Compilatori

Corso di Laurea in Informatica

Mauro Leoncini

A.A. 2024/2025

1. Automi finiti
   - Automi deterministici
   - Automi non deterministici
   - Subset construction
   - Realizzazione di un AFND da una espressione regolare

Compilatori

1. Automi finiti
   - Automi deterministici
   - Automi non deterministici
   - Subset construction
   - Realizzazione di un AFND da una espressione regolare

Ruolo degli automi finiti

- Gli *automi finiti*, spesso chiamati anche (un pò impropriamente) *automi a stati finiti* sono importanti *strumenti modellistici* che trovano amplissima applicazione in molti settori dell'Informatica
- Un numero elevato di strumenti di uso quotidiano sono modellabili come automi finiti: lavatrici e lavastoviglie, distributori di cibo e bevande, sistemi di controllo degli ascensori, distributori automatici di carburante, ...
- In questo corso siamo ineteressati agli automi finiti in quanto descrivono lo stesso insieme di linguaggi (i linguaggi regolari) descritti da espressioni regolari
- In particolare, da ogni espressione regolare è possibile costruire algoritmicamente un automa finito non deterministico da utilizzare nel riconoscimento di token

Definizione informale

- Un *automa finito deterministico* (*AFD*), può essere visto come un calcolatore elementare dotato di stato interno e supporto unidirezionale di input
- Il funzionamento dell'automa consiste di transizioni di stato a seguito della lettura di un simbolo da un dispositivo di input
- Ad ogni stato q sono in generale associate azioni (come la stampa di messaggi) che l'automa esegue quando transita in q

Un primo esempio (concreto ma molto semplificato)

- Un distributore eroga un dato prodotto al prezzo di 1 euro.
- Il distributore accetta monete da 50 centesimi e da 1 euro e può dare il resto
- Il funzionamento è modellabile come AFD con 2 soli stati

	Ingressi		
Stato attuale	50c	1€	Resto
A	B 0	A Prodotto	A 0
B	A Prodotto	B Prodotto	A 50c

Descrizione formale

Un *AFD* M è una quintupla

$$M = (\Sigma, Q, q_0, Q_f, \delta)\,,$$

in cui

- Σ è l'alfabeto di input
- Q è un insieme finito i cui elementi sono detti *stati* dell'automa
- q_0 è un elemento speciale di Q, detto *stato iniziale*
- $Q_f \subseteq Q$ è l'insieme degli *stati finali*, detti anche di *accettazione* dell'input
- δ è la funzione che determina le *transizioni* di stato. Essa mappa coppie $\langle stato, simbolo\rangle$ in stati: $\delta : Q \times \Sigma \to Q$

Computazioni di un automa

- Definibili in modo intuitivo come sequenza di *passi*
- Ad ogni passo, l'automa si trova in uno stato q (inizialmente $q = q_0$), legge un simbolo x dall'input e transita nello stato $\delta(q, x)$
- La computazione termina al verificarsi di una delle seguenti situazioni:
  - **mancanza di input** non vi sono più simboli di input, oppure
  - **transizione non specificata** in corrispondenza dello stato attuale e del simbolo letto, la funzione di transizione non è specificata
- Il numero di transizioni effettuate prima della terminazione è detto *lunghezza* della computazione e ne rappresenta una misura del costo

Rappresentazione di automi

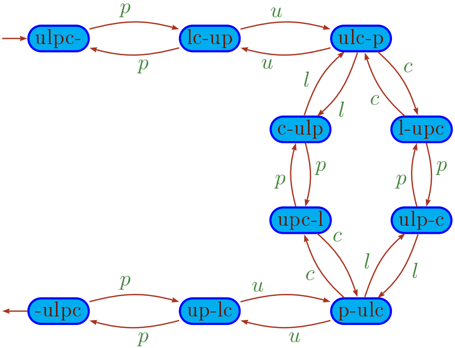
- Un utile formalismo (molto diffuso perché “intuitivo”) è quello dei *diagrammi di transizione*
- Un diagramma di transizione è un grafo i cui nodi ed archi rappresentano, rispettivamente, stati e transizioni
- Ogni arco è etichettato da un simbolo di input
- Lo stato iniziale viene evidenziato mediante una freccia entrante (e non uscente da alcun altro nodo)
- Gli stati finali sono indicati tramite doppia cerchiatura oppura da una freccia uscente (e non entrante in alcun altro nodo)

Un esempio introduttivo

- L'automa di cui presentiamo il diagramma di transizione (slide successiva) è tratto dal *Dragon Book* originale.
- Descrive la soluzione del ben noto problema del lupo, della pecora e del cavolo che una persona deve traghettare dalla sponda sinistra a quella destra di un fiume
- La barca usata dall'uomo può portare un solo altro "passeggero" (oltre all'uomo)
- Gli stati dell'automa descrivono una possibile situazione che consiste nell'indicare chi sta sulla sponda sinistra e chi sta su quella destra
- Le "transizioni" di stato possono indicare l'uomo (quando traghetta da solo) oppure il passeggero, che nello stato di partenza deve stare dalla stessa parte dell'uomo
- Il vincolo è che non succeda mai che cavolo e pecora, come pure lupo e pecora, stiano da soli sulla stessa sponda

Esempio introduttivo

Il lupo, la pecora e il cavolo



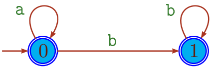
Di veda Hopcroft, Ullman (1979)

Automi riconoscitori di linguaggi

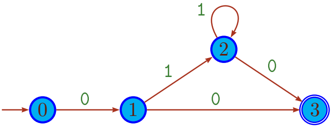
- Un *AFD riconosce* (o *accetta*) una stringa X in input se la computazione determinata dai caratteri di X termina in uno stato di Q_f per mancanza di input
- Ad esempio, nel caso del semplice rompicapo "Lupo, pecora e cavolo", la sequenza (stringa di input) `pulpcup` è riconosciuta dall'automa, mentre la stringa `pulcpup` non lo è
- Un *AFD* M *riconosce un linguaggio* $\mathcal{L}$ se e solo se $\mathcal{L}$ coincide con l'insieme delle stringhe riconosciute da M

Esempi

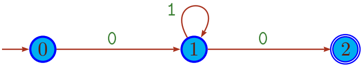
- Il seguente *AFD* $M_{n,m}$ riconosce il linguaggio $L_{n,m} = \{\mathtt{a}^n\mathtt{b}^m | n, m \geq 0\} = \mathbf{a}^*\mathbf{b}^*$



- Il seguente *AFD* M_{ss} riconosce il linguaggio $L_{ss} = \{\mathtt{0}\mathtt{1}^k\mathtt{0} | k \geq 0\} = \mathbf{01^*0}$

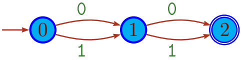


- Esiste un automa più semplice per L_{ss}?

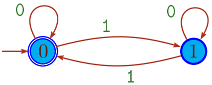


Esempi

- Il seguente *AFD* M_2 riconosce il linguaggio $L_2 = \{X \in \mathcal{B}^* : |X| = 2\}$



- Il seguente *AFD* M_{parity} riconosce il cosiddetto *linguaggio parità*, ovvero l'insieme delle stringhe $X \in \mathcal{B}^*$ che contengono un numero pari di 1



Rappresentazione di un AFD

- La rappresentazione di un AFD coincide "essenzialmente" con la rappresentazione della funzione di transizione δ
- Questa può essere semplicemente data come *tabella*, con $m = |Q|$ righe ed $n = |\Sigma|$ colonne
- Il consumo di memoria $\Theta(nm)$ può essere eccessivo se dalla maggior parte dei nodi del grafo di transizione escono relativamente pochi archi
- Tuttavia, almeno per il momento, non ci interessiamo al problema dell'efficienza

Simulazione di automi deterministici

- La simulazione del comportamento di un AFD $M = (\Sigma, Q, q_0, Q_f, \delta)$ è particolarmente semplice
- L'algoritmo riceve in ingresso M e l'input X per M e produce l'output che darebbe M su input X
- L'algoritmo presentato nella diapositiva seguente si riferisce alla simulazione di un generico automa riconoscitore
- Nella descrizione dell'algoritmo (in pseudocodice) si suppone che:
  - l'input X sia terminato dal carattere \$
  - tale carattere non appartiene all'alfabeto Σ dell'automa
  - se, per una determinata coppia stato-simbolo, $\langle q, x \rangle$, la funzione di transizione è indefinita, si pone $\delta(q, x) = \perp$

Simulazione di un AFD: algoritmo AFD-Sim

```
q ← q_0
x ← nextchar(X)
while (x ≠ $) do
    if δ(q, x) ≠ ⊥ then
        q ← δ(q, x)
    else
        reject
    x ← nextchar(X)
if q ∈ Q_f then
    accept
else
    reject
```

Simulazione di automi deterministici

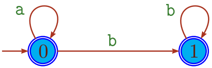
- Si noti che l'algoritmo è un vero e proprio *interprete*, ancorché molto semplice
- Infatti, esso prende in input un programma M (l'automa) e un input X per il programma, ed "esegue" M su input X
- È facile convincersi del fatto che il costo della simulazione è lineare nella lunghezza dell'input (a patto che si possa considerare costante il costo di valutazione della funzione δ, che tipicamente viene implementata mediante una tabella)

Qualche esercizio

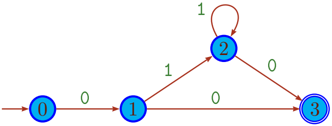
- Per ciascuno dei seguenti linguaggi, si fornisca un AFD che riconosce il linguaggio
  - $\{X | X \in \{\mathtt{0}, \mathtt{1}\}^*, X \text{ non contiene } \mathtt{0} \text{ adiacenti}\}$
  - $\{X | X \in \{\mathtt{0}, \mathtt{1}\}^*, \text{ogni sottostringa di lunghezza 3 in } X \text{ contiene almeno due } \mathtt{1}\}$
  - $\{X | X \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*, \text{due qualsiasi caratteri adiacenti in } X \text{ sono fra loro differenti}\}$

Espressioni regolari e automi

- Data un'espressione regolare $\mathcal{E}$, è possibile definire un automa $\mathcal{M}(\mathcal{E})$ che riconosce il linguaggio definito da $\mathcal{E}$, e viceversa
- Riconsideriamo qualche esempio già visto
- Il linguaggio $L_{n,m} = \mathbf{a}^*\mathbf{b}^*$ e l'automa $M_{n,m}$

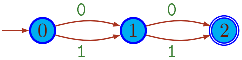


- Il linguaggio $L_{ss} = \mathbf{01}^*\mathbf{0}$ e l'automa M_{ss}

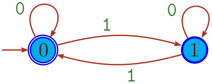


Espressioni regolari e automi

- Il linguaggio $L_2 = (\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1})$ e l'automa M_2



- Il linguaggio "parità", $L_{\text{parity}} = \mathbf{0}^* \left(\mathbf{1}\mathbf{0}^*\mathbf{1}\mathbf{0}^*\right)^*$, e l'automa M_{parity}



- Quale automa corrisponde all'espressione regolare $\mathbf{a}^*\mathbf{b}\mathbf{c}^* + \mathbf{c}^*\mathbf{a}^*\mathbf{b}$?

Compilatori

1. Automi finiti
   - Automi deterministici
   - Automi non deterministici
   - Subset construction
   - Realizzazione di un AFND da una espressione regolare

Automi non deterministici

- Data un'espressione regolare $\mathcal{E}$, abbiamo detto (anche se non dimostrato) che esiste un AFD $\mathcal{M}(\mathcal{E})$ ad essa "equivalente", cioè che riconosce lo stesso linguaggio definito dall'espressione
- In particolare, come abbiamo anticipato nella slide iniziale, è possibile automatizzare il processo che, a partire da $\mathcal{E}$, sintetizza $\mathcal{M}(\mathcal{E})$
- Tale automatizzazione risulta tuttavia più' "facile" se la si suddivide in due passi distinti:
  - dall'espressione regolare ad un *automa non deterministico* equivalente
  - dall'automa non deterministico all'automa deterministico equivalente
- I due passi saranno oggetto della nostra attenzione nelle due prossime sezioni
- In questa sezione dobbiamo invece preliminarmente rivolgerci al concetto di non determinismo negli automi finiti

Definizione di automa non deterministico

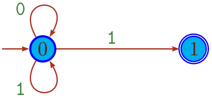
- Al non determinismo sono collegati alcuni dei problemi teorico/computazionali più importanti aperti in Informatica (e anche nella stessa Matematica), inclusa la famosa questione **P** *versus* **NP**
- Nel caso degli automi finiti il non determinismo non rende gli automi più potenti, cioè capaci di riconoscere un insieme più ampio di linguaggi
- Un automa finito si dice *non deterministico* (*AFND*) se, in almeno uno stato q, la transizione *non è univocamente determinata* dal simbolo di input
- In altri termini, dallo stato q e con lo stesso simbolo di input l'automa può transitare "non deterministicamente" in più di uno stato diverso

Transizioni non deterministiche

- Traducendo formalmente il concetto espresso nella slide precedente, possiamo dire che in un automa non deterministico ciò che cambia è la definizione della "funzione" di transizione, che mappa coppie ⟨stato,simbolo⟩ in *sottoinsiemi* (anziché elementi) di Q
- Si può essere tentati di immaginare una transizione *non deterministica* come guidata dalla probabilità: se da uno stato si diramano due o più transizioni, l'automa seguirà una di esse con una certa probabilità!
- Questo è errato! Il non determinismo è un concetto non riducibile a nozioni fisiche note (neppure quantistiche) e non rappresenta, almeno per ora, un modello computazionale realizzabile in pratica
- In relazione agli automi, ne potremo invece apprezzare l'utilità proprio come strumento teorico utile per porre e/o elucidare questioni computazionale concrete.
- Ma vediamo subito alcuni semplici esempi nella slide successiva

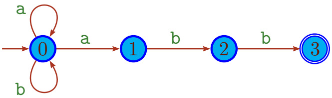
Esempi

- Il seguente automa è non deterministico perché nello stato 0 ci sono due transizioni etichettate con il simbolo 1



In altri termini, la funzione di transizione mappa la coppia $\langle 0, 1\rangle$ nell'insieme $\{0, 1\}$

- Un altro esempio di AFND:



Riconoscimento di stringhe da parte di un AFND

- Si dice che un *AFND* M *riconosce* una stringa X se e soltanto se *esiste* una sequenza di transizioni etichettata con i simboli di X che termina in uno stato finale
- È facile vedere che il primo automa della precedente trasparenza riconosce la stringa in input solo se questa termina con 1
- Si può anche facilmente dimostrare che, per ogni tale stringa, esiste una sequenza di transizioni (mosse) che porta l'automa nello stato 1
- Possiamo quindi concludere che l'automa riconosce il linguaggio $(\mathbf{0}|\mathbf{1})^*\mathbf{1}$

Riconoscimento di stringhe da parte di un AFND

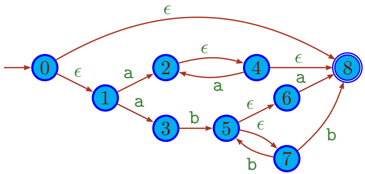
- Si noti come nello stato 0, e con input 1, l'automa debba decidere non deterministicamente se transitare nello stato 1 o restare nello stato 0
- Questo equivale a dire che l'automa deve decidere se è stato letto l'ultimo carattere 1
- L'automa del secondo esempio riconosce invece il linguaggio $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$
- Nello stato 0 e su input a, l'automa deve decidere se quella appena letta è l'ultima a nella stringa di input

Due immagini suggestive

- Alla luce della nozione ce abbiamo fornito di *riconoscimento* (o accettazione) di una stringa, se proprio volessimo pensare ad un computer non deterministico concreto, potremmo ricorrere a una delle due seguenti immagini:
  - un computer non deterministico è una macchina che, posta di fronte ad una scelta, "azzecca" sempre la mossa giusta (macchina *fortunata*); oppure
  - un computer non deterministico è una macchina in grado di eseguire in parallelo tutte le computazioni originate dalle varie opzioni non-deterministiche
- Vedremo che, nel caso degli automi finiti, la seconda opzione in qualche modo si avvicina a quanto possibile fare nel processo di simulazione (deterministica) di un automa non deterministico

ϵ-transizioni

- Una particolare "incarnazione" del non determinismo in un automa finito è costituita dalle cosiddette ϵ-transizioni
- Una tale transizione mappa elementi di $Q \times \{\epsilon\}$ in Q
- Il seguente diagramma costituisce un primo esempio di AFND con ϵ-transizioni



- Una ϵ-transizione che collega due nodi q ed r consente all'automa di passare da q ad r "senza consumare input"

Automi normalizzati

- Nel processo di sintesi di un automa deterministico da una data espressione regolare, che andremo ad analizzare a partire dalla prossima slide, utilizzeremo come passaggio “intermedio” proprio AFND che usano ϵ-transizioni come unica forma di non determinismo
- Addirittura, gli automi che scaturiscono dalla trasformazione di un'espressione regolare sono “normalizzati” nel senso seguente:

  > Da ogni nodo del diagramma di transizione (che non sia una *sink*) si dipartono o un singolo arco etichettato con un simbolo dell'alfabeto oppure al più due archi etichettati ϵ

- Nonostante siano un sotto-insieme degli automi non deterministici, tali automi hanno la stessa capacità riconoscitiva degli automi generali

Compilatori

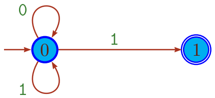
1 Automi finiti
- Automi deterministici
- Automi non deterministici
- Subset construction
- Realizzazione di un AFND da una espressione regolare

Equivalenza di AFD e AFND

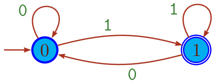
- Inizieremo trattando il secondo passo della trasformazione da espressione regolare ad ASFD
- Dimostreremo che per ogni arbitrario automa non deterministico esiste un automa deterministico che riconosce lo stesso linguaggio
- In questo caso, dunque, andiamo oltre ciò che sarà richiesto dopo avervisto anche il primo passo, che produce solo automi normalizzati
- Più precisamente, quel che faremo è di dimostrare che le computazioni di un generico automa non deterministico possono essere simulate da un automa deterministico.
- Il contrario è banale, poiché gli automi non deterministici generalizzano quelli deterministici
- Tutto ciò proverà dunque che automi finiti deterministici e non deterministici sono equivalenti

Esempi

- Vediamo dapprima un paio di esempi, relativi ad automi già introdotti
- Automi che riconoscono il linguaggio $(\mathbf{0}|\mathbf{1})^*\mathbf{1}$:
  - Automa non deterministico

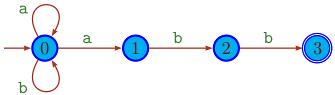


  - Automa deterministico equivalente

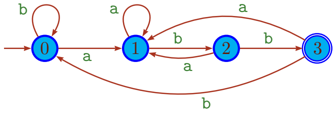


Esempi

- Automi che riconoscono il linguaggio $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$
  - Automa non deterministico



  - Automa deterministico equivalente

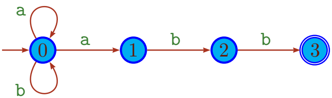


Costruzione dell'automa deterministico: idee

- Gli esempi appena visti sono stati costruiti in modo "ad-hoc", cioè non generalizzabile
- Ciò di cui abbiamo bisogno è invece di un processo che possa essere automatizzato, a partire da un qualsiasi AFND $\mathcal{N}$
- Il processo di costruzione che vedremo è noto come *subset construction*
- Osserviamo che se Q è l'insieme degli stati di $\mathcal{N}$ allora, dopo la lettura di i simboli dell'input, $\mathcal{N}$ può essersi arrestato oppure può trovarsi in uno degli stati di un qualche sotto-insieme di Q.
- L'idea della simulazione allora è in sé semplice: dato $\mathcal{N}$, l'automa (o meglio, <u>un</u> automa) equivalente $\mathcal{D}$ procede tenendo traccia proprio di tutti gli stati in cui può trovarsi $\mathcal{N}$ dopo aver letto i simboli di input $(i = 0, 1, ...)$

Costruzione dell'automa deterministico: esempio

- Ad esempio, su input aab l'automa:



  può trovarsi indifferentemente (o meglio, non deterministicamente) nello stato 0 o nello stato 2.
- In questo caso, l'automa deterministico equivalente $\mathcal{D}$ avrà, fra gli altri, uno stato che corrisponde al sottoinsieme $\{0, 2\}$
- Seguendo questa linea di ragionamento, possiamo dedurre che se $|Q| = m$ allora il numero di stati distinti di $\mathcal{D}$ sarà <u>al più</u> 2^m

Subset construction: premesse

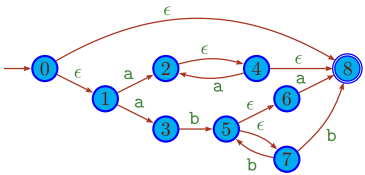
- Consideriamo un generico AFND $\mathcal{N} = (\Sigma, Q, q_0, Q_f, \delta)$
- Sia Z un sottoinsieme Q; la *chiusura* di Z rispetto ad ϵ-transizioni, indicata con ϵ-$CLOSURE(Z)$, è l'insieme ottenuto aggiungendo a Z tutti gli stati raggiungibili a partire da un qualsiasi stato $z \in Z$ seguendo transizioni etichettate con ϵ
- Indichiamo ora con $\mathcal{D} = (\Sigma, Q^d, q_0^d, Q_f^d, \delta^d)$ l'AFD equivalente a $\mathcal{N}$ che vogliamo "costruire"; chiaramente, dobbiamo specificare tutti gli elementi della quintupla eccetto l'alfabeto, che naturalmente è lo stesso di $\mathcal{N}$
- L'algoritmo, riportato nella slide seguente, definisce gli altri elementi in modo incrementale

Subset construction: algoritmo

1. Poniamo $q_0^d = \epsilon\text{-}CLOSURE(\{q_0\})$ e consideriamo tale stato *non marcato*
2. Ripetiamo i passi seguenti fintanto che esistono stati non marcati
3. Sia q^d uno stato non marcato scelto arbitrariamente
4. Per ogni simbolo $a \in \Sigma$, ripetiamo poi ciò che segue:
    1. esaminiamo tutti gli stati in q^d
    2. per ogni tale stato q consideriamo l'insieme di tutti gli stati direttamente raggiungibili da q su input a (ovvero seguendo tutti archi uscenti da q etichettati con a) e indichiamo tale insieme con t
    3. Poniamo $T = \epsilon\text{-}CLOSURE(t)$.
    4. Se T non coincide con nessumo degli stati già ottenuti in precedenza (marcati o non marcati) allora poniamo $\delta^d(q) = T$; inoltre, se T inlcude uno degli stati finali di $\mathcal{N}$ inseriamo T in Q_f^d.
    5. Inseriamo T nell'insieme degli stati non marcati
5. Etichettiamo q^d come stato marcato

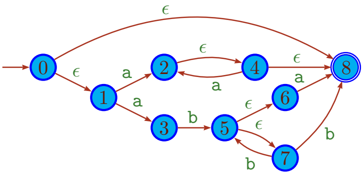
Esempio di subset construction

- Consideriamo il seguente automa non deterministico, già introdotto a proposito delle ϵ-transizioni:



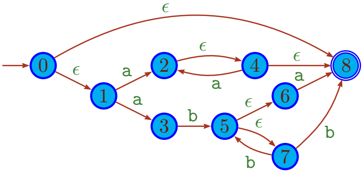
- Se indichiamo con A lo stato iniziale di $\mathcal{D}$, avremo $A = \{0, 1, 8\}$
- Si noti infatti che $\{0, 1, 8\} = \epsilon\text{-}CLOSURE(0)$

Esempio di subset construction (2)



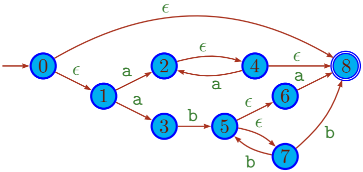
- Esaminiamo ora, a partire dagli stati di A, in quali stati si arriva su input a. Tali stati sono dapprima 2 e 3 ma poi, considerando le ϵ-transizioni, anche gli stati 4 e 8 (vale cioè $\epsilon\text{-}CLOSURE(\{2,3\}) = \{2,3,4,8\}$)
- Poniamo quindi $B = \{2,3,4,8\}$ e $\delta^d(A, \mathsf{a}) = B$
- L'analisi di A è terminata perché dai corrispondenti stati di $\mathcal{N}$ non esce alcuna transizione etichettata b

Esempio di subset construction (3)



- Lo stato $4 \in B$ è l'unico da cui si diparte una transizione etichettata con a
- Da esso si può ritornare nello stato 2 e quindi, mediante ϵ-transizioni, si può tornare nuovamente in 4 oppure in 8 (cioè ϵ-$CLOSURE(\{2\} = \{2, 4, 8\})$
- Poniamo quindi $C = \{2, 4, 8\}$ e $\delta^d(B, \mathsf{a}) = C$
- Analogamente, considerando il carattere b di input, avremo ancora un nuovo stato di $\mathcal{D}$, e precisamente $D = \{5, 6, 7\}$ e $\delta^d(B, \mathsf{b}) = D$

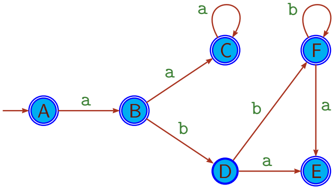
Esempio di subset construction (4)



- Continuando in questo modo introduciamo dapprima la transizione $\delta^d(C, \mathtt{a}) = C$;
- quindi lo stato $E = \{8\}$ e la transizione $\delta^d(D, \mathtt{a}) = E$;
- quindi lo stato $F = \{5, 6, 7, 8\}$ e la transizione $\delta^b(D, \mathtt{b}) = F$;
- quindi la transizione $\delta^b(F, \mathtt{a}) = E$;
- infine la transizione $\delta^b(F, \mathtt{b}) = F$.

Esempio di subset construction (5)

L'automa $\mathcal{D}$ risultante è:



dove

$$
\begin{array}{rcl}
A & = & \{0,1,8\} \\
B & = & \{2,3,4,8\} \\
C & = & \{2,4,8\} \\
D & = & \{5,6,7\} \\
E & = & \{8\} \\
F & = & \{5,6,7,8\}
\end{array}
$$

Esercizio progettuale

Implementare l'algoritmo di subset construction, dopo aver attentamento valutato come rappresentare, mediante struttura dati concrete, i vari elementi che descrivono gli automi.

Compilatori

1 Automi finiti
- Automi deterministici
- Automi non deterministici
- Subset construction
- Realizzazione di un AFND da una espressione regolare

Automi finiti ed espressioni regolari

- Vedremo dunque ora la costruzione che, a partire da una generica espressione regolare $\mathcal{E}$, produce un AFND che riconosce lo stesso linguaggio denotato da $\mathcal{E}$.
- Al di là del nostro insteresse per i compilatori, questa costruzione dimostra che gli automi finiti sono in grado di esprimere linguaggi che <u>includono</u> quelli regolari
- Più avanti vedremo anche che è vero anche il viceversa, e cioè che se un linguaggio è riconoscibile da un automa finito allora esso è regolare.
- Tutto ciò ci porterà a concludere che automi finiti ed espressioni regolari sono modi alternativi per descrivere linguaggi *regolari*

Generalità sulla costruzione

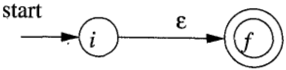
- L'idea alla base della costruzione è di analizzare (seguendo l'ordine imposto da regole di precedenza ed eventuali parentesi) la "struttura" di un'espressione regolare e di costruire i pezzi di automa corrispondenti
- I pezzi di automa saranno poi assemblati sempre tenendo conto delle precedenze
- Naturalmente, come in tutte le opere di assemblaggio, ci servono i *componenti base* da assemblare e questi sono gli automi che corrispondono alle alle espressioni regolari di base.
- Questi sono quindi i primi che andiamo ad analizzare

Generalità sulla costruzione (2)

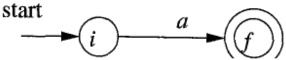
- Come già osservato, gli AFND che costruiremo avranno due soli "tipi" di stato:
  1. stati che chiameremo *deterministici*, dai quali esce una sola transizione etichettata con un simbolo dell'alfabeto Σ di input;
  2. stati *non deterministici* dai quali escono al più due transizioni etichettate ϵ.
- Inoltre, avranno un solo stato iniziale e un solo stato finale.
- Tutti gli schemi che vedremo sono tratti dal più volte citato *Dragon Book*

Costruzione dell'automa

- Poiché le espressioni regolari di base corrispondono alla stringa vuota e ai simboli dell'alfabeto, i "pezzi base" per la costruzione degli automi saranno quelli in grado di riconoscere ϵ e i singoli elementi di Σ.
- Avremo dunque il componente base



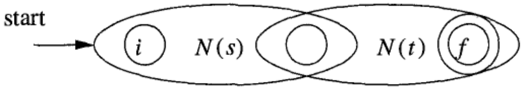
- come pure, per ogni a $\in \Sigma$, anche il componente base:



- Si noti quindi che, per ogni simbolo (lettera o ϵ) si introducono due nuovi stati.
- Nel seguito, indicheremo con $\mathcal{N}(s)$ l'automa corrispondente all'espressione regolare s.

Costruzione dell'automa (2)

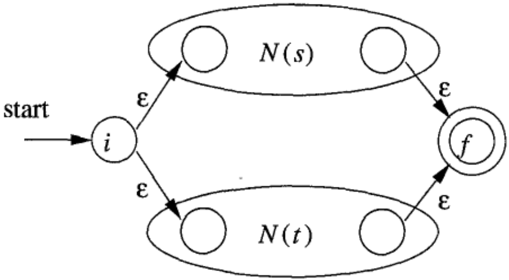
- Come sappiamo, ci sono 3 regole per la composizione delle espressioni regolari. Per ognuna di esse avremo dunque una corrispondente regola di composizione degli automi.
- La prima regola che consideriamo è quella per la concatenazione di due espressioni regolari s e t, illustrata dalla seguente schema:



- Poiché lo stato iniziale di $\mathcal{N}(t)$ coincide con lo stato finale di $\mathcal{N}(s)$, possiamo concludere che l'operazione di concatenazione addirittura riduce il numero di stati utilizzati.

Costruzione dell'automa (3)

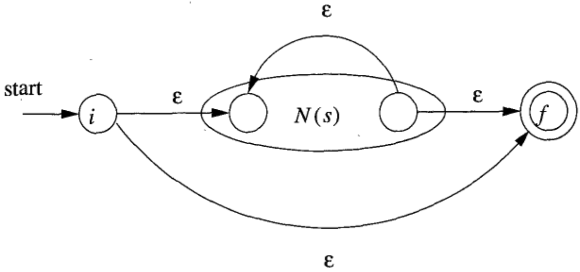
- La seconda regola è relativa all'unione di due espressioni regolari s e t:



- Un'operazione di unione introduce quindi due nuovi stati.

Costruzione dell'automa (4)

- L'ultima regola riguarda la chiusura di un'espressione regolare s:



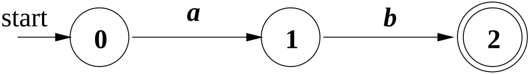
- Anche la chiusura introduce quindi due nuovi stati.

Costruzione dell'automa (5)

- Si noti che l'assemblaggio di due componenti può rendere necessaria una ridefinizione dei nomi degli stati (nel caso in cui i componenti assemblati siano stati etichettati allo stesso modo).
- La costruzione è corretta (proprietà che non dimostreremo formalmente) e gli automi risultanti godono di ulteriori interessanti proprietà:
    1. detto r il numero di operatori ed operandi presenti nell'espressione regolare (cioè la lunghezza della formula, parentesi escluse), il numero di stati è al più $2r$ mentre il numero di transizioni è al più $4r$;
    2. esiste un solo stato iniziale, senza transizioni entranti, e un solo stato finale, senza transizioni uscenti;
    3. se si eccettua il caso base del riconoscimento di ϵ ogni stato non deterministico ha esattamente due transizioni uscenti.

Esempio

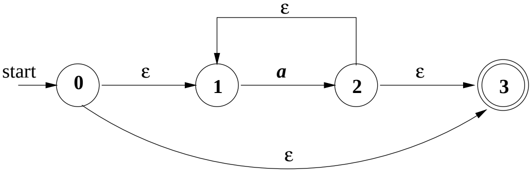
- Costruiamo l'automa corrispondente all'espressione regolare $\mathbf{b}(\mathbf{ab} + \mathbf{a}^*\mathbf{c})$.
- La costruzione deve naturalmente rispettare le regole di precedenza, e dunque riflette la seguente parentesizzazione: $\mathbf{b}((\mathbf{ab}) + ((\mathbf{a}^*)\mathbf{c}))$.
- Come primo passo costruiamo l'AFND per il riconoscimento di $\mathbf{ab}$ a partire dagli automi che riconoscono una sola lettera:



- Si noti che è stata operata una ridefinizione degli stati.

Esempio (2)

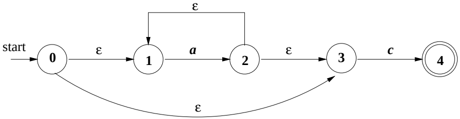
- Come secondo passo costruiamo l'automa per il riconoscimento di $\mathbf{a}^*$ a partire dall'automa che riconosce $\mathbf{a}$.

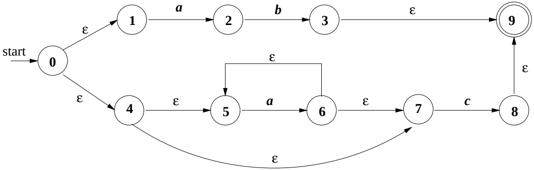


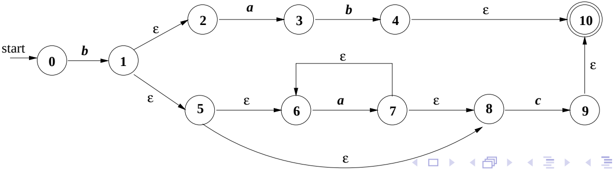
- Anche in questo caso si è operata una ridefinizione degli stati (in modo da avere sempre 0 come stato iniziale).

Esempio (3)

- I passi successivi creano gli automi per il riconoscimento, rispettivamente, di $\mathbf{a}^*\mathbf{c}$, di $\mathbf{ab} + \mathbf{a}^*\mathbf{c}$ e infine di $\mathbf{b}(\mathbf{ab} + \mathbf{a}^*\mathbf{c})$.

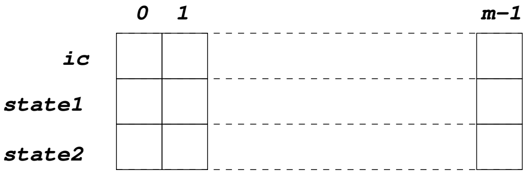






Rappresentazione interna

- Gli automi che derivano dalla costruzione appena descritta sono rappresentabili in modo efficiente dal punto di vista del consumo di memoria.
- La rappresentazione può essere fatta mediante tre *array paralleli*, che chiameremo `ic`, `state1` e `state2`:



- Le posizioni di indice i nei tre array rappresentano lo stato i o, meglio, le transizioni uscenti da i.

Rappresentazione interna (2)

- Il generico stato i può essere di uno dei seguenti tipi:
  1. *deterministico*, con un'unica transizione $i \rightarrow j$ etichettata $a \in \Sigma$. In tal caso avremo:

	i
ic	***a***
state1	***j***
state2	

  2. *non deterministico*, con due transizioni $i \rightarrow j$ e $i \rightarrow k$ etichettate ϵ (possiamo trattare l'unico caso di unica transizione ponendo $k = j$). In tal caso avremo:

	i
ic	ε
state1	***j***
state2	***k***

Esercizio progettuale

Implementare la trasformazione di espressioni regolari base in automi finiti non deterministici. Individuare possibili inconvenienti nel riconoscimento di lessemi che abbiano prefissi che sono a sua volta lessemi.