re dal software. In particolare la teoria dei problemi intrattabili permette di capire se è probabile che si riesca ad affrontare un problema direttamente e a scrivere un programma per risolverlo (in quanto non incluso nella classe intrattabile), oppure se sia necessario escogitare un modo per aggirarlo: trovare un'approssimazione, usare un metodo euristico o di qualche altra natura per limitare la quantità di tempo che il programma impiega per risolvere il problema.

In questo capitolo introduttivo si comincia da una panoramica ad alto livello della teoria degli automi e delle sue applicazioni. Gran parte del capitolo è dedicata all'indagine delle tecniche di dimostrazione e ai modi per ideare dimostrazioni. Ci occupiamo di dimostrazioni deduttive, di riformulazioni di enunciati, di dimostrazioni per assurdo e per induzione, e di altri importanti concetti. L'ultimo paragrafo introduce i concetti che permeano la teoria degli automi: alfabeti, stringhe e linguaggi.

1.1 Perché studiare la teoria degli automi

Ci sono svariate ragioni per cui lo studio degli automi e della complessità è parte essenziale dell'informatica. Questo paragrafo presenta al lettore la motivazione principale e delinea gli argomenti più rilevanti trattati in questo libro.

1.1.1 Introduzione agli automi a stati finiti

Gli automi a stati finiti sono un utile modello di molte categorie importanti di hardware e software. A partire dal Capitolo 2 esemplificheremo l'impiego dei concetti. Per ora ci limitiamo a elencare alcuni dei casi più significativi.

1. Software per progettare circuiti digitali e verificarne il comportamento.
2. L'analizzatore lessicale di un compilatore, ossia il componente che scompone l'input (i dati in ingresso) in unità logiche, come gli identificatori, le parole chiave e la punteggiatura.
3. Software per esaminare vaste collezioni di testi, ad esempio pagine Web, per trovare occorrenze di parole o di frasi.
4. Software per verificare sistemi di qualsiasi tipo, che abbiano un numero finito di stati discreti, come i protocolli di comunicazione oppure i protocolli per lo scambio sicuro di informazioni.

Forniremo una definizione precisa di automi di vario tipo tra breve. Intanto cominciamo questa introduzione informale descrivendo che cos'è e che cosa fa un automa a stati finiti. Ci sono molti sistemi o componenti, come quelli elencati sopra, di cui si può dire che in

ogni istante si trovano in uno "stato" preso da un insieme finito. Uno stato ha lo scopo di ricordare una parte pertinente della storia del sistema. Dal momento che c'è solo un numero finito di stati generalmente non si può ricordare l'intera storia, perciò il sistema dev'essere progettato attentamente affinché ricordi ciò che è importante e dimentichi ciò che non lo è. Il vantaggio di avere solo un numero finito di stati è che il sistema può essere implementato con un insieme fissato di risorse. Per esempio è possibile implementarlo come un circuito, oppure come un semplice programma che può prendere decisioni esaminando solo un numero limitato di dati o usando la posizione nel codice stesso per prendere la decisione.

Esempio 1.1 Un interruttore è forse il più semplice automa a stati finiti non banale. Un dispositivo di questo tipo ricorda se è nello stato *on* (acceso) oppure nello stato *off* (spento) e permette all'utente di premere (*push*) un pulsante il cui effetto è diverso a seconda del suo stato: se l'interruttore è nello stato "spento", allora premere il pulsante lo fa passare nello stato "acceso"; viceversa, se l'interruttore è nello stato "acceso", allora premere il medesimo pulsante lo porta nello stato "spento".

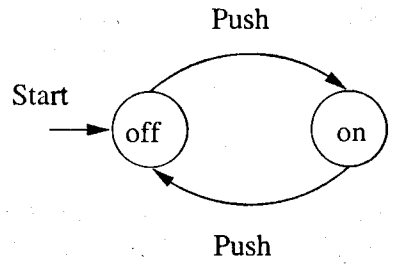

Figura 1.1 Un automa a stati finiti che rappresenta un interruttore.

Il modello di automa a stati finiti per l'interruttore è rappresentato nella Figura 1.1. Come per tutti gli automi a stati finiti, gli stati vengono indicati per mezzo di cerchi; nell'esempio considerato gli stati sono stati chiamati *off* e *on*. Gli archi tra gli stati vengono etichettati dagli input, che rappresentano le influenze esterne sul sistema. Qui ambedue gli archi sono etichettati dall'input *Push*, che rappresenta la pressione del pulsante. Il significato dei due archi è che, indipendentemente dallo stato presente, a fronte di un input *Push* il sistema passa nell'altro stato.

Uno degli stati è detto "stato iniziale", cioè lo stato in cui il sistema si trova inizialmente. Nell'esempio lo stato iniziale è *off*, e per convenzione lo denoteremo con la parola *Start* (avvio) e una freccia che punta a quello stato. Spesso è necessario indicare uno o più stati come gli stati "finali" o "accettanti". Se uno di questi stati viene raggiunto dopo una sequenza di input, tale sequenza di input si considera valida. Per esempio potremmo

considerare lo stato *on*, nella Figura 1.1, come lo stato accettante, in quanto il dispositivo controllato dall'interruttore in quello stato è attivo. Per convenzione gli stati accettanti vengono rappresentati da un doppio cerchio, sebbene nella Figura 1.1 questa convenzione non sia stata rispettata. □

Esempio 1.2 Talvolta ciò che uno stato ricorda è più complesso di una semplice scelta acceso/spento. La Figura 1.2 mostra un altro automa a stati finiti che potrebbe far parte di un analizzatore lessicale. Il compito di questo automa è riconoscere la parola-chiave `then`. L'automa ha quindi bisogno di cinque stati, ognuno dei quali rappresenta una diversa posizione raggiunta nella parola `then`. Le posizioni corrispondono ai prefissi della parola, a partire dalla stringa vuota (vale a dire: finora nessuna parte della parola è stata vista) sino alla parola completa.

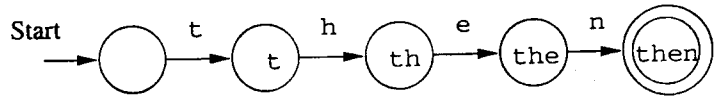

Figura 1.2 Un automa a stati finiti che modella il riconoscimento di `then`.

Nella Figura 1.2 i cinque stati prendono il nome del prefisso di `then` visto fino a quel punto. Gli input corrispondono alle lettere. Possiamo immaginare che l'analizzatore lessicale esamini un carattere per volta tra quelli del programma che sta compilando; ogni nuovo carattere della sequenza da esaminare è l'input dell'automa. Lo stato iniziale corrisponde alla stringa vuota e ogni stato ha una transizione attraverso la successiva lettera di `then` nello stato che corrisponde al prefisso più ampio che viene immediatamente dopo. Si passa allo stato chiamato `then` quando l'input ha formato la parola `then`. Poiché il compito di questo automa è riconoscere quando `then` è stato visto, possiamo considerare quello come l'unico stato accettante. □

1.1.2 Rappresentazioni strutturali

Ci sono due notazioni importanti, diverse dagli automi, ma che hanno un ruolo di primo piano nello studio di questi e delle loro applicazioni.

1. Le *grammatiche* sono modelli utili quando si progetta software che elabora dati con una struttura ricorsiva. L'esempio più noto è il *parser*, un componente del compilatore che tratta gli elementi ricorsivi di un tipico linguaggio di programmazione, come le espressioni aritmetiche, condizionali, ecc. Per esempio una regola grammaticale come $E \Rightarrow E + E$ dichiara che un'espressione può essere formata prendendo due espressioni qualunque e connettendole con un segno più; questa regola mostra come si formano di solito le espressioni dei linguaggi di programmazione. Le grammatiche libere dal contesto (*context-free grammars*), come vengono comunemente chiamate, verranno introdotte nel Capitolo 5.

2. Anche le *espressioni regolari* denotano la struttura di un dato, specialmente di stringhe testuali. Come si vedrà nel Capitolo 3, le configurazioni di stringhe che vengono descritte dalle espressioni regolari sono esattamente le stesse che possono essere descritte da automi a stati finiti. Lo stile di queste espressioni si differenzia significativamente da quello delle grammatiche. Limitiamoci a un semplice esempio. L'espressione regolare in stile UNIX `'[A-Z][a-z]*[ ][A-Z][A-Z]'` rappresenta parole con lettere iniziali maiuscole seguite da uno spazio e due lettere maiuscole. Questa espressione rappresenta configurazioni testuali che potrebbero indicare una città seguita dall'indicazione dello stato, come `Ithaca NY`. Non riesce a rappresentare nomi di città formati da più parole, come `Palo Alto CA`, che potrebbe essere reso dall'espressione più complessa

```
'[A-Z][a-z]*([ ][A-Z][a-z]*)*[ ][A-Z][A-Z]'
```

Quando interpretiamo questa espressione, l'unica cosa che dobbiamo sapere è che `[A-Z]` rappresenta una gamma di caratteri dalla "A" maiuscola alla "Z" maiuscola (ossia qualunque lettera maiuscola) e `[ ]` indica il singolo carattere di spaziatura. Inoltre `*` rappresenta "qualunque numero di" rispetto all'espressione precedente. Le parentesi si usano per raggruppare componenti dell'espressione: non rappresentano caratteri del testo descritto.

1.1.3 Automi e complessità

Gli automi sono essenziali per lo studio dei limiti della computazione. Come accennato nell'introduzione al capitolo, ci sono due punti importanti.

1. Che cosa può fare un computer in assoluto? Tale studio è detto "decidibilità", e i problemi risolvibili da un computer sono detti "decidibili". Il tema è affrontato nel Capitolo 9.

2. Che cosa può fare un computer in maniera efficiente? Tale studio è detto "intrattabilità", e i problemi risolvibili da un computer in un tempo limitato da una funzione lentamente crescente della dimensione dell'input sono detti "trattabili". Spesso si considerano le funzioni polinomiali come "lentamente crescenti", mentre si ritiene che le funzioni che crescono più rapidamente delle polinomiali crescano troppo velocemente. L'argomento viene affrontato nel Capitolo 10.

arco uscente riconduce allo stesso stato. Tale stato corrisponde alla situazione in cui la banca ha ricevuto un messaggio *cancel* prima del messaggio *redeem*. Ciononostante il negozio ha ricevuto un messaggio *pay*, cioè il cliente ha fatto il doppio gioco e ha speso e annullato lo stesso denaro. Incautamente il negozio ha spedito la merce prima di tentare di riscattare il denaro, e quando eseguirà l'azione *redeem* la banca non riconoscerà il messaggio. Infatti essa si trova nello stato 2 in cui ha annullato il denaro e non è disposta a elaborare una richiesta *redeem*.

2.2 Automi a stati finiti deterministici

È arrivato il momento di presentare la nozione formale di automa a stati finiti, in modo da poter precisare alcune argomentazioni e descrizioni informali viste nei Paragrafi 1.1.1 e 2.1. Cominciamo dalla definizione di automa a stati finiti deterministico, un automa che dopo aver letto una qualunque sequenza di input si trova in un singolo stato. Il termine "deterministico" concerne il fatto che per ogni input esiste un solo stato verso il quale l'automa passa dal suo stato corrente. All'opposto, gli automi a stati finiti non deterministici, tema del Paragrafo 2.3, possono trovarsi in diversi stati contemporaneamente. Il termine "automa a stati finiti" farà riferimento alla varietà deterministica, anche se si farà uso di "deterministico" o dell'abbreviazione *DFA* (*Deterministic Finite Automaton*, automa a stati finiti deterministico) per ricordare al lettore di quale tipo di automa si sta parlando.

2.2.1 Definizione di automa a stati finiti deterministico

Un *automa a stati finiti deterministico* consiste dei seguenti componenti.

1. Un insieme finito di *stati*, spesso indicato con Q.
2. Un insieme finito di *simboli di input*, spesso indicato con Σ.
3. Una *funzione di transizione*, che prende come argomento uno stato e un simbolo di input e restituisce uno stato. La funzione di transizione sarà indicata comunemente con δ. Nella rappresentazione grafica informale di automi che abbiamo visto, δ è rappresentata dagli archi tra gli stati e dalle etichette sugli archi. Se q è uno stato e a è un simbolo di input, $\delta(q, a)$ è lo stato p tale che esiste un arco etichettato con a da q a p.[2]
4. Uno *stato iniziale*, uno degli stati in Q.

[2] Più precisamente, il grafo è la descrizione di una funzione di transizione δ, e gli archi del grafo sono costruiti per riflettere le transizioni specificate da δ.

5. Un insieme di stati *finali*, o *accettanti*, F. L'insieme F è un sottoinsieme di Q.

Un automa a stati finiti deterministico verrà spesso indicato con il suo acronimo *DFA*. La rappresentazione più concisa di un DFA è un'enumerazione dei suoi cinque componenti. Nelle dimostrazioni denotiamo un DFA come una quintupla:

$$A = (Q, \Sigma, \delta, q_0, F)$$

dove A è il nome del DFA, Q è l'insieme degli stati, Σ i suoi simboli di input, δ la sua funzione di transizione, q_0 il suo stato iniziale ed F il suo insieme di stati accettanti.

2.2.2 Elaborazione di stringhe in un DFA

La prima cosa che bisogna capire di un DFA è come decide se "accettare" o no una sequenza di simboli di input. Il "linguaggio" del DFA è l'insieme di tutte le stringhe che il DFA accetta. Supponiamo che $a_1 a_2 \cdots a_n$ sia una sequenza di simboli di input. Si parte dal DFA nel suo stato iniziale, q_0. Consultando la funzione di transizione δ, per esempio $\delta(q_0, a_1) = q_1$, troviamo lo stato in cui il DFA entra dopo aver letto il primo simbolo di input, a_1. Il successivo simbolo di input, a_2, viene trattato valutando $\delta(q_1, a_2)$. Supponiamo che questo stato sia q_2. Continuiamo così, trovando gli stati $q_3, q_4, \ldots, q_n$ tali che $\delta(q_{i-1}, a_i) = q_i$ per ogni i. Se q_n è un elemento di F, allora l'input $a_1 a_2 \cdots a_n$ viene accettato, altrimenti viene rifiutato.

Esempio 2.1 Specifichiamo formalmente un DFA che accetta tutte e sole le stringhe di 0 e di 1 in cui compare la sequenza 01. Possiamo scrivere il linguaggio L così:

$$\{w \mid w \text{ è della forma } x01y \text{ per stringhe } x \text{ e } y \text{ che consistono solamente di 0 e di 1}\}$$

Una descrizione equivalente, che usa i parametri x e y a sinistra della barra verticale, è:

$$\{x01y \mid x \text{ e } y \text{ sono stringhe qualsiasi di 0 e di 1}\}$$

Esempi di stringhe appartenenti al linguaggio includono 01, 11010 e 100011. Esempi di stringhe *non* appartenenti al linguaggio includono ϵ, 0, e 111000.

Che cosa sappiamo di un automa che accetta questo linguaggio L? In primo luogo il suo alfabeto di input è $\Sigma = \{0, 1\}$; ha un certo insieme di stati, Q, di cui uno, poniamo q_0, è lo stato iniziale. Quest'automa deve ricordare i fatti importanti relativi agli input già visti. Per decidere se 01 è una sottostringa dell'input, A deve ricordare quanto segue.

1. Ha già visto 01? In caso affermativo accetta ogni sequenza di ulteriori input, cioè da questo momento in poi si troverà solo in stati accettanti.

2. Pur non avendo ancora visto 01, l'input più recente è stato 0, cosicché se ora vede un 1 avrà visto 01. Da questo momento può accettare qualunque seguito?

3. Non ha ancora visto 01, ma l'input più recente è nullo (siamo ancora all'inizio) oppure come ultimo dato ha visto un 1? In tal caso A non accetta finché non vede uno 0 e subito dopo un 1.

Ognuna di queste tre condizioni può essere rappresentata da uno stato. La condizione (3) è rappresentata dallo stato iniziale q_0. All'inizio bisogna vedere uno 0 e poi un 1. Ma se nello stato q_0 si vede per primo un 1, allora non abbiamo fatto alcun passo verso 01, e dunque dobbiamo permanere nello stato q_0. In altri termini $\delta(q_0, 1) = q_0$.

D'altra parte, se ci troviamo nello stato q_0 e vediamo uno 0, siamo nella condizione (2). Vale a dire: non abbiamo ancora visto 01, ma ora abbiamo lo 0. Dunque usiamo q_2 per rappresentare la condizione (2). La transizione da q_0 sull'input 0 è $\delta(q_0, 0) = q_2$.

Ora consideriamo le transizioni dallo stato q_2. Se vediamo uno 0, non siamo in una situazione migliore di prima, ma neanche peggiore. Non abbiamo visto 01, ma 0 è stato l'ultimo simbolo incontrato: stiamo ancora aspettando un 1. Lo stato q_2 descrive questa situazione perfettamente, e dunque poniamo $\delta(q_2, 0) = q_2$. Se ci troviamo nello stato q_2 e vediamo un input 1, sappiamo che c'è uno 0 seguito da un 1. Possiamo passare a uno stato accettante, che si chiamerà q_1 e corrisponderà alla summenzionata condizione (1). Ossia $\delta(q_2, 1) = q_1$.

Infine dobbiamo determinare le transizioni per lo stato q_1. In questo stato abbiamo già incontrato una sequenza 01, quindi, qualsiasi cosa accada, saremo ancora in una situazione in cui abbiamo visto 01. In altri termini $\delta(q_1, 0) = \delta(q_1, 1) = q_1$.

Da quanto detto risulta allora $Q = \{q_0, q_1, q_2\}$. Come affermato in precedenza, q_0 è lo stato iniziale e l'unico stato accettante è q_1. Quindi $F = \{q_1\}$. La definizione completa dell'automa A che accetta il linguaggio L delle stringhe che hanno una sottostringa 01 è

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$$

dove δ è la funzione di transizione descritta sopra. □

2.2.3 Notazioni più semplici per i DFA

Specificare un DFA attraverso una quintupla e una descrizione dettagliata della funzione di transizione δ è allo stesso tempo noioso e di difficile lettura. I sistemi preferibili per la descrizione degli automi sono due:

1. un *diagramma di transizione*, cioè un grafo come quelli presentati nel Paragrafo 2.1

2. una *tabella di transizione*, cioè la specificazione tabellare della funzione δ, da cui si deducono l'insieme degli stati e l'alfabeto di input.

Diagrammi di transizione

Un *diagramma di transizione* per un DFA $A = (Q, \Sigma, \delta, q_0, F)$ è un grafo definito come segue.

a) Per ogni stato in Q esiste un nodo.

b) Per ogni stato q in Q e ogni simbolo di input a in Σ, sia $\delta(q, a) = p$. Allora il diagramma ha un arco dal nodo q al nodo p etichettato a. Se esistono diversi simboli di input che causano transizioni da q a p, il diagramma può avere un arco etichettato dalla lista di tali simboli.

c) Una freccia etichettata *Start* entra nello stato iniziale q_0. Tale freccia non proviene da alcun nodo.

d) I nodi corrispondenti a stati accettanti (quelli in F) sono indicati da un doppio circolo. Gli stati non in F hanno un solo circolo.

Esempio 2.2 La Figura 2.4 mostra il diagramma di transizione per il DFA costruito nell'Esempio 2.1. Nel diagramma vediamo i tre nodi che corrispondono ai tre stati. C'è una freccia *Start* che entra nello stato iniziale q_0, e l'unico stato accettante, q_1, è rappresentato da un doppio circolo. Da ogni stato escono un arco etichettato 0 e un arco etichettato 1 (sebbene i due archi siano combinati in un unico arco con una doppia etichetta nel caso di q_1). Ogni arco corrisponde a uno dei fatti relativi a δ, stabiliti nell'Esempio 2.1. □

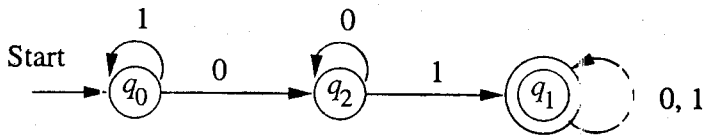

Figura 2.4 Il diagramma di transizione per il DFA che accetta tutte le stringhe con una sottostringa 01.

Tabelle di transizione

Una *tabella di transizione* è una comune rappresentazione tabellare di una funzione come δ, che ha due argomenti e restituisce un valore. Le righe della tabella corrispondono agli stati, le colonne agli input. La voce all'incrocio della riga corrispondente allo stato q e della colonna corrispondente all'input a è lo stato $\delta(q, a)$.

Esempio 2.3 La tabella di transizione relativa alla funzione δ dell'Esempio 2.1 è rappresentata dalla Figura 2.5, corredata di due informazioni specifiche: lo stato iniziale è indicato da una freccia, gli stati accettanti da un asterisco. Dato che possiamo dedurre gli insiemi degli stati e dei simboli di input dalle intestazioni delle righe e delle colonne, dalla tabella di transizione si ricavano tutte le informazioni necessarie per specificare l'automa a stati finiti in maniera univoca. □

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Figura 2.5 Tabella di transizione per il DFA dell'Esempio 2.1.

2.2.4 Estensione della funzione di transizione alle stringhe

Abbiamo spiegato in modo informale che un DFA definisce un linguaggio: l'insieme di tutte le stringhe che producono una sequenza di transizioni di stati dallo stato iniziale a uno stato accettante. Rispetto al diagramma di transizione, il linguaggio di un DFA è l'insieme delle etichette lungo i cammini che conducono dallo stato iniziale a un qualunque stato accettante.

È necessario ora precisare la nozione di linguaggio di un DFA. A questo scopo definiamo una *funzione di transizione estesa*, che descrive che cosa succede quando partiamo da uno stato e seguiamo una sequenza di input. Se δ è la funzione di transizione, allora la funzione di transizione estesa costruita da δ si chiamerà $\hat{\delta}$. La funzione di transizione estesa è una funzione che prende uno stato q e una stringa w e restituisce uno stato p: lo stato che l'automa raggiunge quando parte nello stato q ed elabora la sequenza di input w. Definiamo $\hat{\delta}$ per induzione sulla lunghezza della stringa di input come segue.

BASE $\hat{\delta}(q, \epsilon) = q$. In altre parole, se ci troviamo nello stato q e non leggiamo alcun input, allora rimaniamo nello stato q.

INDUZIONE Supponiamo che w sia una stringa della forma xa, ossia a è l'ultimo simbolo di w e x è la stringa che consiste di tutti i simboli eccetto l'ultimo.[3] Per esempio $w = 1101$ si scompone in $x = 110$ e $a = 1$. Allora

[3]Ricordiamo la convenzione che abbiamo fissato, secondo la quale le lettere all'inizio dell'alfabeto sono simboli, mentre quelle verso la fine dell'alfabeto sono stringhe. Tale convenzione è necessaria affinché la proposizione "della forma xa" abbia un senso.

$$\hat{\delta}(q, w) = \delta\big(\hat{\delta}(q, x), a\big) \tag{2.1}$$

La (2.1) può sembrare contorta, ma il concetto è semplice. Per computare $\hat{\delta}(q, w)$, calcoliamo prima $\hat{\delta}(q, x)$, lo stato in cui si trova l'automa dopo aver elaborato tutti i simboli di w eccetto l'ultimo. Supponiamo che questo stato sia p, ossia $\hat{\delta}(q, x) = p$. Allora $\hat{\delta}(q, w)$ è quanto si ottiene compiendo una transizione dallo stato p sull'input a, l'ultimo simbolo di w. In altri termini $\hat{\delta}(q, w) = \delta(p, a)$.

Esempio 2.4 Costruiamo un DFA che accetti il linguaggio

$$L = \{w \mid w \text{ ha un numero pari di 0 e un numero pari di 1}\}$$

Non dovrebbe sorprendere che il compito degli stati di questo DFA sia quello di contare il numero degli 0 e quello degli 1, ma di contarli modulo 2. In altre parole si usa lo stato per ricordare se il numero degli 0 e il numero degli 1 visti fino a quel momento sono pari o dispari. Quindi ci sono quattro stati, che possono essere interpretati come segue:

q_0: sia il numero degli 0 sia il numero degli 1 visti finora sono pari

q_1: il numero degli 0 visti finora è pari, ma il numero degli 1 è dispari

q_2: il numero degli 1 visti finora è pari, ma il numero degli 0 è dispari

q_3: sia il numero degli 0 sia il numero degli 1 visti finora sono dispari.

Lo stato q_0 è nello stesso tempo lo stato iniziale e l'unico stato accettante. È lo stato iniziale perché prima di leggere qualunque input il numero degli 0 e degli 1 visti sono entrambi zero, e zero è pari. È l'unico stato accettante perché descrive esattamente la condizione per la quale una sequenza di 0 e di 1 appartiene al linguaggio L.

Siamo ora in grado di specificare un DFA per il linguaggio L:

$$A = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

La funzione di transizione δ è descritta dal diagramma di transizione della Figura 2.6. Si noti che ogni input 0 fa sì che lo stato attraversi la linea tratteggiata orizzontale. Pertanto, dopo aver visto un numero pari di 0 ci troviamo al di sopra della linea, nello stato q_0 oppure q_1, mentre dopo averne visto un numero dispari ci troviamo al di sotto, nello stato q_2 oppure q_3. Analogamente, ogni 1 fa sì che lo stato attraversi la linea tratteggiata verticale. Perciò, dopo aver visto un numero pari di 1 siamo a sinistra, nello stato q_0 oppure q_2, mentre dopo averne visto un numero dispari siamo a destra, nello stato q_1 o q_3. Tali osservazioni sono una dimostrazione informale che i quattro stati hanno le

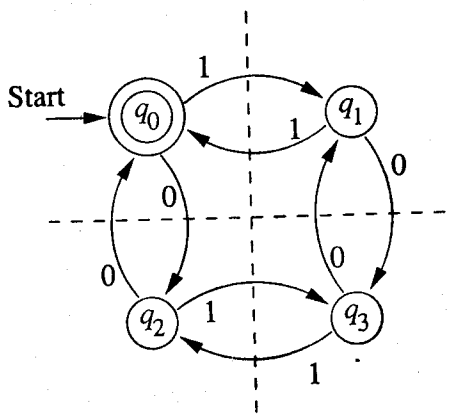

Figura 2.6 Diagramma di transizione per il DFA dell'Esempio 2.4.

interpretazioni a loro attribuite. Si potrebbe dimostrare in termini formali la correttezza delle nostre affermazioni sugli stati, per mutua induzione, sulla scorta dell'Esempio 1.23.

È possibile rappresentare questo DFA anche per mezzo di una tabella di transizione, come illustrato nella Figura 2.7. Noi però non siamo interessati solo all'ideazione di questo DFA; il nostro intento è di usarlo per illustrare la costruzione di $\hat{\delta}$ dalla funzione di transizione δ. Supponiamo che l'input sia 110101. Dato che questa stringa ha un numero pari sia di 0 sia di 1, ci aspettiamo che appartenga al linguaggio, cioè che $\hat{\delta}(q_0, 110101) = q_0$, dato che q_0 è l'unico stato accettante. Verifichiamo quest'asserzione.

	0	1
$* \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Figura 2.7 Tabella di transizione per il DFA dell'Esempio 2.4.

La verifica comporta il calcolo di $\hat{\delta}(q_0, w)$ per ogni prefisso w di 110101, a partire da ϵ e procedendo per aggiunte successive. Il riepilogo di questo calcolo è:

- $\hat{\delta}(q_0, \epsilon) = q_0$
- $\hat{\delta}(q_0, 1) = \delta\big(\hat{\delta}(q_0, \epsilon), 1\big) = \delta(q_0, 1) = q_1$

Notazione standard e variabili locali

Dopo aver letto questo paragrafo si potrebbe immaginare che la notazione abituale sia obbligatoria, cioè che si *debba* usare δ per la funzione di transizione, A per il nome di un DFA, e così via. In realtà tendiamo a usare le stesse variabili per denotare la stessa cosa in tutti gli esempi perché ciò aiuta a ricordare il tipo delle variabili; analogamente in un programma una variabile i è quasi sempre di tipo intero. Di fatto possiamo chiamare i componenti di un automa, o qualunque altro elemento, come vogliamo. Se lo desideriamo, possiamo chiamare un DFA M e la sua funzione di transizione T.

Non ci si deve inoltre stupire del fatto che le stesse variabili hanno significati diversi in contesti diversi. Per esempio a entrambi i DFA degli Esempi 2.1 e 2.4 è stata assegnata una funzione di transizione chiamata δ, ma ognuna delle due funzioni è una variabile locale, pertinente solo al suo esempio. Le due funzioni sono molto diverse e non hanno alcuna relazione l'una con l'altra.

- $\hat{\delta}(q_0, 11) = \delta\big(\hat{\delta}(q_0, 1), 1\big) = \delta(q_1, 1) = q_0$
- $\hat{\delta}(q_0, 110) = \delta\big(\hat{\delta}(q_0, 11), 0\big) = \delta(q_0, 0) = q_2$
- $\hat{\delta}(q_0, 1101) = \delta\big(\hat{\delta}(q_0, 110), 1\big) = \delta(q_2, 1) = q_3$
- $\hat{\delta}(q_0, 11010) = \delta\big(\hat{\delta}(q_0, 1101), 0\big) = \delta(q_3, 0) = q_1$
- $\hat{\delta}(q_0, 110101) = \delta\big(\hat{\delta}(q_0, 11010), 1\big) = \delta(q_1, 1) = q_0$.

□

2.2.5 Il linguaggio di un DFA

Possiamo ora definire il *linguaggio* di un DFA $A = (Q, \Sigma, \delta, q_0, F)$. Questo linguaggio è indicato con $L(A)$ ed è definito da

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \text{ è in } F\}$$

In altre parole il linguaggio di A è l'insieme delle stringhe w che portano dallo stato iniziale q_0 a uno degli stati accettanti. Se L è uguale a $L(A)$ per un DFA A, allora diciamo che L è un *linguaggio regolare*.

Esempio 2.5 Come detto in precedenza, se A è il DFA dell'Esempio 2.1, allora $L(A)$ è l'insieme di tutte le stringhe di 0 e di 1 che contengono la sottostringa 01. Se invece A è il DFA dell'Esempio 2.4, allora $L(A)$ è l'insieme di tutte le stringhe di 0 e di 1 i cui numeri di 0 e di 1 sono entrambi pari. □

2.2.6 Esercizi

Esercizio 2.2.1 La Figura 2.8 rappresenta una pista per biglie. Una biglia viene lanciata a partire da A o da B. Le leve x_1, x_2 e x_3 fanno cadere la biglia a sinistra oppure a destra. Ogni volta che una biglia si imbatte in una leva, dopo che la biglia è passata la leva si ribalta, cosicché la biglia successiva prende l'altra diramazione.

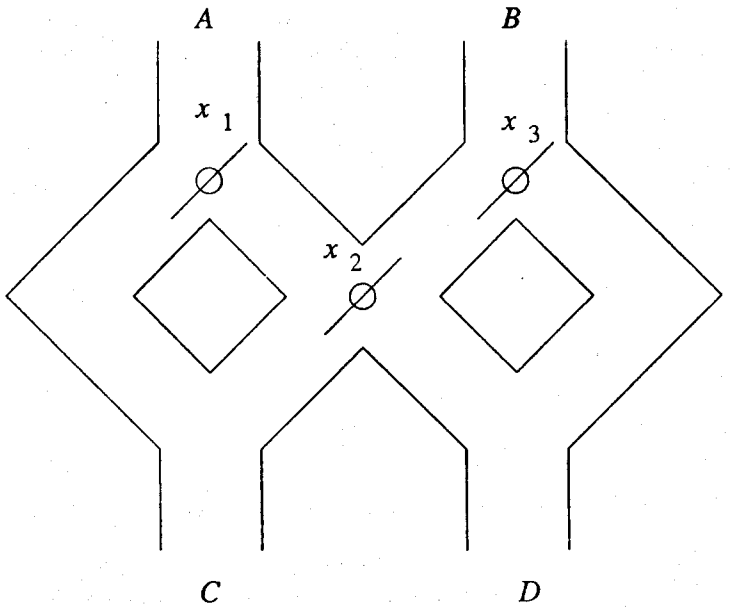

Figura 2.8 Una pista per biglie.

* a) Modellate questa pista con un automa a stati finiti. Gli input A e B rappresentano la via su cui la biglia viene fatta cadere. Supponete che l'accettazione corrisponda all'uscita della biglia da D, la non accettazione all'uscita da C.

! b) Descrivete informalmente il linguaggio dell'automa.

c) Supponete che le leve si ribaltino *prima* di permettere il passaggio della biglia. Come cambiano le risposte a (a) e (b)?

*! **Esercizio 2.2.2** Abbiamo definito $\hat{\delta}$ scomponendo la stringa di input in una stringa seguita da un singolo simbolo (nella parte induttiva, Equazione 2.1). Informalmente possiamo pensare invece che $\hat{\delta}$ descriva ciò che capita lungo un cammino con una certa stringa di etichette; in tal caso non dovrebbe essere rilevante in che modo scomponiamo la stringa di input nella definizione di $\hat{\delta}$. Mostrate che in effetti $\hat{\delta}(q, xy) = \hat{\delta}(\hat{\delta}(q, x), y)$ per qualunque stato q e qualunque coppia di stringhe x e y. *Suggerimento*: svolgete un'induzione su $|y|$.

! **Esercizio 2.2.3** Dimostrate che, per qualunque stato q, stringa x e simbolo di input a, $\hat{\delta}(q, ax) = \hat{\delta}(\delta(q, a), x)$. *Suggerimento*: usate l'Esercizio 2.2.2.

Esercizio 2.2.4 Definite tre DFA che accettino i seguenti linguaggi sull'alfabeto $\{0, 1\}$:

* a) l'insieme di tutte le stringhe che finiscono per 00

b) l'insieme di tutte le stringhe con tre 0 consecutivi (non necessariamente alla fine)

c) l'insieme delle stringhe con 011 come sottostringa.

! **Esercizio 2.2.5** Definite quattro DFA che accettino i seguenti linguaggi sull'alfabeto $\{0, 1\}$:

a) l'insieme di tutte le stringhe tali che ogni blocco di cinque simboli consecutivi contenga almeno due 0

b) l'insieme di tutte le stringhe il cui decimo simbolo a partire da destra sia 1

c) l'insieme delle stringhe che cominciano o finiscono (o entrambe le cose) per 01

d) l'insieme delle stringhe tali che il numero di 0 sia divisibile per cinque e il numero di 1 sia divisibile per 3.

!! **Esercizio 2.2.6** Definite due DFA che accettino i seguenti linguaggi sull'alfabeto $\{0, 1\}$:

* a) l'insieme di tutte le stringhe che cominciano con un 1 e che, interpretate come interi binari, siano multipli di 5 (per esempio le stringhe 101, 1010 e 1111 sono nel linguaggio; 0, 100 e 111 invece no)

b) l'insieme di tutte le stringhe che lette al contrario come un intero binario siano divisibili per 5 (esempi di stringhe nel linguaggio sono 0, 10011, 1001100 e 0101).

Esercizio 2.2.7 Sia A un DFA e q uno stato di A tale che $\delta(q, a) = q$ per tutti i simboli di input a. Dimostrate per induzione sulla lunghezza dell'input che, per tutte le stringhe di input w, $\hat{\delta}(q, w) = q$.

Esercizio 2.2.8 Sia A un DFA e a un simbolo di input di A tale che, per tutti gli stati q di A, valga $\delta(q, a) = q$.

a) Dimostrate per induzione su n che, per ogni $n \geq 0$, $\hat{\delta}(q, a^n) = q$, dove a^n è la stringa formata da n ripetizioni di a.

b) Dimostrate che o $\{a\}^* \subseteq L(A)$ o $\{a\}^* \cap L(A) = \emptyset$.

***! Esercizio 2.2.9** Sia $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ un DFA e supponiamo che per tutti gli a in Σ si abbia $\delta(q_0, a) = \delta(q_f, a)$.

a) Dimostrate che, per ogni $w \neq \epsilon$, $\hat{\delta}(q_0, w) = \hat{\delta}(q_f, w)$.

b) Dimostrate che se x è una stringa non vuota in $L(A)$, allora per ogni $k > 0$ anche x^k (cioè x scritta un numero k di volte) è in $L(A)$.

***! Esercizio 2.2.10** Considerate il DFA con la seguente tabella di transizione:

	0	1
$\rightarrow A$	A	B
$*B$	B	A

Descrivete in termini informali il linguaggio accettato da questo DFA e dimostrate per induzione sulla lunghezza della stringa di input che la descrizione è corretta. *Suggerimento*: nel porre l'ipotesi induttiva è consigliabile formulare un enunciato su quali input portano a ciascuno stato, e non solo su quali input portano allo stato accettante.

! Esercizio 2.2.11 Ripetete l'Esercizio 2.2.10 per la seguente tabella di transizione:

	0	1
$\rightarrow *A$	B	A
$*B$	C	A
C	C	C

2.3 Automi a stati finiti non deterministici

Un automa a stati finiti non deterministico (*NFA, Nondeterministic Finite Automaton*) può trovarsi contemporaneamente in diversi stati. Questa caratteristica viene sovente espressa come capacità di "scommettere" su certe proprietà dell'input. Per esempio, quando un automa viene usato per cercare determinate sequenze di caratteri (come: parole chiave) in una lunga porzione di testo, è utile "scommettere" che ci si trova all'inizio di una di tali

sequenze e usare una sequenza di stati per verificare, carattere per carattere, che compaia la stringa cercata. Vedremo un esempio di questo tipo di applicazione nel Paragrafo 2.4.

Prima di esaminare le applicazioni, è necessario definire gli automi a stati finiti non deterministici e mostrare che accettano gli stessi linguaggi accettati dai DFA. Come i DFA, gli NFA accettano proprio i linguaggi regolari. Tuttavia ci sono buone ragioni per occuparsi degli NFA. Spesso sono più succinti e più facili da definire rispetto ai DFA; inoltre, anche se è sempre possibile convertire un NFA in un DFA, quest'ultimo può avere esponenzialmente più stati di un NFA. Per fortuna casi di questo tipo sono rari.

2.3.1 Descrizione informale degli automi a stati finiti non deterministici

Come un DFA, un NFA ha un insieme finito di stati, un insieme finito di simboli di input, uno stato iniziale e un insieme di stati accettanti. Ha anche una funzione di transizione che chiameremo δ. La differenza tra DFA ed NFA sta nel tipo di δ. Per gli NFA, δ è una funzione che ha come argomenti uno stato e un simbolo di input (come quella dei DFA), ma restituisce un insieme di zero o più stati (invece di un solo stato, come nel caso dei DFA). Cominceremo da un esempio di NFA e poi passeremo a precisare le definizioni.

Esempio 2.6 La Figura 2.9 mostra un automa a stati finiti non deterministico con il compito di accettare tutte e sole le stringhe di 0 e di 1 che finiscono per 01. Lo stato q_0 è lo stato iniziale e possiamo pensare che l'automa si trovi nello stato q_0 (eventualmente insieme ad altri stati) quando non ha ancora "scommesso" che il finale 01 è cominciato. È sempre possibile che il simbolo successivo non sia il primo del suffisso 01, anche se quel simbolo è proprio 0. Dunque lo stato q_0 può operare una transizione verso se stesso sia su 0 sia su 1.

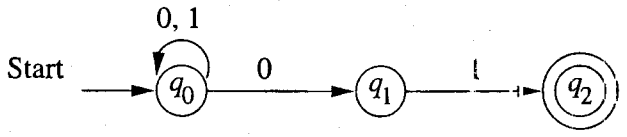

Figura 2.9 Un NFA che accetta tutte le stringhe che finiscono per 01.

Se però il simbolo successivo è 0, l'NFA scommette anche che è iniziato il suffisso 01. Un altro arco etichettato 0 conduce dunque da q_0 allo stato q_1. Si noti che ci sono due archi etichettati 0 in uscita da q_0. L'NFA ha l'opzione di andare verso q_0 oppure verso q_1, ed effettivamente fa entrambe le cose, come si vedrà quando si preciseranno le definizioni. Nello stato q_1 l'NFA verifica che il simbolo successivo sia 1, e se è così, passa allo stato q_2 e accetta.