Esercizi esame FLT

1 Data una grammatica capire se il linguaggio è REGOLARE

1. Genero l'albero di derivazione;
2. Se il linguaggio generato contiene dei **bilanciamenti** il linguaggio **NON** è regolare.

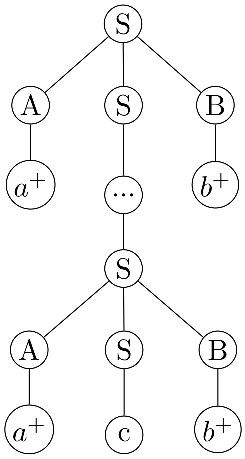
1.1 Dare l'espressione regolare che lo denota

1.2 Grammatica lineare destra che lo genera

1.3 Esempio regolare

Grammatica:

- $S \to ASB \mid c$
- $A \to aA \mid a$
- $b \to b \mid bB$



$S \to ASB$ è una regola ricorsiva autoinclusiva. Ad ogni sua applicazione A genera a sinistra di "c" a^+, e B genera a destra di "c" b^+. Quindi **non** c'è dipendenza (nessuna forma di bilanciamento) fra gli oggetti a destra e a sinistra.

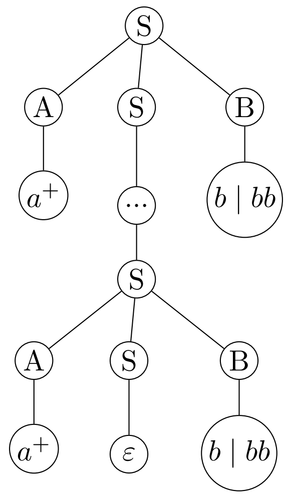
Il linguaggio è $L = a^*cb^*$, è **regolare**, e può essere generato da una grammatica lineare destra:

- $S \to aS \mid cX$
- $X \to bX \mid \varepsilon$

1.4 Esempio non regolare

Grammatica:

- $S \to ASB \mid \varepsilon$
- $A \to aA \mid a$
- $B \to b \mid bb$

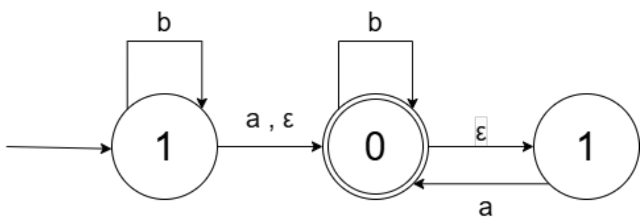


C'è una produzione ricorsiva autoinclusiva ($A \to ASB$) che genera un albero di derivazione con asse di crescita centrale (il fatto che la ricorsione si chiuda con $S \to \varepsilon$, ossia senza mettere un simbolo al centro, non cambia la sostanza del problema).

Abbiamo delle "a" a sinistra, e delle "b" a destra. Il problema è capire se ci sono o no bilanciamenti fra il numero delle "a" e quello delle "b". In questo linguaggio **c'è dipendenza**, ogni volta che genero una o due "b", genero a^+ ovvero, almeno una "a". Questo significa che le "a" devono essere non meno della metà delle "b".

Il linguaggio è $L = a^m b^n$ con $m \geq \frac{n}{2}$. Quindi **NON** è regolare.

2 Dato un automa



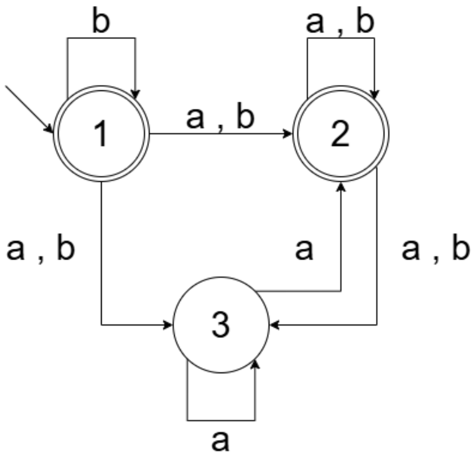
2.1 Eliminare le ε -mosse

- $\varepsilon - CH$: cosa raggiungo partendo dallo stato corrente attraversando solo archi ε, quindi senza consumare input;
- $\sigma(\varepsilon - CH, a)$: a partire dalla $\varepsilon - CH$ dello stato dove arrivo attraversando uno stato a;
- $\varepsilon - CH(\sigma(\varepsilon - CH, a))$: a partire dagli stati raggiunti prima dove arrivo attraversando solo archi ε.

	0	**1**	**2**	
$\varepsilon - CH$	1, 2, 3	2, 3	3	
$\sigma(\varepsilon - CH, a)$	2	2	2	
$\varepsilon - CH(\sigma(\varepsilon - CH, a))$	2, 3	2, 3	2, 3	$\Leftarrow$
$\sigma(\varepsilon - CH, b)$	1, 2	2	/	
$\varepsilon - CH(\sigma(\varepsilon - CH, b))$	1, 2, 3	2, 3	/	$\Leftarrow$

Per costruire l'automa senza ε-mosse per prima cosa calcolo l'insieme degli stati finali. Posso aggiungere lo stato iniziale all'insieme degli stati finali solo se nella sua $\varepsilon - CH$ è presente almeno uno stato finale.

Ora posso creo tutti gli stati e connetto gli arichi in base a cosa dicono le righe: $\varepsilon - CH(\sigma(\varepsilon - CH, a))$.



2.2 Renderlo deterministico

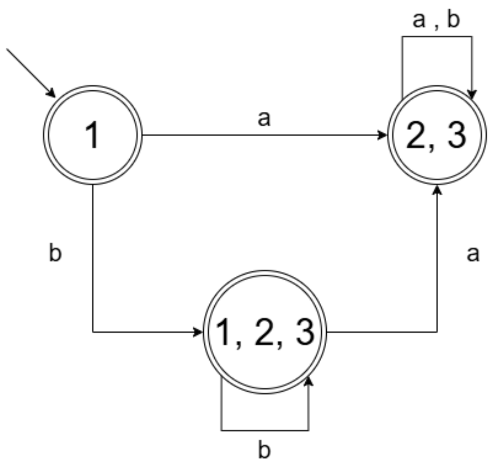
Come prima cosa costruisco una tabella con tutte le possibili combinazioni di stati. Se nella combinazione è presente uno stato finale la marco x^F.

Le caselle che fanno riferimento agli stati singoli vanno riempite con i dati presenti nella tabella precedente $\varepsilon - CH(\sigma(\varepsilon - CH, a))$, le altre caselle vanno riempite facendo l'unione dei vari $\varepsilon - CH(\sigma(\varepsilon - CH, a))$ dii tutti gli stati della casella.

	1^F	2^F	3	$1,2^F$	$1,3^F$	$2,3^F$	$1,2,3^F$
a	2,3	2,3	2,3	2,3	2,3	2,3	2,3
b	1,2,3	2,3	/	1,2,3	1,2,3	2,3	1,2,3

Parto dallo stato iniziale vedo per ogni simbolo in quale stato devo andare, per ogni stato che visito prendo le caselle di tutti i simboli. In questo caso dallo stato 1 vado in $2, 3$ per a e seleziono entrambe le caselle, per b vado in $1, 2, 3$ e per a è presente $2, 3$, già selezionato e per b è presente $1, 2, 3$ lo stato stesso, quindi mi fermo.

Creo gli stati selezionati, se sono marcati allora saranno finali. Per gli archi vedo dallo stato in quale stato devo andare.



2.3 Minimizzarlo

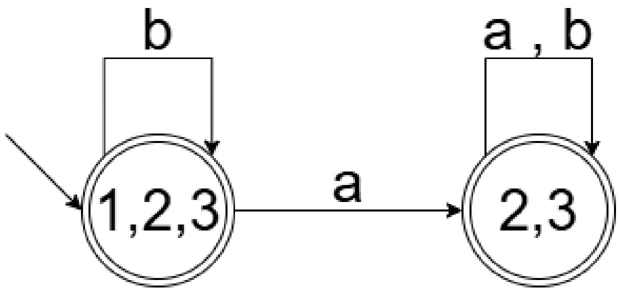
Creo una matrice senza la diagonale.

- Se gli stati sono discordi (uno finale e uno non) posso marcare come diversi;
- Se gli stati sono concordi (entrambi finali o entrambi non finali)
  - Controllo per ogni simbolo dove si arriva con lo stato ¡Dove arrivo partendo da B per il simbolo a, Dove arrivo partendo da A per il simbolo a¿ e così via per tutti gli stati.

* Se

- $A \to 1$
- $B \to 2, 3$
- $c \to 1, 2, 3$

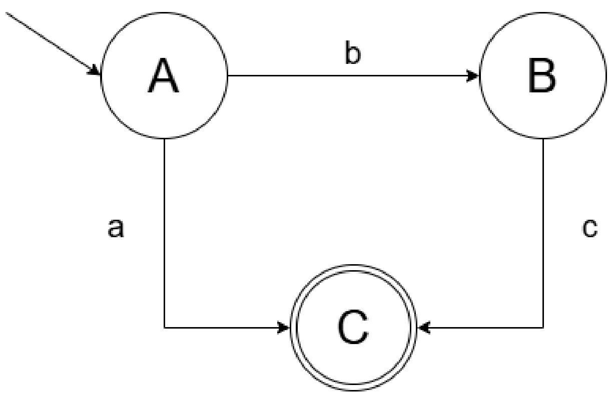
A			
B	<~~B,B~~> <~~B,C~~>		
C	<B,C> <B,C>	<~~B,C~~> <~~B,B~~>	
	A	B	C



3 Dato un automa definire una grammatica

- L'alfabeto è lo stesso;
  Σ
- L'insieme dei non terminali è definito sulla base dell'insieme degli stati;
  $V = Q$
- Il nodo iniziale avrà un corrispondente non terminale;
  $S = Q_0$
- Per qualunque regola di produzione ottengo la regola equivalente.
  $\forall \sigma(q_i, a) = q_j$ ottengo $Q_i \to aQ_j$

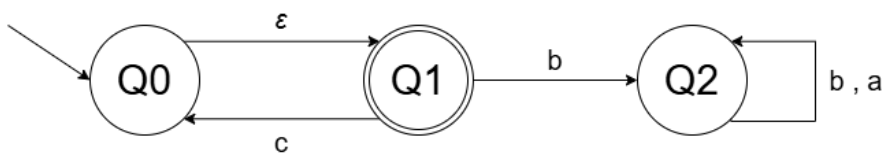
3.1 Esempio 1



$\Sigma = \{a, b, c\} \qquad V = \{A, B, C\} \qquad S = A$

- $A \to aC \mid bB$
- $B \to cC$
- $C \to \varepsilon$

3.2 Esempio 2



$$\Sigma = \{a, b, c, \varepsilon\} \qquad V = \{Q_0, Q_1, Q_2\} \qquad S = Q_0$$

- $Q_0 \to Q_1$
- $Q_1 \to cQ_0 \mid bQ_2 \mid \varepsilon$
- $Q_2 \to aQ_2 \mid bQ_2$

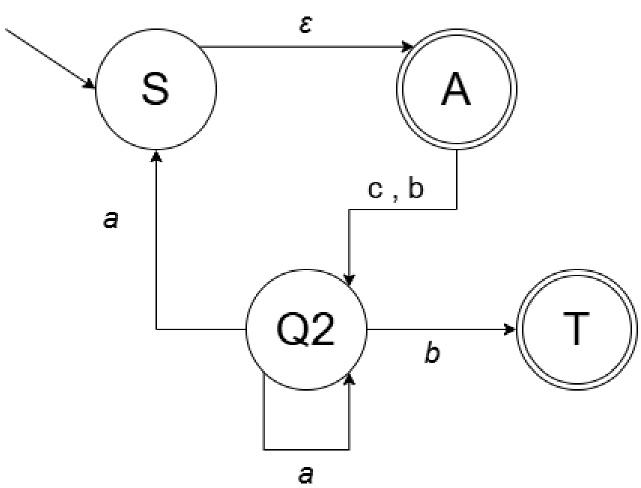
4 Data una grammatica definire l'automa

- L'alfabeto è lo stesso;
  Σ
- Per ogni non terminale devo avere un corrispondente stato dell'automa;
  $Q = V$
- Il nodo iniziale avrà un corrispondente non terminale;
  $Q_0 = V$
- In una grammatica lineare dx posso trovarmi 4 tipi di regole:
  - $A \to aB$, scriverò la funzione di transizione $\sigma(A, a) = B$;
  - $A \to B$, scriverò la funzione di transizione $\sigma(A, \varepsilon) = B$;
  - $\forall A \to a$, devo creare un nuovo stato T finale $\sigma(A, a) = T$, dove $T \in F$;
  - $\forall A \mid A \to \varepsilon$ allora $A \in F$.

4.1 Esempio

- $S \to A \mid aA$
- $A \to cB \mid bB \mid \varepsilon$
- $B \to aS \mid aB \mid b$

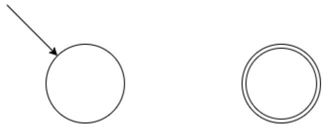
$$\Sigma = \{a, b, c, \varepsilon\} \qquad Q = \{S, A, B, T\} \qquad Q_0 = S$$



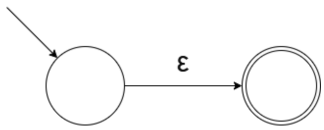
5 Da espressione regolare ad automa a stati finiti

Algoritmo di **Thomson**, solo per grammatiche lineari destre.
Casi base:

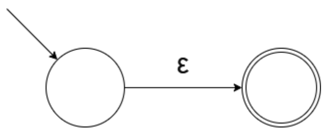
1. L'insieme vuoto;



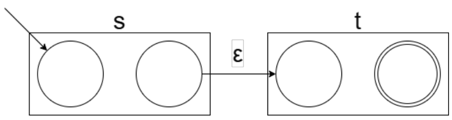
2. La stringa vuota;



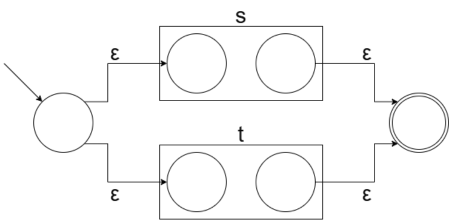
3. Una parola di un singolo simbolo;



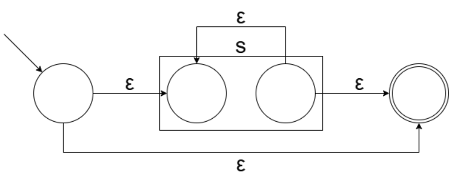
4. (a) $s \cdot t \quad$ e.r $\implies$



   (b) $s \cup t \quad$ e.r $\implies$



5. S^* e.r



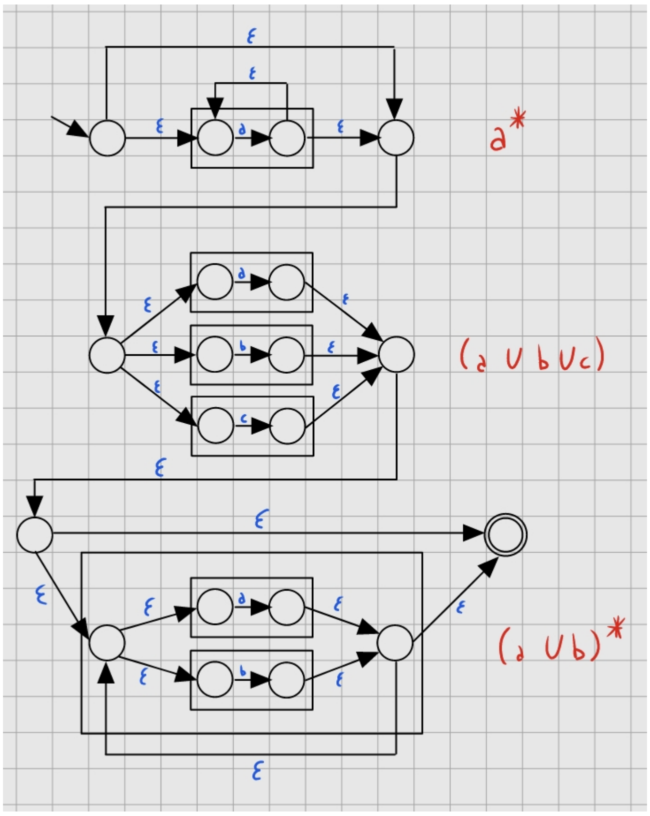
5.1 Esempio

$$a^*(a \cup b \cup c)(a \cup b)^*$$

1. Identifico le sottoespressioni regolari:

   - $a^* \Rightarrow$ caso 5
   - $(a \cup b \cup c) \Rightarrow$ caso 4b

- $(a \cup b)^* \Rightarrow$ caso 4b+5



6 A partire da una grammatica costruire, in modo automatico, l'auto a pila corrispondente

$T = \Sigma \cup V \cup \{Z_0\}$
$F = \{\}$ non ci sono stati finali perché accettiamo per pila vuota.

- Regola di inizializzazione: $\sigma(q_0, \varepsilon, Z_0) = (q_0, \swarrow S)$
- Regola di terminazione: $\sigma(q_0, \swarrow, \swarrow) = (q_0, \varepsilon)$
- $\forall a \in \Sigma$ ogni simbolo dell'alfabeto: $\sigma(q_0, a, a) = (q_0, \varepsilon)$
- $\forall A \to a$ e p:
  - $\alpha = a\beta \Rightarrow \sigma(q_0, a, A) = (q_0, \beta^R)$ inizia con un terminale
  - $\alpha = x\beta \Rightarrow \sigma(q_0, \varepsilon, A) = (q_0, \beta^R x)$ non inizia con un terminale

6.1 Esempio 1

- $S \to aSb \mid Aab$
- $S \to ac$

1. $\sigma(q_0, \varepsilon, Z_0) = (q_0, \swarrow S)$
2. $\sigma(q_0, \swarrow, \swarrow) = (q_0, \varepsilon)$
3. $\sigma(q_0, a, a) = (q_0, \varepsilon)$
4. $\sigma(q_0, b, b) = (q_0, \varepsilon)$
5. $\sigma(q_0, c, c) = (q_0, \varepsilon)$

6. $\sigma(q_0, a, S) = (q_0, bS)$
7. $\sigma(q_0, a, A) = (q_0, c)$
8. $\sigma(q_0, \varepsilon, S) = (q_0, baA)$

6.1.1 Simulazione

PILA	STATO	INPUT	AZIONE
Z_0	q_0	aacabb$\swarrow$	1
$\swarrow$ S	q_0	aacabb$\swarrow$	6
$\swarrow$ bS	q_0	acabb$\swarrow$	8
$\swarrow$ bbaA	q_0	acabb$\swarrow$	7
$\swarrow$ bbac	q_0	cabb$\swarrow$	5
$\swarrow$ bba	q_0	abb$\swarrow$	3
$\swarrow$ bb	q_0	bb$\swarrow$	4
$\swarrow$ b	q_0	b$\swarrow$	4
$\swarrow$	q_0	$\swarrow$	2
	q_0		

6.2 Esempio 2

- $S \to aS$
- $S \to A$
- $A \to aAb$
- $A \to ab$

1. $\sigma(q_0, \varepsilon, Z_0) = (q_0, \swarrow S)$
2. $\sigma(q_0, \swarrow, \swarrow) = (q_0, \varepsilon)$
3. $\sigma(q_0, a, a) = (q_0, \varepsilon)$
4. $\sigma(q_0, b, b) = (q_0, \varepsilon)$
5. $\sigma(q_0, \varepsilon, S) = (q_0, A)$
6. $\sigma(q_0, a, S) = (q_0, S)$
7. $\sigma(q_0, a, A) = (q_0, bA)$
8. $\sigma(q_0, a, A) = (q_0, b)$

6.2.1 Simulazione

PILA	STATO	INPUT	AZIONE
Z_0	q_0	aaaaabbb$\swarrow$	1
$\swarrow$ S	q_0	aaaaabbb$\swarrow$	3
$\swarrow$ S	q_0	aaaabbb$\swarrow$	3
$\swarrow$ S	q_0	aaaabbb$\swarrow$	3
$\swarrow$ A	q_0	aaabbb$\swarrow$	5
$\swarrow$ bA	q_0	aabbb$\swarrow$	7
$\swarrow$ bbA	q_0	abbb$\swarrow$	7
$\swarrow$ bbb	q_0	bbb$\swarrow$	8
$\swarrow$ bb	q_0	bb$\swarrow$	4
$\swarrow$ b	q_0	b$\swarrow$	4
$\swarrow$	q_0	$\swarrow$	4
	q_0		

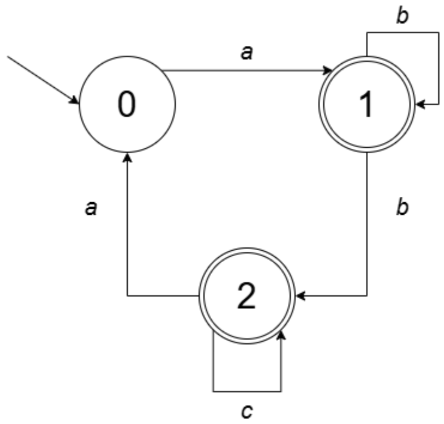
7 Dati i seguenti automi costruite l'automa complemento

Quando l'automa a stati finiti dovrebbe dire SÍ dice NO e viceversa.
$M = \langle Q, \Sigma, \sigma, q_0, F \rangle$
$\overline{M} = \langle \overline{Q}, \Sigma, \overline{\sigma}, \overline{q_0}, \overline{F} \rangle$

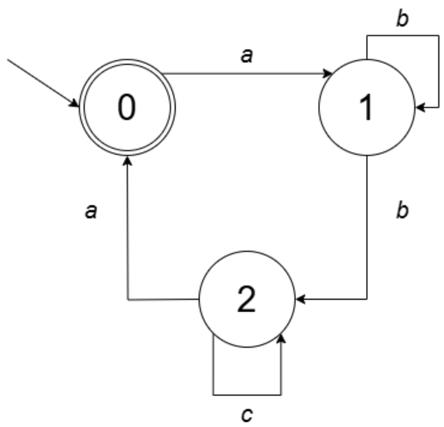
1. Rendo gli stati finali $\rightarrow$ non finali e quelli non finali $\rightarrow$ finali;

2. Creo un nuovo stato P pozzo, finale

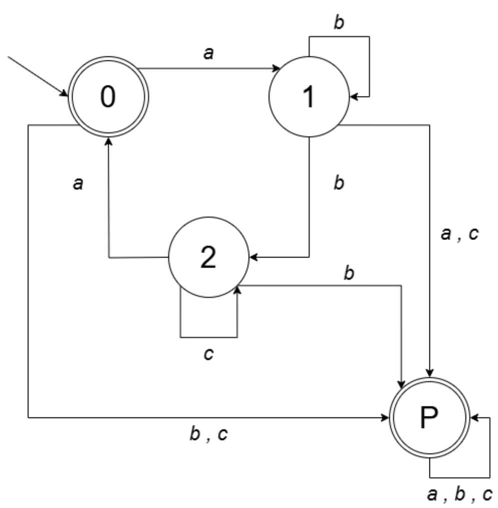
7.1 Esempio



Inverto gli stati finali.



Creo un nuovo stato "P" finale.



- $\overline{Q} = Q \cup \{P\}$
- $\overline{q_0} = q_0$
- $\overline{F} = (Q - F) \cup \{P\}$
- $\forall q, \in Q \forall a \in \Sigma \quad \sigma(q,a) = q' \quad \overline{\sigma}(q,a) = q'$
- $\forall q \in Q, \Sigma a \in \Sigma \quad \sigma(q,a) = / \quad \overline{\sigma}(q,a) = P$
- $\Sigma a \in \Sigma \quad \overline{\sigma}(P,a) = P$

8 Dato un linguaggio fornire una grammatica context free (non contestuale) che genera tale linguaggio

Una grammatica context-free (CFG) è definita da una quadrupla:

$$G = (V, \Sigma, P, S)$$

dove:

- V è l'insieme dei simboli non terminali;
- Σ è l'alfabeto dei terminali;
- P è l'insieme delle produzioni;
- $S \in V$ è il simbolo iniziale.

I passaggi per costruire una CFG sono:

1. Identificare la struttura del linguaggio.
2. Definire l'alfabeto dei terminali Σ.
3. Identificare i non terminali V.
4. Scrivere le regole di produzione P.
5. Verificare che la grammatica generi tutte e solo le stringhe desiderate.

8.1 Esempio 1: Linguaggio $a^n b^n$

Linguaggio:

$$L = \{a^n b^n \mid n \geq 0\}$$

Grammatica:

$$S \to aSb \mid \varepsilon$$

Spiegazione:

- La produzione $S \to aSb$ assicura che ogni a abbia un corrispondente b.
- La produzione $S \to \varepsilon$ permette la terminazione della stringa.

8.2 Esempio 2: Linguaggio delle stringhe binarie palindrome

Linguaggio:

$$L = \{w \mid w = w^R, w \text{ è una stringa su } \{0,1\}\}$$

Grammatica:

$$S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$$

Spiegazione:

- $S \to 0S0$ e $S \to 1S1$ garantiscono simmetria.
- $S \to 0$ e $S \to 1$ gestiscono i casi base.
- $S \to \varepsilon$ permette la stringa vuota.

9 Date le seguenti grammatiche costruire interamente l'automa LR(0) corrispondente

9.1 1. Aggiunta della produzione iniziale

Se la grammatica ha una produzione con simbolo iniziale S, aggiungiamo una nuova produzione:

$$S' \to S$$

Questo serve per garantire che l'automa abbia un unico stato di accettazione.

9.2 2. Costruzione degli stati dell'automa

Gli stati dell'automa LR(0) sono insiemi di elementi LR(0), ossia produzioni con un punto $\cdot$ che indica la posizione attuale di lettura.

- Lo stato iniziale contiene la produzione $S' \to \cdot S$.
- Ogni stato si espande includendo tutte le produzioni con il punto davanti a un non terminale.

9.3 3. Transizioni tra stati

Se in un certo stato c'è una produzione con il punto prima di un simbolo X, allora esiste una transizione verso un nuovo stato in cui il punto è avanzato oltre X.

9.4 4. Chiusura e generazione degli stati

Continuiamo a costruire stati e transizioni finché non troviamo nuovi stati. A ogni passo:

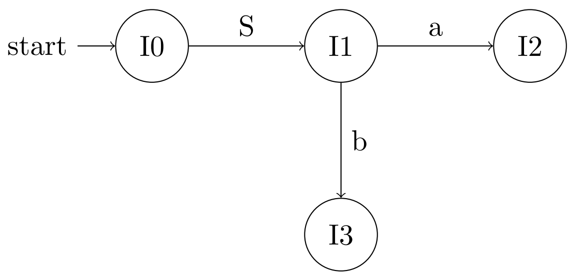
- Se il punto è prima di un simbolo non terminale, aggiungiamo le produzioni corrispondenti.
- Se il punto è alla fine di una produzione, lo stato rappresenta una riduzione.

9.5 5. Generazione della tabella di parsing

- Se una produzione termina con un punto, aggiungiamo un'azione di riduzione.
- Se il punto è davanti a un terminale, aggiungiamo una transizione di spostamento (shift).
- Se lo stato contiene $S' \to S\cdot$, è uno stato di accettazione.

9.6 Automa LR(0)

Di seguito è riportato il diagramma dell'automa LR(0) costruito:



10 Data la seguente grammatica, costruire l'automa e dire se è SLR(1)

1. Aggiunta della produzione iniziale
2. Costruzione degli stati LR(0)
3. Costruzione dell'automa LR(0)
4. Calcolo degli insiemi $FIRST$ e $FOLLOW$
5. Verifica della proprietà SLR(1)

10.1 Grammatica Data

Consideriamo la seguente grammatica:

$$S \to Aa \mid b$$
$$A \to c$$

10.2 Costruzione dell'Automa LR(0)

10.2.1 1. Aggiunta della produzione iniziale

Aggiungiamo una nuova produzione con un simbolo iniziale S' per garantire un unico stato di accettazione:

$$S' \to S$$

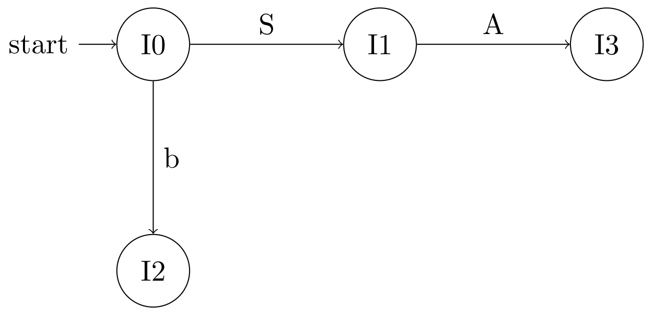
10.2.2 2. Stati dell'automa LR(0)

Gli stati contengono produzioni con un punto $\cdot$ che indica la posizione corrente della lettura:

- Stato I_0: $S' \to \cdot S$
- Espandiamo per ottenere tutti gli stati successivi.

10.2.3 3. Costruzione dell'automa LR(0)

L'automa risultante è il seguente:



10.3 Calcolo degli insiemi FIRST e FOLLOW

10.3.1 1. Calcolo di FIRST

L'insieme $FIRST(X)$ contiene tutti i terminali che possono apparire come primo simbolo di una stringa derivata da X.

$$FIRST(A) = \{c\}$$

$$FIRST(S) = FIRST(Aa) \cup FIRST(b) = \{c, b\}$$

10.3.2 2. Calcolo di FOLLOW

L'insieme $FOLLOW(X)$ contiene tutti i terminali che possono seguire immediatamente X in una derivazione.

- Poiché $S' \to S$, abbiamo:

$$FOLLOW(S) = \{\$\}$$

- Dalla produzione $S \to Aa$, otteniamo:

$$FOLLOW(A) = \{a\}$$

- Da $A \to c$ non si aggiunge nulla a $FOLLOW(A)$.

10.4 Verifica della proprietà SLR(1)

Per verificare se la grammatica è $SLR(1)$, controlliamo se esistono conflitti tra gli stati di riduzione e i terminali nell'insieme $FOLLOW$.

- Se un simbolo terminale in $FOLLOW(X)$ appare in più di un'azione nella tabella di parsing, allora la grammatica **non** è SLR(1).
- In questo caso, non ci sono conflitti tra gli stati, quindi la grammatica è **SLR(1)**.

11 Date le seguenti espressioni costruire le grammatiche context free corrispondenti

Data un'espressione regolare (RE), i passi per costruire una grammatica context-free (CFG) sono:

1. Identificare l'alfabeto dei terminali Σ.
2. Definire i simboli non terminali V.
3. Scrivere le regole di produzione P basate sulle operazioni della RE.
4. Verificare la correttezza della grammatica generando stringhe di prova.

11.1 Conversione delle operazioni di un'espressione regolare in una grammatica

11.1.1 Concatenazione: $RE = AB$

$$S \to AB$$

11.1.2 Unione: $RE = A|B$

$$S \to A \mid B$$

11.1.3 Chiusura di Kleene: $RE = A^*$

$$S \to AS \mid \varepsilon$$

11.1.4 Chiusura positiva: $RE = A^+$

$$S \to AS \mid A$$

11.2 Esempi pratici

11.2.1 Esempio 1: Espressione regolare $(a|b)^*$

Linguaggio: qualsiasi sequenza di "a" e "b", inclusa la stringa vuota.

$$S \to aS \ |bS \ |\varepsilon$$

11.2.2 Esempio 2: Espressione regolare a^+b

Linguaggio: una o più "a" seguite da una "b".

$$S \to Ab$$
$$A \to aA \ |a$$

11.2.3 Esempio 3: Espressione regolare $(a|b)c^*$

Linguaggio: una "a" o una "b", seguita da zero o più "c".

$$S \to aC \ |bC$$
$$C \to cC \ |\varepsilon$$

12 Eliminare le ricorsioni sinistre immediate applicando entrambi i metodi ovvero quello che presenta epsilon produzioni e quello che non le presenta

Una grammatica possiede una **ricorsione sinistra immediata** se contiene una produzione della forma:

$$A \to A\alpha \mid \beta$$

dove:

- A è un non terminale,
- α è una sequenza di simboli (terminali o non terminali),
- β è una produzione alternativa che non inizia con A.

Il problema della ricorsione sinistra è che può portare a loop nella derivazione, quindi è necessario eliminarla.

12.1 Metodo 1: Eliminazione con ε-produzioni

Si introduce un nuovo non terminale A' e si riscrive la grammatica come:

$$A \to \beta A'$$
$$A' \to \alpha A' \mid \varepsilon$$

Esempio: Dato

$$S \to Sa \mid b$$

dopo l'eliminazione della ricorsione sinistra:

$$S \to bS'$$
$$S' \to aS' \ |\varepsilon$$

12.2 Metodo 2: Eliminazione senza ε-produzioni

Invece di usare ε, si usa un'altra regola per generare un numero finito di ripetizioni:

$$A \to \beta \;|\beta A'$$
$$A' \to \alpha \;|\alpha A'$$

Esempio: Dato

$$S \to Sa \mid b$$

si trasforma in:

$$S \to b \;|bS'$$
$$S' \to a \;|aS'$$

13 Teoria

13.1 Dimostrare che l'insieme delle grammatiche regolari è un sottoinsieme delle grammatiche context free

Le grammatiche regolari sono un sottoinsieme delle grammatiche context-free poiché ogni grammatica regolare è, per definizione, una grammatica context-free con vincoli più stringenti sulle produzioni.
Dimostrazione:

- Una grammatica è **regolare** se tutte le sue produzioni sono della forma:

$$A \to aB \quad \text{(forma destra)}$$
$$A \to Ba \quad \text{(forma sinistra)}$$
$$A \to a \quad \text{(produzione terminale)}$$

  dove A, B sono non terminali e a è un terminale.

- Una grammatica è **context-free** se tutte le sue produzioni sono della forma:

$$A \to \gamma$$

  dove A è un singolo non terminale e γ è una stringa arbitraria di terminali e non terminali.

- Poiché le produzioni regolari sono un caso particolare delle produzioni context-free (in cui il lato destro è limitato a una specifica forma), ne segue che ogni grammatica regolare è anche context-free.

Conclusione: L'insieme delle grammatiche regolari è un sottoinsieme proprio delle grammatiche context-free.

13.2 Definire formalmente l'automa a pila deterministico

Un **automa a pila deterministico** (DPDA) è una tupla:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

dove:

- Q è un insieme finito di stati,
- Σ è l'alfabeto di input,
- Γ è l'alfabeto del nastro della pila,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$ è la funzione di transizione,

- $q_0 \in Q$ è lo stato iniziale,
- $Z_0 \in \Gamma$ è il simbolo iniziale della pila,
- $F \subseteq Q$ è l'insieme degli stati finali.

Un DPDA è deterministico se per ogni coppia $(q, a, X) \in Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$, al massimo una transizione è definita in δ.

13.3 Definire formale l'epsilon chiusura

L'ε-chiusura di uno stato in un automa a stati finiti (NFA) è l'insieme degli stati raggiungibili a partire da uno stato dato attraverso transizioni ε. Formalmente, data una NFA $N = (Q, \Sigma, \delta, q_0, F)$, l'$\varepsilon$-chiusura di uno stato $q \in Q$ è definita come:

$$\varepsilon\text{-Chiusura}(q) = \{p \in Q \mid q \xrightarrow{\varepsilon^*} p\}$$

dove $q \xrightarrow{\varepsilon^*} p$ indica che esiste una sequenza di transizioni ε che porta da q a p.