

Linguaggi formali e compilazione

Corso di Laurea in Informatica

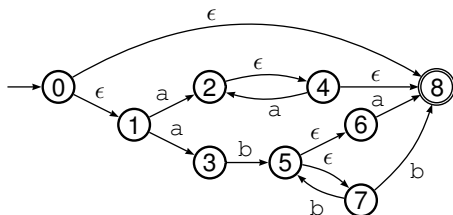
A.A. 2015/2016

Automi a stati finiti

Dalle espressioni regolari agli automi

Generalità delle ϵ -transizioni

- Gli automi non deterministici, come abbiamo visto, possono avere due sorgenti di non determinismo:



1. da uno stesso stato possono uscire più transizioni etichettate con lo stesso simbolo;
2. da uno stato possono anche uscire una o più transizioni etichettate con ϵ .

Generalità delle ϵ -transizioni

Automati a stati finiti

Dalle espressioni regolari
agli automi

- ▶ Si può dimostrare che per ogni ASFND con ϵ -transizioni esiste un ASFND senza ϵ -transizioni equivalente (cioè che riconosce lo stesso linguaggio).
- ▶ È vero anche il contrario, e cioè che dato un automa non deterministico generale (in cui le due “forme” di non determinismo sono presenti) ne esiste uno equivalente in cui sono presenti solo ϵ -transizioni.
- ▶ La trasformazione da espressioni regolari ad automi, che vedremo subito dopo, genera precisamente automi di questo secondo tipo: cioè automi non deterministici in cui il non determinismo è dato solo dalla presenza di ϵ -transizioni.

Automi finiti ed espressioni regolari

Automi a stati finiti

Dalle espressioni regolari
agli automi

- ▶ Vedremo dunque ora la costruzione che, a partire da una generica espressione regolare \mathcal{E} , produce un ASFND che riconosce lo stesso linguaggio denotato da \mathcal{E} .
- ▶ Più avanti (per una sorta di “completezza” del quadro generale sui linguaggi regolari) vedremo anche che è vero il viceversa, e cioè che se un linguaggio è riconoscibile da un automa a stati finiti allora esso è regolare.
- ▶ Potremo quindi concludere che *un linguaggio è regolare se e solo esiste un automa finito che lo riconosce.*

Generalità sulla costruzione

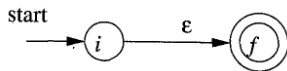
- ▶ L'idea alla base della costruzione è di *analizzare la struttura* dell'espressione regolare e di costruire (e poi assemblare) pezzi di automa corrispondenti via via più complessi.
- ▶ Naturalmente ci servono (come in tutte le opere di assemblaggio) i *componenti base*, e questi sono gli automi corrispondenti alle espressioni regolari di base.
- ▶ Dopodiché avremo bisogno di *regole per comporre gli automi* di base secondo che rifletteranno le regole di definizione delle espressioni regolari (e cioè unione, concatenazione e chiusura).

Generalità sulla costruzione (continua)

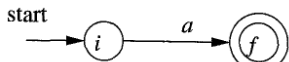
- ▶ Come già anticipato, gli ASFND che verranno generati avranno ϵ -transizioni come unica sorgente di non determinismo.
- ▶ Conviene però procedere (per ragioni che diventeranno chiare in seguito) ad una ulteriore *normalizzazione* degli automi.
- ▶ Gli automi che costruiremo avranno dunque due soli “tipi” di stato:
 1. stati *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 libro *Compilatori: Principi, tecniche e strumenti* (citato sul sito Web).

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 proprio (e solo) in grado di riconoscere ϵ e i singoli elementi di Σ .
- ▶ Avremo dunque il componente base



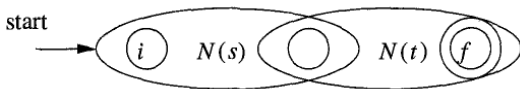
- ▶ come pure, per ogni $a \in \Sigma$, anche il componente base:



- ▶ 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'automata (continua)

- ▶ 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:



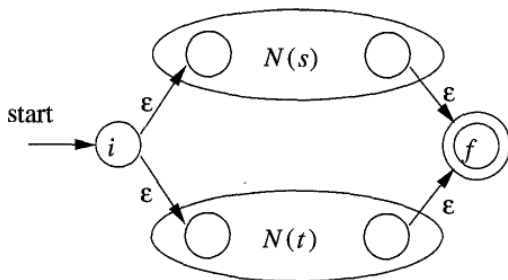
- ▶ Poiché lo stato iniziale di $N(t)$ coincide con lo stato finale di $N(s)$, possiamo concludere che l'operazione di concatenazione addirittura riduce il numero di stati utilizzati.

Costruzione dell'automata (continua)

Automati a stati finiti

Dalle espressioni regolari
agli automi

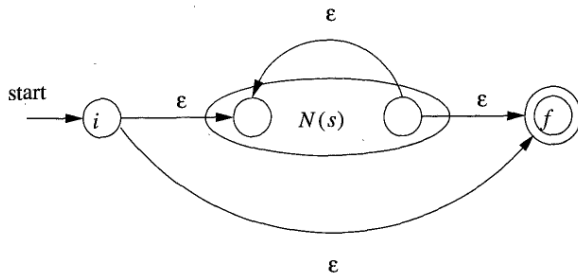
- ▶ La seconda regola è relativa all'unione di due espressioni regolari s e t :



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

Costruzione dell'automa (continua)

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



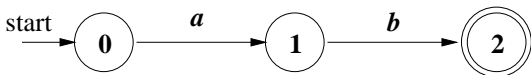
- Anche la chiusura introduce quindi due nuovi stati.

Costruzione dell'automa (continua)

- ▶ Si noti che l'assemblaggio di due componenti può rendere necessaria una ridefinizione dei nomi degli stati (nel caso in cui i componenti assemblati abbiano 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

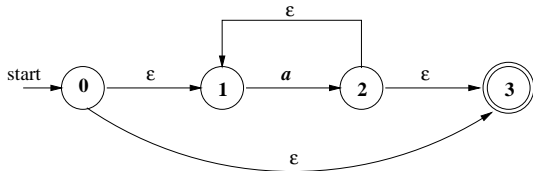
- ▶ Costruiamo l'automato corrispondente all'espressione regolare $\mathbf{b(ab + a^*c)}$.
- ▶ La costruzione deve naturalmente rispettare le regole di precedenza, e dunque riflette la seguente parentesizzazione: $\mathbf{b((ab) + ((a^*)c))}$.
- ▶ Come primo passo costruiamo l'ASFND per il riconoscimento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



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

Esempio (continua)

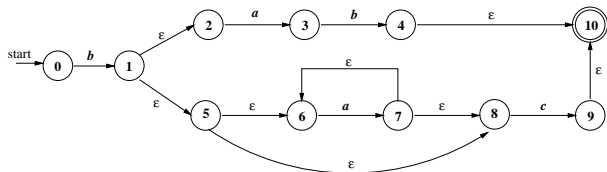
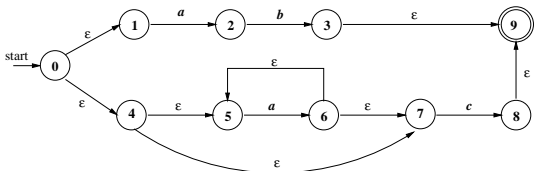
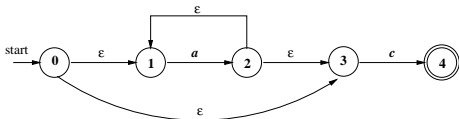
- ▶ Come secondo passo costruiamo l'automa per il riconoscimento di a^* a partire dall'automa che riconosce a .



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

Esempio (continua)

- I passi successivi consistono nella creazione degli automi per il riconoscimento, rispettivamente, di a^*c , di $ab + a^*c$ e infine di $b(ab + a^*c)$.

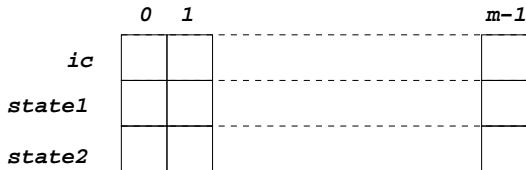


Il punto della situazione

- ▶ Data una qualunque espressione regolare \mathcal{E} , siamo in grado (per il momento, “a mano”) di definire un ASFND $\mathcal{N} = \mathcal{N}(\mathcal{E})$ che riconosce il linguaggio denotato da \mathcal{E} .
- ▶ Dato \mathcal{N} , siamo in grado, mediante l’algoritmo di *Subset construction*, di costruire un automa deterministico equivalente \mathcal{D} .
- ▶ Infine, siamo in grado di simulare \mathcal{D} .
- ▶ Complessivamente disponiamo di un processo algoritmico che consente di riconoscere stringhe descritte da (che hanno un match con) ogni data espressione regolare.
- ▶ L’automatizzazione del processo completo si potrà attuare solo dopo aver introdotto il parsing.

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 (continua)

► 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	

Esercizi proposti

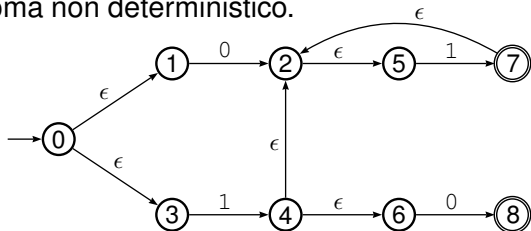
- ▶ Si fornisca un automa a stati finiti che riconosce il linguaggio denotato dalla seguente espressione regolare:

$$0^* (10^* 10^*)^*$$

- ▶ Si costruisca l'ASFND corrispondente (secondo la costruzione vista) a ciascuna delle seguenti espressioni regolari:
 - ▶ $a^*b(a \mid b)$;
 - ▶ $1^*(0 \mid \epsilon)1^*(0 \mid \epsilon)1^*$;
 - ▶ $(ba \mid b)^*ba^*(ab \mid b)$.
- ▶ Si dimostri che nessuno dei seguenti linguaggi è regolare:
 - ▶ $\{a^n b^n \mid n \geq 0\}$;
 - ▶ $\{a^n b^m \mid m > n\}$;
 - ▶ $\{1^n \mid n \text{ primo}\}$.

Alcuni esercizi dati all'esame

- ▶ Utilizzando la costruzione vista a lezione, si mostri l'automa deterministico equivalente al seguente automa non deterministico.



- ▶ Sia $\Sigma = \{x, y, z\}$ e si consideri il linguaggio $L \subseteq \Sigma^*$ così definito:

$$L = \{xy\alpha yx \mid \alpha \in \Sigma^*, \nexists \beta, \gamma \in \Sigma^* \text{ t.c. } \alpha = \beta y x \gamma\}$$

α non può cioè contenere yx come sottostringa. L modella i commenti in linguaggi come C o C++. Si costruisca un ASFD che riconosce le stringhe di L e si scriva poi un'e.r. che definisce L .