Linguaggi e compilatori

Corso di Laurea in Informatica

Mauro Leoncini

A.A. 2023/2024

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

Linguaggi e compilatori

Ruolo degli automi finiti

- Gli *automi finiti*, spesso chiamati anche (un pò impropriamente) *automi a stati finiti* sono importanti *strumenti modellistici*
- Un numero elevato di strumenti di uso quotidiano sono modellabili come automi finiti, dalle lavatrici alle macchine distributrici di alimenti e bevande.
- In questo corso siamo interessati agli automi finiti perché essi consentono di dare una formulazione alternativa, algoritmica, dello stesso insieme di linguaggi descrivibili mediante espressioni regolari.
- Più precisamente, esistono costruzioni algoritmiche che permettono di passare da un'espressione regolare ad un automa che riconosce lo stesso linguaggio descritto dalla e.r. e viceversa.
- La prima di queste costruzioni (da espressioni regolari ad automi) è in fondo proprio ciò che fa `Lex`.

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

Stato attuale	Ingressi 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 (nel caso di *automi riconoscitori*) stati 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

- La computazione di un automa è una sequenza finita 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:
  - **lettura completa della sequenza di input**, oppure
  - **funzione di transizione indefinita per lo stato attuale e il simbolo in lettura**
- Il numero di transizioni effettuate prima della terminazione è detto *lunghezza* della computazione e ne rappresenta una misura del costo
- Per quanto appena detto sulla terminazione, le computazioni di un automa su input costituito da n simboli hanno costo $O(n)$

Rappresentazione di automi

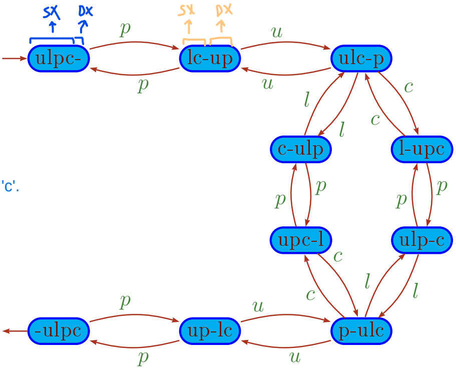
- Un formalismo comune e intuitivo per descrivere un automa è quello dei cosiddetti *diagrammi di transizione*
- Un diagramma di transizione è un grafo i cui nodi ed archi rappresentano, rispettivamente, stati e transizioni
- Ogni arco è etichettato da un simbolo di input
- Lo stato iniziale viene evidenziato 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



- Quando nella transizione c'è 'u' significa che si sposta solo l'umano.
- Quando nella transizione c'è 'l' , 'p' , 'c'. vuol dire che si sposta
  'l' + 'u' or
  'c' + 'u' or
  'p' + 'u'

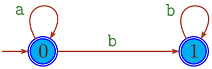
Di veda Hopcroft, Ullman (1979)

Automi riconoscitori di linguaggi

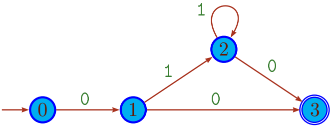
- Un *AFD riconosce* (o *accetta*) una stringa X in input se la computazione determinata dai caratteri di X termina in uno stato di Q_f dopo aver letto tutta la stringa di input
- Ad esempio, nel caso del semplice rompicapo "Lupo, pecora e cavolo", la sequenza (stringa di input) `pulpcup` è riconosciuta 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

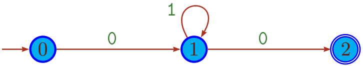
- Il seguente *AFD* $M_{n,m}$ riconosce il linguaggio $L_{n,m} = \{\mathtt{a}^n\mathtt{b}^m | n, m \geq 0\} = \mathbf{a}^*\mathbf{b}^*$



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

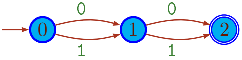


- Esiste un automa più semplice per L_{ss}?

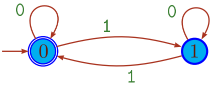


Esempi

- 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

- Una rappresentazione dell'automa utile per scopi implementativi è invece formulabile come *tabella*
- Detta tabella descrive precisamente la funzione di transizione
- Essa ha quindi una riga in corrispondenza di ogni stato e una colonna in corrispondenza di ogni simbolo di input.
- Nella cella individuata dallo stato (riga) q e dal simbolo (colonna) x è contenuto il valore $\delta(q, x)$, se la transizione è specificata oppure un simbolo speciale per indicare che il valore è indefinito
- Fissato lo stato iniziale in corrispondenza della prima riga, senza perdita di generaltà, per una descrizione completa dell'automa alla tabella manca quindi solo l'indicazione di quali siano gli stati finali

Simulazione di automi deterministici

- Disponendo della rappresentazione tabellare dell'automa e di un insieme (o lista) con gli stati finali, la simulazione del comportamento di un AFD M diviene particolarmente semplice
- Ogni passo consiste infatti di un semplice look-up alla tabella con il cambio di stato o l'eventuale arresto.
- L'algoritmo è presentato nella slide seguente in cui si suppone che:
  - l'automa in input è dato dalla tabella (chiamata ancora δ) e dall'insieme Q_f degli stati finali;
  - l'input X è terminato da un particolare carattere (nello specifico \$) che non fa parte dell'alfabeto Σ
  - il caso di valore indefinito per la funzione è rappresentato con un altro simbolo speciale, e precisamente $\perp$
  - la funzione `nextchar` restituisce il prossimo carattere nella stringa di input

Simulazione di un AFD: algoritmo AFD-Sim

Stato iniziale

```
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 effettivamente $O(n)$ a patto che si possa considerare costante il costo di accesso alla tabella

Qualche esercizio

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

Linguaggi e compilatori

Automi non deterministici

- Abbiamo già anticipato il fatto che, data un'espressione regolare $\mathcal{E}$, esiste e si può effettivamente "costruire" un AFD $\mathcal{M}(\mathcal{E})$ che riconosce lo stesso linguaggio definito da $\mathcal{E}$
- La costruzione cui abbiamo fatto riferimento risulta molto più' "agevole" suddividendola 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
- Prima però è bene chiarire che cosa si intenda per "non determinismo", un concetto per molti nuovo e che può creare fraintendimenti

Definizione di automa non deterministico

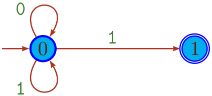
- Al non determinismo sono collegati alcuni dei problemi teorico/computazionali più importanti aperti in Informatica (e anche nella stessa Matematica), inclusa la famosa questione **P** *versus* **NP**
- Tale questione può essere ricondotta proprio alla nostra attuale incapacità di stabilere se modelli di calcolo generali (tipicamente la Macchina di Turing) siano più "potenti" quando abbiano a disposizione la possibilità di compiere scelte non-deterministiche
- Nel caso degli automi finiti la questione è risolta, nel senso che è noto che automi deterministici e non deterministici riconoscono lo stesso insieme di linguaggi con comparabile efficienza
- 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 computazionali concrete.
- Ma vediamo subito alcuni semplici esempi nella slide successiva

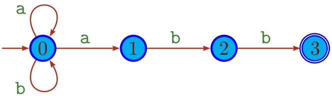
Esempi

- 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

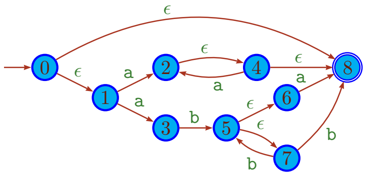
- Si noti come nello stato 0, e con input 1, l'automa debba decidere non deterministicamente se transitare nello stato 1 o restare nello stato 0
- Questo equivale a dire che l'automa deve decidere se è stato letto l'ultimo carattere 1
- L'automa del secondo esempio riconosce invece il linguaggio $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$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

Linguaggi e compilatori

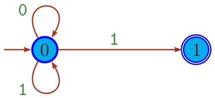
1. Automi finiti
   - Automi deterministici
   - Automi non deterministici
   - Subset construction
   - Realizzazione di un AFND da una espressione regolare

Equivalenza di AFD e AFND

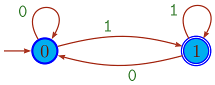
- Inizieremo trattando il secondo passo della trasformazione da espressione regolare ad ASFD
- Dimostreremo che per ogni arbitrario automa non deterministico esiste un automa deterministico che riconosce lo stesso linguaggio
- In questo caso, dunque, andiamo oltre ciò che sarà richiesto dopo avervisto 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

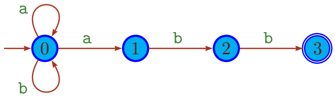


  - Automa deterministico equivalente

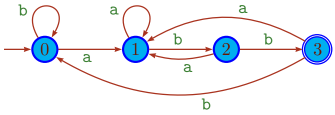


Esempi

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



  - Automa deterministico equivalente

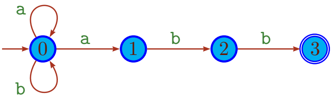


Costruzione dell'automa deterministico: idee

- 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 a l'automa:



  può trovarsi indifferentemente (o meglio, non deterministicamente) nello stato 0 oppure nello stato 1.
- In questo caso, l'automa deterministico equivalente $\mathcal{D}$ avrà dunque uno stato che corrisponde al sottoinsieme $\{0, 1\}$
- Analogamente, poiché su input aab l'automa non deterministico può trovarsi sia nello stato 0 che nello stato 2, il corrispondente automa deterministico dovrà prevedere uno stato corrispondente a $\{0, 2\}$
- In generale, ogni stato dell'automa deterministico corrisponderà ad un sotto-insieme di stati di quello non deterministico

Subset construction: premesse

- L'esempio fatto ci consente, prima ancora di mostrare la costruzione, di anticipare qualcosa riguardo la struttura dell'automa deterministico $\mathcal{D}$ che in base a tale costruzione corrisponde ad un automa finito $\mathcal{N}$ non deterministico.
  - L'alfabeto di input dei due automi sarà chiaramente lo stesso.
  - Riguardo gli stati, possiamo dire che, se $\mathcal{N}$ ne ha k, allora $\mathcal{D}$ ne potrà avere al più 2^k; tale infatti è il numero di sotto-insiemi di un insieme di k elementi
  - Gli stati di accettazione di $\mathcal{D}$ corrisponderanno necessariamente a sottoinsiemi che includono almeno uno degli stati finali di $\mathcal{N}$
  - Poiché lo stato iniziale di $\mathcal{D}$, chiamiamolo $O_{\mathcal{D}}$, dovrà ovviamente essere uno solo, e poiché la costruzione potrebbe invece produrre più sottoinsiemi che includono lo stato iniziale di $\mathcal{N}$, al momento non possiamo anticipare a quale di questi sottoinsiemi corrisponderà $O_{\mathcal{D}}$

Subset construction: "intuizioni" iniziali

Simboli , Stati , Stato_iniziale , Stato_Finale , Transizioni

- 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* Stack.push()
2. Ripetiamo i passi seguenti fintanto che esistono stati non marcati While
3. Sia Q^d uno stato non marcato scelto arbitrariamente Stack.pop()
4. Per ogni simbolo $a \in \Sigma$, ripetiamo poi ciò che segue: for i in range _Alphabet_
    1. esaminiamo tutti gli stati di $\mathcal{N}$ formano 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 alcuno degli stati già ottenuti in precedenza (marcati o non marcati) allora poniamo $\delta^d(Q) = T'$; inoltre, se T' include uno degli stati finali di $\mathcal{N}$ inseriamo T' in Q_f^d. Infine, inseriamo T' nell'insieme degli stati non marcati
5. Etichettiamo Q^d come stato marcato

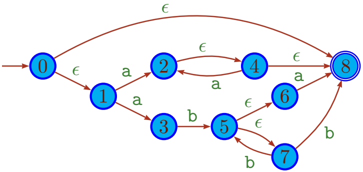
Esempio di subset construction (1)

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

Non ha costo

- Partendo dallo stato zero, prima di leggere qualsiasi input si può trovare nello stato:
  - 0
  - 1
  - 8

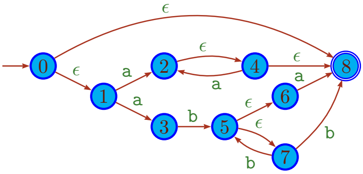
  Senza consumare input.



AUTOMA NON DETERMINISTICO

- 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)

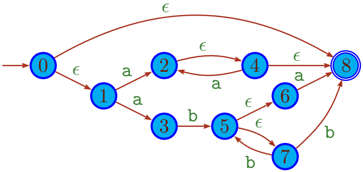


- Esaminiamo ora, a partire dagli stati di A, in quali stati si arriva su input a. Tali stati sono dapprima 2 e 3 ma poi, considerando le ϵ-transizioni, anche gli stati 4 e 8 (vale cioè $\epsilon\text{-}CLOSURE(\{2,3\}) = \{2,3,4,8\}$)

Da A in input "a" si va in B

- 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)



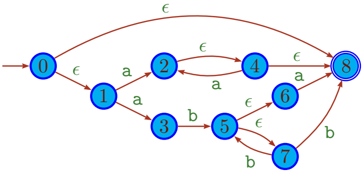
- Lo stato $4 \in B$ è l'unico da cui si diparte una transizione etichettata con a
- Da esso si può ritornare nello stato 2 e quindi, mediante ϵ-transizioni, si può tornare nuovamente in 4 oppure in 8 (cioè $\epsilon\text{-}CLOSURE(\{2\} = \{2, 4, 8\})$

Da B con input "a" vai a C

- Poniamo quindi $C = \{2, 4, 8\}$ e $\delta^d(B, \mathtt{a}) = C$
- Analogamente, considerando il carattere b di input, avremo ancora un nuovo stato di $\mathcal{D}$, e precisamente $D = \{5, 6, 7\}$ e $\delta^d(B, \mathtt{b}) = D$

Esempio di subset construction (4)

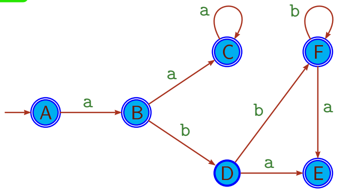
- Sappiamo che l'algoritmo finirà perchè c'è un bound sul numero di sottoinsiemi -> 2^n



- 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

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

Risultato finale -> Dopo aver eseguito subset construction.

AUTOMA DETERMINISTICO

Esercizio progettuale: 2 punti di incremento sul voto finale della prima parte

2 PUNTI!!!

- Implementare l'algoritmo di subset construction, dopo aver attentamento valutato come rappresentare gli automi
- Suggerimento (per iniziare a fare pratica di generic programming in C++): descrivere gli stati degli automi non-deterministici come numeri interi e quelli degli automi deterministici mediante ⟨set⟩ di interi
- Dettagli su drive per chi desidera partecipare a questo primo "progettino"

Linguaggi e 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

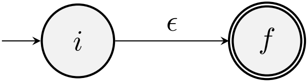
Generalità sulla costruzione (2)

- Come già osservato, gli AFND che costruiremo avranno due soli "tipi" di stato:
  1. stati che chiameremo *deterministici*, dai quali esce <u>una sola</u> transizione etichettata con un simbolo dell'alfabeto Σ di input;
  2. stati *non deterministici* dai quali escono <u>al più</u> 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

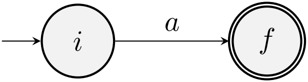
In pratica spiega come dall'albero dell'espressione regolare si genera l'automa

- Le espressioni regolari alla base della "costruzione ricorsiva" corrispondono alla stringa vuota e ai simboli dell'alfabeto. Ne consegue che gli *automi base* saranno



bab|a*c → Automa

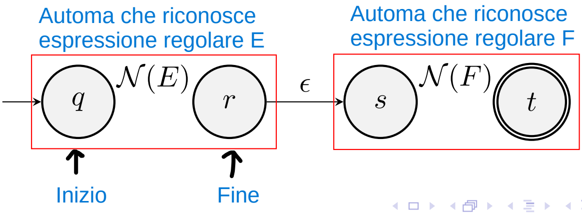
e, per ogni elemento $a \in \Sigma$,



- Per ogni simbolo (lettera o ϵ) si introducono quindi due stati.
- Nel seguito, indicheremo con $\mathcal{N}(E)$ l'AFND corrispondente all'espressione regolare E.

Costruzione dell'automa (2) REGOLA1--> CONCATENAZIONE

- A ciascuna regola di composizione delle espressioni regolari corrisponde una regola di composizione degli automi.
- Iniziamo dalla concatenazione. Consideriamo gli automi $\mathcal{N}(E)$ e $\mathcal{N}(F)$ corrispondenti alle due espressioni regolari E e F.
- Nello schema seguente (e in quelli successivi) di ogni automa componente mettiamo in evidenza solo gli stati iniziale e finale.
- L'automa corrispondenre all'espressione regolare EF si ottiene semplicemente collegando lo stato finale di $\mathcal{N}(E)$ con lo stato finale di $\mathcal{N}(F)$



Costruzione dell'automa (3)

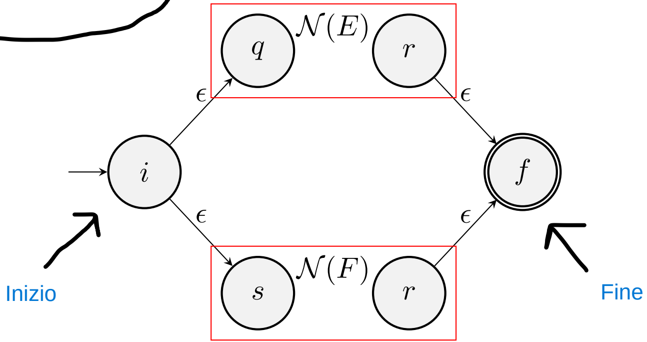
REGOLA1--> CONCATENAZIONE

- Lo stato iniziale del nuovo automa $\mathcal{N}(EF)$ coincide con lo stato iniziale di $\mathcal{N}(E)$ mentre lo stato finale di $\mathcal{N}(EF)$ coincide con quello finale di $\mathcal{N}(F)$
- Si noti che l'automa risultato della concatenazione ha un numero di stati che è la somma degli stati dei cue automi concatenati.
- Si noti inoltre che o i nomi degli stati di $\mathcal{N}(E)$ e $\mathcal{N}(F)$ sono disgiunti oppure in $\mathcal{N}(EF)$ è necessaria una qualche ri-denominazione. Questo varrà anche per le altre due costruzioni.

Costruzione dell'automa (4)

REGOLA2--> UNIONE

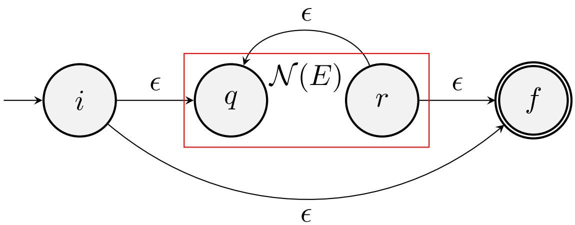
- La seconda regola riguarda l'unione di due espressioni regolari E e F
- In questo caso vengono introdotti due nuovi stati, che diventano lo stato iniziale di $\mathcal{N}(E|F)$, collegato agli stati iniziali di $\mathcal{N}(E)$ e $\mathcal{N}(F)$, e quello finale, a sua volta raggiunto dagli stati finali di $\mathcal{N}(E)$ e $\mathcal{N}(F)$



Costruzione dell'automa (5)

REGOLA3--> CHIUSURA

- L'ultima regola riguarda la chiusura di un'espressione regolare E
- L'automa $\mathcal{N}(E^*)$ prevede due nuovi stati, indicati con i e f, che diventano rispettivamente lo stato iniziale e finale
- i viene collegato allo stato iniziale di $\mathcal{N}(E)$ mentre lo stato finale di $\mathcal{N}(E)$ viene collegato a f
- Viene inoltre inserito un collegamento fra gli stati finale e iniziale dell'automa $\mathcal{N}(E)$



Costruzione dell'automa (6)

- Data una generica espressione regolare E, l'automa $\mathcal{N}(E)$ viene realizzato applicando le costruzioni appena viste nell'ordine naturale determinato dalle regole di precedenza:
  - prima la chiusura riflessiva,
  - poi la concatenazione,
  - infine l'unione
- Come nel caso delle espressioni aritmetiche, possono poi essere presenti le parentesi, che naturalmente sono utili solo nel caso in cui si voglia alterare l'ordine naturale.
- Ad esempio, la coppia di parentesi nell'espressione $\boxed{\mathbf{b(ab|a^*c)}}$ gioca un ruolo importante; con essa l'espressione può essere riscritta come:

$$\boxed{\mathbf{bab|ba^*c}}$$

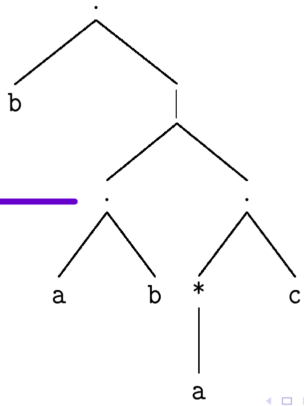
mentre senza di essa sarebbe

$$\mathbf{bab|a^*c}$$

Costruzione dell'automa (7)

- L'ordine di interpretazione di un'espressione regolare può essere efficacemente descritta mediante un *Abstact Syntax Tree*
- Ad esempio, sempre con riferimento all'espressione $\mathbf{b}(\mathbf{ab}|\mathbf{a}^*\mathbf{c})$, la corretta interpretazione corrisponde all'albero

Concatenazione



- Più è in basso, più la prorità è alta.

Esempio completo

- Costruiamo dunque l'automa corrispondente all'espressione regolare $\mathbf{b}(\mathbf{ab}|\mathbf{a}^*\mathbf{c})$ procedendo secondo i passi elementari determinati da una semplice visita *in ordine posticipato* del suo AST
- Come primo passo "costruiamo" l'AFND per il riconoscimento di $\mathbf{b}$.
- Questo in realtà significa semplicemente "prendere" l'automa che riconosce a lettera b:

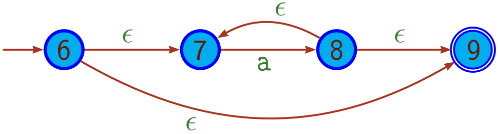
1. CASO BASE

- Messo temporaneamente "da parte" il precedente automa, l'ordine posticipato di visita impone di considerare due automi base per a e ancora per b e poi di concatenarli

2. CONCATENAZIONE

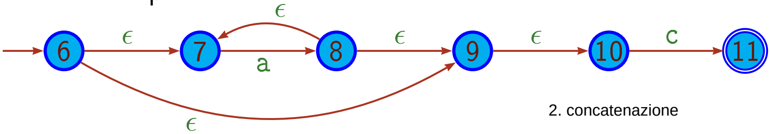
Esempio completo (2)

- Proseguendo nella visita, è ora necessario considerare un altro automa base per a e costruire l'automa per la sua chiusura riflessiva e transitiva



3. Chiusura riflessiva

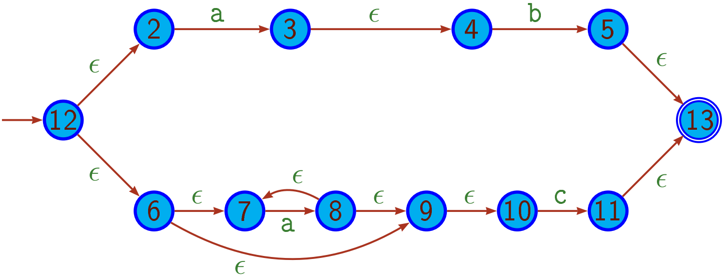
- Il passaggio successivo, dopo aver "prelevato" (dal pool degli automi base) un automa per la lettera c, è la concatenazione di quest'ultimo con l'automa per a*



2. concatenazione

Esempio completo (3)

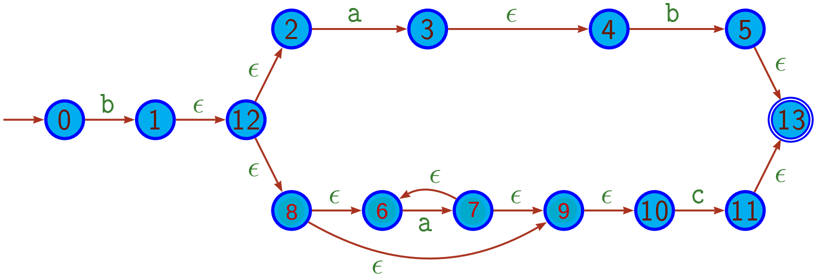
- La visita in ordine posticipato ci porta ora a realizzare l'automa corrispondente all'unione delle due espressioni regolari ab e a*c



Esempio completo (4)

AUTOMA FINALE : SOLUZIONE

- L'ultimo passo consiste nel concatenare l'automa per b, messo a suo tempo "da parte", con l'ultimo componente appena costruito, cioè l'automa per ab|a*c



Stati e transizioni

- Ci domandiamo: potevamo in qualche modo anticipare il numero di stati e di transizioni, rispettivamente 14 e 16, che possiamo "contare" nell'automa appena costruito?
- Osserviamo che
  - ogni lettera dell'espressione regolare porta ad utilizzare 1 automa "base" con 2 stati e una transizione
  - La concatenazione di due automi aumenta di un'unità il numero di transizioni
  - L'unione di due automi e la chiusura di un automa sono costruzioni che portano a introdurre 2 nuovi stati e di 4 nuove transizioni ciascuna
- Nell'espressione $\mathbf{b}(\mathbf{ab}|\mathbf{a}^*\mathbf{c})$ compaiono 5 lettere, che richiedono 10 stati e 5 transizioni.
- A queste dobbiamo aggiungere 3 transizioni per le tre concatenazioni e 4 stati e 8 transizioni per le due operazioni di unione e chiusura.
- I totali sono proprio i 14 stati e le 16 transizioni dell'automa effettivamente costruito

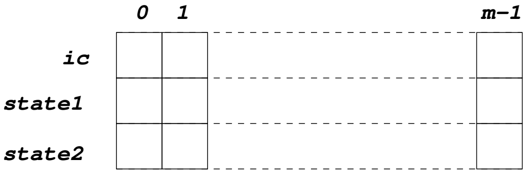
Rappresentazione interna $\longrightarrow$ 3 array

- Osserviamo innanzitutto che gli automi risultato della costruzione descritta hanno le seguenti caratteristiche “strutturali:
  1. hanno un solo stato iniziale, senza transizioni entranti, e un solo stato di accettazione, senza transizioni uscenti;
  2. ad esclusione dello stato di accettazione, ogni stato può avere o una sola transizione uscente etichettata con un simbolo dell'alfabeto, oppure una o due transizioni uscenti etichettate ϵ
- Chiameremo *deterministici* gli stati dai quali esce una transizione etichettata con un simbolo dell'alfabeto e stati *non deterministici* gli altri
- Le caratteristiche appena enucleate consentono di rappresentare gli automi in modo efficiente dal punto di vista del consumo di memoria

Rappresentazione interna (2) (Automa deterministico e non)

- La rappresentazione può essere fatta mediante tre *array paralleli*, che chiameremo `ic`, `state1` e `state2`:

m stati



- Le posizioni di indice i nei tre array rappresentano lo stato i o, meglio, le transizioni uscenti da i.
- Osserviamo poi che è sempre possibile numerare gli stati in modo che, se l'automa ha m stati, allora allo stato iniziale viene attribuito l'indice 0 e allo stato finale viene attribuito l'indice $m-1$, corrispondenti alla prima e all'ultima posizione degli array paralleli

Rappresentazione interna (3)

- La generica posizione i dell'array può corrispondere:
  1. ad uno stato *deterministico*, quindi con un'unica transizione $i \to j$ etichettata $a \in \Sigma$. In tal caso avremo:

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

     in cui la corrispondente "entry" nell'array `state2` è non significativa
  2. ad uno stato *non deterministico*, con al più due transizioni etichettate ϵ. Con riferimento alla figura, si assume che i casi $j \neq k$ e $j = k$ rappresentino rispettivamente stati con due o una transizione uscente

	~~i~~	0
ic	ε	
state1	~~j~~	1
state2	~~k~~	8

Esercizio progettuale completo: 3 punti di incremento sul voto finale della prima parte

- Implementare la trasformazione completa da ER a AFD
- Si può supporre che l'espressione regolare in input sia fornita già nella sua rappresentazione ad albero; questo perché il passaggio dalla rappresentazione lineare a quella ad albero è precisamente il compito del parsing, che affronteremo nelle prossime lezioni.
- Una rappresentazione "lineare" di un albero può essere la seguente.
- Per ogni nodo X
  - se X è una foglia (che quindi denota una lettera) allora la sua rappresentazione è X
  - se X è un nodo interno (che quindi rappresenta uno dei tre possibili operatori: concatenazione, unione o chiusura), allora la sua rappresentazione è:

$$\begin{array}{rl} (X\ (SX)) & \text{se } X = * \text{ (operatore di chiusura)} \\ (X\ (SX)\ (DX)) & \text{se } X \in \{\cdot, |\} \end{array}$$

// concatenazione o or

Rappresentazione lineare di un albero

- Ad esempio, l'espressione $\mathbf{b(ab|a^*c)}$ usata come esempio verrebbe rappresentata nel modo seguente

$$\boxed{(\cdot(\mathrm{b})(|(\cdot(\mathrm{a})(\mathrm{b}))(\cdot(*(\mathrm{a}))(\mathrm{c}))))}$$

- Come ulteriore esempio, consideriamo l'espressione sull'alfabeto $\{0\ 1\}$ che denota il linguaggio composto dalle stringhe che contengono almeno un 1, cioè $(0|1)^*1(0|1)^*$

$$(\cdot(\cdot(*(|(0)(1)))(1))(*(|(0)(1))))$$

- Anche per questa seconda parte del progetto i dettagli si trovano nella cartella condivisa su drive.