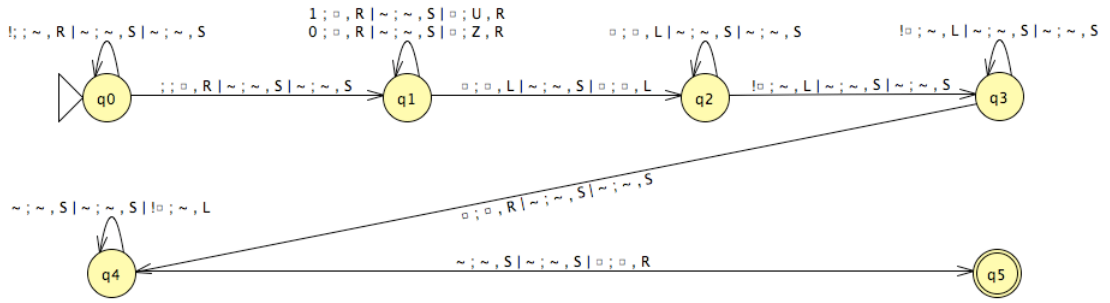
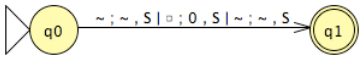


Figura 2.3: il secondo passo della macchina di Turing universale.



2. Scorre il primo nastro verso destra e, per ogni simbolo diverso da □, lo copia sul terzo nastro (spostando la testina di questo nastro a destra) e lo cancella.

3. Posiziona la testina del primo nastro e quella del terzo nastro sul simbolo diverso da □ più a sinistra.

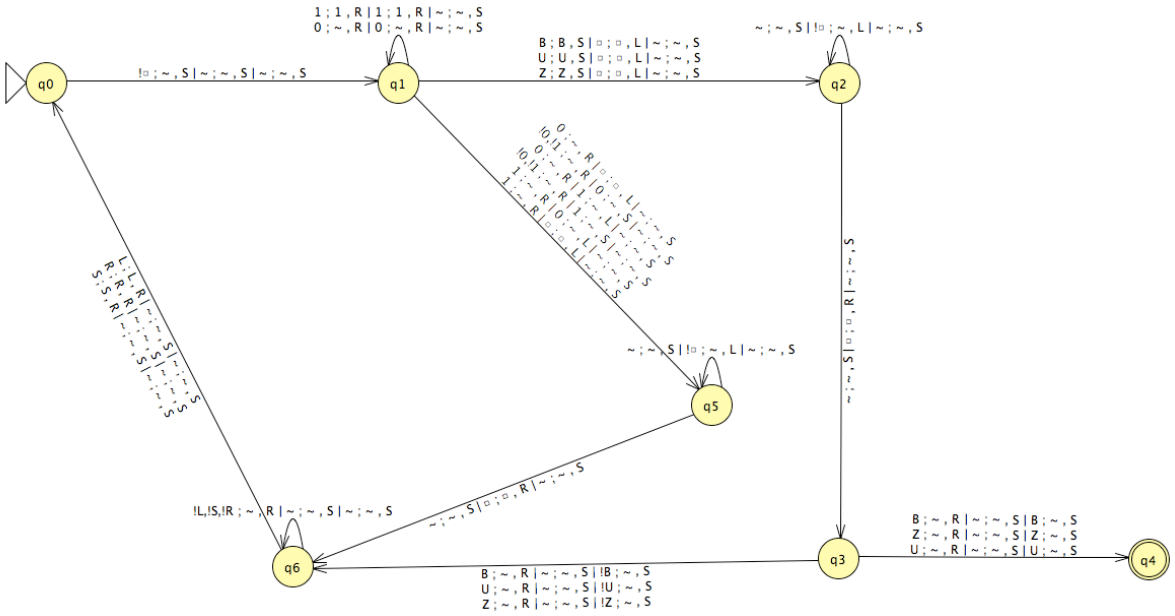
2.3.2 Il secondo passo della macchina universale

La sotto-macchina che esegue il secondo passo della macchina di Turing universale, ovvero che inizializza il contenuto del secondo nastro con la codifica dello stato iniziale, deve semplicemente scrivere su tale nastro il simbolo `0` e posizionare la testina su di esso: ricordiamo infatti che, in base alle assunzioni fatte nel paragrafo precedente, lo stato iniziale di una qualunque macchina di Turing è quello di indice 0. Il grafo delle transizioni di tale sotto-macchina è mostrato nella Figura 2.3.

2.3.3 Il terzo passo della macchina universale

La ricerca della transizione che deve essere eventualmente applicata è probabilmente il passo più complicato da realizzare della macchina di Turing universale e può esse-

Figura 2.4: il terzo passo della macchina di Turing universale.



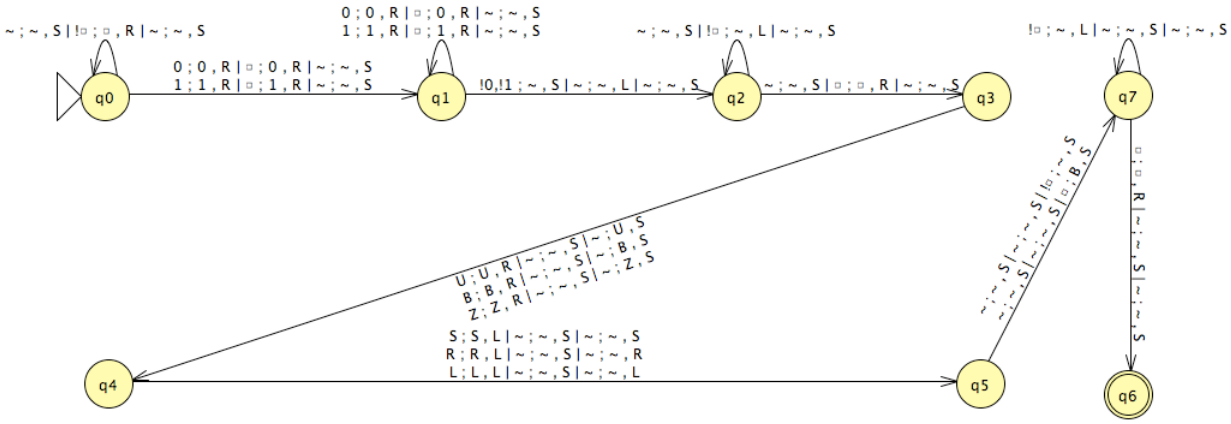
re effettuata risolvendo, anzitutto, un problema di *pattern matching* tra il contenuto del primo nastro e quello del secondo nastro, ovvero cercando sul primo nastro la prima occorrenza del contenuto del secondo nastro. Una volta trovata tale occorrenza la ricerca prosegue verificando se il simbolo immediatamente successivo è uguale al simbolo attualmente scandito sul terzo nastro: in tal caso, la sotto-macchina può terminare posizionando la testina sulla seconda parte della transizione appena identificata. Altrimenti, deve proseguire cercando sul primo nastro un'altra occorrenza del contenuto del secondo nastro.

In particolare, la sotto-macchina che realizza il terzo passo della macchina di Turing universale è definita dal grafo delle transizioni mostrato nella Figura 2.4 e opera nel modo seguente.

1. Scorre il primo e il secondo nastro verso destra fintanto che trova simboli uguali (osserviamo che il secondo nastro include solo simboli `0` e `1`).

2. Se non incontra un simbolo diverso da `0` e `1` su entrambi i nastri, allora posiziona nuovamente la testina del secondo nastro sul primo simbolo a sinistra

Figura 2.5: il quarto passo della macchina di Turing universale.



diverso da □, posiziona la testina del primo nastro sul primo simbolo della transizione successiva a quella attualmente esaminata (ovvero, sul primo simbolo successivo a un simbolo incluso in $\{\texttt{L}, \texttt{R}, \texttt{S}\}$) e torna al passo precedente.

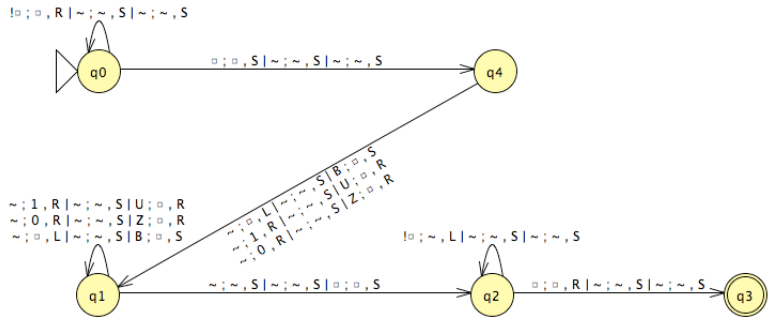
3. Se incontra un simbolo diverso da `0` e `1` su entrambi i nastri, allora posiziona nuovamente la testina del secondo nastro sul primo simbolo a sinistra diverso da □ (preparandosi, così, a un'eventuale successiva ricerca) e verifica se il simbolo letto sul primo nastro è uguale a quello letto sul terzo nastro. In tal caso, sposta la testina del primo nastro di una posizione a destra e termina. Altrimenti, posiziona la testina del primo nastro sul primo simbolo della transizione successiva a quella attualmente esaminata e torna al primo passo.

2.3.4 Il quarto passo della macchina universale

Trovata la transizione da applicare, l'effettiva esecuzione della transizione stessa viene realizzata mediante una semplice sotto-macchina, il cui grafo delle transizioni è mostrato nella Figura 2.5 e che opera nel modo seguente.

1. Cancella l'intero contenuto del secondo nastro e copia su di esso la sequenza di simboli `0` e `1` contenuta sul primo nastro e il cui primo simbolo è attualmente scandito (al termine della copia la testina del primo nastro si troverà sul simbolo immediatamente a destra della sequenza binaria). Quindi, posiziona la testina del secondo nastro sul primo simbolo a sinistra diverso da □.

Figura 2.6: il quinto passo della macchina di Turing universale.



2. Sostituisce il simbolo attualmente scandito sul terzo nastro con quello attualmente scandito sul primo nastro e sposta la testina di quest'ultimo nastro di una posizione a destra.

3. In base al simbolo letto sul primo nastro, sposta la testina del terzo nastro: se il nuovo simbolo letto su tale nastro è □, allora lo sostituisce con il simbolo `B`.

4. Posiziona la testina del primo nastro sul primo simbolo a sinistra diverso da □ (preparandosi, così, a un'eventuale nuova ricerca di una transizione) e termina.

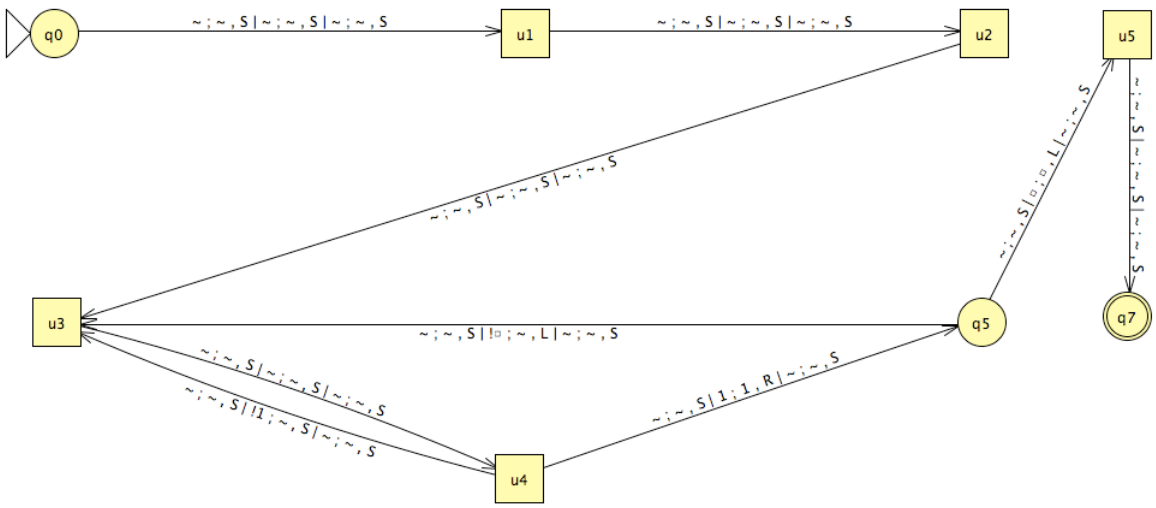
2.3.5 Il quinto passo della macchina universale

Terminata l'esecuzione della transizione, la macchina di Turing universale deve verificare se lo stato attuale della macchina di Turing simulata sia quello finale: in base alle assunzioni fatte nel paragrafo precedente, a tale scopo è sufficiente verificare se il contenuto attuale del secondo nastro è uguale alla stringa `1`. Se così non è allora la macchina di Turing universale torna a eseguire il terzo passo, altrimenti cancella il contenuto del primo nastro (anche se ciò non sarebbe strettamente necessario) e vi copia l'output della macchina simulata, decodificato facendo uso di simboli `0` e `1`. In particolare, quest'ultima operazione viene eseguita dalla sotto-macchina il cui grafo delle transizioni è mostrato nella Figura 2.6.

2.3.6 La macchina universale

In conclusione, facendo riferimento a tutte le sotto-macchine sopra definite, possiamo ora definire formalmente la macchina di Turing universale `U` mediante il grafo delle

Figura 2.7: la macchina di Turing universale.



transizioni mostrato nella Figura 2.7. Osserviamo che, sebbene tale macchina faccia uso di tre nastri e di un alfabeto di lavoro contenente dieci simboli (ovvero □, `0`, `1`, `B`, `U`, `Z`, `L`, `R`, `S` e `;`), in base a quanto abbiamo mostrato nel capitolo precedente e nel primo paragrafo di questo capitolo, essa può essere trasformata in una macchina di Turing con un solo nastro e con un alfabeto di lavoro contenente i tre soli simboli □, `0` e `1` (ovviamente tale trasformazione richiede che la codifica delle macchina di Turing mostrata nel paragrafo precedente sia pre-elaborata in modo da usare solo simboli binari). Ciò implica che la macchina di Turing universale $\mathtt{U}$ può simulare se stessa con input la codifica di un'altra macchina di Turing $\mathtt{T}$ e una sequenza di simboli binari x: il risultato di tale simulazione sarà uguale all'esecuzione di $\mathtt{T}$ con input x.

Osserviamo, infine, che ogni passo della macchina di Turing $\mathtt{T}$ simulata da quella universale richiede da parte di quest'ultima un numero di passi che dipende esclusivamente dalla lunghezza della codifica di $\mathtt{T}$: in altre parole, se fissiamo la macchina di Turing $\mathtt{T}$ da simulare, il numero di passi eseguiti da $\mathtt{U}$ è proporzionale a quello dei passi eseguiti da $\mathtt{T}$. In generale, però, per ogni stringa x di lunghezza n, il primo, il quarto e il quinto passo di $\mathtt{U}$ richiedono un numero di passi lineare in n, mentre il secondo passo ne richiede un numero costante. Il terzo passo, invece, richiede un numero di passi quadratico in n, in quanto, dopo ogni fallita ricerca della transizione

da eseguire, la macchina deve "riavvolgere" il secondo nastro. Pertanto, possiamo concludere che $t_U(n) \in O(n^2)$.

Esercizi

Esercizio 2.1. Dato l'alfabeto $\Sigma = \{0, 1, B, U, Z, L, R, S\}$, si definisca il codice binario c_Σ a esso corrispondente.

Esercizio 2.2. Dato $k \geqslant 3$ e dato un alfabeto Σ di n simboli, in modo simile a quanto fatto nel primo paragrafo di questo capitolo si definisca il codice c_Σ^k che utilizza k simboli e, per ogni stringa x su Σ di lunghezza m, si confronti la lunghezza di $c_\Sigma(x)$ con quella di $c_\Sigma^k(x)$.

Esercizio 2.3. Data una macchina di Turing T definita mediante il suo grafo delle transizioni, si calcoli il numero di diverse possibili codifiche di T.

Esercizio 2.4. Si scriva la codifica delle macchine di Turing viste nel secondo paragrafo del primo capitolo.

Esercizio 2.5. Una macchina di Turing multi-testina è simile a una macchina di Turing ordinaria a eccezione del fatto che il nastro è scandito da un numero limitato di testine, che si muovono indipendentemente l'una dall'altra. A ogni transizione è associata un'etichetta formata da una lista di triple

$$((\sigma_1, \ldots, \sigma_k), (\tau_1, \ldots, \tau_k), (m_1, \ldots, m_k))$$

I simboli $\sigma_1, \ldots, \sigma_k$ e $\tau_1, \ldots, \tau_k$ che appaiono all'interno di una tripla dell'etichetta di una transizione indicano, rispettivamente, i k simboli attualmente letti dalle k testine e i k simboli da scrivere (assumendo che se due testine devono scrivere due simboli diversi nella stessa cella, la priorità viene assegnata alla testina con indice maggiore), mentre i valori $m_1, \ldots, m_k$ specificano i movimenti che devono essere effettuati dalle k testine. Dimostrare che il modello delle macchine di Turing con k testine, dove k è un qualunque numero intero con $k \geqslant 2$, è equivalente a quello delle macchine di Turing ordinarie.

Capitolo 3

Limiti delle macchine di Turing

SOMMARIO

In questo capitolo mostriamo l'esistenza di funzioni che non possono essere calcolate da alcuna macchina di Turing. In particolare, dopo aver introdotto il concetto di funzione calcolabile e di linguaggio decidibile, mostriamo anzitutto (in modo non costruttivo), mediante la tecnica di diagonalizzazione, che devono esistere linguaggi non decidibili. Quindi, diamo alcuni esempi naturali di tali linguaggi (come, ad esempio, il linguaggio della terminazione), facendo anche uso della tecnica di riduzione. Successivamente, dimostriamo il teorema della ricorsione che, informalmente, afferma che esistono macchine di Turing in grado di produrre la loro stessa descrizione. Infine, concludiamo il capitolo applicando il teorema della ricorsione per fornire una dimostrazione alternativa della non decibilità del linguaggio della terminazione e per dimostrare il teorema del punto fisso e quello di Rice, il quale afferma fondamentalmente che nessuna proprietà non banale delle macchine di Turing può essere decisa in modo automatico.

3.1 Funzioni calcolabili e linguaggi decidibili

ABBIAMO VISTO nel primo capitolo come le configurazioni siano uno strumento utile per rappresentare in modo sintetico l'esecuzione di una macchina di Turing con input una stringa specificata e per poter definire il concetto di sotto-macchina. Il motivo principale per cui le abbiamo introdotte, tuttavia, è quello di poter definire formalmente il concetto di funzione calcolabile da una macchina di Turing e quello di linguaggio decidibile. A tale scopo, ricordiamo anzitutto che la configurazione iniziale di una macchina di Turing con input una stringa x sull'alfabeto Σ non contenente $\square$, è definita come ^{q_0}x dove q_0 denota lo stato iniziale della macchina. Tale definizione corrisponde a quanto detto nel primo capitolo, in cui abbiamo assunto che, all'inizio della sua computazione, una macchina di Turing abbia la testina

Relazioni e funzioni

Dato un insieme X, una **relazione** r su X è un sottoinsieme di $X \times X$. Se, per ogni $x \in X$, $(x,x) \in r$, allora la relazione è detta essere **riflessiva**. Se, per ogni x e y in X, $(x,y) \in r$ se e solo se $(y,x) \in r$, allora la relazione è detta essere **simmetrica**. Infine, se, per ogni x, y e z in X, $(x,y) \in r$ e $(y,z) \in r$ implicano che $(x,z) \in r$, allora la relazione è detta essere **transitiva**. Dati due insiemi X e Y, una **funzione** $f : X \to Y$ è un sottoinsieme di $X \times Y$ tale che, per ogni $x \in X$, esiste al più un $y \in Y$ per cui $(x,y) \in f$. Se $(x,y) \in f$, allora scriveremo $f(x) = y$ e diremo che f è *definita* per x e che y è l'**immagine** di x. Se una funzione f è definita per ogni $x \in X$, allora f è detta essere **totale**. Se, per ogni $y \in Y$, esiste un $x \in X$ per cui $f(x) = y$, allora la funzione è detta essere **suriettiva**. Se, per ogni x_1 e x_2 in X per cui f è definita, si ha che $x_1 \neq x_2$ implica $f(x_1) \neq f(x_2)$, allora la funzione è detta essere **iniettiva**. Una funzione totale, iniettiva e suriettiva è detta essere **biettiva**.

posizionata sul primo simbolo della stringa di input e si trovi nel suo (unico) stato iniziale.

Definizione 3.1: funzione calcolabile

Dati due alfabeti Σ e Γ tali che $\square \notin \Sigma \cup \Gamma$ e data una funzione $f : \Sigma^* \to \Gamma^*$, f è **calcolabile** se esiste una macchina di Turing T tale che, per ogni $x \in \Sigma^*$, esiste una sequenza $c_1, \ldots, c_n$ di configurazioni di T per cui valgono le seguenti affermazioni.

1. c_1 è la configurazione iniziale di T con input x.
2. Per ogni i con $1 \leqslant i < n$, c_i produce c_{i+1}.
3. c_n è una configurazione finale $y\,q\,f(x)\square z$, dove q è uno stato finale di T e y e z sono due stringhe (eventualmente vuote) sull'alfabeto di lavoro di T.

Abbiamo già visto nel primo capitolo diversi esempi di funzioni calcolabili, ovvero la funzione che complementa ogni bit di una stringa binaria o quella che ne calcola il complemento a due, la funzione di somma di due numeri interi rappresentati in binario o quella che ordina una stringa binaria.

Osserviamo che l'esecuzione di una macchina di Turing con input una stringa x può portare a tre possibili risultati. Il primo caso si ha quando la macchina termina in una configurazione finale $y\,q\,w\square z$, con $w \in \Gamma^*$: in tal caso, la macchina ha calcolato il valore w a partire dal valore x. Il secondo caso si ha quando la macchina non termina, ovvero la CPU non entra mai in uno stato finale. Il terzo e ultimo caso si ha quando la macchina termina in una configurazione non finale $y\,q\,w\square z$ in quanto non esistono transizioni da applicare: in questo caso, pur terminando l'esecuzione, la macchina non ha calcolato alcun valore. Osserviamo che tra questi due ultimi casi, il secondo è preferibile al primo, in quanto, come vedremo più avanti, decidere se

una macchina terminerà la sua esecuzione risulta essere un compito particolarmente difficile: per questo motivo, preferiremo sempre avere a che fare con macchine di Turing che terminano la propria esecuzione per qualunque stringa di input (anche senza calcolare alcun valore).

3.1.1 Linguaggi decidibili

Avendo definito il concetto di funzione calcolabile, siamo ora in grado di introdurre quello di linguaggio decidibile. A tale scopo, associamo a ogni linguaggio una funzione, la cui calcolabilità determina la decidibilità del linguaggio corrispondente. Dato un linguaggio L su un alfabeto Σ, la **funzione caratteristica** $\chi_L : \Sigma^* \to \{0,1\}$ di L è definita nel modo seguente. Per ogni $x \in \Sigma^*$,

$$\chi_L(x) = \begin{cases} 1 & \text{se } x \in L, \\ 0 & \text{altrimenti} \end{cases}$$

Definizione 3.2: linguaggio decidibile

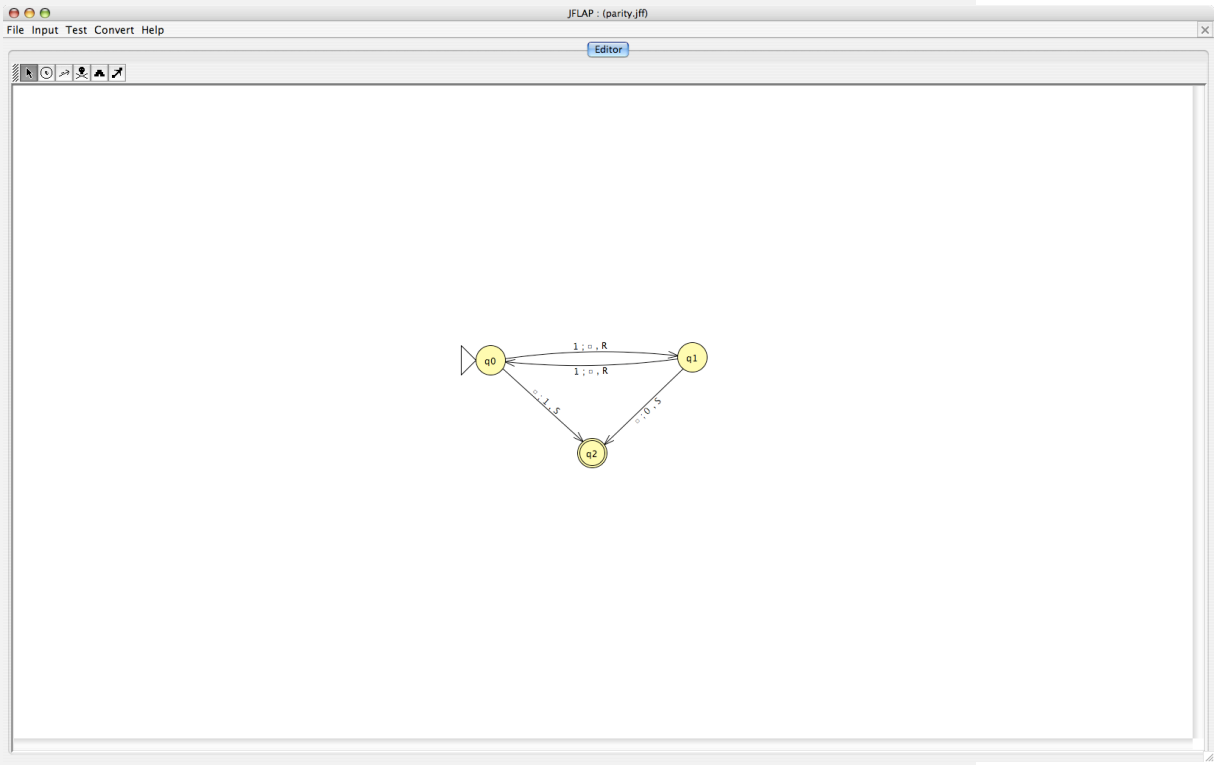
Un linguaggio L è **decidibile** se la sua funzione caratteristica è calcolabile.

Un linguaggio L su un alfabeto Σ è, quindi, decidibile se esiste una macchina di Turing T che *decide* L, ovvero per cui valgono le seguenti due affermazioni.

- Per ogni $x \in L$, T con input x termina nella configurazione finale $y\,q\,1\,\square w$, dove q è uno stato finale di T e y e w sono due stringhe (eventualmente vuote) sull'alfabeto di lavoro di T.
- Per ogni $x \notin L$, T con input x termina nella configurazione finale $y\,q\,0\,\square w$, dove q è uno stato finale di T e y e w sono due stringhe (eventualmente vuote) sull'alfabeto di lavoro di T.

Consideriamo, ad esempio, il linguaggio $L_{\text{parity}} = \{1^k : k \text{ è pari}\}$ e dimostriamo che tale linguaggio è decidibile, definendo una macchina di Turing che ne calcoli la funzione caratteristica. Tale macchina scorre la stringa in input verso destra, sostituendo tutti i simboli 1 con il simbolo $\square$ e alternandosi tra due stati fino a raggiungere il primo $\square$: il primo stato memorizza il fatto che il numero di simboli 1 letti sinora è pari, mentre il secondo stato memorizza il fatto che tale numero è dispari. Se la macchina si trova nel primo stato al termine della lettura dell'input, allora scrive un simbolo 1 e si ferma, altrimenti scrive un simbolo 0 e si ferma.

macchina di Turing per il linguaggio L_{parity}



Teorema 3.1

Se L è un linguaggio decidibile, allora anche L^c è decidibile.

Dimostrazione. Una macchina di Turing T^c che decide L^c può essere facilmente ottenuta a partire da una macchina di Turing T che decide L, sostituendo il simbolo `0` con il simbolo `1` e viceversa al momento in cui T entra in uno stato finale. ◇

Teorema 3.2

Un linguaggio L su un alfabeto Σ è decidibile se e solo se esiste una macchina di Turing T con un solo stato finale tale che, per ogni $x \in \Sigma^*$, T con input x termina e termina in una configurazione finale se e solo se $x \in L$.

Dimostrazione. Supponiamo che L sia decidibile e, anzitutto, osserviamo che la sua funzione caratteristica è una funzione totale $\chi_L : \Sigma^* \to \{0, 1\}$. Sia S la macchina di

Turing che calcola χ_L: modifichiamo S per ottenere T nel modo seguente. Aggiungiamo all'insieme degli stati finali uno stato $q_{\texttt{end}}$ e facciamo diventare non finali tutti gli stati finali di S. Per ogni stato finale q di S introduciamo una transizione a partire da questo stato che collega q a $q_{\texttt{end}}$ e la cui etichetta contiene la tripla $(\texttt{1}, \square, \texttt{S})$. Poiché χ_L è totale, per ogni $x \in \Sigma^*$, esiste uno stato finale q di S per cui S con input x termina nella configurazione $y\,q\,\texttt{1}\square w$ (se $x \in L$), oppure nella configurazione $y\,q\,\texttt{0}\square w$ (se $x \notin L$), dove y e w sono due stringhe (eventualmente vuote) sull'alfabeto di lavoro di S. Nel primo caso, T termina nello stato $q_{\texttt{end}}$, mentre nel secondo caso termina nello stato q, che non è finale per T, in quanto non sono definite transizioni a partire da q che leggano il simbolo 0. Pertanto, T con input x termina sicuramente e termina in una configurazione finale se e solo se $x \in L$. Viceversa, supponiamo che esista una macchina di Turing T che soddisfa le due condizioni del teorema: modifichiamo T, in modo da ottenere una macchina S che decide L, nel modo seguente. Aggiungiamo agli stati finali di T uno stato $q_{\texttt{end}}$, facciamo diventare non finale l'unico stato finale di T e aggiungiamo un nuovo stato (non finale) $q_{\texttt{0/1}}$. Introduciamo, quindi, una transizione dallo stato finale di T a $q_{\texttt{0/1}}$ che, qualunque sia il simbolo letto, scrive 1 e si sposta a destra. Introduciamo, inoltre, una transizione da $q_{\texttt{0/1}}$ a $q_{\texttt{end}}$ che, qualunque sia il simbolo letto, scriva $\square$ e si sposti a sinistra. Infine, per ogni stato non finale q di T e per ogni simbolo σ per cui non siano definite transizioni a partire da q che leggano il simbolo σ, definiamo una transizione da q a $q_{\texttt{0/1}}$ che, qualunque sia il simbolo letto, scrive 0 e si sposta a destra. Evidentemente, se T termina in una configurazione finale, allora S termina nella configurazione finale $y\,q_{\texttt{end}}\,\texttt{1}\square w$, mentre se T termina in uno stato non finale q, allora S termina nella configurazione finale $y\,q_{\texttt{end}}\,\texttt{0}\square w$ (dove y e w sono due stringhe, eventualmente vuote, sull'alfabeto di lavoro di T): poiché T termina in una configurazione finale se e solo se $x \in L$, allora S calcola la funzione caratteristica di L. ◇

Applicando la dimostrazione del teorema precedente alla macchina di Turing per il linguaggio parità precedentemente descritta, otteniamo la macchina mostrata nella parte sinistra della Figura 3.1. Ovviamente, potremmo definire in modo più diretto una macchina di Turing equivalente, facendo passare lo stato q0 nello stato finale nel momento in cui il simbolo letto sia $\square$ e non definendo alcuna transizione a partire da q1 che legga il simbolo $\square$: tale macchina è mostrata nella parte destra della figura.

3.2 Il metodo della diagonalizzazione

PER MOSTRARE l'esistenza di un linguaggio non decidibile faremo uso di una tecnica introdotta da Cantor alla fine del XIX secolo. Quest'ultimo era interessato a capire come fosse possibile confrontare la cardinalità di due insiemi infiniti. In effetti, mentre è immediato decidere se un insieme finito è più o meno grande di un

Figura 3.1: due diverse macchine di Turing per il linguaggio L_{parity}.

altro insieme finito, non è altrettanto evidente decidere quale tra due insiemi infiniti risulti essere il più grande. Ad esempio, se consideriamo l'insieme $\mathbb{N}$ dei numeri interi e quello Σ^* di tutte le stringhe su un alfabeto Σ, entrambi questi insiemi sono infiniti (e quindi più grandi di un qualunque insieme finito): tuttavia, come possiamo dire se $\mathbb{N}$ è più grande di Σ^* o viceversa? La soluzione proposta da Cantor è basata sulla possibilità di mettere in corrispondenza biunivoca gli elementi di un insieme con quelli di un altro insieme.

Definizione 3.3: insiemi equi-cardinali

Due insiemi A e B sono **equi-cardinali** o, equivalentemente, hanno la *stessa cardinalità* se esiste una funzione totale e biettiva $f : A \to B$.

Esempio 3.1: numeri naturali e numeri naturali positivi

Consideriamo l'insieme $\mathbb{N}$ dei numeri naturali e l'insieme $\mathbb{N}^+$ dei numeri naturali positivi. Sia $\mathbb{N}$ che $\mathbb{N}^+$ sono insiemi infiniti. Inoltre, è ovvio che $\mathbb{N}^+$ è un sotto-insieme di $\mathbb{N}$: questi due insiemi hanno comunque la stessa cardinalità. Possiamo, infatti, definire la funzione totale e biettiva $f : \mathbb{N}^+ \to \mathbb{N}$ nel modo seguente: per ogni numero naturale $n > 0$, $f(n) = n - 1$.

Esempio 3.2: numeri naturali e numeri pari

Consideriamo l'insieme $\mathbb{N}$ dei numeri naturali e l'insieme P dei numeri naturali pari. Sia $\mathbb{N}$ che P sono insiemi infiniti. Inoltre, è ovvio che P è un sotto-insieme di $\mathbb{N}$: questi due insiemi hanno comunque la stessa cardinalità. Possiamo, infatti, definire la funzione totale e biettiva $f : P \to \mathbb{N}$ nel modo seguente: per ogni numero naturale pari n, $f(n) = \frac{n}{2}$.

Definizione 3.4: insiemi numerabili

Un insieme A è detto essere **numerabile** se ha la stessa cardinalità dell'insieme dei numeri naturali $\mathbb{N}$.

Il termine "numerabile" deriva dal fatto che, se un insieme A è numerabile, allora è possibile elencare uno dopo l'altro senza ripetizioni gli elementi di A, ovvero associare a ciascuno degli elementi di A una posizione univoca all'interno di tale elenco. In base a quanto esposto nei due esempi precedenti, possiamo concludere che sia l'insieme dei numeri naturali positivi che quello dei numeri pari sono insiemi numerabili: in entrambi i casi la numerazione dei loro elementi è quella più naturale. I prossimi due esempi mostrano, invece, altri due insiemi numerabili di numeri, la cui numerazione non è altrettanto naturale.

Esempio 3.3: numeri naturali e numeri interi

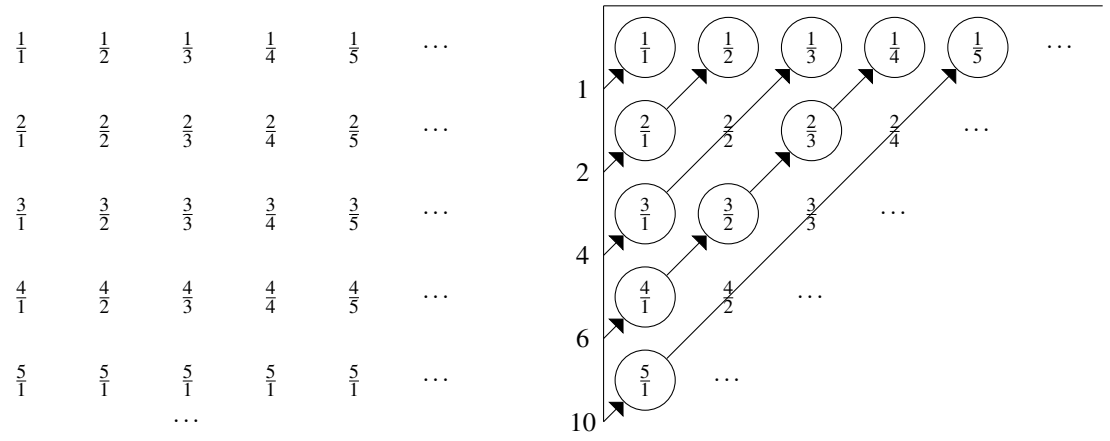
Consideriamo l'insieme $\mathbb{Z}$ dei numeri interi. È ovvio che $\mathbb{N}$ è un sotto-insieme di $\mathbb{Z}$. Possiamo ora mostrare che $\mathbb{Z}$ è numerabile, ovvero che ha la stessa cardinalità di $\mathbb{N}$. La funzione totale e biettiva $f : \mathbb{Z} \to \mathbb{N}$ è definita nel modo seguente: per ogni numero intero n,

$$f(n) = \begin{cases} 0 & \text{se } n = 0, \\ 2n-1 & \text{se } n > 0, \\ -2n & \text{altrimenti} \end{cases}$$

In altre parole, associamo il numero 0 (inteso come numero naturale) allo 0 (inteso come numero intero), i numeri naturali dispari ai numeri interi positivi e i numeri naturali pari ai numeri interi negativi, definendo una numerazione dei numeri interi che inizia nel modo seguente: $0, 1, -1, 2, -2, 3, -3, 4, -4, 5, -5$.

Esempio 3.4: numeri naturali e numeri razionali

Figura 3.2: numerabilità dei numeri razionali positivi



Consideriamo l'insieme $\mathbb{N}^+$ dei numeri naturali positivi e l'insieme $\mathbb{Q}^+$ dei numeri razionali positivi. Sia $\mathbb{N}^+$ che $\mathbb{Q}^+$ sono insiemi infiniti. Inoltre, è ovvio che $\mathbb{N}^+$ è un sotto-insieme di $\mathbb{Q}^+$: questi due insiemi hanno comunque la stessa cardinalità (quindi, $\mathbb{Q}^+$ è un insieme numerabile). Per definire la funzione biettiva $f : \mathbb{Q}^+ \to \mathbb{N}^+$, costruiamo una matrice infinita contenente tutti i numeri razionali positivi con eventuali ripetizioni, come mostrato nella parte sinistra della Figura 3.2: l'elemento in i-esima riga e j-esima colonna è il numero razionale $\frac{i}{j}$. Per ottenere una numerazione degli elementi della matrice, non possiamo ovviamente pensare di numerare prima tutti gli elementi della prima riga, poi quelli della seconda riga e così via, poiché in tal modo non termineremmo mai di numerare gli elementi della prima riga che è infinita. Per ovviare a tale inconveniente, possiamo adottare una strategia simile alla visita in ampiezza di un grafo, procedendo per diagonali di dimensione crescente (si veda la parte destra della figura). La prima di tali diagonali è formata dal solo elemento $\frac{1}{1}$, a cui viene quindi associato il numero naturale positivo 1. La seconda diagonale contiene gli elementi $\frac{2}{1}$ e $\frac{1}{2}$, ai quali vengono quindi associati i numeri naturali positivi 2 e 3. Nel caso della terza diagonale, sorge il problema dovuto al fatto che tale diagonale contiene l'elemento $\frac{2}{2} = \frac{1}{1}$: se associassimo un numero naturale positivo a ciascun elemento della diagonale, avremmo che allo stesso numero razionale positivo sarebbero associati più numeri naturali positivi diversi. Quindi, l'elemento $\frac{2}{2}$ viene saltato nella numerazione e ai due restanti elementi della terza diagonale (ovvero, $\frac{3}{1}$ e $\frac{1}{3}$) vengono associati i numeri naturali positivi 4 e 5. Continuando in questo modo, otteniamo una numerazione di tutti gli elementi di $\mathbb{Q}^+$ e, quindi, la funzione totale e biettiva f.

Dimostrazioni per assurdo

Un modo di dimostrare un enunciato del tipo "`se A è vero, allora B è vero`", consiste nel ragionare per assurdo, assumendo che `B` sia falso e mostrando come la congiunzione di `A` e della negazione di `B` porti a una conseguenza logica notoriamente falsa. Una ben nota dimostrazione per assurdo è quella del seguente enunciato: "se il lato di una quadrato ha lunghezza unitaria, allora la lunghezza d della sua diagonale non è un numero razionale" (equivalentemente, tale enunciato afferma che il numero $\sqrt{2}$ non è un numero razionale). Assumendo, infatti, che d sia un numero razionale (ovvero, *assumendo che `B` sia falso*), abbiamo che $d = \frac{n}{m}$ con n e m relativamente primi (ovvero, il massimo comun divisore di n e m è uguale a 1). Inoltre, avendo il lato del quadrato lunghezza unitaria (*essendo `A` vero*), dal teorema di Pitagora segue che $d^2 = 2$: quindi, $n^2 = 2m^2$, ovvero n^2 è un numero pari. Poichè solo il quadrato di un numero pari può essere pari, abbiamo che n è pari e, quindi, che n^2 è un multiplo di 4. Inoltre, poiché n e m sono relativamente primi, abbiamo che m e, quindi, m^2 sono dispari: ciò implica l'esistenza di un numero dispari il cui doppio è un multiplo di 4 e quest'affermazione è notoriamente falsa.

Una domanda naturale che possiamo porci, a questo punto, è se esistano insiemi infiniti di numeri che non siano numerabili, ovvero che non possano essere elencati in modo univoco senza ripetizioni. Il primo candidato a godere di tale proprietà che può venire in mente, è l'insieme R dei numeri reali: intuitivamente, tale insieme sembra essere molto più grande dell'insieme dei numeri interi e di quello dei numeri razionali, in quanto contiene numeri che non possono essere rappresentati con un numero finito di cifre. Il prossimo teorema conferma che tale intuizione è in effetti giusta: la dimostrazione del teorema ci consentirà, tra l'altro, di introdurre il **metodo della diagonalizzazione**.

Teorema 3.3

L'insieme R dei numeri reali non è numerabile.

Dimostrazione. La dimostrazione procede per assurdo, assumendo che R sia numerabile, ovvero che esista una funzione totale e biettiva $f : R \to N^+$. Possiamo quindi numerare tutti i numeri reali (senza ripetizioni) e costruire una matrice M di cifre decimali tale che la cifra alla riga i e alla colonna j sia la j-esima cifra della parte decimale dell'i-esimo numero reale (ovvero del numero reale x tale $f(x) = i$). Ad esempio, supponiamo che la tabella mostrata nella parte sinistra della Figura 3.3 sia l'inizio di un'ipotetica numerazione dei numeri reali: allora, la parte iniziale della matrice M sarebbe quella mostrata nella parte destra della figura. Definiamo ora un numero reale $x_{\text{diag}} < 1$ nel modo seguente: per ogni numero naturale positivo i, l'i-esima cifra della parte decimale di x_{diag} è uguale a $[(M[i,i]+1) \bmod 10]$. In altre parole, l'i-esima cifra della parte decimale di x_{diag} è posta uguale all'i-esima cifra del numero reale corrispondente a i incrementata di 1 (modulo 10). Ad esempio, il numero x_{diag} corrispondente all'ipotetica numerazione mostrata in precedenza inizierà

Figura 3.3: metodo della diagonalizzazione

i	x
1	7.4875690...
2	33.3333333 ...
3	0.8362719 ...
4	65.4971652 ...
5	0.3384615 ...
6	0.3442623 ...
⋮	⋮

4	8	7	5	6	9	0	...
3	**3**	3	3	3	3	3	...
8	3	**6**	2	7	1	9	...
4	9	7	**1**	6	5	2	...
3	3	8	4	**6**	1	5	...
3	4	4	2	6	**2**	3	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

nel modo seguente: $0.547273\ldots$. Chiaramente, per ogni numero naturale positivo $\mathtt{i}$, $\mathtt{i}$ non è l'immagine di $\mathtt{x}_{\text{diag}}$ in quanto $\mathtt{x}_{\text{diag}}$ differisce dal numero reale la cui immagine è $\mathtt{i}$ per almeno una cifra decimale (ovvero, l'$\mathtt{i}$-esima):[1] quindi, a $\mathtt{x}_{\text{diag}}$ non corrisponde alcun numero naturale positivo, contraddicendo l'ipotesi (fatta per assurdo) che $\mathtt{f}$ sia una funzione totale e biettiva da $\mathtt{R}$ in $\mathtt{N}^+$. Pertanto, abbiamo dimostrato che non può esistere una tale funzione e che $\mathtt{R}$ non è numerabile. ◇

Dalla dimostrazione del teorema precedente, dovrebbe risultare chiaro il motivo per cui la tecnica utilizzata sia comunemente chiamata "diagonalizzazione". In generale, se $\mathtt{A}$ e $\mathtt{B}$ sono due insiemi numerabili e se $\mathtt{f}_{\mathtt{A},\mathtt{B}} : \mathtt{N} \times \mathtt{N} \to \mathtt{X}$, con $|\mathtt{X}| \geqslant 2$, è una funzione totale che, per ogni coppia $\mathtt{i}$ e $\mathtt{j}$ di numeri naturali, specifica una determinata relazione tra l'$(\mathtt{i}+1)$-esimo elemento di $\mathtt{A}$ e il $(\mathtt{j}+1)$-esimo elemento di $\mathtt{B}$, tale tecnica consiste nel definire un *oggetto diagonale* $\mathtt{d}$ il quale si "rapporta" con almeno un elemento di $\mathtt{B}$ in modo diverso da ciascun altro elemento di $\mathtt{A}$: in particolare, per ogni numero naturale $\mathtt{i}$, $\mathtt{d}$ è quell'oggetto la cui relazione con l'$(\mathtt{i}+1)$-esimo elemento di $\mathtt{B}$ è nell'insieme $\mathtt{X} - \{\mathtt{f}_{\mathtt{A},\mathtt{B}}(\mathtt{i},\mathtt{i})\}$. Osserviamo che se siamo in grado di dimostrare che l'oggetto diagonale appartiene ad $\mathtt{A}$, allora abbiamo ottenuto una contraddizione. Infatti, in quanto membro di $\mathtt{A}$, $\mathtt{d}$ corrisponde in modo univoco a un numero naturale $\mathtt{i}_{\mathtt{d}}$ (poiché $\mathtt{A}$ è numerabile): tuttavia, la relazione tra $\mathtt{d}$ e l'$(\mathtt{i}_{\mathtt{d}}+1)$-esimo elemento di $\mathtt{B}$ è per definizione diversa da $\mathtt{f}_{\mathtt{A},\mathtt{B}}(\mathtt{i}_{\mathtt{d}},\mathtt{i}_{\mathtt{d}})$ (generalmente, tale contraddizione è dovuta all'assunzione che l'insieme $\mathtt{A}$ sia numerabile). Nel caso della dimostrazione del teorema precedente, $\mathtt{A}$ è l'insieme dei numeri reali, $\mathtt{B}$ è l'insieme dei numeri naturali, $\mathtt{X} = \{0,1,\ldots,9\}$ e, assumendo che $\mathtt{A}$ sia numerabile, $\mathtt{f}_{\mathtt{A},\mathtt{B}}(\mathtt{i},\mathtt{j})$ determina la $(\mathtt{j}+1)$-esima cifra decimale dell'$(\mathtt{i}+1)$-esimo numero reale, per ogni coppia di

[1] In realtà, è possibile che due numeri reali siano uguali pur avendo una diversa rappresentazione decimale (si pensi, ad esempio, a $0.49999999\ldots$ e a $0.5000000\ldots$): possiamo comunque risolvere questo problema evitando di scegliere le cifre 0 e 9 quando costruiamo $\mathtt{x}_{\text{diag}}$.

numeri naturali i e j: l'oggetto diagonale x_{diag} è un numero reale che si distingue per almeno una cifra decimale da ogni elemento di A (da cui la contraddizione e l'impossibilità di numerare l'insieme dei numeri reali).

3.2.1 Linguaggi non decidibili

Utilizzando il metodo della diagonalizzazione possiamo mostrare l'esistenza di linguaggi non decidibili. A tale scopo, mostreremo anzitutto che l'insieme di tutte le macchine di Turing è numerabile: poiché una macchina di Turing può decidere al più un linguaggio, ciò implica che l'insieme di tutti i linguaggi decidibili non è più che numerabile. D'altra parte, mostreremo come esista un insieme numerabile di linguaggi decidibili: quindi, l'insieme di tutti i linguaggi decidibili è numerabile. Infine, dimostreremo che l'insieme di tutti i linguaggi sull'alfabeto binario non è numerabile: ciò implica che deve esistere un linguaggio che non è decidibile. Nel seguito di questo capitolo, ci limiteremo a considerare l'insieme $\mathcal{L}$ di tutti i linguaggi sull'alfabeto binario e l'insieme $\mathcal{T}$ di tutte le macchine di Turing il cui alfabeto di lavoro è $\{\square, 0, 1\}$ (in base a quanto detto nel capitolo precedente, questa restrizione non causa alcuna perdita di generalità).

Lemma 3.1

L'insieme di tutti i linguaggi decidibili appartenenti a $\mathcal{L}$ è numerabile.

Dimostrazione. Sia $\mathcal{D}$ l'insieme di tutti i linguaggi decidibili appartenenti a $\mathcal{L}$. La dimostrazione consiste nel far vedere che $\mathcal{T}$ è numerabile (il che implica che $\mathcal{D}$ non è più che numerabile) e che esiste un insieme numerabile di linguaggi decidibili contenuto in $\mathcal{L}$ (il che implica che $\mathcal{D}$ è almeno numerabile). Per quanto riguarda la prima affermazione, abbiamo già visto nel precedente capitolo come sia possibile associare in modo univoco, a ogni macchina di Turing in $\mathcal{T}$, una stringa sull'alfabeto $\{0,1,B,U,Z,L,R,S\}$. D'altra parte, per ogni alfabeto, l'insieme di tutte le stringhe su di esso è numerabile: è sufficiente, infatti, elencare le stringhe in base all'ordinamento lessicografico e associare il numero i all'$(i+1)$-esima stringa in tale elenco. Poiché un sottoinsieme di un insieme numerabile è anch'esso numerabile, abbiamo che l'insieme di tutte le codifiche di una macchina di Turing in $\mathcal{T}$ è numerabile: pertanto, $\mathcal{T}$ è numerabile e, quindi, $\mathcal{D}$ non è più che numerabile. Rimane ora da dimostrare che tale insieme è almeno numerabile, mostrando l'esistenza di un suo sottoinsieme numerabile. Tale sottoinsieme $\mathcal{B}$ è costituito dai seguenti linguaggi contenuti in $\mathcal{L}$: per ogni $i \geqslant 0$, L_i contiene solo la rappresentazione binaria di i. È facile definire, per ogni $i \geqslant 0$, una macchina di Turing in $\mathcal{T}$ che decide L_i: inoltre, $\mathcal{B}$ è chiaramente numerabile. Quindi, l'insieme $\mathcal{D}$ include un sottoinsieme numerabile e, pertanto, è anch'esso almeno numerabile: il lemma risulta dunque essere dimostrato. $\diamond$

Lemma 3.2

$\mathcal{L}$ non è numerabile.

Dimostrazione. Supponiamo, per assurdo, che $\mathcal{L}$ sia numerabile, ovvero che esista un funzione biettiva $f : \mathcal{L} \to \mathbb{N}$: per ogni numero naturale i, indichiamo con L_i il linguaggio la cui immagine è i. Consideriamo poi la numerazione $\sigma_0, \sigma_1, \sigma_2, \ldots$ di tutte le stringhe su Σ ottenuta in base all'ordinamento lessicografico. Ragionando per diagonalizzazione, definiamo un linguaggio L_{diag} nel modo seguente: per ogni numero naturale i, $\sigma_i \in L_{diag}$ se e solo se $\sigma_i \notin L_i$. Chiaramente, per ogni numero naturale i, $L_{diag} \neq L_i$ in quanto L_{diag} differisce da L_i per almeno una stringa (ovvero, σ_i): quindi, a L_{diag} non corrisponde alcun numero naturale, contraddicendo l'ipotesi (fatta per assurdo) che f sia una funzione biettiva da $\mathcal{L}$ in $\mathbb{N}$. Pertanto, abbiamo dimostrato che non può esistere una tale funzione e che $\mathcal{L}$ non è numerabile. ◇

Teorema 3.4

Esiste un linguaggio in $\mathcal{L}$ che non è decidibile.

Dimostrazione. La dimostrazione segue immediatamente dai due lemmi precedenti. ◇

3.3 Il problema della terminazione

LA DIMOSTRAZIONE del Teorema 3.4, pur testimoniando il fatto che esistano linguaggi non decidibili, risulta essere alquanto artificiale e, in qualche modo, di poco interesse pratico. In questo paragrafo mostreremo, sempre mediante il metodo della diagonalizzazione, che esistono linguaggi "interessanti" che non sono decidibili. In particolare, considereremo il problema di decidere se un dato programma con input un dato valore termina la sua esecuzione. Non è necessario, crediamo, evidenziare l'importanza di un tale problema: se fossimo in grado di risolverlo in modo automatico, potremmo, ad esempio, incorporare tale soluzione all'interno dei compilatori e impedire, quindi, la distribuzione di programmi che non abbiano termine. Sfortunatamente, il risultato principale di questo paragrafo afferma che nessun programma è in grado di determinare la terminazione di tutti gli altri programmi.

Nel seguito di questo capitolo, indicheremo con $\mathcal{C}$ l'insieme delle codifiche binarie c_T (secondo quanto visto nel primo paragrafo del capitolo precedente) delle codifiche di una macchina di Turing $T \in \mathcal{T}$ (secondo quanto visto nel secondo paragrafo del capitolo precedente). Inoltre, per ogni stringa binaria x e per ogni $T \in \mathcal{T}$, indicheremo con $T(x)$ la computazione della macchina di Turing T con input x: tale computazione può terminare in una configurazione finale, può terminare in una configurazione non finale oppure può non terminare. Infine, date k stringhe binarie $x_1, \ldots, x_k$, indicheremo con $\langle x_1, \ldots, x_k \rangle$ la stringa binaria $d(x_1)01d(x_2)01\cdots01d(x_k)$ dove, per

ogni i con $1 \leqslant i \leqslant k$, $d(x_i)$ denota la stringa binaria ottenuta a partire da x_i raddoppiando ciascun suo simbolo: ad esempio, $\langle 0110, 1100 \rangle$ denota la stringa binaria 00111100001111100000. Osserviamo che, grazie alla presenza del separatore 01, questo modo di specificare sequenze di stringhe binarie non è ambiguo, nel senso che, data la stringa binaria $\langle x_1, \ldots, x_k \rangle$, è possibile determinare in modo univoco le k stringhe binarie $x_1, \ldots, x_k$.

Il **problema della terminazione** consiste nel decidere il seguente linguaggio.

$$L_{\text{stop}} = \{\langle c_T, x \rangle : c_T \in \mathcal{C} \wedge x \in \{0,1\}^* \wedge T(x) \text{ termina}\}$$

Osserviamo che il linguaggio L_{stop} include due tipi di stringhe: le stringhe $\langle c_T, x \rangle$ per cui T con input x termina in una configurazione finale e quelle per cui T con input x termina in una configurazione non finale.

Teorema 3.5

L_{stop} non è decidibile.

Dimostrazione. Supponiamo, per assurdo, che L_{stop} sia decidibile: in base al Teorema 3.2, deve esistere una macchina di Turing $T_{\text{stop}} \in \mathcal{T}$ tale che, per ogni stringa binaria y,

$$T_{\text{stop}}(y) \text{ termina in una } \begin{cases} \text{configurazione finale} & \text{se } y = \langle c_T, x \rangle, c_T \in \mathcal{C} \text{ e } T(x) \text{ termina,} \\ \text{configurazione non finale} & \text{altrimenti} \end{cases}$$

Definiamo, ora, una nuova macchina di Turing $T_{\text{diag}} \in \mathcal{T}$ che usa T_{stop} come sottomacchina nel modo seguente: per ogni stringa binaria z,

$$T_{\text{diag}}(z) \begin{cases} \text{termina} & \text{se } z = c_T \in \mathcal{C} \text{ e } T_{\text{stop}}(\langle c_T, c_T \rangle) \\ & \text{termina in una configurazione non finale} \\ \text{non termina} & \text{altrimenti} \end{cases}$$

(è ovviamente facile forzare una macchina a non terminare). Consideriamo il comportamento di T_{diag} con input la sua codifica, ovvero $c_{T_{\text{diag}}}$. Dalla definizione di T_{stop} e di T_{diag}, abbiamo che $T_{\text{diag}}(c_{T_{\text{diag}}})$ termina se e solo se $T_{\text{stop}}(\langle c_{\text{diag}}, c_{\text{diag}} \rangle)$ termina in una configurazione non finale se e solo se $T_{\text{diag}}(c_{T_{\text{diag}}})$ non termina. Abbiamo, pertanto, generato una contraddizione, dimostrando così che non può esistere la macchina di Turing T_{stop} e che, quindi, L_{stop} non è decidibile. ◇

Anche se può non sembrare del tutto evidente, la dimostrazione del teorema precedente fa uso della tecnica di diagonalizzazione così come l'abbiamo descritta nel paragrafo precedente. In questo caso, l'insieme A e l'insieme B coincidono entrambi con $\mathcal{C}$, l'insieme X contiene i due valori `true` e `false` e la funzione $f_{A,B}$ è definita

nel modo seguente: per ogni coppia di numeri naturali i e j, $f_{A,B}(i,j) = \texttt{true}$ se e solo se la macchina di Turing corrispondente all'$(i+1)$-esima codifica in $\mathcal{C}$ con input la $(j+1)$-esima codifica in $\mathcal{C}$ termina. La codifica binaria della macchina di Turing T_{diag} della dimostrazione del teorema precedente corrisponde all'oggetto diagonale, il quale appartenendo a $\mathcal{C}$ genera una contraddizione.

3.3.1 Linguaggi semi-decidibili

Il Teorema 3.5 mostra che esistono linguaggi interessanti da un punto di vista applicativo, che non possono essere decisi da alcuna macchina di Turing. La dimostrazione di tale risultato usa esplicitamente il fatto che una macchina di Turing che decide un linguaggio debba terminare per ogni input. Ci possiamo quindi porre, in modo naturale, la seguente domanda: se rilassiamo il vincolo della terminazione per ogni input, è ancora vero che esistono problemi interessanti che non possono essere risolti da una macchina di Turing? Per rispondere a tale domanda, introduciamo anzitutto il concetto di semi-decidibilità.

Definizione 3.5: linguaggi semi-decidibili

Un linguaggio $L \in \mathcal{L}$ è detto essere **semi-decidibile** se esiste una macchina di Turing $T \in \mathcal{T}$ tale che, per ogni stringa binaria x, se $x \in L$, allora $T(x)$ termina in una configurazione finale, altrimenti $T(x)$ non termina.

Esempio 3.5: problema della terminazione

La semi-decidibilità di L_{stop} deriva facilmente dall'esistenza della macchina di Turing universale U. Infatti, possiamo definire una macchina T_{stop} che, per ogni stringa binaria y, verifica anzitutto se $y = \langle c_T, x \rangle$ con $c_T \in \mathcal{C}$: in caso ciò non sia vero, T_{stop} non termina. Altrimenti, T_{stop} esegue U con input c_T e x: se tale computazione termina, allora T_{stop} termina in una configurazione finale. Pertanto, se $y \in L_{stop}$, allora $T_{stop}(y)$ termina in una configurazione finale, altrimenti non termina: ciò dimostra che L_{stop} è semi-decidibile.

Se un linguaggio $L \in \mathcal{L}$ è decidibile, allora L è semi-decidibile: in effetti, è sufficiente modificare la macchina T che decide L in modo che, ogni qualvolta T termina con la testina posizionata sul simbolo 0, la macchina modificata entri in uno stato non finale in cui rimanga poi indefinitamente. Inoltre, essendo anche L^c decidibile (si veda il Teorema 3.1), abbiamo che L^c è semi-decidibile. Il prossimo risultato afferma che la semi-decidibilità di L e di L^c è in realtà una condizione non solo necessaria ma anche sufficiente per la decidibilità di L.

Lemma 3.3

Un linguaggio $L \in \mathcal{L}$ è decidibile se e solo se L e L^c sono entrambi semi-decidibili.

Dimostrazione. Abbiamo già visto che se L è decidibile, allora L e L^c sono entrambi semi-decidibili. Per dimostrare il viceversa, supponiamo che esistano due macchine di Turing $T, T^c \in \mathcal{T}$ tali che, per ogni stringa binaria x, valga una delle seguenti due affermazioni.

- $x \in L$ e $T(x)$ termina in una configurazione finale.
- $x \notin L$ e $T^c(x)$ termina in una configurazione finale.

In altre parole, per ogni input x, $T(x)$ termina oppure $T^c(x)$ termina. Possiamo allora definire una macchina di Turing T' che esegua entrambe le computazioni in parallelo: in particolare, per ogni $i \geqslant 0$, T' esegue i passi di $T(x)$ e i passi di $T^c(x)$ e, se nessuna delle due computazioni termina, passa al valore successivo di i. La macchina T' si ferma nel momento in cui una delle due computazioni termina (cosa che deve necessariamente accadere). Se la prima computazione a fermarsi è $T(x)$, allora T termina in una configurazione finale, altrimenti (ovvero, se la prima computazione a fermarsi è $T^c(x)$) T' termina in una configurazione non finale. In base al Teorema 3.2, T' è una macchina di Turing che testimonia la decidibilità di L e il teorema risulta essere così dimostrato. ◇

Dal fatto che L_{stop} è semi-decidibile, dal teorema precedente e dal Teorema 3.5, segue immediatamente il prossimo risultato, il quale mostra che esistono linguaggi "interessanti" che non sono semi-decidibili.

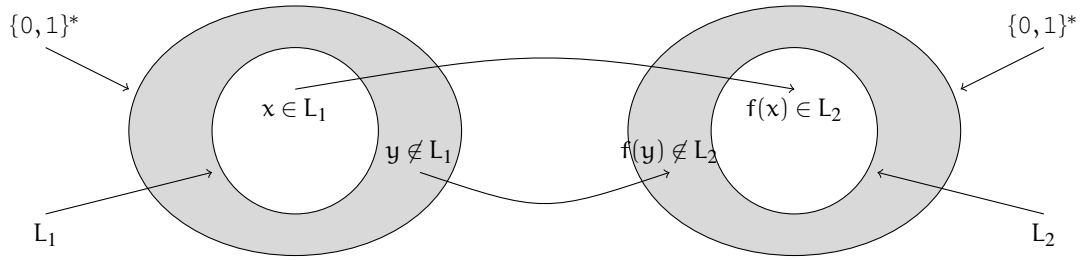
Corollario 3.1

L^c_{stop} non è semi-decidibile.

3.4 Riducibilità tra linguaggi

UNA VOLTA dimostrata l'esistenza di un linguaggio non decidibile, possiamo mostrare che molti altri problemi non sono decidibili facendo uso della tecnica di riduzione. Intuitivamente, tale tecnica consiste nel dimostrare che, dati due linguaggi L_1 e L_2, L_1 non è più difficile di L_2 o, più precisamente, che se esiste una macchina di Turing che decide L_2, allora esiste anche una macchina di Turing che decide L_1. Un lettore abituato allo sviluppo di algoritmi per la risoluzione di problemi conoscerà già la tecnica di riduzione, essendo probabilmente questa una tra le più utilizzate in tale

Figura 3.4: riducibilità tra linguaggi



contesto: in questo contesto, tuttavia, faremo principalmente uso della riducibilità per dimostrare risultati negativi, ovvero per dimostrare che determinati linguaggi non sono decidibili o che non lo sono in modo efficiente (ovvero, come vedremo nella terza parte di queste dispense, che non lo sono in tempo polinomiale). A tale scopo, diamo ora una definizione formale del concetto intuitivo, dato in precedenza, di riducibilità.

Definizione 3.6: linguaggi riducibili

Un linguaggio $L_1 \in \mathcal{L}$ è detto essere **riducibile** a un linguaggio $L_2 \in \mathcal{L}$ se esiste una funzione totale calcolabile $f : \{0,1\}^* \to \{0,1\}^*$, detta **riduzione**, tale che, per ogni stringa binaria x, $x \in L_1$ se e solo se $f(x) \in L_2$.

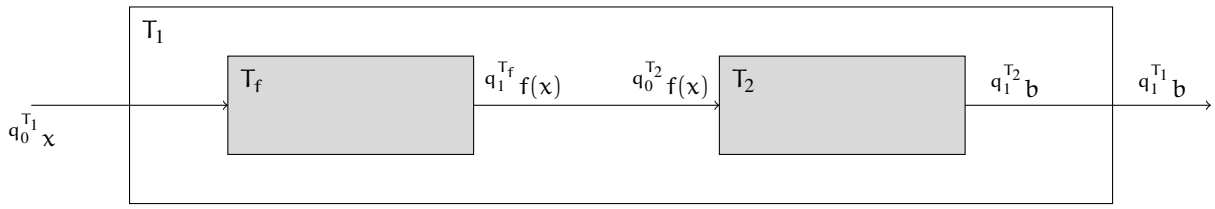
In base alla definizione precedente, una riduzione tra un linguaggio L_1 e un linguaggio L_2 deve essere definita per ogni stringa binaria e deve trasformare stringhe appartenenti a L_1 in stringhe appartenenti a L_2 e stringhe non appartenenti a L_1 in stringhe non appartenenti a L_2 (si veda la Figura 3.4). Osserviamo che la riduzione non deve necessariamente essere suriettiva, ovvero che possono esistere stringhe binarie che non sono immagine di alcuna stringa binaria mediante f. Il risultato seguente formalizza quanto detto in precedenza, ovvero che se un linguaggio L_1 è riducibile a un linguaggio L_2, allora L_1 non è più difficile di L_2 (si veda anche la Figura 3.5).

Lemma 3.4

Siano L_1 e L_2 due linguaggi in $\mathcal{L}$ tali che L_1 è riducibile a L_2. Se L_2 è decidibile, allora L_1 è decidibile.

Dimostrazione. Sia f la funzione totale calcolabile che testimonia il fatto che L_1 è riducibile a L_2 e sia T_f una macchina di Turing che calcola f (senza perdita di generalità, possiamo assumere che, per ogni stringa binaria x, $T_f(x)$ termina con la sola stringa

Figura 3.5: composizione di riduzioni e decisori



$f(x)$ sul nastro e con la testina posizionata sul primo simbolo a sinistra diverso da $\square$). Inoltre, sia T_2 una macchina di Turing che decide L_2. Definiamo allora una macchina di Turing T_1 che decide L_1 nel modo seguente (fondamentalmente, T_1 non è altro che la composizione di T_f con T_2). Per ogni stringa binaria x, T_1 avvia l'esecuzione di T_f con input x: quando T_f termina l'esecuzione lasciando la sola stringa $f(x)$ sul nastro con la testina posizionata sul suo primo simbolo, T_1 avvia l'esecuzione di T_2 con input $f(x)$ (in altre parole, effettua una transizione dallo stato finale di T_f allo stato iniziale di T_2). Per ogni input x, $T_1(x)$ termina con la testina posizionata su un simbolo `1` oppure su un simbolo `0` e termina con la testina posizionata su un simbolo `1` se e solo se $T_2(f(x))$ termina con la testina posizionata su un simbolo `1`. D'altra parte, $T_2(f(x))$ termina con la testina posizionata su un simbolo `1` se e solo se $f(x) \in L_2$ e $f(x) \in L_2$ se e solo se $x \in L_1$. Quindi, per ogni input x, $T_1(x)$ termina con la testina posizionata su un simbolo `1` se solo se $x \in L_1$: ovvero, T_1 decide L_1 e il lemma risulta essere dimostrato. $\diamond$

Come abbiamo già detto, in queste dispense faremo principalmente un uso negativo del teorema precedente allo scopo di dimostrare che determinati linguaggi non sono decidibili: in particolare, utilizzeremo la seguente immediata conseguenza del teorema stesso.

Corollario 3.2

Siano L_1 e L_2 due linguaggi in $\mathcal{L}$ tali che L_1 è riducibile a L_2. Se L_1 non è decidibile, allora L_2 non è decidibile.

La prima applicazione che vedremo del precedente corollario risponde alla domanda che naturalmente ci potremmo porre di sapere se la difficoltà di risoluzione del problema della terminazione risieda nel fatto che non abbiamo posto alcun vincolo sulle stringhe di input: il prossimo esempio mostra come ciò non sia vero (osserviamo che l'esempio può essere facilmente adattato a una qualsiasi stringa binaria diversa da `0`).

Esempio 3.6: problema della terminazione con input fissato

Consideriamo il seguente linguaggio: $L_{stop-0} = \{c_T : c_T \in \mathcal{C} \wedge T(0) \text{ termina}\}$. Dimostriamo ora che L_{stop} è riducibile a L_{stop-0}: dal Teorema 3.5 e dal Corollario 3.2, segue che L_{stop-0} non è decidibile. Data una stringa binaria y, la riduzione f per prima cosa verifica se $y = \langle c_T, x\rangle$ con $c_T \in \mathcal{C}$: in caso contrario (per cui $y \notin L_{stop}$), $f(y)$ produce la codifica di una qualunque macchina di Turing che con input 0 non termina (per cui $f(y) \notin L_{stop-0}$). Altrimenti, $f(\langle c_T, x\rangle)$ produce la codifica $c_{T'}$ di una macchina di Turing T' che, con input una stringa binaria z, per prima cosa cancella z e lo sostituisce con x e, quindi, esegue T con input x. Chiaramente, $\langle c_T, x\rangle \in L_{stop}$ se e solo se $c_{T'} = f(\langle c_T, x\rangle) \in L_{stop-0}$.

Nell'esempio precedente, abbiamo definito il comportamento della riduzione nel caso in cui la stringa y da trasformare non appartenesse a L_{stop} per motivi puramente sintattici (ovvero, $y \neq \langle c_T, x\rangle$ con $c_T \in \mathcal{C}$). In generale, questo aspetto della riduzione non è difficile da gestire nel momento in cui si conosca almeno una stringa binaria w che non appartenga al linguaggio di destinazione: per questo motivo, negli esempi che seguiranno eviteremo di definire esplicitamente il comportamento della riduzione nel caso in cui la stringa binaria da trasformare non sia sintatticamente corretta.

Esempio 3.7: problema dell'accettazione

Consideriamo il seguente linguaggio: $L_{acc} = \{\langle c_T, x\rangle : c_T \in \mathcal{C} \wedge T(x) \text{ termina in una configurazione finale}\}$. Dimostriamo ora che L_{stop} è riducibile a L_{acc}: dal Teorema 3.5 e dal Corollario 3.2, segue che L_{acc} non è decidibile. Date due stringhe binarie $c_T \in \mathcal{C}$ e x, $f(\langle c_T, x\rangle)$ produce la coppia $\langle c_{T'}, x\rangle$ dove $c_{T'}$ è la codifica di una macchina di Turing T' che, con input una stringa binaria z, esegue T con input z: se $T(z)$ termina, allora T' termina in una configurazione finale. Chiaramente, $\langle c_T, x\rangle \in L_{stop}$ se e solo se $T'(x)$ termina in una configurazione finale se e solo se $\langle c_{T'}, x\rangle = f(\langle c_T, x\rangle) \in L_{acc}$.

Esempio 3.8: problema del linguaggio vuoto

Consideriamo il seguente linguaggio: $L_{empty} = \{c_T : c_T \in \mathcal{C} \wedge \forall x \in \{0,1\}^*[T(x) \text{ non termina in una configurazione finale}]\}$. Dimostriamo ora che L_{acc} è riducibile a L^c_{empty}: dall'esempio precedente, dal Corollario 3.2 e dal Teorema 3.1, segue che L^c_{empty} e L_{empty} non sono decidibili. Date due stringhe binarie $c_T \in \mathcal{C}$ e x, $f(\langle c_T, x\rangle)$ produce la codifica $c_{T'}$ di una macchina di Turing T' che, con input una stringa binaria y, per prima cosa cancella y e lo sostituisce con x e, quindi, esegue T con input x. Abbiamo che $\langle c_T, x\rangle \in L_{acc}$ se e solo se $T(x)$ termina in una configurazione finale se e solo se, per ogni stringa binaria y, $T'(y)$ termina in una configurazione finale se e solo se $c_{T'} \in L^c_{empty}$ (in quanto T' si comporta allo stesso modo indipendentemente dalla stringa di input y).

Esempio 3.9: problema dell'equivalenza tra linguaggi

Consideriamo il seguente linguaggio.

$$L_{eq} = \{\langle c_{T_1}, c_{T_2}\rangle \quad : \quad c_{T_1} \in \mathcal{C} \wedge c_{T_2} \in \mathcal{C} \wedge \forall x \in \{0,1\}^* [T_1(x) \text{ termina in una configurazione finale se e solo se } T_2(x) \text{ termina in una configurazione finale}]\}$$

(in altre parole, L_{eq} include tutte le coppie di codifiche di macchine di Turing che decidono lo stesso linguaggio). Riduciamo ora L_{empty} a L_{eq}: dall'esempio precedente e dal Corollario 3.2, segue che L_{eq} non è decidibile. Data una stringa binaria $c_T \in \mathcal{C}$, $f(c_T)$ produce la coppia $\langle c_T, c_R\rangle$ dove c_R è la codifica di una macchina di Turing R che non accetta alcuna stringa. Chiaramente, $c_T \in L_{empty}$ se e solo $\langle c_T, c_R\rangle \in L_{eq}$.

3.5 La tesi di Church-Turing

Sebbene il concetto di algoritmo abbia avuto una lunga storia nel campo della matematica, la sua definizione formale non fu introdotta prima dell'inizio del diciannovesimo secolo: prima di allora, i matematici avevano una nozione intuitiva di cosa fosse un algoritmo e su di essa facevano affidamento per usare e descrivere algoritmi. Tuttavia, tale nozione intuitiva era insufficiente per comprendere appieno le potenzialità del calcolo algoritmico: per questo motivo, come già detto nell'introduzione, diversi ricercatori nel campo della logica matematica proposero una definizione formale del concetto di algoritmo.

Molti modelli di calcolo alternativi alle macchine di Turing sono stati introdotti all'incirca nello stesso periodo di quello introdotto da Turing. Successivamente ulteriori modelli di calcolo sono stati proposti ispirandosi al modello della macchina di Von Neumann e agli odierni calcolatori. Per tutti i modelli proposti si può dimostrare che sono in grado di calcolare esattamente lo stesso insieme di funzioni che sono calcolabili mediante una macchina di Turing. Pertanto, possiamo ragionevolmente assumere che tutti i risultati negativi ottenuti nel precedente capitolo valgano *indipendentemente* dal modello di calcolo utilizzato, mostrando la difficoltà computazionale intrinseca ad alcuni linguaggi (come, ad esempio, quello della terminazione).

Questi risultati giustificano la **tesi di Church-Turing** già esposta nell'introduzione del corso e che può essere formulata nel modo seguente.

> È CALCOLABILE TUTTO CIÒ CHE PUÒ ESSERE CALCOLATO DA UNA MACCHINA DI TURING.

In altre parole, possiamo concludere questa parte del corso affermando che il concetto di calcolabilità secondo Turing cattura il concetto intuitivo di calcolabilità e può a tutti gli effetti essere usato in sua vece.

Esercizi

Esercizio 3.1. Dimostrare che se $L \in \mathcal{L}$ oppure $L^c \in \mathcal{L}$ è un linguaggio finito, allora L e L^c sono due linguaggi decidibili.
Esercizio 3.2. Dimostrare che se $L_1, L_2 \in \mathcal{L}$ sono due linguaggi decidibili, allora anche $L_1 \cup L_2$ e $L_1 \cap L_2$ sono decidibili.
Esercizio 3.3. Dimostrare che la relazione di equi-cardinalità è una relazione di equivalenza.
Esercizio 3.4. Sia $A = \{(i,j,k) : i,j,k \in N\}$. Dimostrare che A è numerabile.
Esercizio 3.5. Dimostrare che l'insieme di tutti i sottoinsiemi finiti di $\{0,1\}^*$ è un insieme numerabile.
Esercizio 3.6. Dimostrare che l'insieme di tutte le stringhe infinite binarie contenenti esattamente due 1 è numerabile.
Esercizio 3.7. Dimostrare che l'insieme di tutte le stringhe infinite binarie non è numerabile.
Esercizio 3.8. Dire, giustificando la risposta, se le seguenti affermazioni sono vere o false.

1. Se $L \in \mathcal{L}$ è un linguaggio decidibile e $L' \in \mathcal{L}$ è un linguaggio semi-decidibile, allora il linguaggio $L \cap L'$ è decidibile.

2. Se $L, L' \in \mathcal{L}$ sono due linguaggi semi-decidibili, allora il linguaggio $L \cap L'$ è semi-decidibile.

3. Se $L, L' \in \mathcal{L}$ sono due linguaggi semi-decidibili, allora il linguaggio $L - L'$ è semi-decidibile.

4. Se $L \in \mathcal{L}$ è un linguaggio semi-decidibile ma non decidibile, allora una qualunque macchina di Turing che semi-decide L deve non terminare per un numero infinito di stringhe binarie di input.

5. Se $L \in \mathcal{L}$ è un linguaggio semi-decidibile ma non decidibile, allora non esiste un linguaggio infinito $L' \subseteq L$ che sia decidibile.

6. Dati due linguaggi decidibili $L_1, L_2 \in \mathcal{L}$, il seguente linguaggio è decidibile.

$$L = \{xy : x \in L_1 \wedge y \in L_2\}$$

7. Siano $L_1, L_2, L_3 \in \mathcal{L}$ tre linguaggi semi-decidibili tali che $L_1 \cap L_2 = \emptyset$, $L_1 \cap L_3 = \emptyset$, $L_2 \cap L_3 = \emptyset$ e $L_1 \cup L_2 \cup L_3 = \{0,1\}^*$. Allora L_1, L_2 e L_3 sono decidibili.

Esercizio 3.9. Facendo uso della tecnica della riducibilità, dimostrare che il seguente linguaggio non è decidibile.

$$L = \{c_T \ : \ c_T \in \mathcal{C} \text{ e se } T(x) \text{ termina in una configurazione finale, allora } x \text{ è formata da una sequenza di } \texttt{0} \text{ seguita da una sequenza di } \texttt{1}\}$$

Esercizio 3.10. Facendo uso della tecnica della riducibilità, dimostrare che il seguente linguaggio non è decidibile.

$$L = \{c_T : c_T \in \mathcal{C} \text{ e } T(\texttt{0}) \text{ termina in una configurazione finale}\}$$

Esercizio 3.11. Facendo uso della tecnica della riducibilità, dimostrare che il seguente linguaggio non è decidibile.

$$L = \{c_T : c_T \in \mathcal{C} \text{ e se } T(x) \text{ termina in una configurazione finale, allora } |x| \text{ è pari}\}$$

Esercizio 3.12. Facendo uso della tecnica della riducibilità, dimostrare che il seguente linguaggio non è decidibile.

$$L = \{c_T : c_T \in \mathcal{C} \text{ ed esiste una stringa } x \text{ per cui } T(x) \text{ produce in output la stringa } \texttt{0}\}$$

Esercizio 3.13. Dire se la seguente affermazione è vera o falsa: "se $L_1, L_2 \in \mathcal{L}$ sono due linguaggi tali che L_1 è riducibile a L_2 e L_2 è regolare, allora L_1 è regolare". Giustificare la risposta.

Esercizio 3.14. È ovvio che se un linguaggio $L_1 \in \mathcal{L}$ è riducibile a un linguaggio $L_2 \in \mathcal{L}$ e se L_1 non è semi-decidibile, allora anche L_2 non è semi-decidibile. È anche evidente che se L_1 è riducibile a L_2, allora anche L_1^c è riducibile a L_2^c. Usando questi due fatti e il Corollario 3.1, dimostrare che il linguaggio L_{eq} definito nell'Esempio 3.9 e il suo complementare L_{eq}^c non sono semi-decidibili.

Esercizio 3.15. Dimostrare che il seguente linguaggio non è semi-decidibile.

$$L = \{c_T : c_T \in \mathcal{C} \text{ e } T(\texttt{101}) \text{ non termina in una configurazione finale}\}$$