Linguaggi di Programmazione e Compilatori

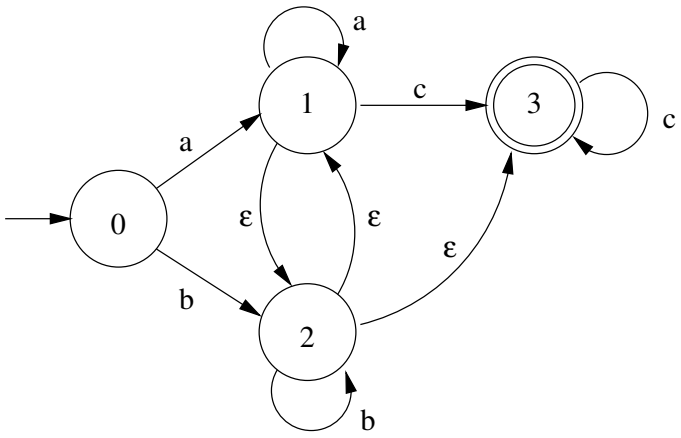
I° Appello del 9/7/2004

Scrivere **in stampatello** COGNOME e NOME su ogni foglio consegnato e sul testo, che va consegnato insieme al compito.

Nota Nel testo le espressioni regolari vengono scritte con la usuale convenzione di precedenza: l'operatore * lega più della concatenazione che, a sua volta, lega più dell'operatore |. Inoltre si possono usare le solite abbreviazioni ($^+$ e ?).

ESERCIZIO 1 (3 punti)

Si scriva un'espressione regolare che denoti il linguaggio accettato dal seguente automa:



SOLUZIONE

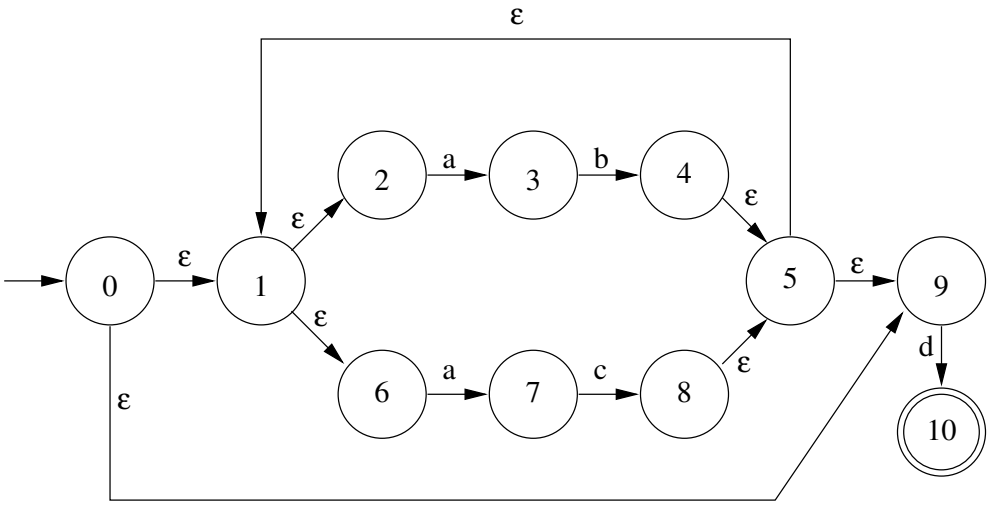
L'espressione è $(a|b)^+c^*$.

ESERCIZIO 2 (3 punti)

Si costruisca, attraverso la costruzione di Thompson, un NFA che accetti il linguaggio denotato dall'espressione regolare $(ab|ac)^*d$.

SOLUZIONE

La struttura sintattica dell'espressione prevede dapprima la stella, poi una concatenazione e infine un or tra due concatenazioni. Seguendo la definizione induttiva della costruzione di Thompson si ottiene il seguente NFA:

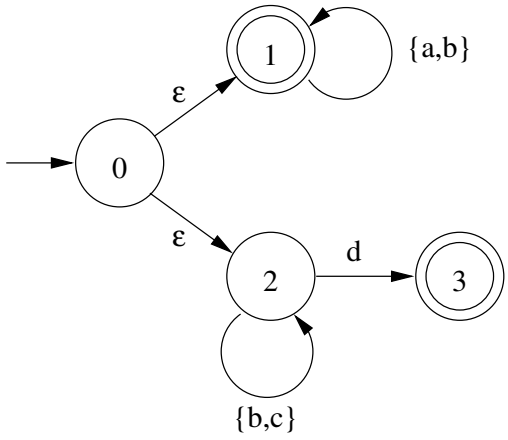


ESERCIZIO 3 (4 punti)

Si scriva un automa minimo per il linguaggio $(a|b)^* \mid (b|c)^*d$.

SOLUZIONE

Possiamo dapprima utilizzare il non determinismo per definire in maniera semplice un NFA per il linguaggio (in questo modo evitiamo eventuali errori dovuti alla complessità di scrivere direttamente un automa deterministico). L'automa è il seguente:



A questo punto usiamo la costruzione dei sottoinsiemi per rendere deterministico l'automa e poi minimizziamo (se necessario).

La tabella *move* dell'automa ottenuta con la costruzione dei sottoinsiemi è la seguente (gli stati di accettazione sono $\{A, B, C, E\}$):

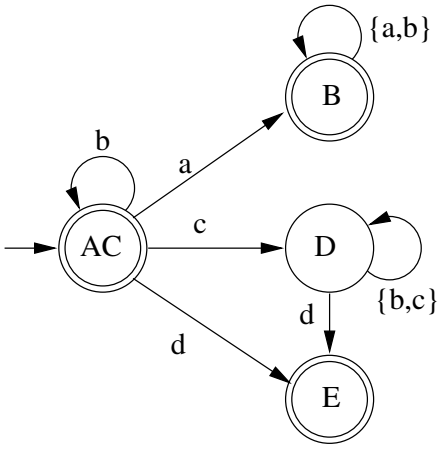
Stato	a	b	c	d
$A = \{0, 1, 2\}$	B	C	D	E
$B = \{1\}$	B	B		
$C = \{1, 2\}$	B	C	D	E
$D = \{2\}$		D	D	E
$E = \{3\}$				

È palese dalla tabella che lo stato A e lo stato C sono equivalenti, mentre per il resto gli stati si comportano diversamente. Tuttavia, per completezza di esposizione, applichiamo l'algoritmo di minimizzazione.

Innanzitutto aggiungiamo uno stato fittizio F, non di accettazione, che ci serve per completare la tabella (ponendo tutte le entrate vuote, e quelle sulla riga di F, ad F). La prima partizione che consideriamo è $(ABCE), (DF)$. Consideriamo il gruppo (DF) e vediamo subito che $move(D, d) = E$ e $move(F, d) = F$. I due stati non sono equivalenti poiché l'input d porta D ed F in stati che sono in gruppi differenti nella partizione corrente. La nuova partizione da considerare è quindi $(ABCE), (D), (F)$.

A questo punto l'unico gruppo da considerare è $(ABCE)$. Si ha $move(A, d) = E$, $move(B, d) = F$, $move(C, d) = E$, $move(E, d) = F$. La nuova partizione è quindi $(AC), (BE), (D), (F)$.

Abbiamo già osservato che fra A e C non troveremo differenze e quindi non ci resta che verificare che invece B ed E non sono equivalenti. Basta osservare che $move(B, a) = B$ e $move(E, a) = F$. Rimuovendo lo stato fittizio e accorpando A e C otteniamo il seguente automa minimo per il linguaggio:



ESERCIZIO 4 (4 punti)

Si consideri un analizzatore lessicale progettato per riconoscere i pattern della lista seguente:

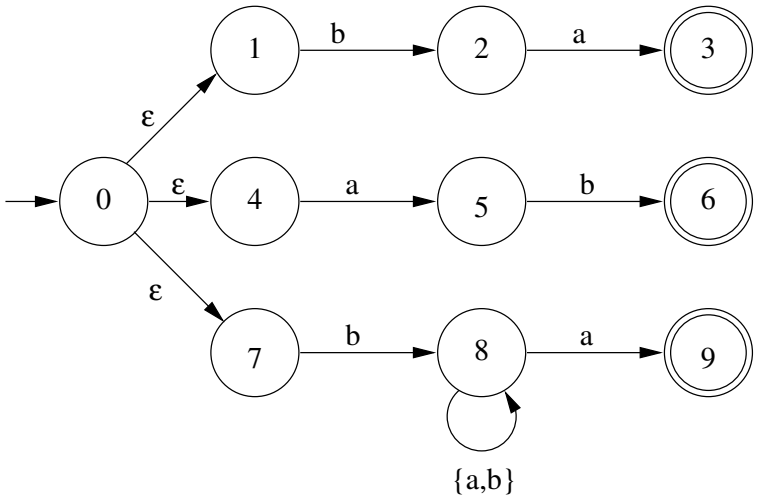
$$\begin{aligned} p_1 &\rightarrow ba \\ p_2 &\rightarrow ab \\ p_3 &\rightarrow b(a|b)^*a \end{aligned}$$

Si dica, giustificando la risposta, qual é la sequenza di token, e relativo lexeme, trovata dall'analizzatore se la stringa di input è *abbabaa*.

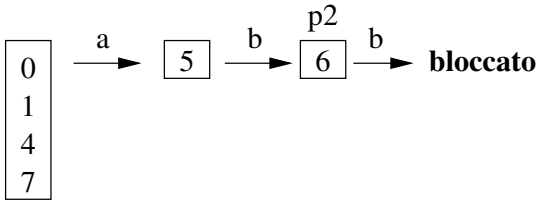
Si assuma un analizzatore che cerchi di riconoscere ad ogni passo il token con il lexeme più lungo possibile e che, in caso di due o più token il cui pattern fa match con il lexeme più lungo, scelga il token il cui pattern è più in alto nella lista data (es. p_1 è più in alto di p_2).

SOLUZIONE

Seguiamo l'algoritmo dell'analizzatore lessicale che usa NFA, visto a lezione. L'NFA risultante per i tre pattern è il seguente:

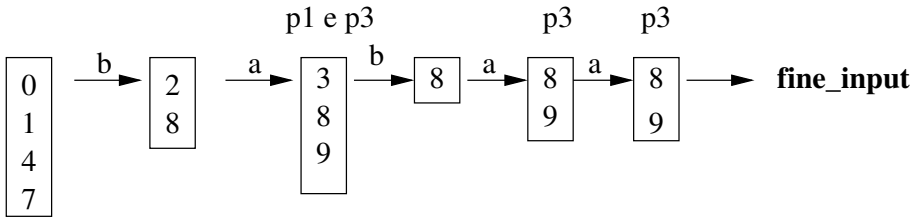


Si noti che abbiamo un po' semplificato l'automa che verrebbe fuori utilizzando la costruzione di Thompson. L'importante è comunque che per ogni pattern ci sia un solo stato finale. Passiamo a simulare l'automa sulla stringa di input avendo cura di andare avanti fino ad una situazione di blocco da cui, tornando all'ultimo insieme di stati contenente uno stato finale, possiamo dedurre il token ed il lexeme riconosciuti. Per il primo token abbiamo i seguenti passi:



Prima del blocco era stato riconosciuto il pattern p_2 dopo l'input ab. Pertanto

il primo token della sequenza è il token numero 2 con lexeme *ab*. A questo punto facciamo ripartire l'analisi dal carattere successivo ad *ab* ed otteniamo la seguente sequenza di passi:



Si noti che il bloccaggio è dovuto alla fine dell'input. Si noti inoltre che dopo *ba* l'automa ha riconosciuto due pattern: p_1 e p_3. Se fosse stato questo il token con lexeme più lungo avremmo dovuto preferire p_1. Tuttavia, il lexeme più lungo è *babaa* corrispondente al pattern p_3. E questo è il secondo token.

ESERCIZIO 5 (9 punti)

Si consideri la seguente grammatica:

$$\begin{array}{lcl} S & \rightarrow & aSb \mid Ad \mid Bc \\ A & \rightarrow & Aa \mid c \\ B & \rightarrow & ddA \mid dC \\ C & \rightarrow & ac \end{array}$$

1. Si esprima il linguaggio generato dalla grammatica mediante espressioni insiemistiche
2. La grammatica è $LL(k)$ per qualche k?
3. Si costruisca la tabella di un parser predittivo non ricorsivo top-down per il linguaggio.

SOLUZIONE

1) Il linguaggio è

$$\{a^n\, c\, a^*\, d\, b^n \mid n \geq 0\} \cup \{a^n\, d\, a\, c\, c\, b^n \mid n \geq 0\} \cup \{a^n\, d\, d\, c\, a^*\, c\, b^n \mid n \geq 0\}$$

2) La grammatica non è $LL(k)$ per nessun k poiché presenta una ricorsione a sinistra immediata nella produzione $A \to Aa$.

3) Proviamo ad eliminare la ricorsione a sinistra ed anche a fattorizzare le produzioni per B. Otteniamo la seguente grammatica:

$$\begin{array}{lcl} S & \to & aSb \mid Ad \mid Bc \\ A & \to & cA' \\ A' & \to & aA' \mid \epsilon \\ B & \to & dB' \\ B' & \to & dA \mid C \\ C & \to & ac \end{array}$$

Si ha FOLLOW$(S) = \{\$, b\}$, FOLLOW$(A) = \{d, c\}$ = FOLLOW(A'), FOLLOW$(B) = \{c\}$ = FOLLOW(B') = FOLLOW(C).

La grammatica così modificata è $LL(1)$ e la tabella di parsing è la seguente:

	a	b	c	d	$
S	$S \to aSb$		$S \to Ad$	$S \to Bc$	
A			$A \to cA'$		
A'	$A' \to aA'$		$A' \to \epsilon$	$A' \to \epsilon$	
B				$B \to dB'$	
B'	$B' \to C$			$B' \to dA$	
C	$C \to ac$				

ESERCIZIO 6 (8 punti)

Si consideri la seguente grammatica:

$$\begin{array}{lcl} S & \to & A \mid Bbb \\ A & \to & aB \\ B & \to & aAb \mid b \end{array}$$

1. Si esprima il linguaggio generato dalla grammatica mediante espressioni insiemistiche
2. La grammatica è LR(1)?
3. La stringa $aaAb$ è un viable prefix? Se si, si dica quali sono gli item $LR(0)$ validi per questo viable prefix.

SOLUZIONE

1) Il linguaggio è

$$\{a^{2n+1}\, b^{n+1} \mid n \geq 0\} \cup \{a^{2n}\, b^{n+3} \mid n \geq 0\}$$

2) Proviamo a costruire dapprima la collezione di item $LR(0)$. Se non ci sono conflitti negli stati la grammatica è $SLR(1)$ e quindi anche $LR(1)$. Aumentiamo la grammatica con la produzione $S' \to S$ come al solito.

$I_0 =$	$S' \to \cdot S$ $S \to \cdot A$ $S \to \cdot Bbb$ $A \to \cdot aB$ $B \to \cdot aAb$ $B \to \cdot b$	$I_1 = goto(I_0, S) =$	$S' \to S\cdot$
$I_2 = goto(I_0, A) =$	$S \to A\cdot$	$I_3 = goto(I_0, B) =$	$S \to B \cdot bb$
$I_4 = goto(I_0, a) =$	$A \to a \cdot B$ $A \to a \cdot Ab$ $B \to \cdot aAb$ $B \to \cdot b$ $A \to \cdot aB$	$I_5 = goto(I_0, b) =$	$B \to b\cdot$
$I_6 = goto(I_3, b) =$	$B \to Bb \cdot b$	$I_7 = goto(I_4, B) =$	$A \to aB\cdot$
$I_8 = goto(I_4, A) =$	$A \to aA \cdot b$	$I_9 = goto(I_6, b) =$	$B \to Bbb\cdot$
$I_{10} = goto(I_8, b) =$	$B \to aAb\cdot$	$goto(I_4, a) = I_4$ $goto(I_4, b) = I_5$	

Si ha che FOLLOW(S') = FOLLOW(S) = {\$}. E poi FOLLOW(A) = FOLLOW(B) = $\{b, \$\}$.
Negli stati non è presente nessun conflitto. Si noti, anzi, che in nessuno stato vengono specificati insieme shift e reduce oppure due diverse reduce. La grammatica pertanto é SLR(1) e, quindi, anche LR(1).

3) Possiamo utilizzare il fatto che la costruzione della collezione canonica $LR(0)$ corrisponde alla specifica di un automa costruito a partire dallo stato I_0, avente uno stato (finale) per ogni insieme I_i e le transizioni specificate dai *goto*. Sappiamo che questo automa riconosce tutti i viable prefixes e pertanto testiamo se la stringa data è accettata dall'automa.

Un cammino sull'automa etichettato con la nostra stringa $aaAb$ è il seguente $0 \xrightarrow{a} 4 \xrightarrow{a} 4 \xrightarrow{A} 8 \xrightarrow{b} 10$. Siccome tutti gli stati sono finali la stringa è accettata e quindi è un viable prefix.

Un altro risultato ci dice, inoltre, che gli item $LR(0)$ contenuti nello stato in cui si arriva testando un viable prefix sull'automa sono tutti gli item validi per quel viable prefix. Pertanto bisogna semplicemente guardare gli item contenuti nello stato I_{10} a cui si arriva con il viable prefix dato. C'è un unico item valido: $B \rightarrow aAb\cdot$