Fondamenti di Informatica

per la Sicurezza

a.a. 2007/08

Automi

Stefano Ferrari

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI TECNOLOGIE DELL'INFORMAZIONE

Automa a stati finiti (1)

Un automa a stati finiti è un modello matematico caratterizzato da:

- un input costituito da una sequenza di elementi discreti;
- eventualmente, un output (anch'esso a valori discreti);
- un insieme finito di stati nei quali l'automa può trovarsi durante l'esecuzione.

Automa a stati finiti (2)

Un automa a stati finiti si può immaginare come un dispositivo dotato di una testina.

L'automa legge spostandosi sempre nella stessa direzione lungo un nastro di lunghezza illimitata contenente dei simboli.

A seconda del simbolo letto, l'automa si porta in un altro stato.

Ripete queste operazioni fino a quando non legge un simbolo terminale.

Definizione formale

Un automa a stati finiti deterministico (DFA) è una quintupla $\langle Q, \Sigma, \delta, q_0, F \rangle$ dove:

- Q è l'insieme degli stati (finito);
- Σ è l'alfabeto di input;
- $\delta : Q \times \Sigma \rightarrow Q$ è la funzione di transizione;
- q_0 è lo stato iniziale;
- $F \subset Q$ è l'insieme degli stati finali.

Funzione di transizione

La funzione di transizione può essere rappresentata in forma tabellare.

Esempio:

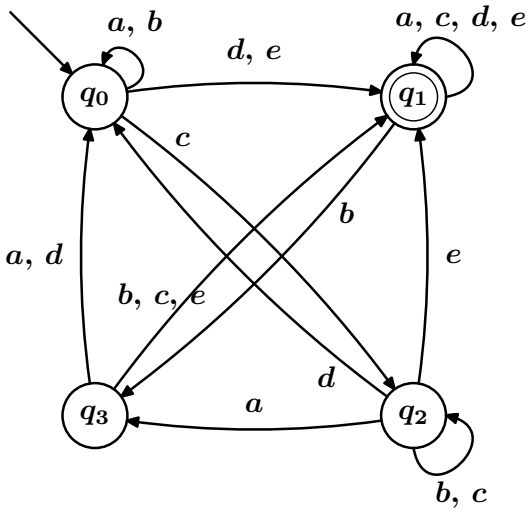
δ	a	b	c	d	e
q_0	q_0	q_0	q_2	q_1	q_1
q_1	q_1	q_3	q_1	q_1	q_1
q_2	q_3	q_2	q_2	q_0	q_1
q_3	q_0	q_1	q_1	q_0	q_1

dove:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{a, b, c, d, e\}$

Rappresentazione grafica

Un DFA può essere rappresentato da un grafo:



dove:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{a, b, c, d, e\}$
- $F = \{q_1\}$

Linguaggio accettato da un DFA (1)

Il concetto di funzione di transizione può essere estesa in modo da essere applicata ripetutamente ad una successione di simboli in input.

Si dice che un DFA accetta una stringa se, dopo averla letta, esso si ritrova in uno stato finale.

L'insieme delle stringhe accettate da un DFA viene detto linguaggio accettato dal DFA stesso.

Linguaggio accettato da un DFA (2)

Più formalmente:

- l'estensione di δ alle stringhe, $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$, si definisce in modo univoco come:

$$\hat{\delta}(q, x) = \begin{cases} q & \text{se } x = \epsilon \\ \delta(\hat{\delta}(q, w), y) & \text{se } x = wy,\ w \in \Sigma^*,\ y \in \Sigma \end{cases}$$

- una stringa x viene accettata dal DFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ se $\hat{\delta}(q_0, x) \in F$;
- il linguaggio $\mathcal{L}(M) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\}$ è il linguaggio accettato da M.

Automa non-deterministico

Un automa a stati finiti non-deterministico (NFA) è una quintupla $\langle Q, \Sigma, \delta, q_0, F \rangle$ dove:

- Q è un insieme finito di stati;
- Σ è un alfabeto (alfabeto di input);
- $\delta : Q \times \Sigma \longrightarrow \wp(Q)$ è la funzione di transizione;
- q_0 è lo stato iniziale;
- $F \subset Q$ è l'insieme degli stati finali.

Nota: ora è ammesso $\delta(q, a) = \emptyset$ per qualche $q \in Q$ e $a \in \Sigma$.

Linguaggio accettato da un NFA

Dalla funzione δ si definisce in modo univoco $\hat{\delta} : Q \times \Sigma^* \longrightarrow \wp(Q)$:

$$\hat{\delta}(q, x) = \begin{cases} \{q\} & \text{se } x = \epsilon \\ \bigcup_{p \in \hat{\delta}(q, w)} \delta(p, y) & \text{se } x = wy,\ w \in \Sigma^*,\ y \in \Sigma \end{cases}$$

Una stringa x viene accettata dal NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ se $\hat{\delta}(q_0, x) \cap F \neq \emptyset$.

Il linguaggio $\mathcal{L}(M) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$ è il linguaggio accettato da M.

Confronto DFA-NFA

La "potenza di calcolo" di un NFA è la stessa di un DFA.

Ciò significa che per ogni linguaggio accettato da un NFA, esiste un DFA che accetta lo stesso linguaggio.

La classe di linguaggi accettati dai DFA è la classe dei linguaggi regolari.

Si può generare in modo automatico il DFA equivalente di un dato NFA.

A volte è più comodo progettare un NFA e poi convertirlo.

Limiti dei DFA

Un DFA è un dispositivo a memoria finita, quindi è adatto a risolvere solo quei problemi che consentono di limitare a priori la lunghezza delle sequenze d'ingresso di cui tenere memoria.

Da cosa deriva la mancanza di espressività?

- Il numero di stati è fissato a priori non consente di trattare quei problemi che richiedono un conteggio senza limite fissato.
  - Esempio: un DFA a 4 stati può riconoscere $aaacbbb$ ma non $aaaacbbbb$.
- Rendere il numero degli stati infinito non è praticabile.

Esercizio 1 (1)

Costruire un DFA su $\Sigma = \{0, 1\}$ che accetti l'insieme di tutte le stringhe aventi tre 0 consecutivi.

Osservazioni:

- l'alfabeto è dato ($\Sigma = \{0, 1\}$);
- si possono individuare le seguenti possibili situazioni:

  (a) sono già stati letti tre (o più) "0" consecutivi;

  (b) gli ultimi due simboli letti sono "0";

  (c) l'ultimo simbolo letto è "0";

  (d) l'ultimo simbolo letto è "1".

Esercizio 1 (2)

Indicando con q_i lo stato in cui l'automa si troverà dopo aver letto i simboli "0" consecutivi:

- situazione (a) $\leftrightarrow q_3$;
- situazione (b) $\leftrightarrow q_2$;
- situazione (c) $\leftrightarrow q_1$;
- situazione (d) $\leftrightarrow q_0$;

Note:

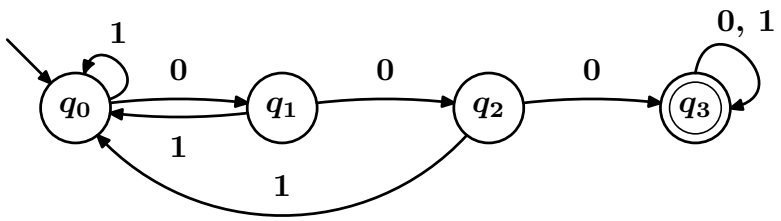
- dopo aver letto tre "0", l'automa accetterà la stringa qualunque siano i simboli ancora da leggere;
- q_3 assume quindi il ruolo di stato finale;
- q_0 assume il ruolo di stato iniziale;
- uno stato dal quale non si esce viene detto "pozzo".

Esercizio 1 (3)

La funzione di transizione può essere:

δ	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_3	q_0
q_3	q_3	q_3

e il grafico:



Esercizio 2 (1)

Costruire un DFA su $\Sigma = \{0, 1\}$ che accetti l'insieme di tutte le stringhe che se interpretate in notazione binaria risultino divisibili per 2.

Osservazioni:

- l'alfabeto è dato ($\Sigma = \{0, 1\}$);
- i numeri pari in notazione binaria terminano per 0;
- un numero in notazione binaria che termina per 0 è pari;
- il problema diventa costruire l'automa che accetta solo le stringhe binarie che terminano per 0.

Esercizio 2 (2)

Se, banalmente, indichiamo con q_i lo stato in cui viene a trovarsi l'automa dopo aver letto il simbolo i:

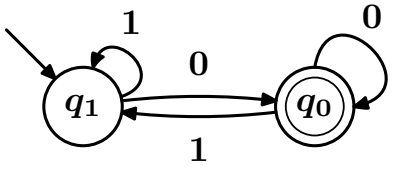
- ponendo q_0 come unico stato finale facciamo sì che l'automa accetti la stringa solo se l'ultimo simbolo che ha letto è stato 0;
- ponendo q_1 come stato iniziale evitiamo che venga accettata la stringa vuota.

Esercizio 2 (3)

Quindi, l'automa può essere formalizzato come:

- $Q = \{q_0, q_1\}$
- $\Sigma = \{0, 1\}$
- q_1 è lo stato iniziale
- $F = \{q_0\}$
-

δ	0	1
q_0	q_0	q_1
q_1	q_0	q_1



Esercizio 3 (1)

Costruire un DFA su $\Sigma = \{0, 1\}$ che accetti l'insieme di tutte le stringhe che hanno come penultimo simbolo "0".

Osservazioni:

- l'alfabeto è dato ($\Sigma = \{0, 1\}$);
- abbiamo bisogno di una finestra di due simboli (gli ultimi due simboli letti);
- consideriamo quindi quattro situazioni: 00, 01, 10 e 11.

Esercizio 3 (2)

Ad esempio, se l'automa si trova nello stato 01 e legge "1", si deve portare nello stato corrispondente a 11.

Con questa regola:

- le situazioni accettate saranno 00 e 01;
- scegliendo 11 come situazione iniziale, evitiamo di accettare stringhe con meno di due caratteri;
- codifichiamo con lo stato q_i la situazione in cui gli ultimi due simboli letti sono interpretati come la rappresentazione binaria di i.

Esercizio 3 (3)

Quindi, l'automa può essere formalizzato come:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- q_3 è lo stato iniziale
- $F = \{q_0, q_1\}$

-

δ	0	1
q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3

Automa a pila (1)

Un automa a pila è una macchina ideale che estende le possibilità degli automi a stati finiti.

Si tratta di un automa a stati finiti dotato di una struttura dati aggiuntiva per la memorizzazione delle informazioni, la pila:

- struttura dati di tipo Last In First Out (LIFO).

Automa a pila (2)

Un automa a pila è quindi composto da:

- controllo (stati + funzione di transizione);
- input (nastro);
- output (pila):
  - push(w): pone la stringa w sulla pila;
  - pop(w): preleva la stringa in testa alla pila;
  - empty: verifica se la pila è vuota (restituisce un valore booleano).

Funzionamento dell'automa a pila

L'automa legge dal nastro e dalla pila contemporaneamente.

Ogni passo di esecuzione comporta:

- la lettura di un simbolo da nastro e l'avanzamento della testina di una posizione;
- la lettura di un simbolo dalla testa della pila;
- il cambio dello stato di controllo;
- la scrittura in testa alla pila.

Nota: alcune operazioni possono essere saltate (lettura/scrittura della stringa vuota).

Stringhe accettate dall'automa a pila

L'automa ha due possibili condizioni per l'accettazione della stringa:

- se si trova in uno stato finale;
- se la pila è vuota.

Le due condizioni sono equivalenti:

- dato un automa che termina per pila vuota, se ne può costruire uno che termina per stato finale.

Linguaggi accettati dall'automa a pila

Il non-determinismo, aggiunge potenza espressiva all'automa a pila.

L'automa a pila deterministico riconosce un sottoinsieme dei linguaggi context-free (tipo 2).

L'automa a pila non-deterministico riconosce i linguaggi context-free.

Per esempio:

- un automa a pila riesce a riconoscere le parentesi ben accoppiate ($a^n b^n$);
- solo un automa non-deterministico riesce a riconoscere le stringhe palindrome.

Limiti degli automa a pila

Da cosa deriva la mancanza di espressività?

- la struttura dati di supporto, la pila, ha un accesso limitato, che si limita alla testa della pila;
  - come accedere al primo degli elementi inseriti?

Per poter riconoscere linguaggi di tipo 1 o 0 bisogna disporre di una struttura dati più flessibile.

Macchina di Turing

Macchina ideale pensata da Alan Turing negli anni '30 per formalizzare la nozione di agente di calcolo (computer).

Una macchina di Turing (MdT) è composta da:

- **nastro**: infinito, abilitato alla lettura ed alla scrittura;
- **controllo**: automa a stati finiti deterministico.

Funzionamento della MdT

Ad ogni passo di computazione, una MdT:

- legge il simbolo sul nastro in corrispondenza della testina;
- in base alla coppia stato-simbolo:
  - eventualmente modifica il simbolo;
  - sposta il nastro di una casella, a destra o a sinistra;
  - eventualmente cambia lo stato.

La computazione della MdT termina quando non è definita alcuna transizione per la coppia stato-simbolo.

Varianti della MdT

Esistono diverse varianti della MdT deterministica.

Ad esempio:

- multinastro;
- multitestina;
- nastro semi-infinito;
- non-deterministica.

Si può dimostrare che per ognuna di queste varianti si può costruire una MdT deterministica equivalente.

MdT universale

È possibile costruire una MdT che simuli una qualsiasi MdT!

Questa MdT viene detta universale.

All'inizio, sul nastro della MdT universale, opportunamente codificati, si trovano:

- la descrizione della MdT da simulare (cioè della parte di controllo);
- il suo input.

Linguaggi accettati

Una MdT riconosce i linguaggi di tipo 0.

Una MdT non deterministica con lunghezza del nastro limitata in modo proporzionale alla lunghezza dell'input riconosce i linguaggi di tipo 1.

Caratteristiche della MdT (1)

La MdT è stata ideata per modellare il processo di calcolo.

Infatti, ha le seguenti caratteristiche:

1. il controllo è costituito da un numero finito di istruzioni;
   - un algoritmo è di lunghezza finita;
2. la MdT è l'agente di calcolo che esegue l'elaborazione;
   - esiste un esecutore che può effettuare le istruzioni dell'algoritmo;
3. il controllo è affidato ad un automa deterministico;
   - il calcolo è deterministico;

Caratteristiche della MdT (2)

4. la MdT opera per passi discreti;
   - l'esecuzione avviene per passi discreti;
5. il nastro è un dispositivo di memorizzazione per l'MdT;
   - l'esecutore dispone di una memoria ausiliaria per immagazzinare i risultati intermedi dell'elaborazione;
6. i dati sono memorizzati sul nastro, il quale è illimitato;
   - non c'è limite alla lunghezza dei dati;
7. la capacità del nastro è illimitata;
   - non c'è limite alla quantità di memoria ausiliaria;

Caratteristiche della MdT (3)

8. le istruzioni che la MdT può eseguire ad ogni passo sono semplici e ben definite;
   - le istruzioni hanno una complessità finita;
9. non esiste nessuna limitazione al numero di passi di calcolo;
   - l'esecuzione termina dopo un numero di passi finito, ma illimitato;
0. è possibile creare una MdT che continua ad eseguire delle istruzioni senza terminare mai;
   - l'esecuzione può non terminare.

Calcolo

Una MdT calcola funzioni (su $\mathbb{N}$):

- funzioni intuitivamente calcolabili;
- l'input è sul nastro all'inizio del calcolo;
- l'output è sul nastro alla fine del calcolo.

Anche gli altri automi meno potenti possono essere modificati per effettuare dei calcoli (cioè generare un output).

Terminazione

La computazione di un automa a stati finiti o a pila termina sempre.

La computazione di una MdT per un certo dato di input può non terminare.

È l'equivalente algoritmico di una funzione non definita per un dato argomento (funzione parziale).

Linguaggi, grammatiche, automi (1)

Le grammatiche danno una descrizione generativa dei linguaggi.

Una grammatica è simile ad una teoria:

- assiomi (simboli iniziali);
- regole di inferenza (regole di produzione);
- predicati validi (stringhe del linguaggio);
- dimostrazioni (produzioni).

Linguaggi, grammatiche, automi (2)

Un automa realizza una funzione di riconoscimento della stringa come appartenente al linguaggio.

La MdT ha la potenza computazionale necessaria e sufficiente per l'accettazione dei linguaggi di tipo 0:

- se G è una grammatica di tipo 0 e $L = \mathcal{L}(G)$ è il linguaggio da essa generato, allora esiste una MdT, M_L, che accetta L;
- se L è un linguaggio accettato da una MdT, M_L, allora esiste una grammatica di tipo 0, G, tale che $L = \mathcal{L}(G)$.

Linguaggi e computabilità

Restringere lo studio della computabilità ai linguaggi non è restrittivo.

Inoltre, lo studio dei vari tipi di linguaggi è utile perché fornisce:

- una scala di complessità di calcolo;
- uno strumento di descrizione di algoritmi:
  - riconoscibilità (sintassi);
  - interpretazione (semantica).

Accettabilità e decidibilità

La non terminazione rende necessaria una ulteriore distinzione:

- un linguaggio viene detto accettabile (o riconoscibile) da un automa se esso è in grado di accettare tutte le sue parole;
  - nulla viene richiesto se la parola in input non appartiene al linguaggio dato;
- un linguaggio viene detto decidibile da un automa se esso è in grado di accettare tutte le sue parole e di rifiutare tutte le altre parole.

In altri termini, se un linguaggio è decidibile, lo deve essere anche il suo complemento.

Decidibilità dei linguaggi

I linguaggi di tipo 1, 2 e 3 sono decidibili.

I linguaggi di tipo 0 sono riconoscibili.

Tuttavia esistono linguaggi decidibili che non sono di tipo 1.

Ogni definizione finita della decidibilità non può comprendere tutti gli insiemi decidibili.

Problemi indecidibili (1)

È noto un certo insieme di problemi non decidibili:

- **halting problem**: non esiste alcuna MdT, M_H, che, data la descrizione di una MdT, M, e una stringa di input, w, decida se la computazione $M(w)$ termina, oppure no;
- non esiste nessuna MdT, M_E, che, data la descrizione di una MdT, M, possa decidere se il linguaggio riconosciuto da M, $\mathcal{L}(M)$, sia vuoto, oppure no;
- non esiste alcuna MdT, M_R, che, data la descrizione di una MdT, M, decida se il linguaggio da essa riconosciuto, $\mathcal{L}(M)$, sia di un tipo dato.

Problemi indecidibili (2)

Anche nella pratica si incontrano problemi indecidibili:

- un programma chiamerà una sua procedura nel corso dell'esecuzione per un dato input?
- una variabile assumerà un dato valore durante l'esecuzione?
  - uscirà dal ciclo?
- un dato programma, in presenza di un dato input, fornirà un particolare output?
- due programmi dati calcolano la stessa funzione?

Problemi indecidibili (3)

- un programma calcola una funzione costante?
  - il suo output è indipendente dall'input?
- un programma calcola una funzione totale?
  - termina per ogni input?
- un programma calcola una funzione positiva?
- una data formula del calcolo dei predicati è un teorema?
- una data formula dell'aritmetica, è un teorema?

Limite del calcolo

La MdT rappresenta il limite per i dispositivi finiti.

Non è l'unico paradigma del genere, ma tutti gli altri si sono dimostrati equivalenti.

La dimostrazione dell'indecidibilità di alcuni problemi equivale infatti al teorema di Gödel.

Tuttavia, tra la maggior parte dei problemi pratici sono decidibili (e, quindi, calcolabili).