Capitolo 1

Automi: metodo e follia

La teoria degli automi è lo studio di dispositivi astratti di calcolo, o "macchine". Negli anni '30, prima dell'avvento dei computer, A. Turing studiò una macchina astratta che aveva tutte le capacità degli elaboratori odierni, almeno per quanto riguarda ciò che possono calcolare. Il fine di Turing era descrivere precisamente il confine tra quello che un dispositivo di calcolo può fare e quello che non può fare; le sue conclusioni non riguardano solo le sue *macchine di Turing* astratte, ma anche le macchine reali di adesso.

Negli anni '40 e '50 diversi ricercatori studiarono alcuni tipi più semplici di macchine, che oggi sono dette *automi a stati finiti*. Questi automi, pensati originariamente per fornire un modello del funzionamento cerebrale, risultarono utili per molti altri scopi, che saranno introdotti nel Paragrafo 1.1. Nello stesso periodo, nei tardi anni '50, il linguista N. Chomsky iniziò a studiare le grammatiche formali. Per quanto non siano delle macchine in senso proprio, queste grammatiche sono strettamente collegate agli automi astratti e oggi stanno alla base di alcuni importanti componenti software, tra cui parti dei compilatori.

Nel 1969 S. Cook approfondì gli studi di Turing su ciò che si può calcolare. Cook riuscì a distinguere i problemi risolvibili in modo efficiente da un elaboratore da quelli che possono essere risolti in linea di principio, ma che di fatto richiedono così tanto tempo da rendere inutilizzabile un computer se non per istanze del problema di dimensione limitata. Questa seconda classe di problemi viene definita *intrattabile*, o *NP-hard*. È molto probabile che nemmeno il progresso esponenziale nella velocità di calcolo dell'hardware (legge di Moore) avrà un effetto determinante sulle nostre capacità di risolvere casi significativi di problemi intrattabili.

Tutti questi sviluppi teorici hanno un rapporto diretto con quanto gli informatici fanno attualmente. Alcuni concetti, come gli automi a stati finiti e certi tipi di grammatiche formali, vengono usati nella progettazione e realizzazione di importanti tipi di software. Altri concetti, come la macchina di Turing, aiutano a comprendere che cosa ci si può aspetta-

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

che il risultato si può scrivere come somma di multipli di 3 e di 5, e aggiungere un ulteriore 3 alla somma per poter scrivere $n + 1$.

In termini più formali si osserva che $n - 2 \geq 8$, così possiamo supporre $S(n - 2)$. In altre parole $n - 2 = 3a + 5b$ per due interi a e b. Allora $n + 1 = 3 + 3a + 5b$, così $n + 1$ può essere scritto come la somma di 3, moltiplicato per $a + 1$, e di 5, moltiplicato per b. Ciò dimostra $S(n + 1)$ e conclude il passo induttivo. □

1.4.3 Induzioni strutturali

Nella teoria degli automi esistono diverse strutture definite ricorsivamente su cui si devono dimostrare enunciati. Importanti esempi sono le nozioni familiari di alberi e di espressioni. Similmente alle induzioni, tutte le definizioni ricorsive hanno un caso di base, in cui si definiscono una o più strutture elementari, e un passo induttivo, in cui si definiscono strutture più complesse nei termini di strutture definite in precedenza.

Esempio 1.19 Una definizione ricorsiva di albero.

BASE Un singolo nodo è un albero, e tale nodo è la *radice* dell'albero.

INDUZIONE Se $T_1, T_2, \ldots, T_k$ sono alberi, allora si può formare un nuovo albero in questo modo:

1. si comincia con un nuovo nodo N, che è la radice dell'albero
2. si aggiunge una copia degli alberi $T_1, T_2, \ldots, T_k$
3. si aggiungono lati dal nodo N alle radici di ogni albero $T_1, T_2, \ldots, T_k$.

La Figura 1.7 mostra la costruzione induttiva di un albero con radice N a partire da k alberi più piccoli. □

Esempio 1.20 Consideriamo un'altra definizione ricorsiva. Questa volta definiamo *espressioni* con gli operatori matematici $+$ e $*$, aventi come operandi sia numeri sia variabili.

BASE Qualunque numero o lettera (ossia una variabile) è un'espressione.

INDUZIONE Se E ed F sono espressioni, allora lo sono anche $E + F$, $E * F$ e (E).

Per esempio sia 2 sia x sono espressioni (caso di base). Il passo induttivo dice che $x + 2$, $(x + 2)$ e $2 * (x + 2)$ sono espressioni. Si noti come ognuna di queste espressioni dipende dal fatto che le precedenti sono a loro volta espressioni. □

Data una definizione ricorsiva, si possono dimostrare teoremi su di essa con la seguente forma di dimostrazione, detta *induzione strutturale*. Sia $S(X)$ un enunciato sulle strutture X definite in modo ricorsivo.

Push, il che richiede almeno una pressione sul pulsante. Poiché l'ipotesi è falsa, possiamo concludere nuovamente che l'enunciato se-allora è vero.

INDUZIONE Ora supponiamo che $S_1(n)$ e $S_2(n)$ siano vere, e proviamo a dimostrare $S_1(n+1)$ e $S_2(n+1)$. Anche questa dimostrazione si suddivide in quattro parti.

1. [$S_1(n+1)$; se] L'ipotesi per questa parte è che $n+1$ sia pari. Di conseguenza n è dispari. La parte "se" dell'enunciato $S_2(n)$ dice che dopo n pressioni l'automa si trova nello stato *on*. L'arco da *on* a *off* recante l'etichetta *Push* dice che la $(n+1)$-esima pressione farà passare l'automa nello stato *off*. Ciò completa la dimostrazione della parte "se" di $S_1(n+1)$.

2. [$S_1(n+1)$; solo-se] L'ipotesi è che l'automa si trovi nello stato *off* dopo $n+1$ pressioni. L'esame dell'automa indica che l'unico modo di pervenire allo stato *off* è di trovarsi nello stato *on* e di ricevere un input *Push*. Perciò, se ci si trova nello stato *off* dopo $n+1$ pressioni, l'automa deve essersi trovato nello stato *on* dopo n pressioni. Allora possiamo ricorrere alla parte "solo-se" dell'enunciato $S_2(n)$ per concludere che n è dispari. Dunque $n+1$ è pari, come si voleva dimostrare per la parte solo-se di $S_1(n+1)$.

3. [$S_2(n+1)$; se] Questa parte è essenzialmente uguale alla parte (1), con i ruoli di S_1 ed S_2 e con i ruoli di "dispari" e "pari" scambiati. Il lettore dovrebbe essere in grado di costruire agevolmente questa parte della dimostrazione.

4. [$S_2(n+1)$; solo-se] Questa parte è essenzialmente uguale alla parte (2), con i ruoli di S_1 ed S_2 e con i ruoli di "dispari" e "pari" scambiati.

□

Dall'esempio 1.23 si può ricavare il modello di tutte le induzioni mutue.

- Ogni enunciato dev'essere dimostrato separatamente nella base e nel passo induttivo.
- Se si tratta di enunciati se-e-solo-se, allora entrambe le direzioni di ogni enunciato devono essere dimostrate, sia nella base sia nel passo induttivo.

1.5 I concetti centrali della teoria degli automi

In questo paragrafo verranno introdotte le definizioni dei termini più importanti che permeano la teoria degli automi. Questi concetti sono "alfabeto" (un insieme di simboli), "stringa" (una lista di simboli di un alfabeto) e "linguaggio" (un insieme di stringhe dallo stesso alfabeto).

1.5.1 Alfabeti

Un *alfabeto* è un insieme finito e non vuoto di simboli. Per indicare un alfabeto si usa convenzionalmente il simbolo Σ. Fra gli alfabeti più comuni citiamo:

1. $\Sigma = \{0, 1\}$, l'alfabeto *binario*
2. $\Sigma = \{a, b, \ldots, z\}$, l'insieme di tutte le lettere minuscole
3. l'insieme di tutti i caratteri ASCII o l'insieme di tutti i caratteri ASCII stampabili.

1.5.2 Stringhe

Una *stringa* (o *parola*) è una sequenza finita di simboli scelti da un alfabeto. Per esempio 01101 è una stringa sull'alfabeto binario $\Sigma = \{0, 1\}$. La stringa 111 è un'altra stringa sullo stesso alfabeto.

La stringa vuota

La *stringa vuota* è la stringa composta da zero simboli. Questa stringa, indicata con ϵ, è una stringa che può essere scelta da un qualunque alfabeto.

Lunghezza di una stringa

Spesso è utile classificare le stringhe a seconda della loro *lunghezza*, vale a dire il numero di posizioni per i simboli della stringa. Per esempio 01101 ha lunghezza 5. Comunemente si dice che la lunghezza di una stringa è "il numero di simboli" nella stringa stessa; quest'enunciato è un'espressione accettabile colloquialmente, ma formalmente non corretta. Infatti ci sono solo due simboli, 0 e 1, nella stringa 01101, ma ci sono 5 *posizioni*, e la lunghezza della stringa è 5. Tuttavia ci si può aspettare di vedere usato di solito "il numero di simboli" quando invece s'intende il "numero di posizioni".

La notazione standard per la lunghezza di una stringa w è $|w|$. Per esempio $|011| = 3$ e $|\epsilon| = 0$.

Potenze di un alfabeto

Se Σ è un alfabeto possiamo esprimere l'insieme di tutte le stringhe di una certa lunghezza su tale alfabeto usando una notazione esponenziale. Definiamo Σ^k come l'insieme di stringhe di lunghezza k, con simboli tratti da Σ.

Esempio 1.24 Si noti che $\Sigma^0 = \{\epsilon\}$ per qualsiasi alfabeto Σ. In altre parole ϵ è la sola stringa di lunghezza 0.

Se $\Sigma = \{0, 1\}$, allora $\Sigma^1 = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$,

Convenzioni tipografiche per simboli e stringhe

Useremo comunemente lettere minuscole prese dall'inizio dell'alfabeto (oppure cifre) per denotare i simboli, e lettere minuscole dalla fine dell'alfabeto, di solito w, x, y e z, per denotare le stringhe. Si consiglia di abituarsi a tale convenzione perché facilita il riconoscimento del tipo di oggetto in discussione.

$$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

e così via. Si osservi che si crea una certa confusione tra Σ e Σ^1. Il primo è un alfabeto: i suoi membri, 0 e 1, sono simboli. Il secondo è un insieme di stringhe; i suoi membri sono le stringhe 0 e 1, ognuna delle quali è di lunghezza 1. Qui non cercheremo di usare due notazioni distinte per i due insiemi, ma ci affideremo piuttosto al contesto per chiarire se $\{0, 1\}$ o insiemi simili sono alfabeti o insiemi di stringhe. □

L'insieme di tutte le stringhe su un alfabeto Σ viene indicato convenzionalmente con Σ^*. Per esempio $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$. Formulato altrimenti

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$$

Talvolta si desidera escludere la stringa vuota da un insieme di stringhe. L'insieme di stringhe non vuote sull'alfabeto Σ è indicato con Σ^+. Di conseguenza valgono le due equivalenze seguenti.

- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$
- $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$

Concatenazione di stringhe

Siano x e y stringhe. Allora xy denota la *concatenazione* di x e y, vale a dire la stringa formata facendo una copia di x e facendola seguire da una copia di y. Più precisamente, se x è la stringa composta da i simboli $x = a_1 a_2 \cdots a_i$ e y è la stringa composta da j simboli $y = b_1 b_2 \cdots b_j$, allora xy è la stringa di lunghezza $i + j$: $xy = a_1 a_2 \cdots a_i b_1 b_2 \cdots b_j$.

Esempio 1.25 Poniamo $x = 01101$ e $y = 110$. Allora $xy = 01101110$ e $yx = 11001101$. Per qualunque stringa w sono valide le equazioni $\epsilon w = w\epsilon = w$. In altre parole ϵ è l'*identità per la concatenazione*, dato che, concatenata con una qualunque stringa, dà come risultato la stringa stessa (nello stesso modo in cui 0, l'identità per l'addizione, può essere sommato a qualunque numero x e dà come risultato x). □

1.5.3 Linguaggi

Un insieme di stringhe scelte da Σ^*, dove Σ è un particolare alfabeto, si dice un *linguaggio*. Se Σ è un alfabeto e $L \subseteq \Sigma^*$, allora L è un *linguaggio su* Σ. Si noti che un linguaggio su Σ non deve necessariamente includere stringhe con tutti i simboli di Σ; perciò, una volta che abbiamo stabilito che L è un linguaggio su Σ, sappiamo anche che è un linguaggio su qualunque alfabeto includa Σ.

La scelta del termine "linguaggio" può sembrare strana; d'altra parte i linguaggi comuni possono essere visti come insiemi di stringhe. Un esempio è l'inglese, in cui la raccolta delle parole accettabili della lingua è un insieme di stringhe sull'alfabeto che consiste di tutte le lettere. Un altro esempio è il C, o qualunque altro linguaggio di programmazione, in cui i programmi accettabili sono un sottoinsieme di tutte le possibili stringhe che si possono formare con l'alfabeto del linguaggio. Quest'alfabeto è un sottoinsieme dei caratteri ASCII. L'alfabeto può variare leggermente tra diversi linguaggi di programmazione, ma generalmente include le lettere maiuscole e minuscole, le cifre, la punteggiatura e i simboli matematici.

Nello studio degli automi ci si imbatte in molti altri linguaggi. Alcuni sono esempi astratti, come quelli che elenchiamo.

1. Il linguaggio di tutte le stringhe che consistono di n 0 seguiti da n 1, per $n \geq 0$: $\{\epsilon, 01, 0011, 000111, \ldots\}$.

2. L'insieme delle stringhe con un uguale numero di 0 e di 1:

$$\{\epsilon, 01, 10, 0011, 0101, 1001, \ldots\}$$

3. L'insieme dei numeri binari il cui valore è un numero primo:

$$\{10, 11, 101, 111, 1011, \ldots\}$$

4. Σ^* è un linguaggio per qualunque alfabeto Σ.

5. $\emptyset$, il linguaggio vuoto, è un linguaggio rispetto a qualunque alfabeto.

6. Anche $\{\epsilon\}$, il linguaggio che consta della sola stringa vuota, è un linguaggio rispetto a ogni alfabeto. Si noti che $\emptyset \neq \{\epsilon\}$; il primo non ha stringhe mentre il secondo ne ha una.

L'unica restrizione di rilievo sui linguaggi è che tutti gli alfabeti sono finiti. Di conseguenza i linguaggi, sebbene possano avere un numero infinito di stringhe, devono consistere di stringhe tratte da un determinato alfabeto finito.

I formatori di insiemi come un modo di definire linguaggi

Spesso si descrive un linguaggio usando un *formatore di insiemi*:

$$\{w \mid \text{enunciato su } w\}$$

Quest'espressione va letta come "l'insieme delle parole w tali che vale l'enunciato su w a destra della barra verticale". Alcuni esempi:

1. $\{w \mid w$ consiste di un numero uguale di 0 e di $1\}$
2. $\{w \mid w$ è un intero binario che è primo$\}$
3. $\{w \mid w$ è un programma C sintatticamente corretto$\}$.

Si suole anche sostituire w con un'espressione con parametri e descrivere le stringhe del linguaggio enunciando le condizioni sui parametri. Ecco alcuni esempi, il primo con un parametro n, il secondo con i parametri i e j.

1. $\{0^n1^n \mid n \geq 1\}$. Leggi "l'insieme di 0 elevato alla n, 1 alla n tale che n è maggiore o uguale a 1"; questo linguaggio è formato dalle stringhe $\{01, 0011, 000111, \ldots\}$. Si noti che, come con gli alfabeti, è possibile elevare un singolo simbolo alla potenza n per rappresentare n copie di quel simbolo.
2. $\{0^i1^j \mid 0 \leq i \leq j\}$. Questo linguaggio consiste di stringhe con un certo numero di 0 (eventualmente nessuno) seguiti da almeno altrettanti 1.

1.5.4 Problemi

Nella teoria degli automi un *problema* è la questione se una data stringa sia o no membro di un particolare linguaggio. Come si vedrà, tutto ciò che definiamo correntemente un "problema" può essere espresso in termini di appartenenza a un linguaggio. In termini più precisi, se Σ è un alfabeto e L è un linguaggio su Σ, allora il problema L è:

- data una stringa w in Σ^*, decidere se w appartiene a L.

Esempio 1.26 Il problema di verificare se un numero è primo si può esprimere con il linguaggio L_p, che consiste di tutte le stringhe binarie il cui valore come numero binario è un numero primo. In altre parole, data una stringa di 0 e di 1, diremo "sì" se la stringa

Linguaggio o problema

Linguaggi e problemi sono in realtà la stessa cosa. Scegliere l'uno o l'altro termine dipende dal punto di vista. Se ci si occupa di stringhe prese come tali, per esempio quelle dell'insieme $\{0^n1^n \mid n \geq 1\}$, allora si è portati a pensare all'insieme di stringhe come a un linguaggio. Negli ultimi capitoli del libro tenderemo ad assegnare una "semantica" alle stringhe, per esempio pensandole come codifiche di grafi, espressioni logiche o persino numeri interi. Nei casi in cui ci occupiamo di quanto viene rappresentato dalla stringa anziché della stringa stessa, tenderemo a considerare un insieme di stringhe come un problema.

è la rappresentazione binaria di un numero primo e "no" in caso contrario. Per alcune stringhe questa decisione è facile. Per esempio 0011101 non può essere la rappresentazione di un primo, per la semplice ragione che ogni intero, con l'eccezione di 0, ha una rappresentazione binaria che comincia con 1. D'altra parte è meno palese se la stringa 11101 appartenga a L_p. Dunque qualunque soluzione a questo problema dovrà avvalersi di risorse computazionali di qualche tipo: tempo e spazio, per esempio. □

Un aspetto potenzialmente insoddisfacente della nostra definizione di "problema" è che comunemente non si pensa a un problema in termini di decisione (p è o non è vero?), bensì come se si trattasse di una richiesta di computare o trasformare un certo input (trovare il miglior modo di svolgere un dato compito). Per esempio il compito di un *parser* in un compilatore C può essere visto come un problema nel nostro senso formale, in cui si dà una stringa ASCII e si chiede di decidere se la stringa sia o no un elemento di L_c, l'insieme dei programmi C validi. Tuttavia un *parser* non si limita a decidere: produce infatti un albero sintattico, voci in una tabella di simboli ed eventualmente altro. Non solo: il compilatore nel suo insieme risolve il problema di trasformare un programma C in codice oggetto per una certa macchina, ben più di una semplice risposta "sì" o "no" alla domanda sulla validità di un programma.

Nonostante ciò, la definizione di problema come linguaggio ha resistito nel tempo come il modo più opportuno di trattare le questioni cruciali della teoria della complessità. In questa teoria l'interesse è volto a scoprire limiti inferiori della complessità di determinati problemi. Di particolare importanza sono le tecniche per dimostrare che certi problemi non possono essere risolti in tempo meno che esponenziale nella dimensione del loro input. Risulta che le versioni "sì-no" oppure "basate su linguaggi" di problemi noti sono in questo senso altrettanto difficili delle versioni "risolvi".

In altre parole, se possiamo dimostrare che è difficile decidere se una data stringa appartiene al linguaggio L_X delle stringhe valide in un linguaggio di programmazione X,