

Riconoscimento e parsificazione delle frasi

4.1 Introduzione

I linguaggi liberi, per essere riconosciuti, richiedono risorse di memoria maggiori di quelli regolari. Nel capitolo si studiano i relativi algoritmi, visti prima come automi astratti con una pila di memoria, poi come procedure di analisi sintattica o parsificazione¹, che costruiscono l'albero sintattico delle frasi. Per inciso, la parsificazione ha poco interesse per i linguaggi regolari poiché la loro struttura sintattica può essere predeterminata (lineare a destra o a sinistra), mentre per i linguaggi liberi essa prende forme diverse.

Come per le grammatiche unilineari, così per quelle libere vi è corrispondenza tra le regole grammaticali e le mosse dell'automa, che però ora possiede, oltre agli stati, una memoria a pila. Una differenza vistosa tra le due famiglie è che per i linguaggi liberi non sempre si può ottenere un automa deterministico. Inoltre la presenza di due memorie, la pila e gli stati, introduce una varietà di modi di funzionamento che complica il quadro teorico, rispetto al caso degli automi finiti.

Dopo la presentazione generale degli automi a pila, si studieranno quelli deterministici che definiscono la famiglia molto utile *DET* dei linguaggi liberi deterministici, di cui si vedranno le principali proprietà.

Il resto del capitolo presenta gli algoritmi di parsificazione: prima gli algoritmi veloci, operanti in tempo lineare, poi un algoritmo parsificatore di tipo generale.

I parsificatori in tempo lineare studiati sono di due tipi: discendenti ($LL(k)$) e ascendenti ($LR(k)$), a seconda dell'ordine in cui l'albero sintattico è costruito. Entrambi i tipi sono diffusamente impiegati nella compilazione. Tali algoritmi presuppongono che il linguaggio sia deterministico, e impongono ulteriori limitazioni sulla forma delle regole della grammatica.

La successiva presentazione dei metodi di costruzione dei parsificatori procede attraverso un percorso espositivo omogeneo, che poggia sulla rappresentazio-

¹Dal latino *pars, partis*, nel senso di dividere la frase nelle sue parti.

ne della grammatica come rete ricorsiva di automi. Tale unificazione mira a evidenziare le simiglianze tra gli algoritmi e le rispettive condizioni di applicabilità, evitando fastidiose ripetizioni di concetti molto simili.

Per i parsificatori discendenti è poi esposta la realizzazione mediante procedure ricorsive, che ha il pregio di permettere una facile costruzione manuale. Per completezza il capitolo tratta anche la costruzione dei parsificatori deterministici per le grammatiche estese con espressioni regolari.

L'ultimo metodo presentato è quello di Earley, adatto a parsificare qualsiasi grammatica, ma di complessità maggiore pur se ancora polinomiale. Il quadro complessivo copre dunque tutta la gamma degli algoritmi normalmente usati nella compilazione e nel trattamento dei documenti.

Il capitolo termina con una discussione dei criteri di scelta dei parsificatori.

4.1.1 Automi a pila

Gli algoritmi di riconoscimento usati dai compilatori sono essenzialmente degli automi finiti dotati d'una memoria ausiliaria organizzata come una pila illimitata di simboli

$$A_1 A_2 \dots \overbrace{A_k}^{\text{cima}}$$

La stringa di ingresso o sorgente, delimitata a destra dal terminatore, è scritta

$$a_1 a_2 \dots \overbrace{a_i}^{\text{carattere corrente}} \dots a_n \vdash$$

Le operazioni applicabili alla pila sono:

- impilamento*: $push(B)$ pone il simbolo B in cima, cioè alla destra di A_k ; più operazioni di impilamento possono essere combinate e rappresentate da $push(B_1, B_2, \dots, B_n)$
- test di pila vuota*: $empty$, predicato vero solo se $k = 0$;
- disimpilamento*: pop , purché la pila non sia vuota, toglie da essa il simbolo A_k .

Conviene immaginare che la pila abbia dipinto sul fondo un simbolo speciale, spesso scritto Z_0 , detto il *fondo*. Esso potrà essere soltanto letto (né tolto né impilato). La sua presenza in cima alla pila significa che essa è vuota.

Come in un automa finito, i caratteri della stringa sono esaminati da sinistra a destra, dalla testina di lettura. Il carattere, che sta sotto la testina in un certo momento, è detto *corrente*. In un dato istante la *configurazione* dell'automata è caratterizzata da: il carattere corrente, lo stato corrente, il contenuto della pila. Con una mossa l'automata:

- legge il carattere corrente avanzando con la testina, oppure compie una mossa spontanea senza muovere la testina;
- legge il simbolo in cima e lo toglie dalla pila, oppure legge Z_0 se la pila è vuota;

- in funzione dei valori dello stato corrente, del carattere corrente e del simbolo letto dalla pila, calcola il nuovo stato e eventualmente impila uno o più simboli.

4.1.2 Dalla grammatica all'automa a pila

Si mostra che le regole grammaticali possono essere viste come istruzioni d'una macchina a pila indeterministica che riconosce il linguaggio; essa è tanto semplice da non usare gli stati ma soltanto la pila come memoria.

Intuitivamente, l'automa opera in modo *predittivo* o finalizzato (goal oriented), usando la pila come una agenda delle future azioni da compiere. I simboli della pila sono i caratteri nonterminali e terminali della grammatica. Se la pila contiene, dalla cima al fondo, i simboli $A_k \dots A_1$, la macchina eseguirà per prima l'azione associata a A_k , poi la A_{k-1} , e così via fino all'ultima azione A_1 . L'azione associata a A_k ha l'obiettivo di riconoscere se in ingresso, a partire dal carattere corrente a_i , vi è una stringa w derivabile da A_k . In caso positivo, l'azione farà avanzare la testina di lettura di $|w|$ posizioni. Naturalmente l'obiettivo può articolarsi ricorsivamente in sotto obiettivi, se per riconoscere A_k è necessario riconoscere altri simboli terminali o non. Inizialmente l'obiettivo è l'assioma della grammatica: compito del riconoscitore è infatti riconoscere se la stringa sorgente deriva dall'assioma.

Algoritmo 4.1. Dalla grammatica all'automa a pila indeterministico con un solo stato.

Data la grammatica libera $G = (V, \Sigma, P, S)$, si precisa ora la corrispondenza tra le regole e le mosse dell'automa. Nella tabella la lettera b sta per un terminale, le lettere A, B per un nonterminale e A_i per un simbolo qualsiasi.

<i>regola</i>	<i>mossa</i>	<i>commento</i>
1 $A \rightarrow BA_1 \dots A_n$ $n \geq 0$	if <i>cima</i> = <i>A</i> then pop; push($A_n \dots A_1 B$) end if	Per riconoscere <i>A</i> si devono riconoscere $BA_1 \dots A_n$
2 $A \rightarrow bA_1 \dots A_n$ $n \geq 0$	if <i>car_corr</i> = <i>b</i> \wedge <i>cima</i> = <i>A</i> then pop; push ($A_n \dots A_1$); avanza testina lettura	<i>b</i> era il primo carattere atteso ed è stato letto; restano da riconoscere $A_1 \dots A_n$
3 $A \rightarrow \varepsilon$	if <i>cima</i> = <i>A</i> then pop	È stata riconosciuta la stringa vuota che deriva da <i>A</i>
4 per ogni carattere $b \in \Sigma$	if <i>car_corr</i> = <i>b</i> \wedge <i>cima</i> = <i>b</i> then pop; avanza testina lettura	<i>b</i> era il primo carattere atteso ed è stato letto
5 - - -	if <i>car_corr</i> = \perp \wedge pila è vuota then accetta; alt.	La stringa è stata scan- dita per intero e non restano in agenda altri obiettivi

La forma della regola determina la mossa. Se la parte destra inizia con un terminale (2), la mossa è condizionata dalla lettura dello stesso. Invece altre regole (1 e 3) danno luogo a mosse spontanee, che non leggono il carattere corrente. Poi vi sono le mosse (4) che controllano che un terminale, quando affiora in cima alla pila (essendo stato precedentemente impilato da una mossa del tipo 1 o 2) coincida con il carattere corrente. Infine la mossa (5) accetta la stringa se la pila è vuota alla lettura del terminatore.

Inizialmente la pila contiene soltanto il simbolo di fondo pila Z_0 e l'assioma S , e la testina di lettura è posta sul primo carattere della stringa sorgente. A ogni passo l'automa sceglie (indeterministicamente) una delle regole applicabili e esegue la corrispondente mossa.

L'automa riconosce la stringa se, esiste un calcolo che termina con la mossa 5.

Si dice che questo modello d'automa riconosce le frasi *a pila vuota*.

Si noti che l'automa non ha bisogno di stati diversi, cioè la pila è l'unica memoria. Più avanti però si aggiungeranno gli stati, al fine di rendere deterministico l'automa.

Esempio 4.2. Le mosse del riconoscitore predittivo del linguaggio

$$L = \{a^n b^m \mid n \geq m \geq 1\}$$

sono riportate accanto alle regole della grammatica:

Regola	Mossa
1 $S \rightarrow aS$	if $car_corr = a \wedge cima = S$ then pop; push(S); avanza
2 $S \rightarrow A$	if $cima = S$ then pop; push(A)
3 $A \rightarrow aAb$	if $car_corr = a \wedge cima = A$ then pop; push(bA); avanza
4 $A \rightarrow ab$	if $car_corr = a$ and $cima = A$ then pop; push(b); avanza
5	if $car_corr = b \wedge cima = b$ then pop; avanza
6	if $car_corr = \perp \wedge pila \text{ è vuota}$ then accetta; alt.

La scelta tra le mosse 1 e 2 non è deterministica, poiché la 2 può essere scelta anche quando a è il carattere corrente; e similmente la scelta tra 3 e 4.

Non è difficile convincersi che l'automa così costruito riconosce una stringa se, e solo se, la grammatica la genera. Infatti per ogni calcolo che termina con successo esiste una derivazione corrispondente, e viceversa; in altre parole l'automa simula le derivazioni sinistre della grammatica.

Ad es. alla derivazione:

$$S \Rightarrow A \Rightarrow aAb \Rightarrow aabb$$

corrisponde la seguente traccia del calcolo, terminante con esito positivo:

$$\begin{array}{rcl}
 \hline Pila & | & x \\
 \hline Z_0 S & | & aabb \quad \vdash \\
 \\
 Z_0 A & | & aabb \quad \vdash \\
 \\
 Z_0 b A & | & abb \quad \vdash \\
 \\
 Z_0 bb & | & bb \quad \vdash \\
 \\
 Z_0 b & | & b \quad \vdash \\
 \\
 Z_0 & | & \quad \vdash
 \end{array}$$

Ma l'algoritmo non può sapere a priori quale sarà la derivazione giusta, e deve esplorare tutte le possibilità, anche quelle che falliranno, come la seguente:

$$\begin{array}{rcl}
 \hline Pila & | & x \\
 \hline Z_0 S & | & aabb \quad \vdash \\
 \\
 Z_0 S & | & abb \quad \vdash \\
 \\
 Z_0 S & | & bb \quad \vdash \\
 \\
 Z_0 A & | & bb \quad \vdash
 \end{array}$$

errore

Si noti anche che una stringa è accettata per mezzo di diverse computazioni se, e solo se, essa possiede diverse derivazioni sinistre, ossia se è ambigua per la grammatica.

Le regole della tabella di p. 148 sono a ben vedere bidirezionali, e possono essere applicate in senso inverso, per passare da un automa a pila del tipo considerato, alla grammatica che genera lo stesso linguaggio.

Mettendo insieme la trasformazione diretta e inversa, un risultato teorico importante è stato acquisito.

Proprietà 4.3. La famiglia dei linguaggi liberi coincide con quella dei linguaggi riconosciuti a pila vuota da un automa a pila indeterministico, avente un solo stato.

Si sottolinea che la costruzione precedente non vale per gli automi a pila aventi più stati, per i quali si dovranno utilizzare metodi un po' più articolati.

L'obiettivo di questo capitolo potrebbe sembrare raggiunto poiché si dispone ora d'un procedimento per costruire il riconoscitore del linguaggio definito da una grammatica. Purtroppo l'automata non è deterministico e esplora tutte le vie, con una complessità di calcolo non polinomiale rispetto alla lunghezza della stringa sorgente. Seguiranno quindi altri algoritmi più efficienti.

Complessità di calcolo dell'automata a pila

Si calcola ora un limite superiore sul numero di passi necessari nel caso peggiore per riconoscere una stringa, con l'automata a pila sopra descritto. Per semplicità si suppone che la grammatica G del linguaggio sia nella forma di Greibach (p. 66) in cui, come si ricorda, ogni regola inizia con un terminale e non contiene altri terminali. Pertanto non vi sono mosse spontanee (tipi (1) e (3) della tabella di p. 148) e l'automata non impila mai un carattere terminale.

Data una stringa x di lunghezza n , la derivazione $S \xRightarrow{+} x$, se esiste, ha esattamente n passi, e altrettante mosse compie l'automata per riconoscere x . Sia K il massimo tra i numeri delle alternative $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_k$ dei non-terminali della grammatica. A ogni passo della derivazione, per espandere il nonterminale può essere scelta una tra $k \leq K$ alternative, quindi il numero delle possibili derivazioni di lunghezza n è al più K^n . Poiché nel caso peggiore, l'algoritmo può essere obbligato a esaminare tutte queste possibilità, prima di trovare quella giusta, la complessità di calcolo è esponenziale.

Tuttavia questo risultato è migliorabile; alla fine del capitolo un ragionamento più fine condurrà a un algoritmo di riconoscimento di complessità polinomiale, valido per ogni grammatica libera.

4.1.3 Definizione dell'automata a pila

Nell'esporre i vari modelli della macchina a pila, si cercherà di ridurre i dettagli, contando sulla capacità del lettore, di adattare a questi automi i concetti

già studiati per quelli finiti.

Per definire un automa M a pila occorrono sette entità:

1. Q , l'insieme finito degli stati dell'unità di controllo;
2. Σ , l'alfabeto d'ingresso;
3. Γ , l'alfabeto della pila;
4. δ , la funzione di transizione;
5. $q_0 \in Q$, lo stato iniziale;
6. $Z_0 \in \Gamma$, il simbolo iniziale della pila;
7. $F \subseteq Q$, l'insieme degli stati finali.

L'automato così definito è in generale indeterministico. Il dominio e codominio della funzione di transizione sono espressi da prodotti cartesiani tra insiemi:

$$\begin{array}{c|c} \text{dominio} & \text{codominio} \\ \hline Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma & \text{le parti finite dell'insieme } Q \times \Gamma^* \end{array}$$

Le mosse, cioè i valori di δ , possono essere così classificate e descritte:

- mossa con lettura:

$$\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_n, \gamma_n)\}$$

con $n \geq 1$, $a \in \Sigma$, $Z \in \Gamma$ e con $p_i \in Q$, $\gamma_i \in \Gamma^*$

L'automato nello stato q con Z in cima alla pila, leggendo a , può entrare in uno degli stati p_i , $1 \leq i \leq n$, dopo aver eseguito in successione le operazioni pop, push(γ_i).

Note: la scelta dell'azione i -esima tra le n possibili non è deterministica; l'avanzamento della testina di ingresso è automatico; il simbolo in cima è sempre disimpilato; la stringa impilata può essere vuota.

- mossa spontanea:

$$\delta(q, \varepsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_n, \gamma_n)\}$$

con le stesse indicazioni di prima. L'automato, nello stato q con Z in cima alla pila, senza leggere alcun carattere di ingresso, entra in uno degli stati p_i , $1 \leq i \leq n$, dopo aver eseguito in successione le operazioni pop, push(γ_i).

Nella definizione è evidente che il comportamento può essere indeterministico, sia perché il codominio è un insieme di possibili scelte, sia per la presenza di mosse spontanee.

Si chiama *configurazione istantanea* della macchina M una terna $(q, y, \eta) \in Q \times \Sigma^* \times \Gamma^+$, che descrive:

- q , lo stato attuale del controllo;
- y , la parte (suffisso) ancora da leggere della stringa x ;
- η , il contenuto della pila.

La configurazione *iniziale* è (q_0, x, Z_0) o $(q_0, x \neg, Z_0)$, se si usa il terminatore. Il passaggio o transizione da una configurazione a un'altra, si indica con $(q, y, \eta) \mapsto (p, z, \lambda)$. Un calcolo è una catena di zero più transizioni, indicato con \mapsto^* . Al solito, si scrive la croce al posto della stella per denotare un calcolo di almeno una transizione.

Le mosse, con lettura e senza, danno luogo alle seguenti transizioni:

Conf. presente	Conf. successiva	Mossa applicata
$(q, az, \eta Z)$	$(p, z, \eta \gamma)$	mossa con lettura: $\delta(q, a, Z) = \{(p, \gamma), \dots\}$
$(q, az, \eta Z)$	$(p, az, \eta \gamma)$	mossa spontanea: $\delta(q, \varepsilon, Z) = \{(p, \gamma), \dots\}$

Benché ogni mossa cancelli il simbolo in cima, esso può essere impilato di nuovo dalla stessa mossa, se non deve essere rimosso dalla pila.

Una stringa x è *riconosciuta* (o accettata) dalla macchina M mediante stato finale se esiste un calcolo che legge per intero la stringa e termina in uno stato finale:

$$(q_0, x, Z_0) \mapsto^* (q, \varepsilon, \lambda), \quad q \text{ è uno stato finale, } \lambda \in \Gamma^*$$

Al solito il linguaggio riconosciuto è l'insieme delle stringhe accettate.

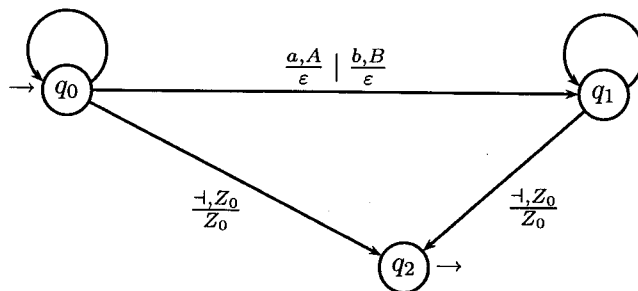
Si noti che, all'atto del riconoscimento, la pila contiene una stringa λ , sulla quale non è posta alcuna condizione, e che talora potrà essere la stringa vuota.

Diagramma stato-transizione per automi a pila

Si possono rappresentare graficamente i diagrammi stato-transizione, benché l'effetto sia di lettura meno immediata rispetto agli automi finiti, dovendosi appesantire la rappresentazione con le operazioni sulla pila, come mostra il prossimo esempio.

Esempio 4.4. Il linguaggio $L = \{uu^R \mid u \in \{a, b\}^*\}$ dei palindromi (p. 31) di lunghezza pari è accettato mediante stato finale dal riconoscitore a pila:

$$\frac{a, A}{AA} \mid \frac{a, B}{BA} \mid \frac{a, Z_0}{Z_0 A} \mid \frac{b, A}{AB} \mid \frac{b, B}{BB} \mid \frac{b, Z_0}{Z_0 B} \qquad \frac{a, A}{\varepsilon} \mid \frac{b, B}{\varepsilon}$$



L'alfabeto della pila ha tre simboli: Z_0 , il simbolo iniziale, A e B che memorizzano rispettivamente l'avvenuta lettura d'una a e d'una b . Ad es. l'arco

$q_0 \xrightarrow{\frac{a, B}{B, A}} q_0$ denota la mossa con lettura $(q_0, BA) \in \delta(q_0, a, B)$.

Il funzionamento indeterministico si manifesta in q_0 , dove, leggendo a dall'ingresso e A dalla pila, la macchina può restare in q_0 impilando AA , oppure andare in q_1 senza impilare.

Data la stringa $x = aa$ si hanno, tra altri possibili, i seguenti due calcoli.

<u>Pila</u>	<u>x</u>	<u>Stato</u>	<u>Commento</u>
Z_0	$aa \dashv$	q_0	
Z_0A	$a \dashv$	q_0	
Z_0AA	\dashv	q_0	rifiuto: nessuna mossa è definita per (q_0, \dashv, A)

<u>Pila</u>	<u>x</u>	<u>Stato</u>	<u>Commento</u>
Z_0	$aa \dashv$	q_0	
Z_0A	$a \dashv$	q_0	
Z_0	\dashv	q_1	
Z_0	\dashv	q_2	riconoscimento nello stato finale

La stringa aa è accettata perché esiste un calcolo che la scandisce per intero e giunge allo stato finale. Similmente la stringa vuota è riconosciuta con la mossa da q_0 a q_2 .

Varietà di automi a pila

L'automa a pila sopra definito si differenzia da quello precedentemente (p. 148) costruito in modo diretto dalla grammatica, in due aspetti: possiede gli stati interni e usa una diversa condizione per accettare le stringhe.

Modalità di accettazione

Già si sono incontrate due varianti sul riconoscimento al termine del calcolo: la macchina entra in uno stato finale, o la pila è vuota. Il primo caso, riconoscimento *a stato finale*, prescinde dal contenuto della pila. Il secondo caso, riconoscimento *a pila vuota*, prescinde invece dallo stato in cui si trova l'automa.

È anche possibile una modalità di riconoscimento combinata: *a stato finale e a pila vuota*.

Proprietà 4.5. Per la famiglia degli automi a pila indeterministici dotati d'uno o più stati interni, le modalità di accettazione

1. a pila vuota
2. a stato finale
3. combinata (stato finale e pila vuota)

hanno la stessa capacità di riconoscimento del linguaggio.

Infatti se l'automa riconosce a stato finale, esso può essere facilmente modificato in modo da riconoscere a pila vuota: quando raggiunge uno stato finale, esso entra in un nuovo stato, nel quale poi resta, svuotando la pila con mosse spontanee, fino a quando il simbolo di fondo pila affiora.

Viceversa, se l'automa riconosce a pila vuota, le modifiche, per ottenere il riconoscimento a stato finale, sono le seguenti:

- si aggiunge uno stato nuovo finale f e la mossa che in esso entra quando la pila si vuota;
- quando, per effetto d'una mossa che porta l'automa originale nello stato q , la pila cessa di essere vuota, il nuovo automa va dallo stato f allo stato q .

Analoghe considerazioni valgono per il terzo caso.

Funzionamento senza cicli spontanei e in linea

Un automa potrebbe eseguire un numero illimitato di mosse senza leggere un carattere d'ingresso; in tal caso esso entra in un *ciclo* detto *spontaneo*, composto esclusivamente di mosse spontanee. Tale comportamento potrebbe da un lato impedire all'automa di leggere per intero la stringa sorgente; dall'altro lato l'automa, a lettura completata, potrebbe aver bisogno d'un numero illimitato di mosse spontanee prima di decidere se accettare o meno. Ma questi comportamenti poco graditi si possono eliminare dagli automi a pila senza perdita di generalità.

Il seguente ragionamento² mostra che si può sempre costruire un automa equivalente privo di cicli spontanei. Tale automa può essere programmato in modo da leggere l'intera stringa sorgente, quale che sia, e di fermarsi subito al termine della lettura.

Per primo, si trasforma l'automa dato in modo che, in ogni configurazione, se b è il carattere corrente, sia definita una mossa che legge b o una mossa spontanea o entrambe.

Si osservi che, se l'automa ha un ciclo spontaneo e dunque un calcolo ciclico

²Per un approfondimento di questo e del successivo punto si può vedere ad es. [32, 24].

ripetibile all'infinito, tale calcolo necessariamente passa per una configurazione con uno stato p e una pila γA , tali che l'automa può eseguire un numero illimitato di mosse spontanee senza consumare la cima A della pila. Secondo, si modifica la macchina aggiungendo due nuovi stati; se durante il ciclo spontaneo l'automa non può mai raggiungere una configurazione finale, si aggiunge un nuovo stato d'errore p_E e la mossa

$$p \xrightarrow{\frac{\epsilon, A}{A}} p_E$$

Altrimenti, si crea anche un nuovo stato finale p_F e le mosse

$$p \xrightarrow{\frac{\epsilon, A}{A}} p_F, \quad p_F \xrightarrow{\frac{\epsilon, A}{A}} p_E$$

Infine lo stato d'errore p_E può essere programmato in modo da consumare quanto resta da leggere della stringa sorgente.

Si dice che un automa funziona *in linea* (on line) se esso, alla lettura dell'ultimo carattere della stringa, può subito decidere se accettarla, senza dover fare ulteriori mosse.

Un automa a pila può sempre essere convertito in un automa equivalente del tipo in linea. Grazie al risultato precedente si può supporre che l'automa sia privo di cicli spontanei. Al termine della lettura della stringa, esso nello stato p potrebbe aver bisogno d'un numero finito di mosse spontanee che esaminano una parte finita ("tappo") della pila, di lunghezza massima k , prima accettare (o di rifiutare). Nel primo caso il tappo si dirà di accettazione, nel secondo di rifiuto. Si modifica l'automa in modo che esso durante il calcolo memorizzi il tappo anche nello stato. Allora esso, invece di entrare nello stato p , entrerà in uno stato che rappresenta la combinazione di p e del tappo. Se il tappo era di accettazione, l'automa accetta altrimenti rifiuta.

4.2 Linguaggi liberi e automi a pila: una sola famiglia

Si dimostra ora che il linguaggio accettato da un automa a pila dotato di stati è libero e di conseguenza, poiché si sa (p. 150) che ogni linguaggio libero è riconosciuto da un automa a pila, vale l'enunciato seguente.

→ **Proprietà 4.6.** La famiglia *LIB* dei linguaggi liberi coincide con quella dei linguaggi riconosciuti da automi a pila.

Dimostrazione. Dato il linguaggio $L = L(M)$, riconosciuto da un automa a pila M dotato d'uno o più stati, con modalità di accettazione a pila vuota, si costruirà ora la grammatica G che lo genera. Essa non è pulita e dovrà essere rifinita eliminando le regole inutili. La seguente tabella mostra i componenti dell'automa (dato) e della grammatica (da costruire):

<i>automa</i>	
Alfabeto terminale:	$\Sigma = \{a, \dots\}$
Stati:	$Q = \{q_0, q, q_i, \dots\}$
Stato iniziale:	q_0
Simboli di pila:	$\Gamma = \{A, B, B_i, \dots\}$
Simbolo iniziale:	Z_0
Mossa:	$q \xrightarrow[B_n B_{n-1} \dots B_1]{a, A} q_1$
<i>grammatica</i>	
Nonterminali:	$\langle q_i, A, q_j \rangle$ dove $q_i, q_j \in Q$ e $A \in \Gamma$
Assioma	S
Regole iniziali:	$\forall q \in Q : S \rightarrow \langle q_0, Z_0, q \rangle$
Regole:	$\langle q, A, q_m \rangle \rightarrow a \langle q_1, B_1, q_2 \rangle \langle q_2, B_2, q_3 \rangle \dots \langle q_{m-1}, B_{m-1}, q_m \rangle$ per ogni $a \in \Sigma \cup \{\varepsilon\}$, per ogni stato $q, q_1, q_2, \dots, q_m \in Q$, e per ogni simbolo di pila $A, B_1, B_2, \dots, B_m \in \Gamma$ tali che esiste la mossa $q \xrightarrow[B_m B_{m-1} \dots B_1]{a, A} q_1$. Nel caso particolare $m = 0$: la regola è $\langle q, A, q_1 \rangle \rightarrow a$

Le regole della grammatica sono costruite in modo che, a ogni calcolo valido dell'automa, corrisponda una derivazione sinistra. Come nella vecchia corrispondenza tra grammatiche e automi (p. 148), un nonterminale è associato a un simbolo della pila; in più un nonterminale è ora contraddistinto da due stati, che hanno questo significato. Dal nonterminale $\langle q, A, r \rangle$ si deriva la stringa z , se, e solo se, l'automa, partendo nello stato q con A in cima alla pila, esegue un calcolo che legge la stringa z , raggiunge lo stato r e cancella A dalla pila.

Omettendo la dimostrazione³ della correttezza della costruzione, si passa a un esempio.

Esempio 4.7. Grammatica equivalente all'automa a pila (Harrison).
Il linguaggio

$$L = \{a^n b^m a^n \mid m, n \geq 1\} \cup \{a^n b^m c^m \mid m, n \geq 1\}$$

è l'unione di due linguaggi fatti di tre parti concatenate. Le prime due parti sono le stesse, mentre la terza parte usa la lettera a nel primo linguaggio e la lettera c nel secondo. Nel primo linguaggio il primo e il terzo gruppo sono

³Vedasi ad es. [24, 27, 28, 43].

eguali; nel secondo, il secondo e terzo gruppo hanno eguale lunghezza. Il linguaggio è accettato a pila vuota da un automa a tre stati, che opera, con i seguenti passi:

1. il simbolo iniziale in pila è C e lo stato iniziale è 0;
2. impila le lettere a (come A) e poi le b (come B), via via che sono lette;
3. leggendo la prima lettera della terza parte, va nello stato 1 se è a o nello stato 2 se è c ;
4. nello stato 1 cancella tutte le B dalla pila, e poi pareggia le rimanenti a dell'ingresso con le A presenti nella pila;
5. nello stato 2, pareggia le rimanenti c in ingresso con le B presenti nella pila, poi cancella le A dalla pila;

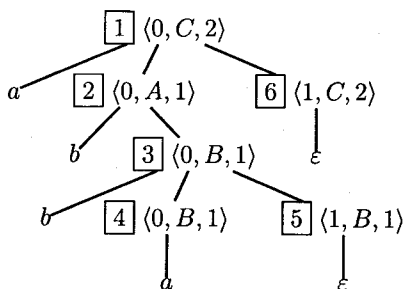
Nella prima colonna della prossima tabella sono mostrate le mosse dell'auto-
ma, con accanto le regole costruite per la grammatica. Le regole inutili sono
state omesse.

<i>mossa</i>	<i>regole</i>
$0 \xrightarrow[\substack{a,C \\ CA}}{0} 0$	$\langle 0, C, 2 \rangle \rightarrow a \langle 0, A, 1 \rangle \langle 1, C, 2 \rangle \mid a \langle 0, A, 2 \rangle \langle 2, C, 2 \rangle$
$0 \xrightarrow[\substack{a,A \\ AA}}{0} 0$	$\langle 0, A, 1 \rangle \rightarrow a \langle 0, A, 1 \rangle \langle 1, A, 1 \rangle, \quad \langle 0, A, 2 \rangle \rightarrow a \langle 0, A, 2 \rangle \langle 2, A, 2 \rangle$
$0 \xrightarrow[\substack{b,A \\ B}}{0} 0$	$\langle 0, A, 1 \rangle \rightarrow b \langle 0, B, 1 \rangle, \quad \langle 0, A, 2 \rangle \rightarrow b \langle 0, B, 2 \rangle$
$0 \xrightarrow[\substack{b,B \\ BB}}{0} 0$	$\langle 0, B, 1 \rangle \rightarrow b \langle 0, B, 1 \rangle \langle 1, B, 1 \rangle, \quad \langle 0, B, 2 \rangle \rightarrow b \langle 0, B, 2 \rangle \langle 2, B, 2 \rangle$
$0 \xrightarrow[\substack{a,B \\ \epsilon}}{1} 1$	$\langle 0, B, 1 \rangle \rightarrow a$
$1 \xrightarrow[\substack{\epsilon,B \\ \epsilon}}{1} 1$	$\langle 1, B, 1 \rangle \rightarrow \epsilon$
$1 \xrightarrow[\substack{a,A \\ \epsilon}}{1} 1$	$\langle 1, A, 1 \rangle \rightarrow a$
$1 \xrightarrow[\substack{\epsilon,C \\ \epsilon}}{2} 2$	$\langle 1, C, 2 \rangle \rightarrow \epsilon$
$0 \xrightarrow[\substack{c,B \\ \epsilon}}{2} 2$	$\langle 0, B, 2 \rangle \rightarrow c$
$2 \xrightarrow[\substack{c,B \\ \epsilon}}{2} 2$	$\langle 2, B, 2 \rangle \rightarrow c$
$2 \xrightarrow[\substack{\epsilon,A \\ \epsilon}}{2} 2$	$\langle 2, A, 2 \rangle \rightarrow \epsilon$
$2 \xrightarrow[\substack{\epsilon,C \\ \epsilon}}{2} 2$	$\langle 2, C, 2 \rangle \rightarrow \epsilon$

L'assioma è $\langle 0, C, 2 \rangle$.

Segue un calcolo dell'auto-
ma e il corrispondente albero di derivazione, con i
passi numerati:

$$\begin{aligned}
\langle 0, abba, C \rangle &\mapsto \langle 0, bba, CA \rangle \\
&\mapsto \langle 0, ba, CB \rangle \\
&\mapsto \langle 0, a, CBB \rangle \\
&\mapsto \langle 1, \varepsilon, CB \rangle \\
&\mapsto \langle 1, \varepsilon, C \rangle \\
&\mapsto \langle 2, \varepsilon, \varepsilon \rangle
\end{aligned}$$



La corrispondenza tra i due modelli si comprende confrontando il calcolo

$$\langle 0, abba, C \rangle \xrightarrow{3} \langle 0, a, CBB \rangle$$

con la derivazione sinistra

$$\langle 0, C, 2 \rangle \xrightarrow{3} abb\langle 0, B, 1 \rangle \langle 1B, 1 \rangle \langle 1, C, 2 \rangle$$

grazie alle seguenti osservazioni:

- il prefisso della stringa, letto dall'automa, e quello generato dalla derivazione sono identici: *abb*
- il contenuto della pila *CBB* è la stringa riflessa di quella ottenuta concatenando i simboli centrali delle terne (nonterminali) presenti nella stringa derivata;
- due terne (nonterminali) adiacenti nella stringa derivata sono “incatenate” dall'eguaglianza degli stati contraddistinti dallo stesso tipo di freccia:

$$\langle 0, B, 1 \rangle \xrightarrow{1} \langle 1, B, 1 \rangle \xrightarrow{1} \langle 1, C, 2 \rangle$$

Grazie a queste corrispondenze, che valgono in ogni passo, i due modelli definiscono lo stesso linguaggio.

4.2.1 Intersezione di linguaggi liberi e regolari

Come illustrazione dei concetti precedenti, l'affermazione (cap. 2, p. 78), che l'intersezione d'un linguaggio libero con uno regolare è un linguaggio libero, è ora facile da giustificare. Data la grammatica G e l'automa finito A , si mostra come ottenere l'automa a pila M che riconosce $L(G) \cap L(A)$.

Prima si costruisce, con il metodo di p. 148, l'automa N , avente un solo stato, che a pila vuota riconosce il linguaggio $L(G)$.

Poi si costruisce la macchina M prodotto cartesiano delle due macchine N e A , applicando sostanzialmente la stessa costruzione per gli automi finiti (p. 138). L'unica modifica riguarda la pila: la macchina prodotto M esegue sulla

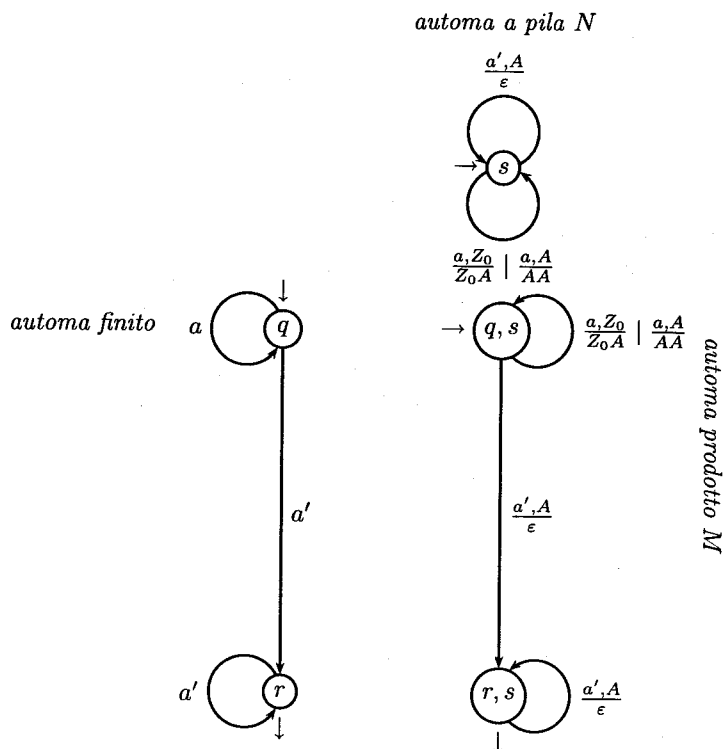
pila le stesse operazioni dell'automa componente N . La macchina ottenuta ha dunque come insieme degli stati il prodotto cartesiano degli stati delle macchine componenti. Essa riconosce mediante stato finale e pila vuota; sono finali gli stati che contengono uno stato finale dell'automa finito A . ←

→ Circa il determinismo, la macchina è deterministica se entrambe le macchine componenti lo sono.

È facile comprendere che l'automa a pila riconosce a stato finale una stringa se, e solo se, entrambe le macchine componenti la accettano, ossia se essa appartiene all'intersezione dei due linguaggi.

Esempio 4.8. Si vuole (come nell'es. 2.82 di p. 80) l'intersezione del linguaggio di Dyck di alfabeto $\Sigma = \{a, a'\}$ con il linguaggio regolare $a^*a'^+$. Il risultato è il linguaggio $\{a^n a'^n \mid n \geq 1\}$.

Il linguaggio di Dyck è facilmente riconosciuto a pila vuota da un automa (deterministico) con un solo stato. I riconoscitori dei due linguaggi e l'automa del prodotto cartesiano sono:



Ad es. l'arco da $\{q, s\}$ a $\{r, s\}$, associato alla lettura di a' , opera sulla pila come l'automa N cancellando una A e va dallo stato q allo stato r come l'automa finito.

Poiché l'automa a pila componente accetta a pila vuota, e quello finito riconosce nello stato finale r , la macchina costruita riconosce a pila vuota nello stato finale (r, s) .

4.2.2 Automi a pila e linguaggi deterministici

È importante approfondire lo studio dei riconoscitori deterministici e dei loro linguaggi, perché sono i più usati nei compilatori a ragione della loro efficienza. Nel comportamento d'un automa a pila (definito come a p. 151), si possono incontrare tre forme di indeterminismo.

- 1. Incertezza tra più mosse di lettura, se per stato q , carattere a e simbolo A di pila, la funzione di transizione ha più valori: $|\delta(q, a, A)| > 1$.
- 2. Incertezza tra una mossa spontanea e una mossa di lettura, ossia sono definite entrambe le mosse $\delta(q, \varepsilon, A)$ e $\delta(q, a, A)$.
- 3. Incertezza tra più mosse spontanee; ossia per qualche stato q e simbolo A di pila, la funzione $\delta(q, \varepsilon, A)$ ha più valori: $|\delta(q, \varepsilon, A)| > 1$.

Se, nella funzione di transizione, nessuna delle forme 1, 2 e 3 è presente, l'automa a pila è *deterministico*. Il linguaggio riconosciuto da un automa deterministico è detto (libero) *deterministico*, e la famiglia di tali linguaggi è chiamata *DET*.

Esempio 4.9. Forme di indeterminismo.

Il riconoscitore (p. 148) del linguaggio

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

ha un indeterminismo della forma 1

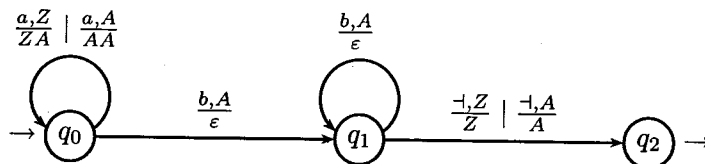
$$\delta(q_0, a, A) = \{(q_0, b), (q_0, bA)\}$$

e anche della forma 2

$$\delta(q_0, \varepsilon, S) = \{(q_0, A)\} \quad \delta(q_0, a, S) = \{(q_0, S)\}$$

(L'unico stato è indicato come q_0).

Lo stesso linguaggio è riconosciuto deterministicamente dall'automa $M_2 = (\{q_0, q_1, q_2\}, \{a, b\}, \{A, Z\}, \delta, q_0, Z, \{q_2\})$:



Intuitivamente M impila le a , codificate come A ; leggendo la prima b cancella una A e va nello stato q_1 . Poi, per ogni b letta, disimpila una A . Se vi fossero più b che a , esso cadrebbe in errore. Leggendo il terminatore, esso passa nello stato q_2 finale, quale che sia il simbolo in cima.

Proprietà di chiusura dei linguaggi deterministici

Essendo i linguaggi deterministici una sottoclasse dei linguaggi liberi, essi godono di proprietà specifiche. Si aggiorna qui il quadro delle proprietà, delle famiglie di linguaggi, continuando quello presentato a p. 78. Si indica con L, D e R un linguaggio appartenente rispettivamente alla famiglia LIB, DET e REG .

Operazione	Proprietà	(Proprietà nota)
Riflessione	$D^R \notin DET$	$D^R \in LIB$
Stella	$D^* \notin DET$	$D^* \in LIB$
Complemento	$\neg D \in DET$	$\neg L \notin LIB$
Unione	$D_1 \cup D_2 \notin DET, D \cup R \in DET$	$D_1 \cup D_2 \in LIB$
Concatenamento	$D_1.D_2 \notin DET, D.R \in DET$	$D_1.D_2 \in LIB$
Intersezione	$D \cap R \in DET$	$D_1 \cap D_2 \notin LIB$

Seguono alcune giustificazioni ed esempi.⁴

Riflessione: il linguaggio

$$L_1 = \{a^n b^n e\} \cup \{a^n b^{2n} d\}, n \geq 1$$

soddisfa l'eguaglianza $|x|_a = |x|_b$ quando la frase termina con e , e l'eguaglianza $2|x|_a = |x|_b$ se la frase termina con d . Esso non è deterministico, ma il linguaggio speculare sì. Infatti la lettura del primo carattere permette all'automata di scegliere quale eguaglianza applicare in seguito.

Complemento: il complemento d'un linguaggio deterministico è tale; la dimostrazione (analogica a quella dei linguaggi regolari di p. 136) costruisce il riconoscitore del complemento scambiando gli stati finali e non, e creando lo stato pozzo.⁵ Se dunque un linguaggio libero ha come complemento un linguaggio non libero, esso non può essere deterministico.

⁴Al solito un enunciato come ad es. $D^R \notin DET$ va inteso nel senso che esiste un linguaggio D tale che D^R non è deterministico.

⁵Si veda ad es. [24, 27]. Esiste anche un metodo [26] per trasformare la grammatica d'un linguaggio deterministico in quella del suo complemento.

Unione: l'es. 4.12 di p. 164 mostrerà che l'unione di linguaggi deterministici (tali sono ovviamente L' e L'') non è in generale deterministica.

Per l'identità di De Morgan si ha $D \cup R = \neg(\neg D \cap \neg R)$, che è un linguaggio deterministico, per via dei punti seguenti: il complemento d'un linguaggio deterministico (regolare) è deterministico (regolare); l'intersezione d'un linguaggio deterministico e d'uno regolare è deterministica (punto successivo).

Intersezione $D \cap R$: l'automa prodotto cartesiano di un automa a pila deterministico e d'un automa finito deterministico è deterministico.

Per mostrare che l'intersezione di due linguaggi deterministici può uscire da *DET*, basta ricordare il linguaggio, non libero, dei tre esponenti (p. 76); esso è l'intersezione di due linguaggi facilmente deterministici:

$$\{a^n b^n c^n \mid n \geq 0\} = \{a^n b^n c^* \mid n \geq 0\} \cap \{a^* b^n c^n \mid n \geq 0\}$$

Concatenamento e stella: concatenando due linguaggi deterministici, può accadere che non si riesca a individuare deterministicamente la fine delle stringhe del primo linguaggio, quindi il punto a partire dal quale occorre riconoscere le stringhe del secondo linguaggio. Esempio: dati i linguaggi

$$L_0 = \{a^i b^i a^j \mid i, j \geq 1\} \quad L_1 = \{a^i b^j a^j \mid i, j \geq 1\} \quad R = \{c, c^2\}$$

il linguaggio $L = cL_0 \cup L_1$ è deterministico, ma il concatenamento RL no.⁶

Infatti la presenza d'un prefisso cc può essere interpretata in due modi, provenienti da $c.cL_0$ oppure da $c^2.L_1$.

Analoga situazione si incontra prendendo la stella d'un linguaggio deterministico.

Concatenamento con linguaggio regolare: il riconoscitore di $D.R$ può essere costruito componendo *in cascata* l'automa a pila deterministico e l'automa finito deterministico. Più precisamente, quando l'automa a pila entra nello stato finale che riconosce una frase di D , la nuova macchina passa nello stato iniziale dell'automa che riconosce R e lo simula, da quel punto in avanti.

Si nota dalla tabella che le operazioni di base della teoria dei linguaggi non preservano il determinismo. Questa può creare qualche difficoltà per il progettista d'un linguaggio artificiale: basti pensare che, unendo due linguaggi esistenti deterministici, il risultato può essere indeterministico (o addirittura ambiguo). Lo stesso dicasi per il concatenamento e la stella. Il progettista dovrà assicurarsi che la proprietà di determinismo sia mantenuta, dopo ogni modifica del linguaggio da progettare.

Infine conviene menzionare che, tra *DET* e *LIB*, vi è una sostanziale differenza rispetto alle proprietà decidibili: dati due automi a pila deterministici, esiste un algoritmo⁷ per decidere se essi sono equivalenti, ossia se riconoscono lo stesso linguaggio, mentre per automi a pila generici ciò non è decidibile.

⁶La dimostrazione si trova in [24].

⁷Vedasi [44].

Linguaggi non deterministici

Diversamente dal caso dei linguaggi regolari, vi sono linguaggi liberi che non possono essere riconosciuti da un automa deterministico.

Proprietà 4.10. La famiglia DET dei linguaggi deterministici è contenuta strettamente in quella LIB dei linguaggi liberi.

L'enunciato segue da due fatti noti: l'inclusione $DET \subseteq LIB$, essendo l'automato deterministico a pila un caso particolare dell'automato a pila generale; e la diseuguaglianza $DET \neq LIB$, poiché certe proprietà di chiusura valgono per una famiglia ma non per l'altra. Ciò basterebbe, ma è interessante mostrare direttamente che certi linguaggi liberi non sono deterministici, proprio allo scopo di riconoscere tali paradigmi per evitarli nell'uso pratico.

Lemma del doppio servizio

Uno strumento concettuale per dimostrare che un certo linguaggio libero non è deterministico si basa sull'analisi delle frasi che sono un prefisso di altre frasi.

Sia D un linguaggio deterministico, $x \in D$ una sua frase, e vi sia un'altra frase $y \in D$ tale da essere un prefisso di x , ovvero $x = yz$, dove tutte le stringhe possono essere vuote.

Si definisce ora un nuovo linguaggio, detto il linguaggio del *doppio servizio* di D , ottenuto inserendo un nuovo terminale diesis \sharp , tra le stringhe y e z :

$$ds(D) = \{y\sharp z \mid y \in D \wedge z \in \Sigma^* \wedge yz \in D\}$$

Ad es. per il linguaggio $F = \{a, ab, bb\}$ si ha

$$ds(F) = \{a\sharp b, a\sharp, ab\sharp, bb\sharp\}$$

dove appaiono le frasi originali terminate da \sharp e anche le frasi segmentate dal diesis nel prefisso (appartenente al linguaggio) e nel suffisso.

Lemma 4.11. Se D è un linguaggio deterministico, allora anche il linguaggio $ds(D)$ del doppio servizio è deterministico.

Per dimostrare l'enunciato, si trasforma il riconoscitore deterministico M di D in un automa deterministico M' , che riconosce il linguaggio del doppio servizio, come ora spiegato. Conviene supporre che l'automato M funzioni *in linea* (p. 154); ossia che, via via che scandisce la stringa d'ingresso, esso possa decidere se la stringa letta è una frase valida.

Si consideri il calcolo con cui M accetta la stringa y . Se la stringa y è seguita dal diesis, la nuova macchina M' lo legge e accetta se non vi sono altri caratteri (poiché $y\sharp \in ds(D)$ se $y \in D$). Altrimenti M' prosegue deterministicamente il calcolo scandendo la stringa z , nello stesso modo con cui M avrebbe riconosciuto la stringa yz .

Le ragioni del nome del linguaggio sono ora chiare: l'automa svolge contemporaneamente due servizi, in quanto riconosce un prefisso e una stringa più lunga.

Per applicare il lemma a casi concreti, basta osservare che, se il doppio servizio d'un linguaggio libero L non è libero, L non è deterministico.

Esempio 4.12. Unione non deterministica di linguaggi deterministici.
Il linguaggio

$$L = \{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\} = L' \cup L''$$

unione di due linguaggi deterministici, non è deterministico.

Intuitivamente l'automa dovrebbe leggere e impilare le a e, se la stringa (ad es. $aabb$) appartiene al primo insieme, disimpilare una a alla lettura d'una b ; ma se la stringa appartiene al secondo insieme (ad es. $aabbbb$), sono due le b da leggere per disimpilare una a . Non potendo l'automa sapere quale sia la strada giusta (per saperlo dovrebbe contare le b esaminando una parte illimitata della stringa), è obbligato a tentare entrambe le vie.

Più rigorosamente, se per assurdo L fosse deterministico, anche il linguaggio $ds(L)$ lo sarebbe, e, per la proprietà di chiusura di DET (p. 161) rispetto all'intersezione con REG , anche il linguaggio

$$L_R = ds(L) \cap (a^+ b^+ \# b^+)$$

dovrebbe essere deterministico. Ma L_R non è neppure un linguaggio libero, quindi tanto meno un linguaggio deterministico, perché le sue frasi hanno la forma $a^i b^i \# b^i$, $i \geq 1$, con tre esponenti eguali (p. 76) e si sa che tale linguaggio non è libero.

Esempio 4.13. Palindromi.

Analoghe considerazioni mostrano che non è deterministico il linguaggio dei palindromi, L definito dalla grammatica:

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$$

Anche questa dimostrazione interseca il linguaggio del doppio servizio con un linguaggio regolare, allo scopo di ottenere un linguaggio che fuoriesca dalla famiglia LIB . Si guardi il linguaggio

$$L_R = ds(L) \cap (a^* b a^* \# b a^*)$$

Una stringa della forma $a^i b a^j \# b a^k$ appartiene a L_R se, e solo se, $j = i \wedge k = i$. Ma tale linguaggio è di nuovo quello dei tre esponenti, che non è libero.

Determinismo e inambiguità del linguaggio

Se un linguaggio è accettato da un automa deterministico, ogni frase è riconosciuta con uno e un solo calcolo, e si può dimostrare che il linguaggio può

essere generato da una grammatica inambigua.

Si riguardi il procedimento di p. 155, per costruire la grammatica equivalente all'automa, che ne simula i calcoli mediante le proprie derivazioni. La grammatica genera una frase con una certa derivazione sinistra, se, e soltanto se, l'automa ha un corrispondente calcolo che riconosce la stessa frase. In altre parole, due diverse derivazioni sinistre corrispondono necessariamente a calcoli diversi, che accettano stringhe diverse. Di conseguenza vale il seguente enunciato.

→ **Proprietà 4.14.** Sia M un automa a pila deterministico; allora la corrispondente grammatica di $L(M)$, costruita con il procedimento di p. 155, non è ambigua.

Naturalmente non è detto che ogni grammatica del linguaggio sia inambigua. D'altra parte, si ha come corollario che un linguaggio libero inerentemente ambiguo non è deterministico. Infatti, se esso per assurdo fosse deterministico, la proprietà precedente permetterebbe di costruire una grammatica inambigua, mentre per la definizione stessa (p. 56), ogni grammatica del linguaggio è ambigua. Per un linguaggio inerentemente ambiguo non esiste dunque un riconoscitore a pila deterministico.

Esempio 4.15. Linguaggio inerentemente ambiguo e indeterminismo.

Il linguaggio inerentemente ambiguo dell'es. 2.56 (p. 56) può essere scritto come l'unione dei linguaggi

$$L_A = \{a^i b^i c^* \mid i \geq 0\} \cup \{a^* b^i c^i \mid i \geq 0\} = L_1 \cup L_2$$

entrambi deterministici (un'altra prova della non chiusura di DET rispetto all'unione).

Intuitivamente, per riconoscere il linguaggio, un automa deve attuare due metodi necessariamente diversi per le stringhe del primo e del secondo insieme. Nel primo caso deve registrare nella pila le a lette e depennarle, via via che si leggono le b ; e similmente nel secondo caso, con rispettivamente i simboli b e c . Di conseguenza, le frasi, che appartengono a entrambi gli insiemi, sono accettate con due calcoli diversi, e l'automa è indeterministico.

→ Anche per gli automi, si può definire il concetto di ambiguità. Un *automa* è detto *ambiguo* se esiste una frase del linguaggio che è accettata con due calcoli distinti.

Si noti che il concetto di determinismo è più restrittivo di quello di inambiguità: la famiglia degli automi a pila deterministici è strettamente contenuta in quella degli automi a pila inambigui.

Per chiarire l'affermazione conviene riprendere due esempi.

Esempio 4.16. Il linguaggio L_A dell'es. precedente è ambiguo oltre che non deterministico, perché certe frasi sono necessariamente riconosciute da due calcoli diversi.

D'altra parte, il linguaggio dell'es. 4.12

$$L_I = \{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\} = L' \cup L''$$

pur non essendo deterministico (come prima dimostrato), è inambiguo. Infatti i due linguaggi L', L'' da unire sono disgiunti. I due modi di funzionamento corrispondenti (descritti a p. 164) determinano due calcoli diversi, uno solo dei quali può terminare con successo.

Sottoclassi degli automi deterministici a pila

La famiglia *DET* per definizione è quella dei linguaggi riconosciuti dal modello più generale di automa a pila deterministico, quello che possiede più stati e riconosce a stato finale. Diverse limitazioni sugli stati interni e sui modi di riconoscimento, che sono senza effetto nel caso indeterministico, causano una restrizione dei linguaggi riconoscibili da un automa deterministico. Le principali restrizioni sono qui brevemente ricordate.⁸

- Automa con un solo stato: il riconoscimento a pila vuota risulta meno potente di quello a stato finale.
- Limitazioni sul numero degli stati interni: causano una restrizione nella famiglia dei linguaggi riconoscibili; simile effetto hanno le limitazioni sul numero delle configurazioni finali dell'automa.
- Funzionamento in tempo reale, cioè senza mosse spontanee: restringe la classe riconoscibile.

Semplici sottofamiglie deterministiche

In molti ambiti applicativi dell'informatica i linguaggi sono progettati in modo da essere analizzabili deterministicamente: l'esemplificazione comprende quasi tutti i linguaggi programmatici e i linguaggi XML della Rete. Vi sono più modi per assicurare a priori che un linguaggio sia deterministico, imponendo restrizioni sul linguaggio stesso o sulla sua grammatica. Si ottengono così più sottofamiglie di *DET*, tra le quali spiccano per semplicità le due prossime. Altri casi più importanti per le applicazioni corrispondono ai linguaggi $LL(k)$ e $LR(k)$ accettati dagli algoritmi veloci di analisi sintattica studiati più avanti.

Linguaggi deterministici semplici

Una grammatica è detta *deterministica semplice* se soddisfa le seguenti condizioni:

1. ogni parte destra d'una regola inizia con un carattere terminale (pertanto non vi sono regole nulle);
2. per ogni nonterminale A non esistono due alternative che iniziano con lo stesso carattere terminale:

$$\nexists A \rightarrow a\alpha \mid a\beta, \text{ con } a \in \Sigma, \alpha, \beta \in (\Sigma \cup V)^*, \alpha \neq \beta$$

⁸Per approfondimenti si veda [24, 45].

Un esempio è la grammatica $S \rightarrow aSb \mid c$.

Costruendo, mediante la corrispondenza (p. 148) tra le regole e le mosse, l'automa, esso risulta chiaramente deterministico e consuma un carattere a ogni mossa, ossia funziona in tempo reale.

Ma questa restrizione rappresenta un ostacolo eccessivo per un uso pratico della grammatica.

Linguaggi a parentesi

I linguaggi a parentesi, introdotti nel cap. 2 a p. 43, sono deterministici, come è facile vedere. Infatti ogni frase generata da una grammatica parentesizzata, ha una struttura a parentesi che marca l'inizio e la fine della parte destra d'una regola sintattica. Si suppone che la grammatica sia parentesizzata con segni distinti (p. 45) o che, se essa usa un solo tipo di parentesi, non vi siano regole con la stessa parte destra, al fine di garantire l'assenza di ambiguità.

Un semplicissimo algoritmo di riconoscimento è il seguente. Si scandisce la stringa fino a trovare una coppia di parentesi priva di parentesi al proprio interno. Si ricerca la stringa compresa tra le parentesi (estremi inclusi) tra le parti destre delle regole. Se non la si trova il riconoscimento fallisce. Altrimenti si riduce tale stringa al nonterminale corrispondente (parte sinistra della regola). Poi si riprende la ricerca d'una coppia di parentesi allo stesso modo. Si riconosce la stringa sorgente se all'ultimo passo essa si riduce all'assioma.

Non è difficile programmare l'algoritmo sotto forma d'un automa a pila deterministico.

I documenti della Rete nel formato XML sono punteggiati da marche di apertura e di chiusura, che delimitano le varie parti del documento e permettono un loro efficiente riconoscimento. La famiglia dei linguaggi XML è dunque anch'essa deterministica. Essa è simile a quella dei linguaggi definiti dalle grammatiche parentesizzate, ma se ne distacca nel permettere l'uso degli operatori delle espressioni regolari all'interno delle regole grammaticali.⁹

⁹Le grammatiche XML, pur generando linguaggi liberi, si scostano da quelle BNF in vari aspetti. Per un confronto iniziale si veda [9].

4.3 Analisi sintattica

Il resto del capitolo espone gli algoritmi di parsificazione più diffusi, iniziando dai veloci metodi deterministici discendenti e ascendenti, anche per grammatiche estese con espressioni regolari, e terminando con un algoritmo generale di complessità maggiore, ma sempre polinomiale.

Data una grammatica G , l'analizzatore sintattico o parsificatore legge la stringa *sorgente*, e se appartiene al linguaggio $L(G)$, ne produce una derivazione o un albero sintattico; altrimenti si ferma indicando la configurazione in cui l'errore è comparso (diagnosi); eventualmente prosegue l'analisi, saltando le sottostringhe contaminate dall'errore, ossia ripara l'effetto dell'errore.

Trascurando il trattamento degli errori, un analizzatore è dunque un riconoscitore, arricchito con la capacità di produrre la derivazione della stringa. A tale fine, basta che l'automa, quando compie la mossa corrispondente al riconoscimento d'una regola grammaticale, salvi in una struttura dati l'etichetta che la identifica. La struttura, al termine dell'analisi, rappresenterà l'albero sintattico.

Se la stringa sorgente è ambigua, il risultato è un insieme di alberi (o di derivazioni).

4.3.1 Analisi discendente e ascendente

Si sa che uno stesso albero corrisponde a più derivazioni: quella sinistra, quella destra (e tutte le altre di minore interesse). A seconda che si consideri una derivazione destra o sinistra, e dell'ordine in cui essa è ricostruita, si ottengono due classi importanti di analizzatori.

Analisi discendente: costruisce la derivazione sinistra, procedendo dall'assioma, ossia dalla radice dell'albero, verso le foglie; i passi dell'algoritmo sono in corrispondenza con le derivazioni immediate.

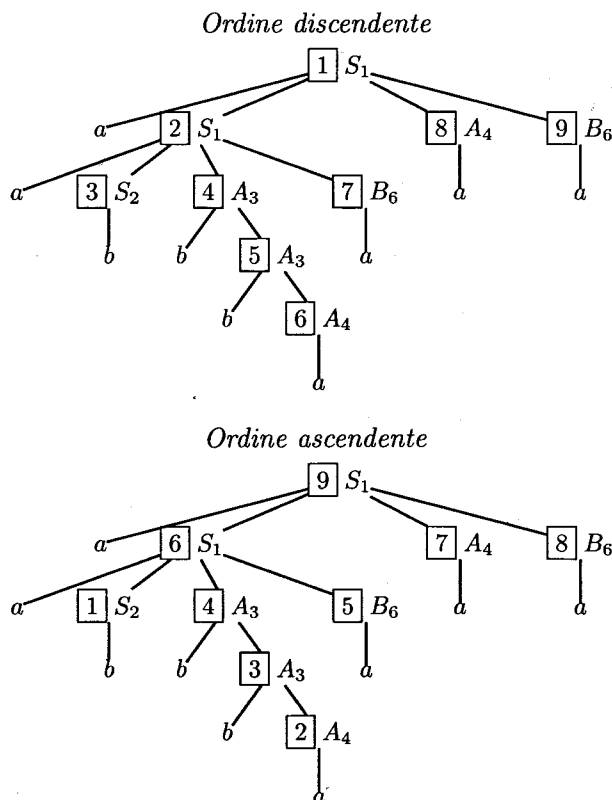
Analisi ascendente: costruisce la derivazione destra della stringa, ma nell'ordine riflesso, cioè, come si vedrà, dalle foglie alla radice dell'albero; i passi dell'algoritmo sono in corrispondenza con le operazioni di riduzione.

Esempio 4.17. Ordini di visita dell'albero.

Per la grammatica

- | | |
|-------------------------|----------------------|
| 1. $S \rightarrow aSAB$ | 2. $S \rightarrow b$ |
| 3. $A \rightarrow bA$ | 4. $A \rightarrow a$ |
| 5. $B \rightarrow cB$ | 6. $B \rightarrow a$ |

e la frase $a^2b^3a^4$, gli ordinamenti corrispondenti alla visita discendente e ascendente sono indicati nelle seguenti figure dai numeri inquadriati, mentre i pedici denotano la regola applicata.



Un analizzatore discendente sviluppa le parti sinistre delle regole nelle corrispondenti parti destre. Se queste contengono dei simboli terminali, essi devono combaciare con quelli presenti nel testo sorgente. Il processo termina quando tutti i simboli nonterminali sono stati trasformati in simboli terminali, che combaciano con il testo (o in stringhe vuote).

Nell'analisi ascendente si riducono in un nonterminale le parti destre delle regole via via incontrate, prima nella stringa sorgente, poi nelle forme di frase da essa ottenute mediante le riduzioni. Il processo termina quando l'intera stringa si è ridotta all'assioma.

In entrambi i casi il processo si interrompe incontrando un errore.

In linea di principio, l'analisi potrebbe invertire l'ordine di scansione, leggendo la stringa dal fondo all'inizio. Ma la totalità dei linguaggi artificiali è progettata per essere elaborata da sinistra a destra (come del resto avviene nella lingue umane dove l'ordine naturale di lettura corrisponde a quello di emissione della frase da parte del parlatore). Per inciso, il cambio di direzione nella scansione del testo può invalidare la proprietà di determinismo d'un linguaggio, perché la famiglia dei linguaggi deterministici non è chiusa rispetto alla riflessione speculare.

4.3.2 La grammatica come rete di automi finiti

Conviene rappresentare la grammatica mediante una rete di automi finiti, allo scopo di facilitare e unificare l'esposizione degli algoritmi di parsificazione.

Data una grammatica G , ogni nonterminale è la parte sinistra di una o più alternative. Se la grammatica è nella forma detta BNF estesa, nella parte destra vi sono anche gli operatori delle espressioni regolari. In tale caso si può supporre che ogni nonterminale abbia una sola regola $A \rightarrow \alpha$, dove α è una e.r. di alfabeto terminale e non. La parte destra α definisce dunque un linguaggio regolare, di cui è facile costruire l'automa finito riconoscitore, M_A .

Nel caso semplice in cui α contenga soltanto simboli terminali, l'automa M_A riconosce il linguaggio $L_A(G)$ generato dalla grammatica partendo dal nonterminale A . Ma in generale in α compaiono anche simboli nonterminali. Una transizione dell'automa etichettata con un nonterminale B è da pensare come l'invocazione d'un altro automa, quello associato alla regola $B \rightarrow \beta$. Ma B può anche coincidere con A , nel qual caso l'invocazione è ricorsiva.

Per evitare confusioni si diranno macchine "prova" gli automi finiti, riservando il termine "automa" a quello a pila che riconosce il linguaggio $L(G)$.

Definizione 4.18. Rete ricorsiva di macchine finite

Le macchine che qui interessano sono deterministiche

- Siano Σ l'alfabeto terminale, $V = \{S, A, B, \dots\}$ i nonterminali, S l'assioma della grammatica libera estesa G .
- Per ogni nonterminale A vi è una sola regola $A \rightarrow \alpha$, la cui parte destra α è una e.r. di alfabeto $\Sigma \cup V$.
- Siano $S \rightarrow \sigma, A \rightarrow \alpha, B \rightarrow \beta, \dots$ le regole della grammatica. Si indicano con R_S, R_A, R_B, \dots i linguaggi regolari, di alfabeto $\Sigma \cup V$, definiti dalle e.r. $\sigma, \alpha, \beta, \dots$.
- M_S, M_A, M_B, \dots sono i nomi delle macchine finite deterministiche che accettano i linguaggi regolari R_S, R_A, \dots . L'insieme delle macchine, ossia la rete, è indicata da M .
- Per chiarezza, conviene supporre che i nomi degli stati interni delle macchine siano tutti distinti. L'insieme degli stati della macchina M_A è Q_A , lo stato iniziale è $q_{A,0}$ e gli stati finali sono F_A . La funzione di transizione di ogni macchina può essere indicata con lo stesso nome δ senza rischio di confusione, visto che gli insiemi degli stati sono disgiunti.
- Per uno stato q della macchina M_A , si indica con $R(M_A, q)$ (o con $R(q)$ se la macchina è sottintesa) il linguaggio regolare, di alfabeto $\Sigma \cup V$, accettato dalla macchina partendo dallo stato q . Nel caso q sia lo stato iniziale, risulta $R(M_A, q_{A,0}) \equiv R_A$.

Una regola $A \rightarrow \alpha$ è perfettamente rappresentata dal grafo della macchina M_A , essendovi corrispondenza biunivoca tra le stringhe definite da α e i cammini del grafo conducenti dallo stato iniziale a uno stato finale.

Poiché l'insieme delle macchine $\mathcal{M} = \{M_S, M_A, \dots\}$ è una pura variante notazionale, i concetti studiati per le grammatiche continuano a valere, in particolare quello di derivazione. Così il linguaggio terminale (di alfabeto Σ) $L(\mathcal{M})$ definito (o riconosciuto) dalla rete di macchine coincide con quello generato dalla grammatica, $L(G)$.

Inoltre servirà il linguaggio terminale definito da una macchina generica M_A , partendo da uno stato di essa anche diverso da quello iniziale. Per uno stato q della macchina M_A si pone

$$L(M_A, q) = \{y \in \Sigma^* \mid \eta \in R(M_A, q) \wedge \eta \xrightarrow{*} y\}$$

Se la macchina è sottintesa basta scrivere $L(q)$. La formula considera ogni stringa η , contenente terminali o nonterminali, accettata dalla macchina M_A partendo dallo stato q . Le derivazioni che originano da η producono le stringhe terminali appartenenti al linguaggio $L(q)$.

In particolare per quanto detto risulta:

$$L(M_A, q_{A,0}) \equiv L(q_{A,0}) \equiv L_A(G)$$

Per l'assioma si ha:

$$L(M_S, q_{S,0}) \equiv L(q_{S,0}) \equiv L(\mathcal{M}) \equiv L(G)$$

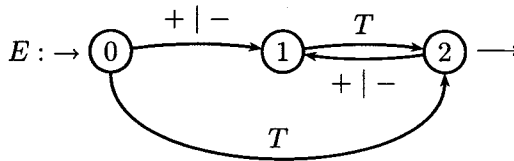
Esempio 4.19. Rete di macchine per le espressioni aritmetiche.

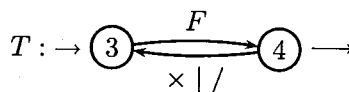
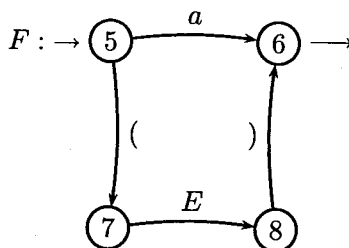
La grammatica contiene tre nonterminali (E è l'assioma), dunque tre regole:

$$E \rightarrow [+ \mid -]T \quad ((+ \mid -)T)^* \quad T \rightarrow F((\times \mid /)F)^* \quad F \rightarrow (a \mid '(E)')$$

La prossima figura mostra le macchine della rete \mathcal{M} .

M_E :



M_T : M_F :

Risulta:

$$R(M_E, 0) = R(M_E) = [+ | -]T ((+ | -)T)^*$$

$$R(M_T, 4) = R(4) = ((\times | /)F)^+$$

$$L(M_E, 0) = L(0) = L_E(G) = L(G) = L(\mathcal{M}) = \{a, a + a, a \times (a - a), \dots\}$$

$$L(M_F, 5) = L(5) = L_F(G) = \{a, (a), (a + a), \dots\}$$

$$L(M_F, 8) = L(8) = \{ '\}'$$

Diagrammi sintattici

Si apre un intermezzo sull'impiego delle reti di macchine nella documentazione dei linguaggi tecnici. Come in altri settori della tecnica, anche qui certi schemi grafici sono impiegati per documentare il progetto, in sostituzione o accanto alle definizioni testuali (espressioni regolari o regole grammaticali).

I *diagrammi sintattici* delle grammatiche libere, anche estese con espressioni regolari, sono una documentazione grafica molto comune nei manuali dei linguaggi tecnici; nonché, sotto il nome di reti di transizioni, nella linguistica computazionale, per il trattamento automatico del linguaggio naturale.

La diffusione dei diagrammi sintattici deriva dalla loro facile leggibilità e dal fatto che forniscono allo stesso tempo la definizione strutturale del linguaggio e gli schemi di flusso delle procedure del parsificatore, come si vedrà nel seguito.

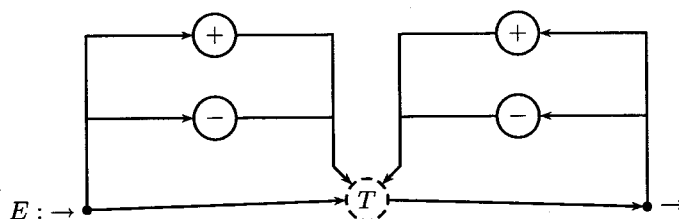
I diagrammi sintattici differiscono dalle reti di macchine soltanto nelle convenzioni grafiche. Primariamente il grafo dell'automa finito è trasformato nel grafo duale, in cui archi e nodi sono scambiati. I nodi del grafo sono i simboli, terminali e non, della parte destra della regola grammaticale $A \rightarrow \alpha$. Gli archi non sono etichettati; ossia gli stati non sono contraddistinti dal nome, ma solo dalla posizione nel grafo. Il diagramma ha convenzionalmente un solo punto di entrata, evidenziato dalla freccia etichettata con il nome del nonterminale A , e un solo punto di uscita, corrispondente allo stato finale.

Un esempio è sufficiente per presentare i diagrammi sintattici.

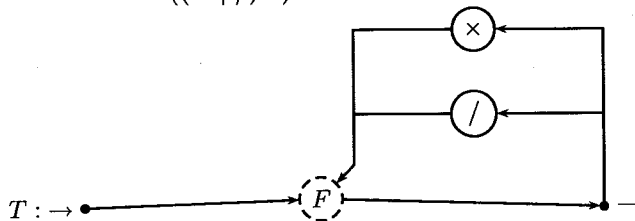
Esempio 4.20. Diagrammi sintattici di espressioni aritmetiche (es. 4.19).

Le macchine della rete di p. 171 si trasformano nei grafi tipici dei diagrammi sintattici: i nodi del grafo sono i simboli della grammatica, tratteggiati i nonterminali, a tratto continuo i terminali;¹⁰ gli stati dell'automa non sono disegnati come nodi né portano etichetta.

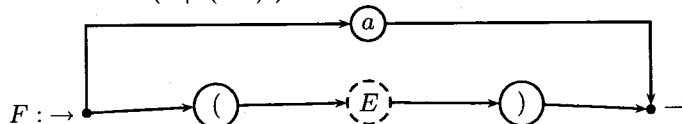
$$E \rightarrow [+ | -]T ((+ | -)T)^*$$



$$T \rightarrow F((\times | /)F)^*$$



$$F \rightarrow (a | ('E'))$$



Attraversando un grafo dall'entrata all'uscita, si ottengono le possibili parti destre della regola sintattica, ad es. nel primo diagramma:

¹⁰ Anche altri stili grafici possono essere impiegati per distinguere visivamente le due classi di simboli.

$$T, \quad +T, \quad T + T, \quad \dots$$

Data una grammatica libera estesa, per ottenere i diagrammi sintattici che la rappresentano, basta costruire, con uno dei metodi noti, i riconoscitori finiti delle e.r. delle parti destre delle regole, e aggiustarli alla convenzione grafica adottata dai diagrammi sintattici.

4.3.3 Procedura ricorsiva indeterministica di riconoscimento

Si mostra che è immediato interpretare una rete di macchine ricorsive come lo schema di flusso dell'algoritmo indeterministico di riconoscimento delle frasi del linguaggio.

Si può pensare che ciascuna macchina sia un sottoprogramma e che il salto da una macchina all'altra sia l'invocazione d'un sottoprogramma, al cui termine avviene il ritorno alla macchina chiamante. Poiché le macchine della rete possono invocarsi ricorsivamente, questo tipo di algoritmo è detto a *discesa ricorsiva*.

È essenziale dire che, affinché la ricorsione termini, la grammatica non deve essere ricorsiva a sinistra. In caso contrario, se vi fosse la regola s-ricorsiva $A \rightarrow A\gamma$, il sottoprogramma associato alla macchina M_A invocherebbe se stesso senza fine.

L'algoritmo realizza un automa a pila, in generale non deterministico, con un solo stato interno, che accetta a pila vuota.

Si descrive dapprima a parole il suo funzionamento. Alla partenza, la macchina attiva è quella dell'assioma e il carattere corrente il primo carattere del testo. Se la stringa è valida, esisterà un calcolo che, dopo aver eventualmente invocato altre macchine, condurrà tale macchina nello stato finale.

L'automa a ogni passo può proseguire il calcolo all'interno della *macchina attiva*, proprio come farebbe un riconoscitore finito, leggendo il prossimo carattere terminale, e cambiando stato. Il cambio di stato è registrato scrivendo nella pila il nuovo stato al posto del vecchio, sicché la cima contiene sempre lo stato corrente della macchina attiva.

La macchina, se dallo stato corrente esce un arco non terminale A , può saltare spontaneamente allo stato iniziale della macchina chiamata M_A (la stessa macchina attiva se la regola è ricorsiva), depositando sulla pila tale stato sopra quello della macchina sospesa.

Quando un calcolo raggiunge lo stato finale d'una macchina, si dice che essa è terminata. L'automa deve ritornare alla macchina sospesa, anche detta chiamante. L'operazione di ritorno comprende due passi: prima cancella lo stato della macchina attiva, e così ripristina lo stato dell'ultima macchina sospesa, che torna a essere attiva; poi esegue la mossa etichettata con il nome della macchina terminata.

Algoritmo 4.21. Riconoscitore a discesa ricorsiva non deterministico.

L'automa a pila è così definito:

begin

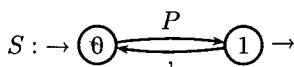
1. la stringa sorgente è x e cc il carattere terminale corrente;
2. i simboli della pila sono l'unione (disgiunta) $Q = Q_A \cup Q_B \cup \dots$ degli stati di tutte le macchine;
3. l'automa a pila ha un solo stato interno che si lascia sottinteso;
4. inizialmente la pila contiene lo stato iniziale $q_{S,0}$;
5. transizioni; sia $s \in Q_A$ il simbolo in cima, cioè lo stato della macchina attiva A ;
- a) (mossa di scansione)
 - se la mossa $s \xrightarrow{cc} s'$ è definita (nella macchina attiva M_A), consuma il carattere corrente e scrivi nella pila s' al posto di s ;
- b) (mossa di chiamata)
 - se la mossa $s \xrightarrow{A} s'$ è definita per un nonterminale (generico) A , effettua una mossa spontanea che depone sulla pila lo stato iniziale $q_{A,0}$ della macchina M_A , che così diviene quella attiva;
- c) (mossa di ritorno)
 - se s è uno stato finale d'una macchina generica M_A , cancellalo dalla pila, effettua dallo stato r emerso sulla cima la mossa $r \xrightarrow{A} s'$ e scrivi nella pila s' al posto di r ;
- d) (mossa di riconoscimento)
 - se s è uno stato finale di M_S (la macchina dell'assioma) e $cc = \neg$, accetta e termina;
- e) in ogni altro caso rifiuta la stringa e termina.

end

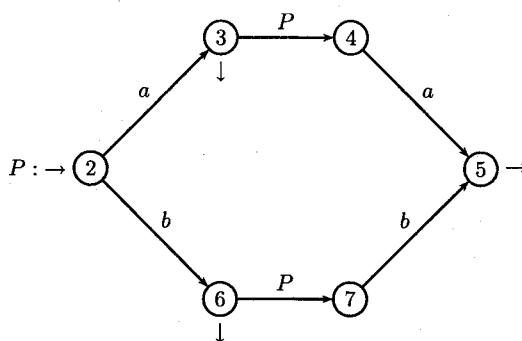
Esempio 4.22. Lista di palindromi dispari.

La rete ha due macchine:

$$S \rightarrow P(,P)^*$$



$$P \rightarrow aPa \mid bPb \mid a \mid b$$



Partendo dallo stato 0 si ha il linguaggio $L(0) \equiv L(G)$ delle liste di palindromi di lunghezza dispari. Dallo stato 2 si ha invece il linguaggio $L(2)$ dei palindromi dispari.

L'analisi della stringa $a, b \vdash$ inizia con lo stato 0 in cima alla pila. La macchina M_S pone sulla pila 2, ossia chiama la macchina M_P , la quale, leggendo a , scrive sulla pila 3 al posto di 2. Poi M_P può terminare, oppure chiamare ricorsivamente se stessa, scelta quest'ultima che fallirebbe. Se M_P decide di terminare, toglie 3 dalla pila, e il calcolo riprende in 0. La macchina M_S impila 1 al posto di 0, poi, leggendo la virgola, impila 0 al posto di 1, e di nuovo chiama M_P , e così via. Il calcolo termina nello stato 1, leggendo il carattere terminatore.

La prossima sezione svilupperà, sotto opportune condizioni, una versione deterministica dell'algoritmo.

4.4 Analisi sintattica discendente deterministica

Molte delle grammatiche dei linguaggi tecnici sono state progettate apposta per permettere un veloce riconoscimento deterministico delle frasi. La pros-

sima strada, detta $LL(k)$ ¹¹, per costruire gli automi a pila deterministici a discesa ricorsiva, è intuitiva e consente grande flessibilità nell'implementazione dei traduttori guidati dalla sintassi.

Si vedranno due varianti implementative: l'automa a pila classico, e il programma a discesa ricorsiva, realizzato mediante la pila delle aree di attivazione dei sottoprogrammi.

Infine seguiranno alcune trasformazioni utili per adattare una grammatica a questo tipo di analisi.

4.4.1 Condizioni per la costruzione del riconoscitore $LL(1)$

Sia data una grammatica G , sotto forma di rete ricorsiva di macchine, ossia ogni regola $A \rightarrow \alpha$ è specificata dalla macchina finita deterministica M_A , che riconosce il linguaggio regolare definito dalla e.r. α .

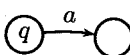
Si ricorda che un linguaggio è detto *annullabile* se contiene la stringa vuota. Riprendendo concetti simili a quelli introdotti per gli automi a stati finiti (algoritmi GMY 3.8.2 e BS 3.8.3 p. 129 e seguenti) si danno le seguenti definizioni:

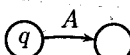
Insieme degli inizi d'uno stato q : per un generico stato q si prendono gli inizi delle stringhe riconosciute partendo da tale stato $Ini(q) = Ini(L(q))$.

Insieme dei seguiti: L'insieme dei seguiti $Seg(A)$ d'un nonterminale A contiene i caratteri terminali che possono seguire A in qualche derivazione. Inoltre il terminatore \dagger appartiene ai seguiti dell'assioma.

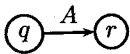
Il calcolo degli inizi e dei seguiti è presentato dai seguenti algoritmi sotto forma di clausole logiche. Siano a un terminale, A, B dei nonterminali e q, r degli stati.

Algoritmo 4.23. Insieme degli inizi del linguaggio $L(q)$.

1. $a \in Ini(q)$ se \exists arco 

2. $a \in Ini(q)$ se \exists arco 

$\wedge a \in Ini(q_{A,0})$, dove $q_{A,0}$ è lo stato iniziale della macchina M_A .

3. $a \in Ini(q)$ se \exists arco 

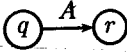
$\wedge L(q_{A,0})$ è annullabile $\wedge a \in Ini(r)$

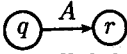
I casi 1. e 2. si spiegano da soli. Nel caso 3. poiché $\varepsilon \in L(q_{A,0})$, dallo stato q si può andare in r senza leggere alcun carattere; allora il primo carattere incontrato sarà quello che si legge a partire da r .

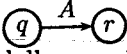
¹¹La sigla $LL(1)$ ha un significato storico. La prima *elle* indica che l'analisi esamina il testo da sinistra (left) a destra, la seconda *elle* indica che la derivazione è sinistra (leftmost) e il numero k precisa la lunghezza in caratteri della prospezione.

Algoritmo 4.24. Insieme dei seguiti d'un nonterminale A .

→ 1. $\vdash \in \text{Seg}(S)$

2. $a \in \text{Seg}(A)$ se \exists arco  $\wedge a \in \text{Ini}(r)$

3. $a \in \text{Seg}(A)$ se \exists arco  nella macchina $M_B, B \neq A$
 \wedge il linguaggio $L(r)$ è annullabile
 $\wedge a \in \text{Seg}(B)$

4. $a \in \text{Seg}(A)$ se \exists arco  \rightarrow
 $\wedge r$ è uno stato finale della macchina M_B , con $B \neq A$
 $\wedge a \in \text{Seg}(B)$

Il caso 1. dice che l'assioma, poiché deriva una frase completa, è seguito dal terminatore.

Il caso 2. aggiunge ai seguiti di A i caratteri iniziali del linguaggio riconosciuto partendo dallo stato destinazione d'una freccia etichettata A .

Nel caso 4., letta una stringa del linguaggio $L_A(G)$, l'automa si trova nello stato finale della macchina M_B , quindi ha terminato la lettura d'una stringa del linguaggio $L_B(G)$. Perciò il carattere seguente appartiene ai seguiti di B e anche a quelli di A .

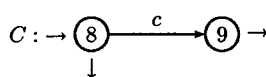
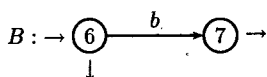
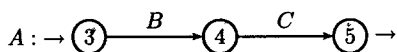
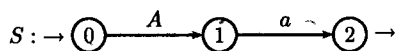
Il caso 3. generalizza il caso 4., nel senso che lo stato r non è direttamente finale, ma è collegato a uno stato finale della macchina M_B , tramite un cammino che può leggere la stringa vuota.

Per eseguire il calcolo degli inizi e dei seguiti, si inizializzano tutti gli insiemi degli inizi e dei seguiti a vuoto. Poi si applicano le clausole degli algoritmi ripetutamente e in qualsiasi ordine, finché gli insiemi degli inizi e dei seguiti non crescono più (raggiungimento d'un punto fisso).

Esempio 4.25. Inizi e seguiti.

Queste situazioni sono illustrate per la grammatica.

$$S \rightarrow Aa \quad A \rightarrow BC \quad B \rightarrow b \mid \varepsilon \quad C \rightarrow c \mid \varepsilon$$



I linguaggi generati da A, B, C sono annullabili. Per l'insieme degli inizi si ha:

$$\begin{aligned} Ini(0) &= Ini(3) \cup Ini(1) \\ &= Ini(6) \cup Ini(4) \cup Ini(1) \\ &= Ini(6) \cup Ini(8) \cup Ini(5) \cup Ini(1) \\ &= \{b\} \cup \{c\} \cup \emptyset \cup \{a\} \end{aligned}$$

Gli insiemi dei seguiti sono:

$$Seg(S) = \{-\} \quad \text{caso } (1)$$

$$Seg(A) = Ini(1) \quad \text{caso } (2)$$

$$= \{a\}$$

$$Seg(B) = Ini(4) \cup Seg(A)$$

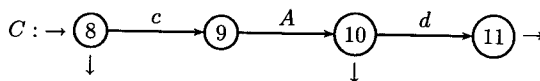
$$= Ini(8) \cup Seg(A) \quad \text{caso } (3)$$

$$= \{c\} \cup \{a\}$$

$$Seg(C) = Seg(A)$$

$$= \{a\} \quad \text{caso } (4)$$

Se nella rete vi sono più comparse d'un nonterminale, i seguiti di ogni comparsa vanno uniti. Per illustrarlo, si modifichi la macchina M_C della rete:



Si ricalcolano i seguiti di A :

$$\begin{aligned}
 \text{Seg}(A) &= \text{Ini}(1) \cup \text{Ini}(10) \cup \text{Seg}(C) \\
 &= \{a\} \cup \{d\} \cup \text{Seg}(A) \\
 &= \{a\} \cup \{d\}
 \end{aligned}$$

Si noti che all'ultimo passaggio il termine $\text{Seg}(A)$, identico alla parte sinistra, è eliminato.

Insieme guida

L'insieme guida prossimamente definito serve da selezionatore per la scelta della mossa negli stati dove si presenta un bivio. L'idea è di calcolare per ogni ramo d'un bivio l'insieme dei primi caratteri che potranno essere incontrati prendendo quella strada. Se tali insiemi guida sono disgiunti per gli archi del bivio, la scelta della mossa è univoca: quella il cui insieme guida contiene il carattere corrente.

Definizione 4.26. Insieme guida (o di prospezione).

L'insieme guida $\text{Gui}(q \rightarrow \dots) \subseteq \Sigma \cup \{\rightarrow\}$ è definito per ogni freccia (di mossa o di accettazione) della rete. A seconda del tipo di freccia e della sua etichetta, la definizione dell'insieme si divide nei seguenti casi.

1. Per un arco $q \xrightarrow{b} r$, con etichetta terminale $b \in \Sigma$, si ha:

$$\text{Gui}(q \xrightarrow{b} r) = \{b\}$$

2. Per un arco $q \xrightarrow{A} r$ della macchina M_B , etichettato dal nonterminale A , si ha:

a) $\text{Gui}(q \xrightarrow{A} r) = \text{Ini}(L(q_{A,0}) L(r))$, se il concatenamento $L(q_{A,0}) L(r)$ non è annullabile;

b) $\text{Gui}(q \xrightarrow{A} r) = \text{Ini}(L(q_{A,0}) L(r)) \cup \text{Seg}(B)$, altrimenti.

3. Per la freccia $q \rightarrow$ d'uno stato finale q della macchina M_B si ha:

$$\text{Gui}(q \rightarrow) = \text{Seg}(B)$$

Il caso 1. è ovvio: il primo carattere è l'etichetta stessa dell'arco.

In 2.a il carattere che fa scegliere la mossa etichettata dal nonterminale A , è un carattere iniziale del linguaggio definito dal nonterminale; ma se esso è annullabile si prende anche un carattere iniziale del linguaggio riconosciuto a partire dallo stato r .

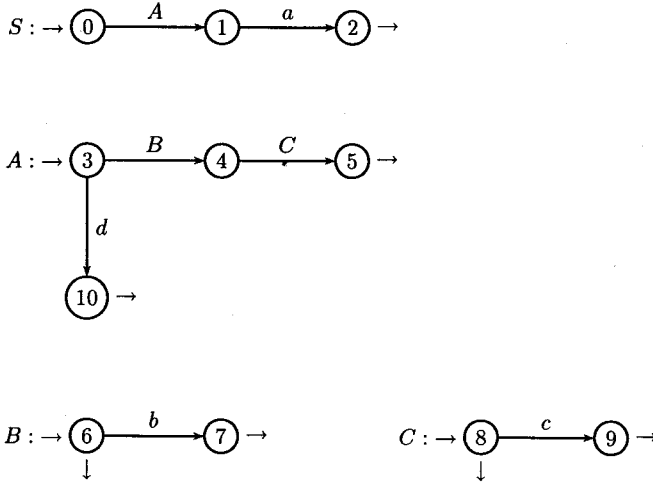
Il caso più complesso è 2.b. Poiché il concatenamento dei linguaggi $L(q_{A,0})L(r)$ è annullabile, il riconoscitore può raggiungere uno stato finale della macchina M_B senza incontrare alcun carattere. In tale caso, il primo carattere incontrato sarà uno di quelli che possono seguire il nonterminale B .

Nel caso 3. il calcolo sulla macchina M_B può terminare quando il carattere corrente sta nei seguiti di B .

Esempio 4.27. Casi dell'insieme guida.

Queste situazioni sono illustrate dalla seguente grammatica.

$$S \rightarrow Aa \quad A \rightarrow BC \mid d \quad B \rightarrow b \mid \varepsilon \quad C \rightarrow c \mid \varepsilon$$



Lo stato 3 della macchina M_A è un bivio, dove, essendo il concatenamento

$$L(6)L(8) = L(6)L(4) = \{\varepsilon, b\}\{\varepsilon, c\}$$

annullabile, risulta:

$$\begin{aligned} Gui(3 \xrightarrow{B} 4) &= Ini(L(6)L(8)) \cup Seg(A) \\ &= Ini(\{\varepsilon, b, c, bc\}) \cup \{a\} \\ &= \{b, c, a\} \end{aligned}$$

$$Gui(3 \xrightarrow{d} 10) = \{d\}$$

Nel bivio 6 si ha:

$$\begin{aligned} Gui(6 \rightarrow) &= Seg(B) \\ &= \{a, c\} \\ Gui(6 \xrightarrow{b} 7) &= \{b\} \end{aligned}$$

L'ultimo bivio da considerare è 8, in cui risulta:

$$\begin{aligned} Gui(8 \rightarrow) &= Seg(C) \\ &= Seg(A) \\ &= \{a\} \\ Gui(8 \xrightarrow{c} 9) &= \{c\} \end{aligned}$$

Poiché in ogni bivio gli insiemi guida delle frecce uscenti sono disgiunti, la grammatica soddisfa la condizione $LL(1)$.

Si completa l'esposizione con una condizione sufficiente, affinché l'automa a pila dell'Algoritmo 4.21 diventi deterministico.

Definizione 4.28. *Condizione $LL(1)$*

Una macchina M_A della rete soddisfa la condizione $LL(1)$ nello stato q se, per ogni coppia di frecce uscenti dallo stato, gli insiemi guida sono disgiunti.

Una regola soddisfa la condizione $LL(1)$ se ogni stato della macchina corrispondente la soddisfa.

Una grammatica gode della proprietà $LL(1)$ se ogni stato delle macchine della rete soddisfa la condizione $LL(1)$.

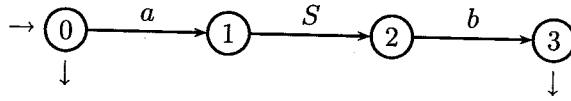
È facile osservare che, se uno stato fosse indeterministico e da esso partissero due frecce con la stessa etichetta, la condizione $LL(1)$ sarebbe violata. Questa è la ragione che motiva l'utilizzo di macchine deterministiche nell'ambito della costruzione dei parsificatori $LL(1)$. Seguono vari esempi di calcolo degli insiemi guida e di verifica della condizione.

Esempio 4.29. Grammatiche $LL(1)$ e non.

1. La grammatica

$$G_1 : \quad S \rightarrow aSb \mid \varepsilon$$

genera il linguaggio $\{a^n b^n \mid n \geq 0\}$. Disegnata la macchina deterministica corrispondente



La verifica della condizione $LL(1)$ richiede il calcolo degli insiemi guida, soltanto dove almeno due frecce escono, cioè nel solo stato 0. Gli insiemi

$$Gui(0 \xrightarrow{a} 1) = \{a\}, \quad Gui(0 \rightarrow) = Seg(S) = \{b, \vdash\}$$

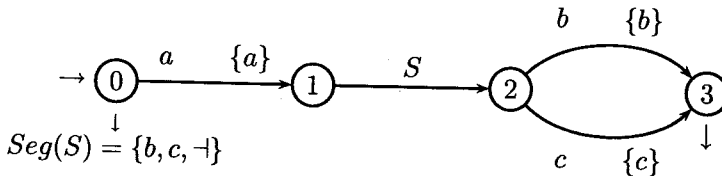
sono disgiunti e la grammatica è $LL(1)$.

Si noti che il secondo insieme contiene b perché S nello stato 2 è seguito da b ; e contiene il terminatore perché S è la macchina assioma.

2. La grammatica

$$G_2: \quad S \rightarrow aSb \mid aSc \mid \varepsilon$$

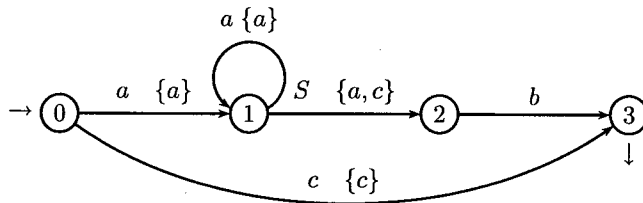
genera il linguaggio $\{a^n(b \mid c)^n \mid n \geq 0\}$. La macchina risulta $LL(1)$ poiché negli stati bivii 0 e 2 gli insiemi guida delle frecce uscenti, tra graffe nel disegno, sono disgiunti.



3. La grammatica

$$G_3: \quad S \rightarrow a^+Sb \mid c$$

genera il linguaggio $\{a^*a^ncb^n \mid n \geq 0\}$. Il grafo della macchina contiene un circuito (autoanello) corrispondente all'operatore croce.

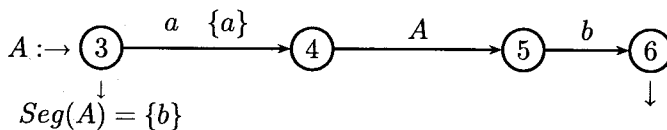
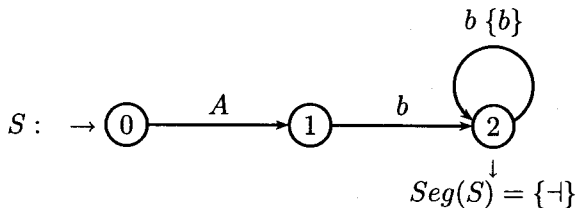


Lo stato 1 viola la condizione $LL(1)$ poiché $Ini(S) = \{a, c\}$ contiene a .
Nel bivio 0 la condizione è invece verificata.

4. La grammatica

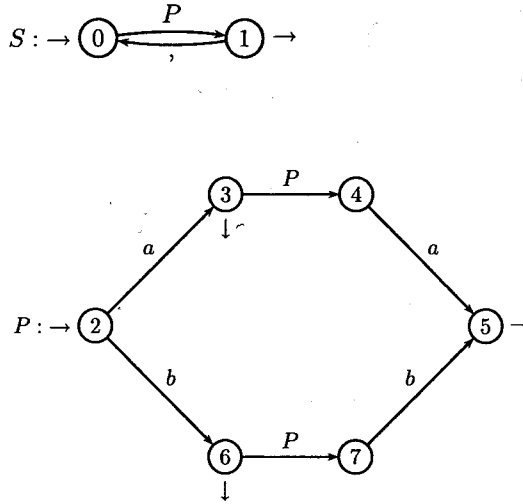
$$G_4: \quad S \rightarrow Ab^+ \quad A \rightarrow aAb \mid \varepsilon$$

genera il linguaggio $\{a^n b^n b^+ \mid n \geq 0\}$.



Le macchine soddisfano la condizione nei loro bivii, 2 e 3.

5. La grammatica delle liste di palindromi (es. 4.22 p. 175)



non è $LL(1)$ nello stato 3, dove la freccia d'uscita ha l'insieme $Gui(3 \rightarrow) = Seg(P) = \{, \} \cup \{a\} \cup \{b\}$, calcolato osservando le tre comparse di P nella rete. Ma l'insieme $Gui(3 \xrightarrow{P} 4) = Ini(L_P(G)) = Ini(2) = \{a, b\}$ è sovrapposto. Lo stesso conflitto avviene nello stato 6.

Invece il bivio 1 della prima macchina soddisfa la condizione, poiché la virgola non appartiene all'insieme $Gui(1 \rightarrow) = Seg(S) = \{, \}$.

Calcolo semplificato degli insiemi guida

Il calcolo degli insiemi guida, se la grammatica non è estesa con espressioni regolari, può essere direttamente condotto sulle regole stesse, senza la necessità di disegnare le macchine corrispondenti.

La definizione si semplifica nel seguente modo. Sia $A \rightarrow \alpha$ una regola della grammatica G di assioma S .

$$\begin{cases} Gui(A \rightarrow \alpha) = Ini(\alpha), & \text{se } \alpha \text{ non è annullabile;} \\ Gui(A \rightarrow \alpha) = Ini(\alpha) \cup Seg(A), & \text{altrimenti.} \end{cases}$$

I termini che compaiono possono essere definiti, invece che sulle macchine, direttamente sulla grammatica:

$$\begin{aligned} Ini(\alpha) &= \{a \in \Sigma \mid \alpha \xrightarrow{*} ay \wedge y \in \Sigma^*\} \\ Seg(A) &= \{a \in \Sigma \cup \{, \} \mid S \dashv \xrightarrow{*} yAaz \wedge y, z \in \Sigma^*\} \end{aligned}$$

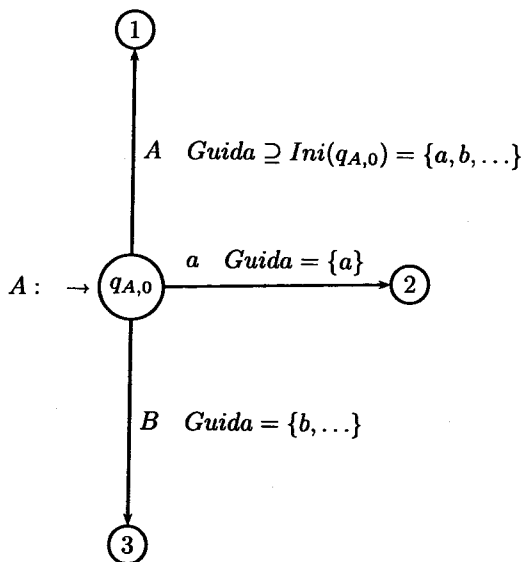
La condizione $LL(1)$ dunque impone che gli insiemi guida siano disgiunti, per ogni coppia di regole alternative $A \rightarrow \alpha, A \rightarrow \alpha'$ della grammatica.

Ricorsione a sinistra

Dalla definizione segue facilmente la seguente proprietà, che esclude dall'analisi discendente le grammatiche aventi regole ricorsive a sinistra.

Proprietà 4.30. Una regola ricorsiva a sinistra viola la condizione $LL(1)$.

Dimostrazione. Per semplicità si esamina una regola con ricorsione soltanto immediata, $A \rightarrow A \dots | a \dots | B \dots$ sotto schematizzata, ma lo stesso ragionamento vale nel caso d'una derivazione ricorsiva di lunghezza maggiore di uno.



Il calcolo dell'insieme guida del primo arco (dall'alto) porta a rientrare ricorsivamente nella macchina stessa, così ottenendo come caratteri iniziali quelli stessi che stanno negli insiemi guida del secondo e terzo arco. Di conseguenza l'insieme guida dell'arco ricorsivo a sinistra contiene l'unione degli altri insiemi guida.

Di conseguenza, le ricorsioni sinistre presenti in una grammatica dovranno essere trasformate in destre, se si vuole applicare l'analisi $LL(1)$.

Riconoscitore $LL(1)$

Il punto d'arrivo del percorso concettuale precedente è il prossimo enunciato

Proprietà 4.31. Se la grammatica gode della proprietà $LL(1)$, l'algoritmo 4.21 (p. 175) di riconoscimento delle stringhe diviene deterministico.

Si guardi la riformulazione dell'algoritmo, sotto la condizione $LL(1)$.

Algoritmo 4.32. Riconoscitore a discesa ricorsiva deterministico

L'automa a pila è così definito:

begin

1. la stringa sorgente è x e cc il carattere terminale corrente;
2. i simboli della pila sono l'unione (disgiunta) $Q = Q_A \cup Q_B \cup \dots$ degli stati di tutte le macchine;
3. l'automa a pila ha un solo stato interno, che si lascia sottinteso;
4. inizialmente la pila contiene lo stato iniziale $q_{S,0}$;
5. transizioni; sia $s \in Q_A$ il simbolo in cima, cioè lo stato della macchina attiva M_A ;
 - a) (mossa di scansione)
se la mossa $s \xrightarrow{cc} s'$ è definita (nella macchina attiva M_A), consuma il carattere corrente e scrivi nella pila s' al posto di s ;
 - b) (mossa di chiamata)
se la mossa $s \xrightarrow{A} s'$ è definita per un nonterminale A , e $cc \in Gui(s \xrightarrow{A} s')$ effettua la mossa spontanea che depone sulla pila lo stato iniziale $q_{A,0}$ della macchina M_A , che così diviene quella attiva;
 - c) (mossa di ritorno)
se s è uno stato finale d'una macchina generica M_A e $cc \in Gui(s \rightarrow)$ cancellalo, effettua dallo stato r emerso sulla cima la mossa $r \xrightarrow{A} s'$ spontanea e scrivi nella pila s' al posto di r ;
 - d) (mossa di riconoscimento)
se s è uno stato finale di M_S (macchina dell'assioma) e $cc = \neg$, accetta e termina;
 - e) in ogni altro caso rifiuta la stringa e termina.

end

Poiché gli insiemi guida delle frecce uscenti dallo stato s sono per l'ipotesi $LL(1)$ disgiunti, al passo 5. le condizioni a), b), c) e d) sono mutuamente esclusive e la scelta dell'automa è deterministica.

La complessità di calcolo è lineare nella lunghezza n della stringa sorgente. Infatti o l'automa legge e consuma un carattere sorgente, o effettua una mossa spontanea nei casi b) e c). Ma il numero di mosse spontanee, comprese tra due letture successive, è limitato superiormente da una costante, grazie al ragionamento seguente. Una mossa spontanea invoca una macchina, che a sua volta può immediatamente (ossia senza leggere un carattere) invocare un'altra macchina, e così via. Poiché la grammatica non può contenere regole (anche mediatamente) ricorsive a sinistra, la lunghezza della catena delle invocazioni è limitata superiormente dal numero dei nonterminali della grammatica. In definitiva il numero di mosse spontanee tra due letture non può superare una costante, indipendente dalla lunghezza della stringa sorgente, quindi la complessità dell'algoritmo è $O(n)$.
Si noti poi che il numero di passi diviene esattamente eguale alla lunghezza

della stringa quando la grammatica $LL(1)$ è nella forma normale di Greibach, che per questa ragione è anche detta in tempo reale. ↑

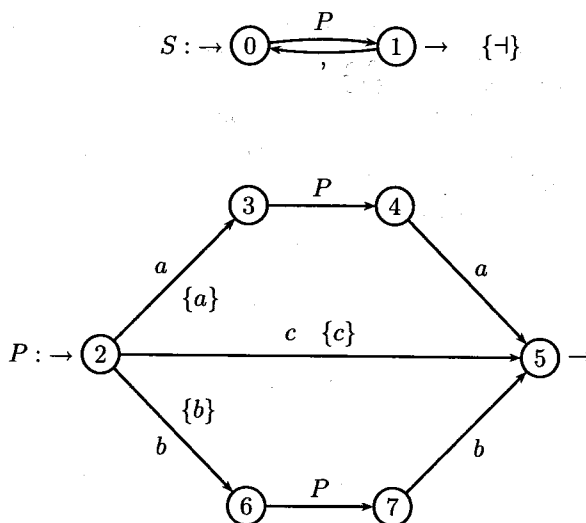
Implementazione del parsificatore a procedure ricorsive

Si illustra con un esempio la realizzazione del parsificatore come insieme di procedure corrispondenti alle macchine della grammatica.

Esempio 4.33. Procedure ricorsive per lista di palindromi con centro.
La seguente grammatica

$$S \rightarrow P(, P)^* \quad P \rightarrow aPa \mid bPb \mid c$$

e rete di macchine ha la proprietà $LL(1)$, come testimoniato dagli insiemi guida riportati nel disegno:



Il programma consiste di due procedure, in corrispondenza con i simboli non-terminali della grammatica.

In sostanza ogni procedura ha lo schema a blocchi che corrisponde alla macchina associata a quel nonterminale.

La procedura, esaminato il carattere corrente, decide, in base alla sua appartenenza a un insieme guida, quale freccia del grafo seguire.

Se la freccia ha etichetta terminale, il prossimo carattere corrente è calcolato dalla funzione Prossimo (cioè dall'analizzatore lessicale o scansore). Se la

```

procedure S
begin
  1. call P;
  2. if cc='-' then accetta e termina;
  3. else if cc=',' then cc := Prossimo; go to 1;
  4. else Errore;
end

```

```

procedure P
begin
  1. if cc='a' then
    begin
      cc:=Prossimo; call P; if cc='a' then
        cc:=Prossimo else Errore
      end
    end
  2. else if cc='b' then
    begin
      cc:=Prossimo; call P; if cc='b' then
        cc:=Prossimo else Errore
      end
    end
  3. else if cc='c' then
    begin
      cc:=Prossimo
    end
  4. else Errore
end

```

freccia è nonterminale, la corrispondente procedura è invocata. Infine se il carattere non sta in nessun insieme guida, si ha un errore, e la stringa sorgente è rifiutata.

Inizialmente si lancia la procedura dell'assioma S , sul primo carattere della stringa.

Vari miglioramenti di stile e di efficienza si possono immaginare nella programmazione delle procedure. Ad es. nella prima regola l'iterazione mediante la stella, potrebbe essere codificata direttamente in un ciclo iterativo *while ... do*.

Trattamento degli errori

In caso di errore, questo semplice parsificatore termina subito, senza fornire spiegazioni, mentre nella pratica è di solito richiesto un messaggio diagnostico. Non è difficile generare automaticamente una semplice diagnostica, confrontando i caratteri attesi in un certo stato (quelli appartenenti agli insiemi guida delle frecce uscenti) con il carattere corrente, e scrivendo: "i caratteri attesi sono ..., invece di ...".

Inoltre, al fine di anticipare la scoperta d'un errore, il parsificatore può eseguire il test dell'insieme guida non soltanto nei bivii, ma anche in quegli stati

da cui esce una sola freccia, se essa ha etichetta nonterminale. Se infatti il carattere corrente non sta nell'insieme guida, la stringa è errata. Si noti che l'errore sarebbe comunque rilevato più avanti.

Infine un buon analizzatore deve essere capace di proseguire il calcolo dopo il primo errore, al fine di esaminare l'intera stringa sorgente con una sola compilazione, anche se essa contiene degli errori.¹²

4.4.2 Come ottenere grammatiche $LL(1)$

Posti di fronte a una grammatica, per evitare l'inutile calcolo degli insiemi guida, conviene verificare che essa non sia ambigua, e che non presenti ricor-sioni sinistre.

In caso contrario, occorre rimuovere l'ambiguità (p. 49), e poi trasformare le ricor-sioni sinistre in destre (p. 64).

Dopo tali trasformazioni si calcolano gli insiemi guida, e se la grammatica non è $LL(1)$, si analizzano meglio le cause della violazione. A tale scopo, è utile un modo più sintetico di definire la condizione $LL(1)$. Per maggior generalità, si dà la definizione con valore parametrico della lunghezza di prospezione (anticipando p. 194).

Proprietà 4.34. Un nonterminale A della grammatica viola la condizione $LL(k)$, $k \geq 1$, se esistono due derivazioni

$$S \xRightarrow{*} uAv \Rightarrow u\alpha v \xRightarrow{*} uzv \quad S \xRightarrow{*} uAv \Rightarrow u\alpha'v' \xRightarrow{*} uz'v'$$

dove A è nonterminale, α, α' sono stringhe di terminali o nonterminali, e le stringhe u, v, z, z', v' sono terminali anche vuote, con la seguente condizione: detto $m = \min(k, |zv|, |z'v'|)$ il minimo tra k e le lunghezze delle stringhe zv e $z'v'$, i prefissi di lunghezza m di zv e di $z'v'$ coincidono.

Nell'enunciato, la precisazione relativa alla lunghezza minima ha soltanto lo scopo di impedire che il prefisso sia più lungo della stringa da cui è estratto. Si prenda $k = 1$. Per convincersi che la formulazione equivale a quella precedente 4.28, si osservi che:

1. se esistessero due tali derivazioni, l'esame dei prossimi $k = 1$ caratteri, ossia degli insiemi guida delle regole non basterebbe per la scelta tra le derivazioni $A \rightarrow \alpha$ e $A \rightarrow \alpha'$;
2. se gli insiemi guida, nello stato della macchina M_A in cui si presenta il bivio tra i rami α e α' , fossero disgiunti, allora i prefissi considerati nell'enunciato, non coinciderebbero.

¹²Per i metodi di trattamento degli errori si può vedere ad es. [23].

Fattorizzazione sinistra

Il prossimo esempio mostra una violazione della condizione e il modo di correggerla.

Esempio 4.35. Fattorizzazione sinistra.

La grammatica

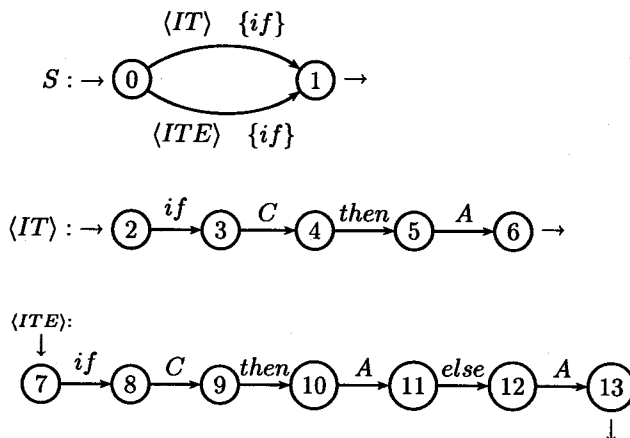
$$\begin{aligned}
 S &\rightarrow \langle IT \rangle \mid \langle ITE \rangle \\
 \langle IT \rangle &\rightarrow \text{if } C \text{ then } A & \langle ITE \rangle &\rightarrow \text{if } C \text{ then } A \text{ else } A \\
 C &\rightarrow \dots & A &\rightarrow \dots
 \end{aligned}$$

schematizza le frasi condizionali con e senza il ramo *else*; i nonterminali C e A , lasciati incompleti, stanno per condizione booleana e istruzione d'assegnamento.

Per applicare la condizione precedente, si esaminino le derivazioni:

$$\begin{aligned}
 S &\Rightarrow \underbrace{\langle IT \rangle}_A \Rightarrow \underbrace{\text{if } C \text{ then } A}_\alpha \stackrel{+}{\Rightarrow} \underbrace{\text{if } x \geq 3 \text{ then } x = 5}_z \\
 S &\Rightarrow \underbrace{\langle ITE \rangle}_A \Rightarrow \underbrace{\text{if } C \text{ then } A \text{ else } A}_{\alpha'} \stackrel{+}{\Rightarrow} \underbrace{\text{if } x \geq 3 \text{ then } x = 5 \text{ else } x = 7}_{z'}
 \end{aligned}$$

Poiché i prefissi di lunghezza 1 (ossia gli inizi) delle stringhe z e z' sono eguali a *if*, il nonterminale S viola la condizione $LL(1)$. Alla stessa conclusione si giunge esaminando gli insiemi guida della prima macchina della rete:



Evidentemente gli insiemi guida degli archi uscenti dallo stato 0 coincidono. Anche aumentando la lunghezza, i prefissi di z e di z' restano eguali:

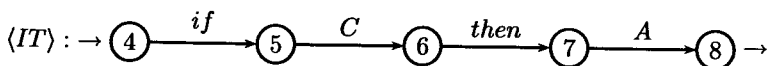
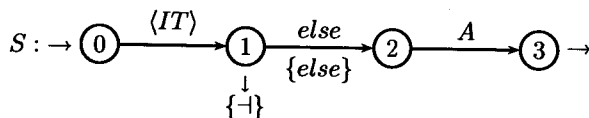
$$\text{if } x, \quad \text{if } x \geq, \quad \text{if } x \geq 3, \quad \text{ecc.}$$

Per differenziare i prefissi dell'esempio, si deve prendere il valore $k = 11$. Ma è evidente che, per ogni valore di k fissato, si può scegliere una stringa (condizione booleana) derivante da C e una stringa (assegnamento) derivante da A , tali da rendere eguali i prefissi di lunghezza k . Ciò significa che con questa grammatica, non si può costruire un parsificatore deterministico discendente, per quanto grande si prenda la lunghezza di prospezione.

La causa del problema è che il condizionale semplice è un prefisso del condizionale doppio. Per rimuoverla, basta ritardare la scelta tra i due costrutti condizionali, fino al momento in cui la presenza o l'assenza di *else* palesa quale sia il costrutto.

Ecco la grammatica dopo la trasformazione:

$$S \rightarrow \langle IT \rangle (\varepsilon \mid \text{else } A) \qquad \langle IT \rangle \rightarrow \text{if } C \text{ then } A$$



Ora l'unico punto di scelta è nello stato 1, dove gli insiemi guida sono disgiunti.

La modifica illustrata è nota come *fattorizzazione sinistra* e spesso riesce a ottenere una grammatica $LL(1)$ equivalente.

La trasformazione consiste nel mettere in evidenza il più lungo prefisso comune ai linguaggi definiti dagli archi uscenti dallo stesso stato. In sostanza si modifica il grafo (o la grammatica) in modo da raccogliere i prefissi comuni in un cammino comune, che sarà posto a monte dei cammini che differenziano i suffissi dei due casi. L'effetto è di spostare più a valle il bivio.

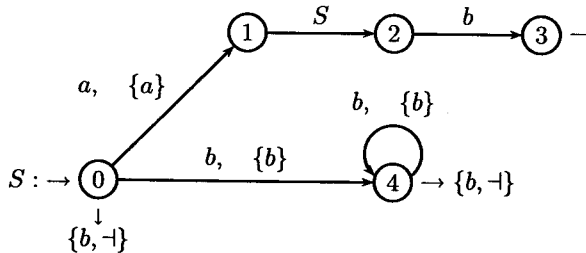
Si noti la somiglianza concettuale con il metodo di determinazione degli automi finiti.

Altre trasformazioni

Esistono altri tipi di infrazioni, alle quali si rimedia con trasformazioni di tipo diverso, come ad es. la seguente.

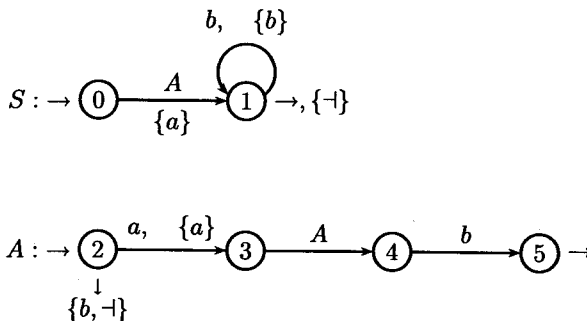
Esempio 4.36. Estrazione d'un costrutto da una ricorsione.
Il linguaggio $\{a^n b^* b^n \mid n \geq 0\}$ è definito dalla grammatica:

$$S \rightarrow aSb \mid b^*$$



Gli insiemi guida sono sovrapposti negli stati 0 e 4. Spostando il termine b^* all'esterno della ricorsione, si ottiene la grammatica equivalente $LL(1)$:

$$S \rightarrow Ab^* \quad A \rightarrow aAb \mid \varepsilon$$



Infine si ricorda che, se la grammatica data genera un linguaggio regolare, è facile ottenere una grammatica equivalente $LL(1)$. Basta infatti costruire l'automa finito deterministico che riconosce il linguaggio: esso presenta soltanto etichette terminali e soddisfa per definizione la condizione $LL(1)$.

4.4.3 Allungamento della prospezione

Per ottenere un parsificatore deterministico, quando la condizione $LL(1)$ cade, invece di modificare la grammatica, conviene spesso adottare una tattica alternativa, che consiste nell'esaminare non solo il carattere corrente, ma anche quello o quelli successivi. L'algoritmo, in altre parole, compie una prospezione di lunghezza $k > 1$ sul testo. Se, così facendo, la scelta della mossa da compiere per uscire dallo stato diventa unica, si dice che esso soddisfa la condizione $LL(k)$.

Una grammatica si dice $LL(k)$ se esiste un intero k tale che, per ogni macchina della rete e per ogni stato, la scelta tra le frecce uscenti dallo stato possa essere fatta utilizzando una prospezione di lunghezza $\leq k$.

È sufficiente un esempio per illustrare il calcolo degli insiemi guida di lunghezza $k = 2$.

Esempio 4.37. Conflitto tra etichette e variabili.

Si definisce una parte d'un linguaggio di programmazione. Esso è una lista di istruzioni (assegnamenti, frasi *for*, frasi *if*, ecc.), con o senza etichetta. Le etichette e i nomi di variabili sono degli *identificatori*.

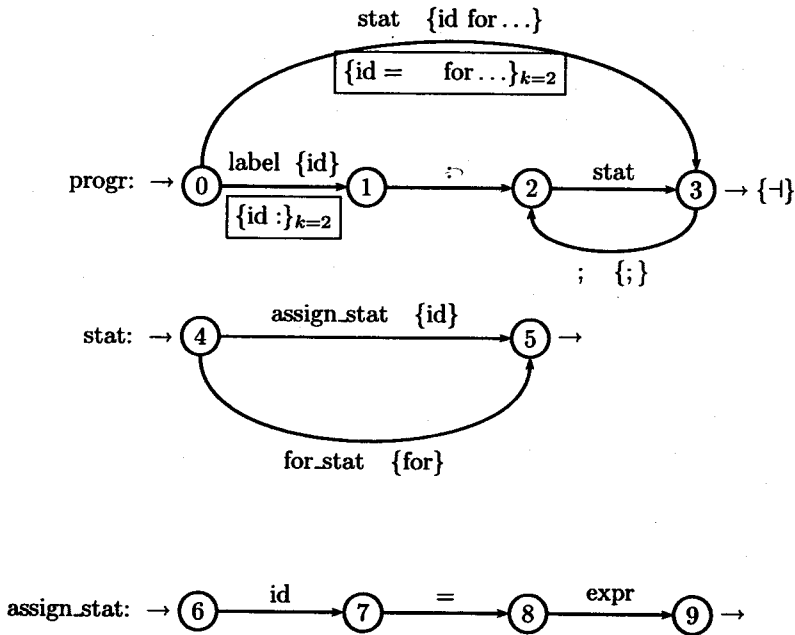
La grammatica estesa è:

```

progr → [label :]stat(; stat)*
      stat → assign_stat | for_stat | ...
assign_stat → id = expr
for_stat → ...
label → id
expr → ...

```

Le macchine della rete sono mostrate con gli insiemi guida di lunghezza 1 e di lunghezza 2 (inquadrati):



Lo stato 0 non è $LL(1)$ perché entrambi gli insiemi guida (tra graffe non inquadrate) contengono l'identificatore. Ma si osserva che, se l'identificatore è un'etichetta, è sempre seguito da due punti; mentre, nel caso dell'arco stat, l'identificatore è la parte sinistra d'un assegnamento, seguito dunque dal segno di eguale.¹³ Poiché, la prospezione del secondo carattere determina la scelta della mossa nello stato 0, esso soddisfa la condizione $LL(2)$. Le due frecce uscenti dallo stato 0 sono contraddistinte dagli insiemi guida di lunghezza due, inquadrate in figura. Poiché gli insiemi sono disgiunti, lo stato 0 soddisfa la condizione $LL(2)$.

Si noti che nello stato 3 è sufficiente la lunghezza $k = 1$, e sarebbe sciocco uniformare la prospezione al massimo dei valori necessari nella grammatica.

Gli elementi d'un insieme guida $LL(k)$ sono formalmente delle stringhe di lunghezza k .¹⁴ In pratica i parsificatori che operano con il metodo $LL(k)$ usano lunghezze diverse di prospezione, con un criterio di economia: in ogni stato si usa la minima lunghezza $m \leq k$, necessaria per risolvere la scelta tra

¹³Nel caso dell'istruzione *for* o *if* si suppone per semplicità che i lessemi *for* o *if* siano parole chiave riservate, vietate come identificatori di variabili.

¹⁴Possono avere lunghezza inferiore quando terminano con il terminatore \rightarrow .

le frecce uscenti dallo stato; se il m -esimo carattere di prospezione è sufficiente a risolvere l'incertezza, è inutile esaminare il successivo.

Sono anche stati sviluppati parsificatori¹⁵ che, oltre a eseguire una prospezione di lunghezza variabile, negli stati dove una prospezione di lunghezza finita non basta, fanno uso d'un a condizione semantica per effettuare la scelta: l'idea sarà presentata nel prossimo capitolo.

In conclusione, il vantaggio di usare una prospezione più lunga d'uno è di evitare la modifica delle regole grammaticali, al contrario delle trasformazioni, come la fattorizzazione sinistra, che producono sgradite deformazioni della grammatica di riferimento del linguaggio.

Limiti della famiglia $LL(k)$

La famiglia $LL(k)$ contiene tutti e soli i linguaggi che possono essere definiti da una grammatica $LL(k)$ per un valore finito di $k \geq 1$.

La prima limitazione è che non tutti i linguaggi deterministici sono generabili da grammatiche $LL(k)$.

Proprietà 4.38. La famiglia dei linguaggi $LL(k)$ è strettamente contenuta nella famiglia DET dei linguaggi deterministici.

Un caso semplice è il seguente.

Esempio 4.39. Linguaggio deterministico ma non $LL(k)$.

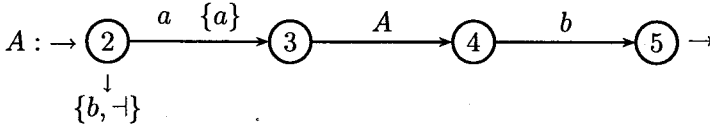
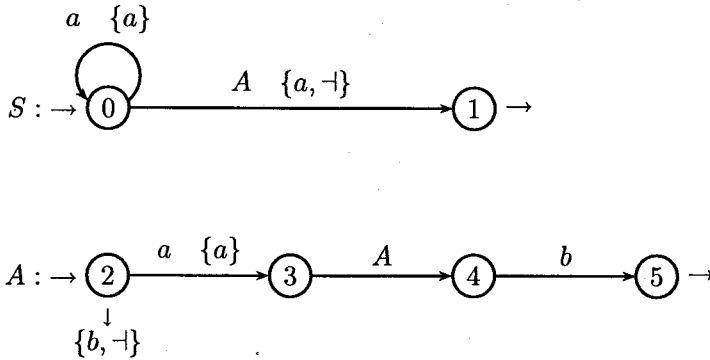
È facile costruire l'automa a pila deterministico del linguaggio $L_1 = \{a^* a^n b^n \mid n \geq 0\}$. Si impila un simbolo A per ogni lettera a letta; al primo b letto si cambia stato, e si disimpila una A ; poi, per ogni successiva b , si disimpila una A ; si riconosce se al termine della stringa la pila contiene zero o più A .

Una grammatica del linguaggio è G_1 :

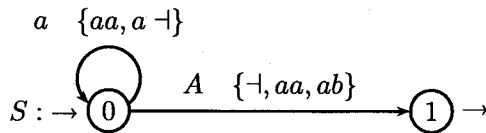
$$S \rightarrow a^* A \qquad A \rightarrow aAb \mid \varepsilon$$

Le macchine della rete, con gli insiemi guida, sono:

¹⁵Come ANTLR [40].



Gli insiemi guida per $k = 1$ nello stato 0 sono sovrapposti. Ma anche con $k = 2$ gli insiemi guida, sotto mostrati, contengono una stringa comune, aa :



In generale, per quanto grande si prenda k , si trova che entrambi gli archi uscenti dallo stato 0 sono compatibili con la prospezione a^k . Dunque G_1 non è una grammatica $LL(k)$, per nessun valore finito di k .

Sorge la domanda se, per lo stesso linguaggio, esista una grammatica che gode della proprietà $LL(k)$; la risposta è negativa.¹⁶

Questo esempio pur semplice evidenzia una limitazione della famiglia dei linguaggi $LL(k)$. Il prossimo mostra che talvolta la decisione di usare il metodo $LL(k)$ impone al progettista del linguaggio alcune cautele nella scelta dei simboli terminali, al fine di evitare sovrapposizioni negli insiemi guida.

Esempio 4.40. Relazioni e espressioni.

La grammatica G :

$$S \rightarrow R \mid (S) \quad R \rightarrow E = E \quad E \rightarrow a \mid (E + E)$$

definisce delle relazioni di eguaglianza tra espressioni aritmetiche additive, quali

$$\begin{array}{ccc} \underbrace{S} & \underbrace{S} & \underbrace{S} \\ \underbrace{S} & \underbrace{R} & \underbrace{R} \\ \underbrace{R} & \underbrace{E} & \underbrace{E} \\ \underbrace{(a = a)} & \underbrace{((a + a) + a) = a} & \end{array}$$

¹⁶La dimostrazione usa il lemma di pompaggio dei linguaggi $LL(k)$ [6].

4.5 Analisi sintattica ascendente deterministica

Il metodo $LL(1)$ non è applicabile se da uno stato d'una macchina, escono due frecce con insiemi guida sovrapposti. Poiché per ipotesi ogni macchina è deterministica, almeno una delle uscite è un arco di etichetta nonterminale \bar{A} o la freccia d'uno stato finale. Nel primo caso, la caduta della condizione $LL(1)$ significa che in quello stato, se il prossimo carattere sta in entrambi gli insiemi guida, non si sa se invocare la macchina A o compiere l'altra mossa. Una tattica efficace è quella di rinviare la decisione, se gli elementi disponibili non permettono di prenderla, e di proseguire il calcolo, tenendo aperte entrambe le strade, fino al momento in cui emergeranno nuove evidenze sufficienti a decidere. Tuttavia nell'intervallo, intercorrente tra la comparsa dell'incertezza e la sua risoluzione, il calcolo deve preservare le informazioni intermedie, che saranno necessarie per eseguire la decisione nel secondo momento.

Questa è l'idea alla base degli algoritmi di analisi sintattica ascendente, storicamente detti $LR(k)$ ¹⁷. Essi costruiscono un automa a pila dotato di stati interni, che applica una prospezione di lunghezza k . Diversamente dal caso $LL(k)$, ora ha senso ridurre a zero la lunghezza della prospezione, poiché nei casi più semplici la scelta può essere determinata dal solo stato in cui si trova l'automa. Per gradualità, l'esposizione inizia da questo caso, considerando prima le grammatiche BNF non estese.

4.5.1 Analisi $LR(0)$

Conviene introdurre l'idea partendo proprio da un esempio di grammatica non accettabile con il metodo $LL(1)$.

Esempio 4.41. Esempio introduttivo $LR(0)$.

Una lista non vuota di costrutti $a^n b^n, n \geq 1$ è definita dalla grammatica:

$$S \rightarrow SA \mid A \qquad A \rightarrow aAb \mid ab$$

rappresentata dalla rete:

¹⁷Nella sigla dell'inventore Donald Knuth, la *l* indica in inglese che la scansione del testo è da sinistra a destra, la *r* indica che la derivazione calcolata è destra, e il parametro è la lunghezza di prospezione in caratteri.