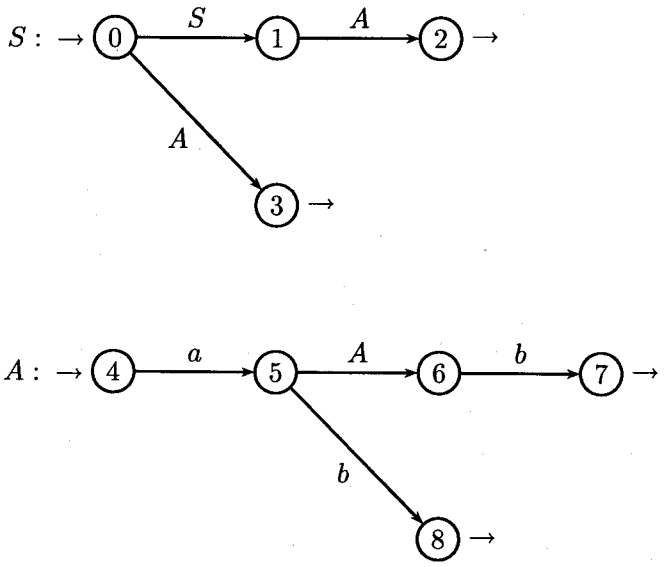


Si osservi che ogni macchina è stata disegnata con l'avvertenza di tenere distinti gli stati finali associati alle diverse alternative grammaticali.
Il metodo $LL(k)$ fallisce nello stato 0, poiché tanto S quanto A iniziano con a, essendo ricorsiva a sinistra la grammatica.
Si descrive il funzionamento dell'automa a pila deterministico costruito con il metodo $LR(0)$.
Inizialmente l'automa si trova nello stato di incertezza o *macrostato* $I_0 = \{0, 4\}$, poiché nello stato iniziale 0 dell'assioma è anche possibile invocare la macchina M_A (per brevità scritta semplicemente A), avente stato iniziale 4. Si può quindi pensare che vi siano due automi a pila deterministici contemporaneamente attivi, l'automa S e l'automa A, che evolveranno simultaneamente. La situazione ricorda il prodotto cartesiano di due macchine finite, con la differenza che ora gli automi devono anche aggiornare la loro pila. Ciò che caratterizza il metodo $LR(k)$ è che vi è *una sola* pila per tutti gli automi contemporaneamente attivi.
Il simbolo iniziale della pila è I_0. Data ad es. la stringa ab, leggendo e consumando la prima a, il parsificatore dal macrostato $I_0 = \{0, 4\}$ va, con una mossa detta di *spostamento*, nel macrostato successivo I_2 così calcolato. Si prende l'unione degli stati prossimi di 0 e di 4

$$\delta(0, a) \cup \delta(4, a) = \emptyset \cup \{5\} = \{5\}$$

Poiché da 5 si può invocare A, si aggiunge lo stato iniziale 4, completando così il macrostato $I_2 = \{4, 5\}$.
La mossa di spostamento si conclude impilando I_2 sopra I_0.
Esaminando il prossimo carattere b, si trova che, tra gli stati 4 e 5 di I_2, soltanto 5 permette la mossa che porta nello stato 8; l'automa impila il macrostato $I_3 = \{8\}$.
Essendo lo stato 8 finale della macchina A è giunto il momento di scegliere la regola grammaticale da applicare, $A \to ab$, riducendo la stringa ab nel nonterminale A.
Tale operazione, detta di *riduzione*, cancella dalla pila $I_2 I_3$, ossia un numero di simboli pari alla lunghezza $|ab| = 2$ della parte destra della regola riconosciuta nello stato 8.
La pila contiene ora $I_0 = \{0, 4\}$. Poiché il simbolo A, parte sinistra della regola riconosciuta, sta su un arco uscente dallo stato 0 (componente di I_0), l'automa esegue uno spostamento e registra sulla pila il macrostato di arrivo della transizione $\delta(0, A) = 3$, denotato $I_4 = \{3\}$. Essendo 3 uno stato finale della macchina assioma, la stringa, di cui è stata completata la lettura, è accettata.

4.5.2 Grammatiche $LR(0)$

I concetti precedenti si formalizzano attraverso una serie di definizioni che condurranno alla famiglia delle grammatiche $LR(0)$.
È data la grammatica $G = (V, \Sigma, P, S)$, con regole che per semplicità sono per ora del tipo non esteso. Ogni nonterminale $A \in V$ è la parte sinistra d'una o più regole alternative $A \to \beta \mid \gamma \mid \ldots$, che sono anche rappresentate da una macchina detta M_A (o A quando non vi sia rischio di confusione). Anche se non necessario per il metodo $LR(k)$, conviene mantenere l'ipotesi che la macchina sia deterministica. Lo stato iniziale di M_A è $q_{A,0}$ e gli stati finali sono F_A.
Inoltre si suppone che ogni alternativa $A \to \beta$ abbia uno stato finale distinto, $q_{A,\beta} \in F_A$; ciò consentirà al parsificatore di selezionare l'alternativa da applicare nella mossa di riduzione.
Per facilitare l'esposizione, conviene aggiungere alla grammatica la regola iniziale $S_0 \to S \dashv$, che dice che ogni frase è seguita dal terminatore. Il simbolo S_0 diviene così l'assioma, mentre S è degradato al ruolo d'un nonterminale qualsiasi.
Gli stati iniziale e finale della macchina dell'assioma sono rispettivamente $q_{iniziale}$ e $q_{terminale}$.
La seguente funzione calcola gli stati raggiungibili da uno stato mediante una catena di zero o più mosse spontanee.

Definizione 4.42. *Chiusura* $LR(0)$
Sia $q \in Q$ *uno stato della rete. La* chiusura *di* q *è:*

$C := \{q\}$;
repeat
$C := C \cup \{q_{A,0} \mid$ *per qualche* $p \in C$ *e* $A \in V$, $\exists$ *la mossa* $\delta(p, A)\}$;
until *nessun nuovo elemento è stato aggiunto a* C;
$chius(q) := C$

Per un insieme $R \subseteq Q$ *di stati la chiusura è definita da:*

$$chius(R) := \{r \mid \text{per qualche } q \in R, r \in chius(q)\}$$

La funzione serve nella costruzione della seguente macchina astratta.

Definizione 4.43. *Macchina* pilota *dell'analisi* $LR(0)$. [18]
Si definisce un automa finito

$$N = (R, \Sigma \cup V, \vartheta, I_0, R)$$

che piloterà il parsificatore: l'insieme degli stati, o meglio macrostati, *è* $R \subseteq$ *parti finite di* Q*; non si usano stati finali, ma per convenzione si prende l'intero insieme* R *come finale;*
l'alfabeto d'ingresso è l'unione di quello terminale e nonterminale della grammatica.
I macrostati, $R = \{I_0, \ldots\}$ *e la funzione di transizione* ϑ *sono calcolati, partendo dal macrostato iniziale, per mezzo della procedura:*

$I_0 = chius(q_{ini})$;
$R := \{I_0\}$;
repeat *per ogni* $I_j \in R$ *e per ogni* $X \in \Sigma \cup V$
$\vartheta(I_j, X) := chius(\{r \mid r \in \delta(q, X),$ *per qualche* $q \in I_j\})$;
if $\vartheta(I_j, X) \notin R$ *then* $R := R \cup \vartheta(I_j, X)$;
until *nessun nuovo macrostato è stato creato dall'iterazione;*

La funzione ϑ produce un macrostato, da aggiungere all'insieme R, se non già presente. Il calcolo della funzione e dei macrostati prosegue finché nessun nuovo macrostato è prodotto dal passo.
Un macrostato ha dunque come componenti uno o più stati, che possono essere finali o non, in qualche macchina della rete.
In un macrostato, uno stato è detto *di spostamento* se da esso, nella macchina della rete, esce un arco verso un altro stato.
Uno stato finale, appartenente a un macrostato, è detto *stato di riduzione.*
Le *riduzioni* associate a un macrostato sono le regole sintattiche il cui stato finale compare nel macrostato:

$$riduz(I_j) = \{A \to \alpha \mid q_{A,\alpha} \in I_j\}$$

[18] Anche detta *riconoscitore dei prefissi ascendenti* per le ragioni poi esposte.

Si noti che uno stato finale può essere allo stesso tempo di riduzione e di spostamento: ciò avviene quando da esso esce anche un arco etichettato. Riunendo i concetti, un macrostato può essere classificato come:

riduzione: se contiene soltanto stati di riduzione;
spostamento: se contiene soltanto stati di spostamento;
misto: se contiene stati di spostamento e stati di riduzione.

La condizione più semplice di determinismo è la seguente.

Definizione 4.44. *Condizione $LR(0)$ e famiglia $LR(0)$.*
Una grammatica soddisfa la condizione $LR(0)$ se ogni macrostato della macchina pilota verifica entrambe le condizioni:

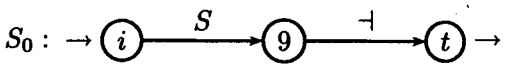
1. *non è misto;*
2. *se è un macrostato di riduzione, contiene un solo stato.*

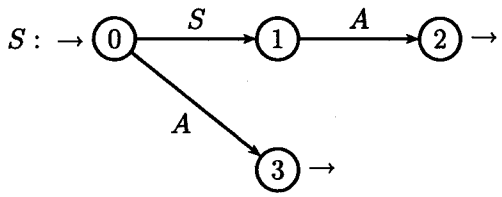
La famiglia dei linguaggi $LR(0)$ *è l'insieme dei linguaggi che possono essere generati da grammatiche della famiglia $LR(0)$.*

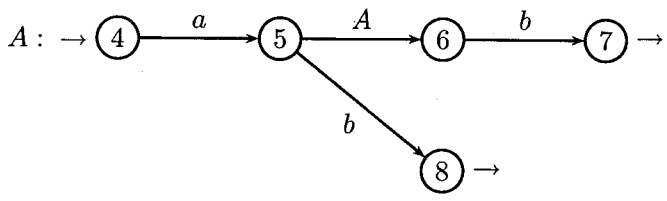
Si osservi che la condizione 1. vieta la compresenza d'un o stato finale e non finale nello stesso macrostato. La condizione 2. vieta la compresenza di due o più stati finali.

Esempio 4.45. Costruzione della macchina pilota per la grammatica dell'es. 4.41, riprodotta con l'aggiunta del nuovo assioma:

$S_0 \to S \dashv$ $\qquad S \to SA \mid A$ $\qquad A \to aAb \mid ab$



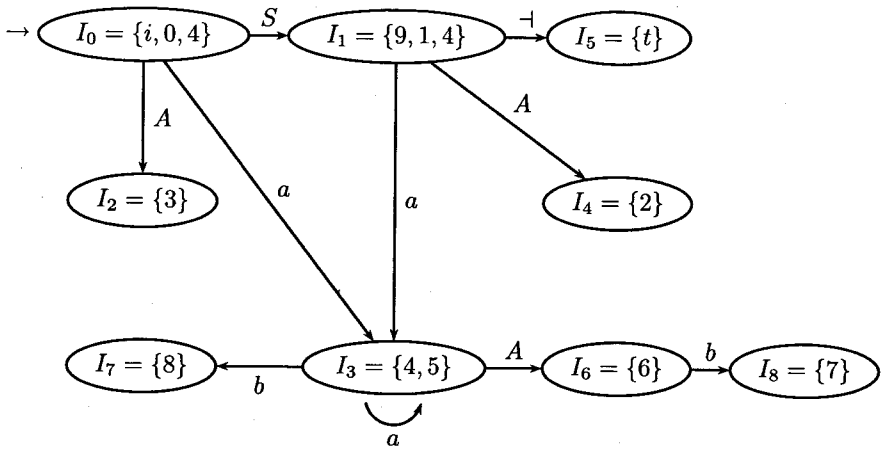




Sono riportati i passi di calcolo dei macrostati e della funzione di transizione del pilota.

	Insieme dei macrostati	*Macrostato creato*	*Mossa creata*
0	$\emptyset$	$\text{chius}(i) = \{i, 0, 4\} = I_0$	
1	$I_0 = \{i, 0, 4\}$	$\text{chius}(\delta(i,S) \cup \delta(0,S) \cup \delta(4,S)) = \text{chius}(\{9,1\}) = \{9,1,4\} = I_1$	$\vartheta(I_0, S) = I_1$
2	$I_0 = \{i, 0, 4\}, I_1$	$\text{chius}(\delta(i,A) \cup \delta(0,A) \cup \delta(4,A)) = \text{chius}(3) = \{3\} = I_2$	$\vartheta(I_0, A) = I_2$
3	$I_0 = \{i, 0, 4\}, I_1, I_2$	$\text{chius}(\delta(i,a) \cup \delta(0,a) \cup \delta(4,a)) = \text{chius}(5) = \{4,5\} = I_3$	$\vartheta(I_0, a) = I_3$
4	$I_0, I_1 = \{9,1,4\}, I_2, I_3$	$\text{chius}(\delta(9,A) \cup \delta(1,A) \cup \delta(4,A)) = \text{chius}(2) = \{2\} = I_4$	$\vartheta(I_1, A) = I_4$
5	$I_0, I_1 = \{9,1,4\}, I_2, I_3, I_4$	$\text{chius}(\delta(9,a) \cup \delta(1,a) \cup \delta(4,a)) = \text{chius}(5) = \{4,5\} = I_3$	$\vartheta(I_1, a) = I_3$
6	$I_0, I_1 = \{9,1,4\}, I_2, I_3, I_4$	$\text{chius}(\delta(9,\dashv) \cup \delta(1,\dashv) \cup \delta(4,\dashv)) = \text{chius}(t) = \{t\} = I_5$	$\vartheta(I_1, \dashv) = I_5$
7	$I_0, I_1, I_2, I_3 = \{4,5\}, I_4, I_5$	$\text{chius}(\delta(4,A) \cup \delta(5,A)) = \text{chius}(6) = \{6\} = I_6$	$\vartheta(I_3, A) = I_6$
8	$I_0, I_1, I_2, I_3 = \{4,5\}, I_4, I_5, I_6$	$\text{chius}(\delta(4,a) \cup \delta(5,a)) = \text{chius}(5) = \{4,5\} \equiv I_3$	$\vartheta(I_3, a) = I_3$
9	$I_0, I_1, I_2, I_3 = \{4,5\}, I_4, I_5, I_6$	$\text{chius}(\delta(4,b) \cup \delta(5,b)) = \text{chius}(8) = \{8\} = I_7$	$\vartheta(I_3, b) = I_7$
10	$I_0, I_1, I_2, I_3, I_4, I_5, I_6 = \{6\}, I_7$	$\text{chius}(\delta(6,b)) = \text{chius}(7) = \{7\} = I_8$	$\vartheta(I_6, b) = I_8$
11	$I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8$		alt

Segue il grafo dell'automa pilota.



I macrostati I_2, I_4, I_5, I_7, I_8 sono di riduzione e contengono un solo stato finale. Gli altri macrostati sono di spostamento, poiché contengono soltanto stati non finali. Non vi sono macrostati misti, né due o più riduzioni nello stesso macrostato di riduzione. Pertanto la condizione è verificata e la grammatica ha la proprietà $LR(0)$.

Proprietà della macchina pilota

Osservando l'esempio è facile riscontrare le particolari proprietà del grafo d'una macchina pilota $LR(0)$:

1. tutte le frecce entranti in un macrostato portano la stessa etichetta (vedasi I_3);
2. ogni macrostato di riduzione è privo di successori;
3. ogni macrostato di spostamento ha almeno un successore.

Si può chiedere quale sia il linguaggio riconosciuto dalla macchina pilota, supponendo che ogni macrostato sia finale. Nell'esempio le stringhe riconosciute sono

$$S, A, a, S\dashv, SA, Sa, aA, aa, ab, SaA, Saa, aAb, aaa, aab, aaAb, \ldots$$

Esse hanno la caratteristica di essere dei prefissi delle forme di frasi generate dalla grammatica G, mediante derivazioni destre.
Ad es. la derivazione destra

$$S_0 \Rightarrow S\dashv \Rightarrow A\dashv \Rightarrow aAb\dashv \Rightarrow aaAbb\dashv \Rightarrow aaabbb\dashv$$

presenta i prefissi accettati dal pilota:

$$S, S \dashv, A, A \dashv, a, aA, aAb, aa, aaA, aaAb, aaab$$

Ma non tutti i prefissi delle forme destre sono accettati, ad es. *aaAbb* e *aaabb* non lo sono.

La spiegazione di questa proprietà si trova nel ragionamento che ha portato alla costruzione del pilota, con la tattica di condurre in contemporanea più calcoli, fino al momento in cui uno di essi raggiunge uno stato finale d'una macchina, ossia un macrostato di riduzione.

Più precisamente, si osservino nel grafo del pilota i cammini che risalgono a ritroso da un macrostato di riduzione.

Sia $A \to \beta$ la riduzione associata al macrostato di riduzione I; allora esiste nel pilota un cammino retroverso, etichettato β^R, che da I conduce a un macrostato contenente come componente lo stato iniziale della macchina A.

Così dal macrostato $I_8 = \{7\}$ i tre passi a ritroso b, A, a risalgono al macrostato I_0, che contiene lo stato iniziale 4 della macchina A; oppure al macrostato I_3, che pure contiene 4.

In che modo differiscono i prefissi riconosciuti e non riconosciuti dal pilota? Un prefisso non riconosciuto come *aaabb* contiene una sottostringa *interna* che è la parte destra d'una regola:

$$aa \underbrace{ab}_{A \to ab} b$$

Invece in un prefisso riconosciuto, come *aaab*, tale sottostringa può solo occupare la posizione finale (suffisso):

$$aa \underbrace{ab}_{A \to ab}$$

4.5.3 Analizzatore a spostamento e riduzione

Se una grammatica è $LR(0)$, dalla sua macchina pilota si ottiene subito l'automa a pila deterministico che riconosce le frasi del linguaggio e ne costruisce gli alberi sintattici. Esso opera nel seguente modo. I simboli della pila sono i macrostati, ai quali si possono affiancare per migliore leggibilità anche i simboli della grammatica. Esaminando il carattere terminale corrente, l'automa, avendo in cima alla pila il macrostato I, esegue la mossa prescritta dall'automa pilota e impila il prossimo macrostato; tale operazione è detta di *spostamento*. Avendo in cima alla pila un macrostato di riduzione, cui è per la seconda condizione $LR(0)$ associata una sola alternativa sintattica, l'automa esegue una serie di operazioni note come *riduzione*. Esse simulano l'omonimo passo nella derivazione d'una frase, prima cancellando, poi inserendo simboli sulla pila.

Al termine della scansione, l'automa accetta la stringa sorgente se la pila è vuota (o se contiene il solo macrostato iniziale).

Algoritmo 4.46. Analizzatore a spostamento e riduzione (senza prospezione). Sia G una grammatica $LR(0)$ e $N = (R, \Sigma \cup V, \vartheta, I_0, R)$ la sua macchina pilota. Si costruisce l'automa a pila A come segue.

Alfabeto di pila: $R \cup \Sigma \cup V$;
Insieme degli stati di A: irrilevante perché contiene un solo stato;
Configurazione iniziale: la pila contiene il macrostato iniziale I_0;
Mosse dell'automa: vi sono due tipi di mosse, quelle di spostamento e quelle di riduzione. Sia I il macrostato in cima alla pila e a il carattere corrente:

Mossa di *spostamento*: se per il macrostato corrente I è definita nel pilota la mossa $\vartheta(I, a) = I'$, l'automa legge $a \in \Sigma$, avanzando la testina, e impila la stringa aI';

Mossa di *riduzione*: se il macrostato corrente I, qui denotato I_n, contiene lo stato di riduzione q, con

$$riduz(q) = \{B \to X_1 X_2 \dots X_n\}$$

dove $n \geqslant 0$ è la lunghezza della parte destra, si compie l'azione seguente.

In cima si troverà necessariamente (per il modo in cui il pilota è stato costruito) una stringa β'

$$\dots I' \overbrace{X_1 I_1 X_2 I_2 \dots X_n I_n}^{\beta'}$$

contenente n macrostati inseriti da mosse precedenti.

La mossa di riduzione è una mossa spontanea (senza avanzamento della testina) che, prima cancella dalla pila la stringa β' (ossia i primi $2n$ simboli dalla cima), poi inserisce la stringa BI'', dove I'' è il macrostato raggiunto leggendo (per così dire) il nonterminale B, ossia è $I'' = \vartheta(I', B)$.

Configurazione di riconoscimento: pila $= I_0$, ingresso $= \dashv$

Se l'automa a pila raggiunge una configurazione in cui la mossa non è possibile, esso rifiuta la stringa.

Si noti che, grazie all'ipotesi $LR(0)$, il macrostato in cima alla pila seleziona il tipo di mossa, accettazione, spostamento o riduzione, e nell'ultimo caso designa univocamente la riduzione da eseguire. Di conseguenza il funzionamento è deterministico.

A meglio vedere, le informazioni messe in pila sono ridondanti: infatti lo spostamento inserisce nella pila un carattere terminale a e il macrostato di arrivo I', ma dal secondo si deduce il primo, perché nel grafo della macchina pilota tutti gli archi entranti in un macrostato portano la stessa etichetta. L'inserimento del carattere terminale ha soltanto scopo esplicativo e permette di seguire più facilmente le simulazioni dell'automa a pila.

Una questione più sottile riguarda gli stati dell'automa a pila. È davvero corretta la precedente affermazione che l'automa non fa uso degli stati? A meglio vedere, la mossa di riduzione è un'operazione più complessa delle istruzioni elementari consentite a un automa a pila. Certamente la riduzione può esere decomposta in una serie di istruzioni elementari, ma facendo ricorso a una memoria finita per contare quanti simboli vanno rimossi dalla pila, e per memorizzare il simbolo nonterminale B della parte sinistra della riduzione. In conclusione, l'automa a pila $LR(0)$ fa un uso nascosto di stati interni nelle mosse di riduzione.

Passando al metodo generale $LR(k)$, gli stati interni avranno un ulteriore motivo: per memorizzare i k caratteri esaminati dalla prospezione (esattamente come nel caso $LL(k)$).

In definitiva, tutti i parsificatori deterministici, fanno uso d'una memoria finita oltre che della pila illimitata (anche se la presentazione degli algoritmi non ne parla per non cadere in una descrizione di livello troppo basso). Il contrario sarebbe sorprendente, infatti si sa che la famiglia DET dei linguaggi deterministici (p. 160) è caratterizzata dall'avere come riconoscitore un automa a pila deterministico dotato di stati interni.

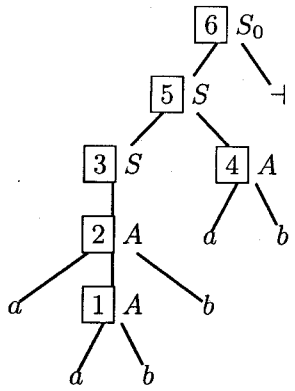
Derivazione destra inversa

Si mostra ora come l'automa a pila parsifica una stringa e in qual ordine ricostruisce l'albero sintattico.

Esempio 4.47. Traccia dell'analisi (es. 4.45).

Pila	x	*Commento*
I_0	$a\ a\ b\ b\ a\ b\ \dashv$	sposta
$I_0\ aI_3$	$a\ b\ b\ a\ b\ \dashv$	sposta
$I_0\ aI_3\ aI_3$	$b\ b\ a\ b\ \dashv$	sposta
$I_0\ aI_3\ aI_3\ bI_7$	$b\ a\ b\ \dashv$	riduci con $A \to ab$
$I_0\ aI_3\ AI_6$	$b\ a\ b\ \dashv$	sposta
$I_0\ aI_3\ AI_6\ bI_8$	$a\ b\ \dashv$	riduci con $A \to aAb$
$I_0\ AI_2$	$a\ b\ \dashv$	riduci con $S \to A$
$I_0\ SI_1$	$a\ b\ \dashv$	sposta
$I_0\ SI_1\ aI_3$	$b\ \dashv$	sposta
$I_0\ SI_1\ aI_3\ bI_7$	$\dashv$	riduci con $A \to ab$
$I_0\ SI_1\ AI_4$	$\dashv$	riduci con $S \to SA$
$I_0\ SI_1$	$\dashv$	sposta
$I_0\ SI_1\ \dashv\ I_5$		riduci con $S_0 \to S \dashv$
I_0		accetta

L'albero sintattico



è costruito dalle mosse di riduzione nell'ordine dei numeri crescenti. Le regole applicate sono

$$A \to ab, A \to aAb, S \to A, A \to ab, S \to SA, S_0 \to S \dashv$$

e corrispondono alla catena di riduzioni:

$$aabbab \dashv \Rightarrow aAbab \dashv \Rightarrow Aab \dashv \Rightarrow Sab \dashv \Rightarrow SA \dashv \Rightarrow S \dashv \Rightarrow S_0$$

L'osservazione dell'ordine di costruzione dell'albero evidenzia che questo è un parsificatore ascendente. Invertendo specularmente l'ordine della riduzione, si ottiene 654321 e la corrispondente derivazione $S_0 \overset{+}{\Rightarrow} aabbab \dashv$, è una derivazione destra, come già osservato (p. 168).

4.5.4 Analisi sintattica con prospezione $LR(k)$

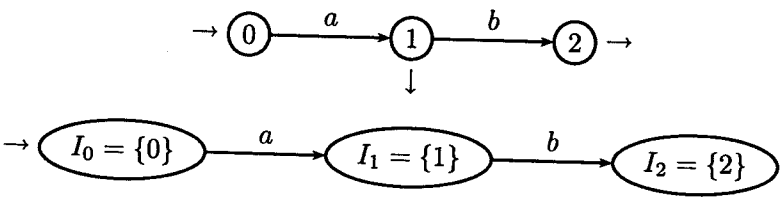
Il prossimo algoritmo di analisi è il più potente tra quelli deterministici. Esso combina efficacemente due accorgimenti: l'uso della prospezione, come nei metodi $LL(k)$, e la tattica di rinvio delle scelte, come nel caso $LR(0)$.
Dopo un esame dei limiti del metodo $LR(0)$, si espone il procedimento $LR(k)$ di calcolo della prospezione per la macchina pilota, fissando a uno la lunghezza della prospezione. Tale scelta da un lato semplifica l'esposizione, dall'altro, diversamente dal caso $LL(k)$, non restringe la famiglia dei linguaggi riconoscibili.

Limiti del metodo $LR(0)$

Avviene di rado che un parsificatore si possa costruire con il metodo $LR(0)$ e ben pochi linguaggi tecnici soddisfano tale condizione.
La limitazione più seria dei linguaggi $LR(0)$ è la seguente: se una stringa appartiene al linguaggio, nessun prefisso di essa può appartenervi, ossia i linguaggi $LR(0)$ sono necessariamente *privi di prefissi*.

Esempio 4.48. Un linguaggio finito non $LR(0)$.
Il linguaggio finito $\{a, ab\}$ non è $LR(0)$, perché una frase è prefisso d'un'altra. Si osservi la grammatica, la macchina corrispondente e la macchina pilota:

$$S \to a \mid ab$$

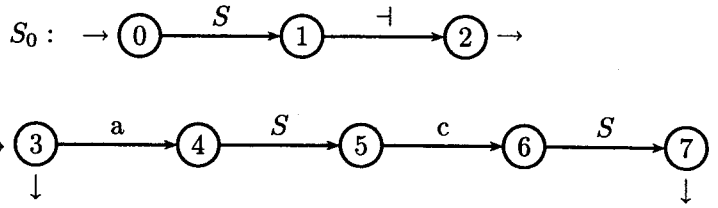


Il macrostato I_1, contenendo lo stato finale 1, è misto: di riduzione con la regola $S \to a$ e di spostamento per l'arco uscente verso 2. Pertanto la condizione $LR(0)$ cade. Intuitivamente, ciò significa che l'analizzatore non può decidere se applicare la riduzione $a \Rightarrow A$ o continuare a spostare; per deciderlo dovrebbe esaminare il prossimo carattere ($\dashv$ per la riduzione e b per lo spostamento), come farebbe un analizzatore $LL(1)$.

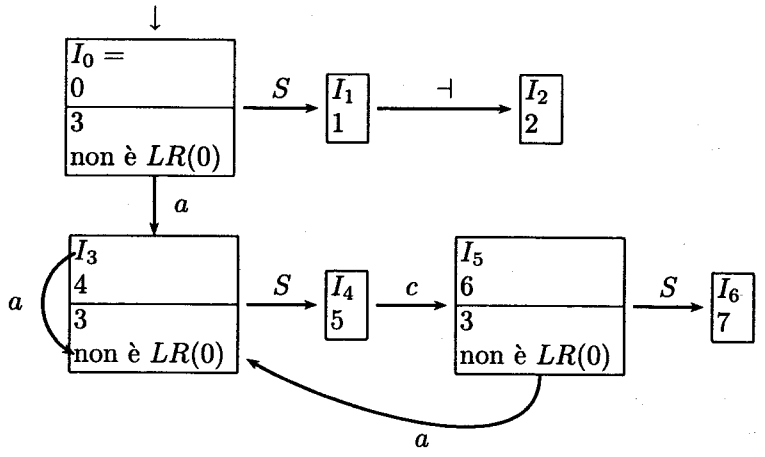
Un secondo esempio contenente dei prefissi è il classico linguaggio delle liste con separatore $e(se)^*$, dove s'incontrano le stringhe e, ese, una prefisso dell'altra. Vi sono dunque linguaggi regolari che escono dalla famiglia $LR(0)$.
Ma la limitazione relativa ai prefissi è ancora più seria, perché si applica alle stringhe generate, non soltanto dall'assioma, ma da qualsiasi nonterminale. Di conseguenza, le liste con separatori non possono comparire neanche come costrutto in un linguaggio tecnico, il che preclude ad es. le liste dei parametri d'una procedura, e tanti casi simili.
Proseguendo la rassegna delle limitazioni, è semplice vedere che la presenza di regole vuote nella grammatica viola la condizione $LR(0)$. Infatti una regola del tipo $A \to \varepsilon \mid \beta$ fa sì che lo stato iniziale $q_{A,0}$ della macchina M_A sia anche finale. Ma da tale stato esce anche l'arco associato alla parte destra (non vuota) β. Di conseguenza, il macrostato contenente lo stato $q_{A,0}$ viola la condizione $LR(0)$.

Esempio 4.49. La grammatica di Dyck non è $LR(0)$.
La nota grammatica

$$S_0 \to S \dashv \qquad S \to aScS \mid \varepsilon$$



porta alla macchina pilota $LR(0)$ seguente:



Nella figura ogni macrostato è sudddiviso dalla riga orizzontale nei sottoinsiemi calcolati dalla funzione di chiusura.
Tutti i macrostati contenenti lo stato 3 sono misti, per la presenza della riduzione $S \to \varepsilon$ e dello spostamento $3 \xrightarrow{a} 4$; essi sono dunque inadeguati per $LR(0)$.

Aggiunta della prospezione alla macchina pilota

Per potenziare il metodo $LR(0)$ e trasformarlo nel metodo pratico $LR(1)$, si aggiungerà a ogni macrostato l'informazione sulla prospezione.
Poiché un macrostato può contenere più stati, è necessario calcolare l'insieme dei caratteri terminali di prospezione per ciascuno stato e per ogni freccia uscente da esso.

Intuitivamente, un macrostato risulta adeguato per l'analisi $LR(1)$ se la prossima mossa dell'automa a pila è univocamente determinata dal macrostato posto in cima alla pila e dall'insieme di prospezione cui appartiene il carattere corrente.

Definizione 4.50. *Componenti della macchina pilota $LR(1)$.*
Un macrostato I della macchina pilota $LR(1)$ è un insieme di coppie, dette candidate, *della forma*

$$\langle q, a\rangle \in Q \times (\Sigma \cup \{\dashv\})$$

dove Q è l'insieme degli stati della rete di macchine.
Per brevità, più candidate aventi in comune lo stesso stato saranno solitamente raggruppate:

$$\langle q, \{a_1, a_2, \ldots, a_k\}\rangle \equiv \{\langle q, a_1\rangle, \langle q, a_2\rangle, \ldots, \langle q, a_k\rangle\}$$

L'insieme $\{a_1, a_2, \ldots, a_k\}$ è detto insieme di prospezione *dello stato q.*

La funzione di chiusura sotto descritta arricchisce quella del caso $LR(0)$, con il calcolo degli insiemi di prospezione.

Algoritmo 4.51. Chiusura $LR(1)$.
Sia $c = \langle q, \pi\rangle$ una candidata. La *chiusura* $LR(1)$ di c, $\text{chius}_1(c)$, è così calcolata:
$C := \{\langle q, \pi\rangle\}$;
repeat

$C := C \cup \{\langle q_{A,0}, \rho\rangle\}$ dove:

$q_{A,0}$ è lo stato iniziale della macchina M_A tale che:
esiste una candidata $\langle p, \pi\rangle \in C \ \wedge$ da p esce un arco $\delta(p, A) = p'$.

La prospezione ρ è così calcolata:

$$\begin{cases} \rho = Ini(L(p')) & \text{se } L(p') \text{ non è annullabile,} \\ \rho = Ini(L(p')) \cup \pi & \text{altrimenti} \end{cases}$$

until nessun nuovo elemento è stato aggiunto a C;
$\text{chius}_1(q) := C$

Commento: se nell'insieme corrente C vi è uno stato p con prospezione π, e nella grammatica esiste l'arco $p \xrightarrow{A} p'$, allora si aggiunge all'insieme C lo stato iniziale $q_{A,0}$ (come nel caso $LR(0)$). Per calcolare l'insieme di prospezione ρ di $q_{A,0}$, si raccolgono i caratteri che possono seguire la stringa che deriva da A. Tali caratteri possono provenire da due casi: gli inizi del linguaggio $L(p')$ riconosciuto partendo dallo stato p'; oppure, se lo stato p' è finale o più in

generale, se $L(p')$ contiene la stringa vuota, anche i caratteri dell'insieme π confluiscono nell'insieme ρ.

Per un insieme I di candidate, la chiusura è l'unione delle chiusure delle candidate appartenenti all'insieme I.

La macchina pilota è costruita in modo simile al caso $LR(0)$, ma ora i componenti dei macrostati sono le candidate e non i semplici stati.

Algoritmo 4.52. Macchina *pilota* dell'analisi $LR(1)$
Si definisce un automa finito

$$N = (R, \Sigma \cup V, \vartheta, I_0, R)$$

che piloterà il parsificatore, così caratterizzato:

- l'insieme dei macrostati è R;
- l'alfabeto è l'unione di quello terminale Σ e nonterminale V della grammatica;
- il macrostato I_0 ha come contenuto iniziale la coppia
  $\langle q_{ini}, \dashv \rangle$ ≡(stato iniziale della rete, terminatore);
- i macrostati, $R = \{I_0, \ldots\}$ e la funzione di transizione ϑ sono calcolati, partendo dal macrostato iniziale, per mezzo della procedura seguente:

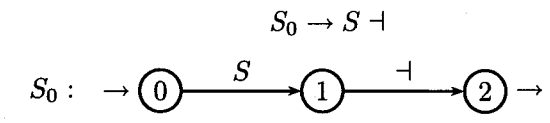
  $I_0 =$ chius$_1(\langle q_{ini}, \dashv \rangle)$;
  $R := \{I_0\}$;
  repeat per ogni macrostato $I_j \in R$ e per ogni simbolo $X \in \Sigma \cup V$

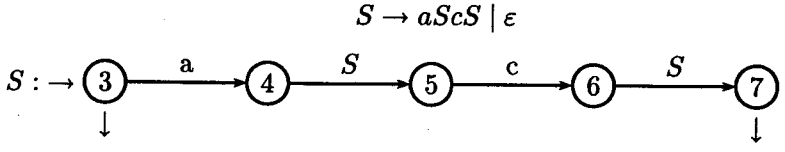
  $\vartheta(I_j, X) :=$ chius$_1(\{\langle r, \pi \rangle\})$ dove
  l'insieme delle candidate $\langle r, \pi \rangle$ è così definito:
  esiste nel macrostato I_j una candidata $\langle q, \rho \rangle$ per la quale
  esiste nella rete l'arco $\delta(q, X) = r$;
  l'insieme di prospezione π è copiato da ρ, ossia $\pi := \rho$;

  if $\vartheta(I_j, X) \notin R$ then $R := R \cup \vartheta(I_j, X)$;

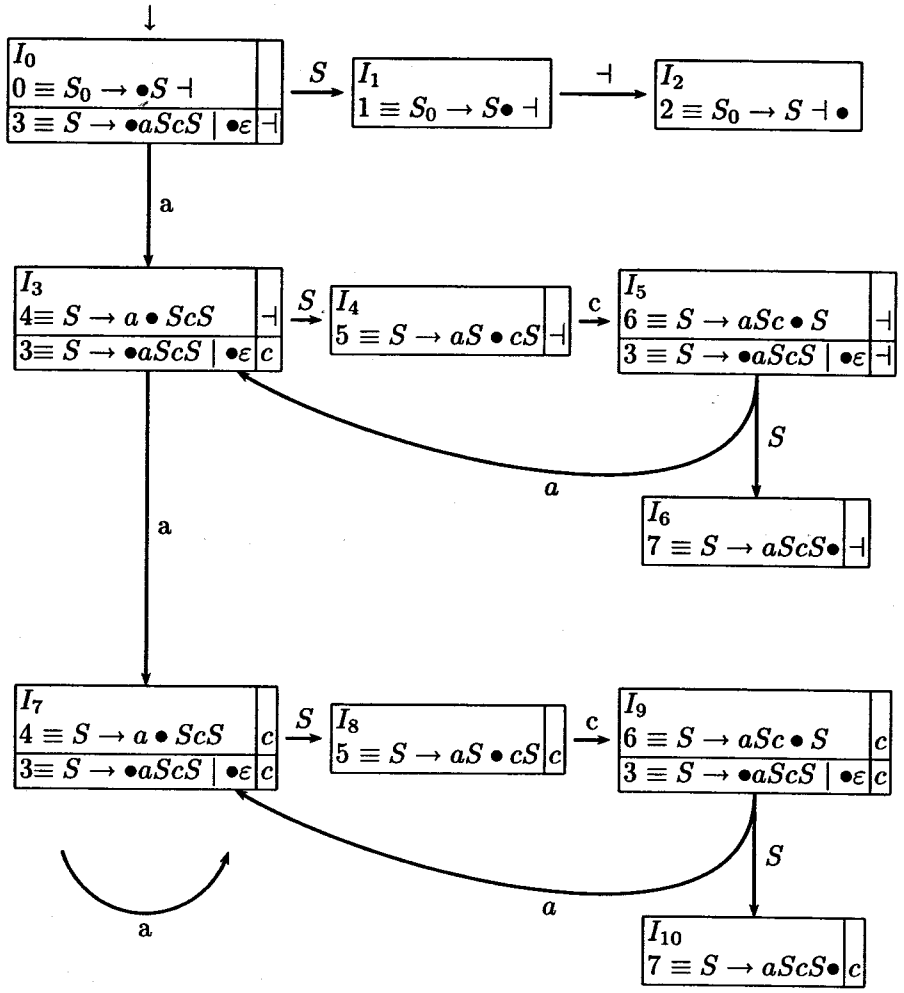
  until nessun nuovo macrostato è stato prodotto dall'iterazione.

Esempio 4.53. Linguaggio di Dyck (es. 4.49) come $LR(1)$.





La macchina pilota $LR(1)$ è sotto mostrata. La colonna di destra d'un macrostato contiene le prospezioni.
Per facilitare la correlazione visiva tra i macrostati e la grammatica, si possono riportare nel disegno del pilota alcune o tutte le regole, marcando il punto corrispondente allo stato.
Una regola grammaticale, come $S \to aScS$, è detta *marcata* se la parte destra contiene una marca (rappresentata dal tondino). Ad es. la regola marcata $S \to a \bullet ScS$ denota lo stato 4 della macchina M_S.
La compresenza di stati e regole marcate è ridondante, perché la marca è soltanto un altro modo di individuare lo stato d'una macchina della rete.



Ciascun macrostato contiene il proprio nome e l'insieme delle candidate. Ad es. le due candidate di I_3 sono $\langle 4, \{\dashv\}\rangle$ e $\langle 3, \{c\}\rangle$.
Per agevolare la lettura, un tratto orizzontale separa le candidate calcolate dall'invocazione della funzione chiusura, ma l'ordine di scrittura delle candidate è irrilevante, poiché un macrostato è un insieme.
Un caso particolare sono le candidate (nei macrostati I_0, I_1, I_2) della regola convenzionale $S_0 \to S \dashv$, in quanto essi non hanno prospezione.
Per illustrare il calcolo delle prospezioni, si consideri I_3. La coppia $\langle 4, \dashv\rangle$ ha la stessa prospezione dello stato 3 di I_0 da cui nasce grazie all'arco $3 \xrightarrow{a} 4$.

Applicando la chiusura alla coppia $\langle 4, \dashv \rangle$ si ottiene lo stato 3 e la prospezione $\{c\}$, poiché $Ini(L(5)) = \{c\}$ e $L(5)$ non è annullabile.
Si noti che lo stato 3 è commentato da due regole marcate, $S \to \bullet aScS$ e $S \to \bullet\varepsilon$; la seconda, si può scrivere come $S \to \varepsilon\bullet$.
Si osserva la crescita del numero dei macrostati rispetto al pilota $LR(0)$, causata dalla presenza delle prospezioni. Infatti due macrostati, come I_3, I_7, che differiscono soltanto nelle prospezioni, sono fusi insieme nella macchina $LR(0)$.

Condizione $LR(1)$

Una regola marcata è detta *completata* se la marca sta in fondo alla parte destra. Es.:

$$S \to \varepsilon\bullet \qquad S \to aScS\bullet \qquad S_0 \to S \dashv \bullet$$

Si riprende la classificazione dei macrostati del caso $k = 0$.
Una *candidata* è detta di *riduzione* se lo stato è finale in una macchina della rete, nel qual caso ad esso è associata una regola marcata completata.
Una *candidata* è detto di *spostamento*, se lo stato è l'origine di una freccia etichettata con un simbolo, terminale o non; ossia se la regola marcata contiene un simbolo a destra del pallino.
Ma la drastica condizione $LR(0)$, che proibiva ai macrostati di contenere riduzioni e spostamenti o riduzioni multiple, è sostituita da un controllo più fine sugli insiemi di prospezione.

Definizione 4.54. *Condizione $LR(1)$ Una grammatica soddisfa la condizione $LR(1)$ se, per ogni macrostato della sua macchina pilota, valgono entrambe le condizioni:*

1. assenza di conflitto riduzione-spostamento*: ogni candidata di riduzione ha un insieme di prospezione disgiunto dall'insieme dei simboli terminali di spostamento (ossia dall'insieme delle etichette terminali delle frecce uscenti dal macrostato);*
2. assenza di conflitto riduzione-riduzione*: se vi sono due candidate di riduzione, i loro insiemi di prospezione sono disgiunti.*

Una grammatica gode della proprietà $LR(1)$ se ogni macrostato della macchina pilota soddisfa la condizione $LR(1)$.

Esempio 4.55. Linguaggio di Dyck (es. 4.53): verifica della condizione $LR(1)$.
I macrostati $I_1, I_2, I_4, I_6, I_8, I_{10}$, aventi una sola candidata, soddisfano banalmente la condizione.
Ciascuno dei macrostati I_0, I_3, I_5, I_7, I_9 possiede la candidata di riduzione $S \to \bullet\varepsilon$, la candidata di spostamento $S \to \bullet aScS$ (entrambe associate allo stato 3), e un'altra candidata di spostamento. Segue la verifica della disgiunzione:

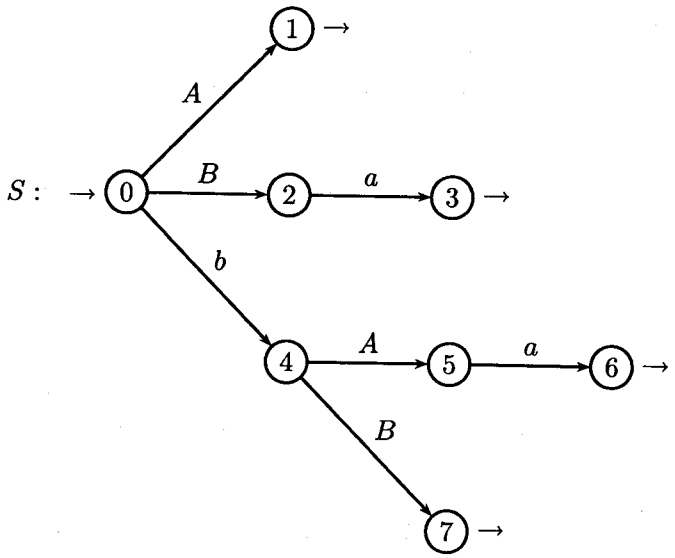
macrostato	*prospezione di* $S \to \varepsilon$	*etichette uscenti*	*condizione*
I_0	$\dashv$	a	$\{\dashv\} \cap \{a\} = \emptyset$
I_3	c	a	$\{c\} \cap \{a\} = \emptyset$
I_5	$\dashv$	a	$\{\dashv\} \cap \{a\} = \emptyset$
I_7	c	a	$\{c\} \cap \{a\} = \emptyset$
I_9	c	a	$\{c\} \cap \{a\} = \emptyset$

La grammatica risulta $LR(1)$.

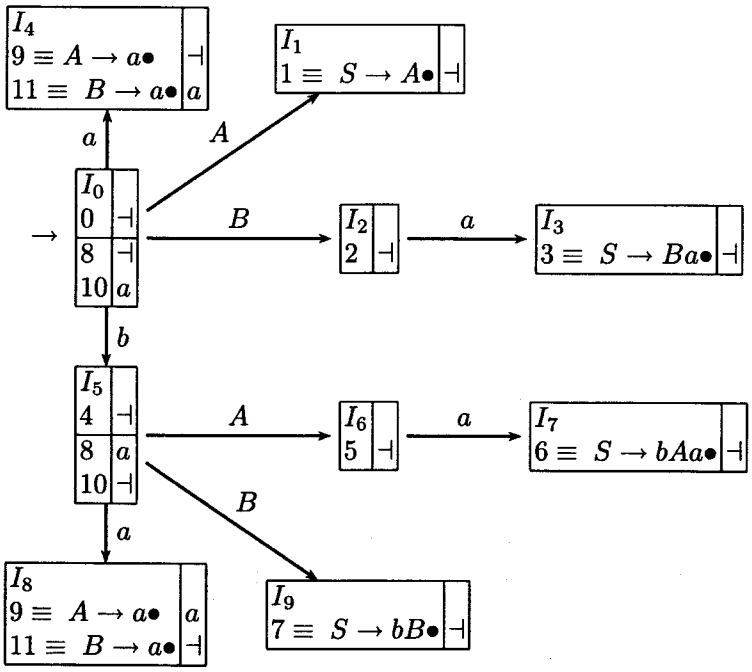
Il prossimo linguaggio (finito) illustra il ramo 2 della condizione.

Esempio 4.56. Condizione $LR(1)$ con due riduzioni nel macrostato.
La grammatica e la rete sono:

$S \to A \mid Ba \mid bAa \mid bB \qquad A \to a \qquad B \to a$



Per non complicare il disegno, nell'automa pilota $LR(1)$ si scrivono, soltanto negli stati di riduzione, le regole marcate:

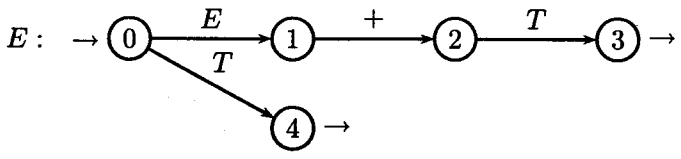


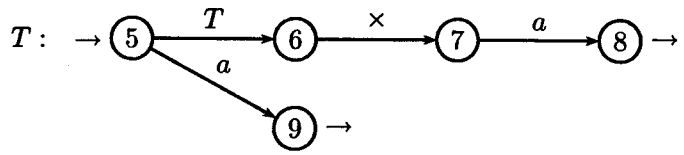
I macrostati che potrebbero violare la condizione sono I_4 e I_8, mentre gli altri soddisfano già la condizione $LR(0)$. Ciascuno dei due macrostati contiene due candidate di riduzione, ma i loro insiemi di prospezione sono disgiunti.
Per altro, poiché da nessuno dei due escono archi, la parte 1 della condizione è manifestamente soddisfatta.
La grammatica risulta $LR(1)$, ma non $LR(0)$, perché in I_4 e in I_8 vi sono due riduzioni, tra le quali la scelta, senza prospezione, sarebbe indeterministica.

Il prossimo esempio chiarisce ulteriormente il ruolo degli insiemi di prospezione.

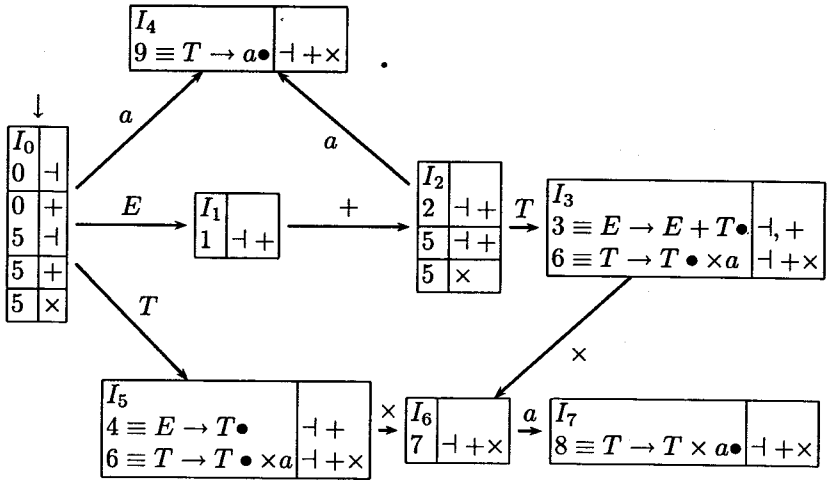
Esempio 4.57. Espressioni aritmetiche, condizione $LR(1)$.

$$E \to E + T \mid T \qquad T \to T \times a \mid a$$





Macchina pilota $LR(1)$:



La grammatica risulta $LR(1)$. Infatti nessun macrostato contiene due stati finali e, nei macrostati misti riduzione/spostamento, la condizione di disgiunzione è soddisfatta, come sotto verificato:

$$I_3 : \times \notin \{\dashv, +\}, \text{ prospezione della riduzione } E \to E + T$$
$$I_5 : \quad \times \notin \{\dashv, +\}, \text{ prospezione della riduzione } E \to T$$

Si noti che in I_3 gli insiemi di prospezione della candidata di spostamento $6 \equiv T \to T \bullet \times a$ e di quella di riduzione $3 \equiv E \to E + T\bullet$ non sono disgiunti: $\{\dashv, +, \times\} \cap \{\dashv, +\} \neq \emptyset$, ma ciò non viola la condizione $LR(1)$. Infatti l'insieme di prospezione d'una candidata di spostamento non è argomento del predicato di verifica, ma serve soltanto per calcolare gli insiemi di prospezione dei macrostati successivi.

Pertanto, una volta che la macchina pilota è stata costruita, gli insiemi di prospezione delle candidate di spostamento possono essere eliminati, perché non servono al parsificatore.

Condizione $LALR(1)$

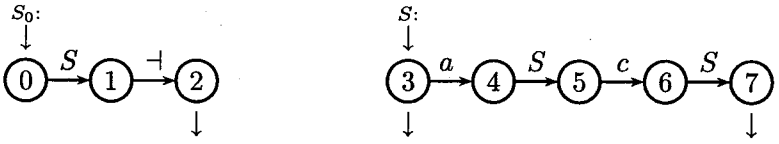
Si considera brevemente una condizione di determinismo intermedia tra quelle $LR(0)$ e $LR(1)$, che ha conosciuto ampia diffusione quando la ridotta memoria dell'elaboratore rendeva sconsigliabile l'uso del metodo $LR(1)$ a causa del numero elevato di macrostati del pilota.
Riprendendo il caso d'una grammatica $LR(1)$ ma non $LR(0)$, si immagini di semplificare il pilota LR(1), fondendo insieme i macrostati che sono indistinguibili nel pilota $LR(0)$, ossia quelli le cui candidate differiscono soltanto nella seconda componente, la prospezione. Al contempo si desidera preservare le informazioni sulla prospezione. Più precisamente, quando si fondono due macrostati, si uniscono gli insiemi di prospezione delle candidate che coincidono nella prima componente (ossia lo stato). Il grafo della macchina pilota[19] $LALR(1)$[20] così ottenuta è isomorfo a quello del pilota $LR(0)$. Esso può però contenere macrostati misti o con riduzioni plurime, purché valga la condizione seguente, identica a quella del caso $LR(1)$.

Una grammatica soddisfa la *condizione* $LALR(1)$ se, per ogni macrostato dell'automa pilota $LALR(1)$, valgono entrambe le condizioni:

1. ogni candidata di riduzione ha un insieme di prospezione disgiunto dall'insieme delle etichette terminali degli archi uscenti dal macrostato;
2. se vi sono due candidate di riduzione, i loro insiemi di prospezione sono disgiunti.

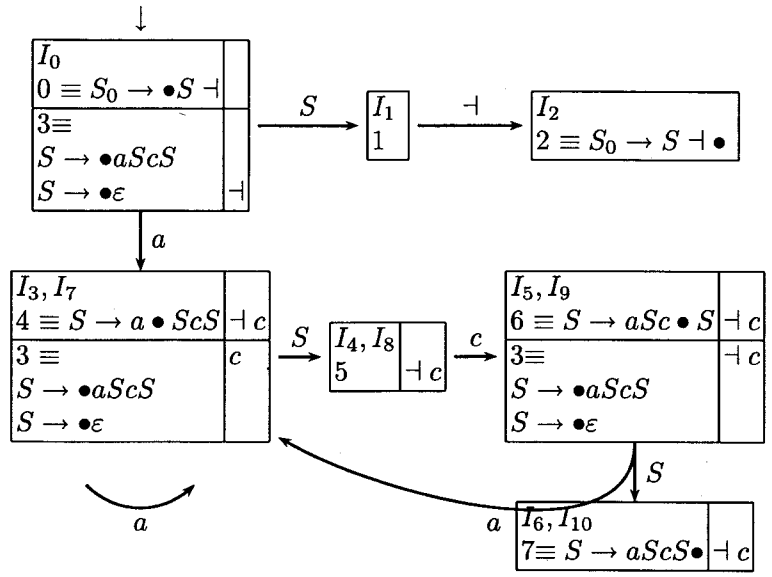
Dalla definizione segue immediatamente che la famiglia delle grammatiche $LALR(1)$ è inclusa in quella delle grammatiche $LR(1)$ e include la famiglia $LR(0)$.
Anche per i linguaggi valgono le stesse inclusioni strette: esistono linguaggi $LR(1)$ ma non $LALR(1)$, e linguaggi $LALR(1)$ ma non $LR(0)$.
Alcuni casi sono illustrati dai prossimi esempi.

Esempio 4.58. Linguaggio di Dyck 4.53 come $LALR(1)$.
La grammatica è nota:



Fondendo i macrostati del pilota $LR(1)$ (di p. 215), non distinguibili nel pilota $LR(0)$ (di p. 210), si ottiene il pilota $LALR(1)$:

[19] Per la costruzione del pilota $LALR(1)$ esistono algoritmi diretti, che evitano di passare attraverso il pilota $LR(1)$.
[20] Sigla di *Look Ahead* $LR(1)$.



il cui grafo è isomorfo a quello del pilota $LR(0)$.
Nessun macrostato contiene più riduzioni, ma vi sono i macrostati misti $I_0, [I_3, I_7], [I_5, I_9]$, che fanno cadere il metodo $LR(0)$. Nei macrostati misti, l'insieme di prospezione della riduzione $S \to \varepsilon$ è disgiunto dalle etichette degli archi uscenti, e la grammatica risulta $LALR(1)$.
Questo pilota svolge, con un numero minore di stati, lo stesso compito del pilota $LR(1)$.

Il prossimo è un esempio in cui i piloti $LR(1)$ e $LALR(1)$ coincidono.

Esempio 4.59. Espressioni aritmetiche (es. 4.57) come $LALR(1)$.
La macchina pilota $LR(1)$ di p. 219 è già isomorfa a quella $LR(0)$, perché non contiene macrostati identici a meno degli insiemi di prospezione. Ciò significa che la grammatica

$$E \to E + T \mid T \qquad T \to T \times a \mid a$$

soddisfa la condizione $LALR(1)$.

Esempio 4.60. Esempio $LR(1)$ non $LALR(1)$.
Nell'es. 4.56 (p. 217), fondendo insieme i macrostati I_4 e I_8, indistinguibili senza la prospezione, si ottiene il macrostato del pilota $LALR(1)$:

I_4, I_8	
$9 \equiv A \to a\bullet$	$\dashv, a$
$11 \equiv B \to a\bullet$	$\dashv, a$

Poiché le due riduzioni hanno insiemi di prospezione sovrapposti (identici), il macrostato non è $LALR(1)$.

Molte esperienze di progetto dei compilatori hanno dimostrato che il metodo $LALR(1)$ è spesso adatto alla costruzione dei parsificatori, di solito al costo di piccole modifiche della grammatica di riferimento del linguaggio.

4.5.5 Algoritmo di parsificazione $LR(1)$

Costruito il pilota $LR(1)$ o $LALR(1)$, il parsificatore agisce come quello $LR(0)$ (p. 207), con in più l'esame della prospezione per scegliere la mossa nei macrostati misti o aventi riduzioni multiple.

Algoritmo 4.61. Costruzione dell'analizzatore a spostamento e riduzione con prospezione.
Sia G una grammatica $LR(1)$ (o $LALR(1)$) e

$$N = (R, \Sigma \cup V, \vartheta, I_0, R)$$

la sua macchina pilota. Si costruisce l'automa a pila A come segue:

Alfabeto di pila: $R \cup \Sigma \cup V$;
Insieme degli stati di A: irrilevante perché contiene un solo stato;
Configurazione iniziale: la pila contiene I_0, il macrostato iniziale;
Mosse dell'automa: Sia I il macrostato in cima alla pila e a il carattere corrente. Due sono i tipi di mosse.

Mossa di spostamento: se per il macrostato I è definita nel pilota la mossa $\vartheta(I, a) = I'$, l'automa legge $a \in \Sigma$, avanzando la testina, e impila la stringa aI';

Mossa di riduzione: se il macrostato corrente, qui denotato I_n, contiene la candidata di riduzione

$$q, \equiv B \to X_1 X_2 \ldots X_n\bullet, \pi$$

(ossia è $riduz(q) = \{B \to X_1 X_2 \ldots X_n\}$) dove $n \geqslant 0$ è la lunghezza della parte destra, e
se il carattere a appartiene all'insieme di prospezione π, si compiono le operazioni seguenti.
In cima alla pila si trova necessariamente una stringa β'

$$I' \overbrace{X_1 I_1 X_2 I_2 \dots X_n I_n}^{\beta'}$$

contenente n macrostati inseriti da mosse precedenti.
La riduzione è una mossa spontanea che, prima cancella dalla pila la stringa β' (ossia gli ultimi $2n$ simboli), poi inserisce la stringa BI'', dove $I'' = \vartheta(I', B)$ è il macrostato raggiunto "leggendo" il nonterminale B.

Configurazione di riconoscimento: pila $= I_0$, ingresso $= \dashv$

Se nessuna mossa è possibile, la stringa è rifiutata.
Si noti che, grazie all'ipotesi $LR(1)$, il macrostato posto in cima e il carattere corrente selezionano univocamente la mossa: spostamento, o riduzione con una ben precisa candidata di riduzione, o rifiuto. Di conseguenza il funzionamento è deterministico.
Segue la traccia dell'analisi sintattica d'una stringa.

Esempio 4.62. Es. 4.57: traccia dell'analisi della stringa $a + a$.
Per le espressioni aritmetiche con pilota $LR(1)$ o $LALR(1)$ (p. 219) si ha:

Pila	x	*Commento*
I_0	$a + a \dashv$	sposta
$I_0\ aI_4$	$+ a \dashv$	riduci con $T \to a$
$I_0\ TI_5$	$+ a \dashv$	$+ \in$ prospezione di 4: riduci con $E \to T$
$I_0\ EI_1$	$+ a \dashv$	sposta
$I_0\ EI_1 + I_2$	$a \dashv$	sposta
$I_0\ EI_1 + I_2\ aI_4$	$\dashv$	riduci con $T \to a$
$I_0\ EI_1 + I_2\ TI_3$	$\dashv$	riduci con $E \to E + T$
I_0	$\dashv$	accetta

4.5.6 Proprietà delle sottofamiglie deterministiche e confronti

Il confronto teorico tra le varie sottofamiglie dei linguaggi deterministici è troppo articolato per essere qui discusso in modo completo. Ci si limita all'enunciazione delle proprietà più fondamentali, cominciando dalla famiglia $LR(k)$ e proseguendo poi con le famiglie dei linguaggi regolari e di quelli $LL(k)$.

Proprietà dei linguaggi e delle grammatiche $LR(k)$

Per completare il quadro teorico, si enunciano, senza dimostrazione, alcune proprietà, in parte già accennate, riguardanti le grammatiche $LR(k)$ e i loro linguaggi.[21]

[21] Per le dimostrazioni e gli approfondimenti si rimanda a [24, 46, 47].

Proprietà 4.63. La famiglia DET dei linguaggi deterministici (che si ricorda sono quelli accettati a stato finale dagli automi a pila deterministici) coincide con quella dei linguaggi generati dalle grammatiche $LR(1)$.

Ovviamente questo non implica che ogni grammatica, il cui linguaggio è deterministico, sia necessariamente $LR(1)$: infatti essa potrebbe essere ambigua o richiedere una prospezione di lunghezza $k > 1$. Ma esisterà una grammatica equivalente, che gode della proprietà $LR(1)$.
Affidandosi all'intuizione del lettore, si immagini di allungare la prospezione a valori $k > 1$ impiegando un riconoscitore $LR(k)$. Evidentemente esso non è altro che un automa a pila deterministico, il cui pilota contiene più macrostati del pilota $LR(1)$, nati dalla differenziazione dele prospezioni. Dalla precedente proprietà discende la prossima.

Proprietà 4.64. La famiglia dei linguaggi generati dalle grammatiche $LR(k)$, per ogni $k > 1$, coincide con la famiglia dei linguaggi generati dalle grammatiche $LR(1)$, dunque con la famiglia DET dei linguaggi deterministici.

Un linguaggio libero ma indeterministico, come quelli studiati a p. 163, non può quindi avere una grammatica $LR(k)$.
Se ogni linguaggio deterministico è generabile da una grammatica $LR(1)$, a che serve considerare grammatiche con un parametro k maggiore? Per rispondere si passa ora al confronto tra le grammatiche, anziché tra i linguaggi. Vi sono grammatiche LR(2) ma non LR(l), e più in generale vale l'enunciato seguente.

Proprietà 4.65. Per ogni valore di $k \geqslant 1$, esistono grammatiche che soddisfano la condizione $LR(k)$, ma non la condizione $LR(k-1)$.

Anche se la proprietà 4.64 assicura che ogni grammatica $LR(k)$, $k > 1$, può essere sostituita da una grammatica $LR(1)$ equivalente, la seconda può risultare meno naturale della prima (si vedranno tra breve degli esempi in 4.5.7).

Per ultimo si cita un risultato teorico negativo.

Proprietà 4.66. Data una grammatica, non è decidibile se esista un intero $k > 0$, per cui essa risulti $LR(k)$; di conseguenza non è decidibile se il linguaggio generato da una grammatica libera è deterministico.

Se però il valore di k è fissato, per es. a 1, si può decidere se la grammatica è $LR(k)$, costruendo il pilota e verificando che non presenti macrostati inadeguati.

Confronti tra *REG*, *LL(k)* e *LR(k)*

Può interessare il confronto tra vari modelli teorici di famiglie di linguaggi deterministici. I modelli considerati sono i linguaggi regolari REG, definiti da espressioni regolari, automi finiti o grammatiche unilineari; i linguaggi $LR(k)$, con le loro varianti ($LR(0)$ e $LALR(1)$); e i linguaggi $LL(k)$. Quanto

alle grammatiche, si fa qui l'ipotesi che esse non siano estese con espressioni regolari.
Si inizia con il confronto tra linguaggi regolari e $LL(1)$.

Proprietà 4.67. Ogni linguaggio regolare è generato da una grammatica $LL(1)$.

La dimostrazione è semplice. Si può supporre che il linguaggio regolare sia definito da un automa finito deterministico, dunque la rete contiene una sola macchina, i cui archi portano etichette terminali soltanto. La condizione LL(1) (p. 182) è soddisfatta per le seguenti ragioni: se due frecce escono da uno stato, esse portano etichette terminali diverse; se uno stato è finale, la freccia di terminazione ha come insieme guida il delimitatore $\dashv$, il quale non può etichettare nessun arco della macchina.

Il confronto tra i casi $LL(k)$ e $LR(k)$ risulta più articolato, in quanto dipende dal considerare le grammatiche o i linguaggi, nonché dal valore del parametro k.
Alcuni fatti noti possono essere così riassunti.

- Ogni linguaggio $LL(k)$ con $k \geqslant 1$, risulta deterministico, cioè $LR(1)$ per la proprietà 4.63. Infatti il parsificatore d'un linguaggio $LL(k)$ è un automa deterministico a pila.
- Vi sono linguaggi deterministici che non sono definibili con una grammatica $LL(k)$, per nessun valore di k.
  Basta ricordare gli esempi 4.39 e 4.40 (p. 196, 197).

Dai linguaggi si passa ora al confronto tra le grammatiche LR e LL, a pari valore della lunghezza di prospezione. La seguente inclusione testimonia la maggiore potenza espressiva delle grammatiche LR rispetto alle LL.

Proprietà 4.68. Per ogni valore $k \geqslant 1$, se una grammatica soddisfa la condizione $LL(k)$ essa soddisfa anche la condizione $LR(k)$.[22]

Ciò si giustifica, considerando per semplicità il caso $k = 1$. Si guardi a p. 213 la grammatica del linguaggio di Dyck, che è evidentemente $LL(1)$, e l'automa pilota $LR(1)$ ivi disegnato. Esso ha la particolarità che ogni macrostato contiene un solo stato nella sezione che sta sopra alla riga orizzontale. Ciò significa che, durante la costruzione del pilota, prima dell'operazione di chiusura $textchius_1$, ogni macrostato contiene un solo stato. Questa proprietà scende necessariamente dall'essere verificata la condizione $LL(1)$. Essa, unita alla condizione che gli insiemi guida delle alternative sono disgiunti, ha come conseguenza che non vi possono essere conflitti in nessun macrostato.

Se si prendono lunghezze di prospezione diverse, 0 e 1, si trova che le due classi di grammatiche $LR(0)$ e $LL(1)$ non sono incluse una nell'altra. Infatti si ricorda che:

[22] La dimostrazione si può trovare in [47].

1. una grammatica con regole epsilon non è $LR(0)$ ma può essere $LL(1)$;
2. una grammatica con ricorsioni a sinistra non è $LL(1)$ ma può essere $LR(0)$.

Per confrontare i linguaggi $LL(1)$ e $LR(0)$ si richiamano dei fatti già noti:

1. un linguaggio contenente delle frasi, i cui prefissi appartengono al linguaggio, non è $LR(0)$ ma può essere $LL(1)$;
2. il linguaggio $\{a^*a^nb^n \mid n \geqslant 0\}$ è $LR(0)$ ma non è $LL(1)$ (p. 196).

Di conseguenza, le famiglie dei linguaggi $LL(1)$ e $LR(0)$ sono distinte e incomparabili.
Il confronto tra le classi $LL(1)$ e $LALR(1)$ è un po' sottile:[23] basti dire che quasi tutte le grammatiche $LL(1)$, con l'eccezione di certi casi di interesse solo teorico, sono anche $LALR(1)$; in altre parole la famiglia $LALR(1)$ è in pratica più ampia della famiglia $LL(1)$.

4.5.7 Come ottenere grammatiche $LR(1)$

Le grammatiche dei linguaggi tecnici soddisfano molto spesso la condizione LR(1), ma talvolta è necessario trasformarle per poter progettare un parsificatore deterministico. Si immagini dunque di trovarsi di fronte a una grammatica non ambigua, ma che viola la condizione LR(1). I casi da considerare sono due: la grammatica è $LR(k)$ ma con prospezione maggiore di uno; la grammatica non è $LR(k)$.

Grammatica LR(2) con conflitto riduzione-riduzione per $k = 1$

Ponendosi nel primo caso, si supponga che la grammatica sia $LR(2)$ ma non $LR(1)$. La violazione per $k = 1$ si ha in particolare se in qualche macrostato I del pilota vi sono due candidate di riduzione

$$A \to \alpha\bullet, \{a\} \qquad B \to \beta\bullet, \{a\}$$

con lo stesso primo carattere di prospezione a, ma discriminate dal secondo carattere.
La modifica da farsi è detta *scansione anticipata*; essa allunga le parti destre delle regole, appendendo ad esse il carattere comune, in modo che nella grammatica modificata le due regole vengano a avere i secondi caratteri negli insiemi di prospezione.
Più precisamente l'intervento introduce due nuovi nonterminali e le regole

$$\langle Aa\rangle \to \alpha a \qquad \langle Ba\rangle \to \beta a$$

Naturalmente si devono anche aggiustare le regole contenenti A o B, per preservare l'equivalenza della grammatica.
La scansione anticipata deve garantire che esiste la derivazione

[23] Si rimanda a [6].

$$\langle Aa\rangle \stackrel{+}{\Rightarrow} \gamma a$$

se, e solo se, nella grammatica iniziale esiste la derivazione

$$A \stackrel{+}{\Longrightarrow} \gamma$$

e se il carattere a può seguire A.

Esempio 4.69. Scansione anticipata.
La grammatica G_1

$$\begin{array}{lll} S \to Abb & A \to aA & B \to aB \\ S \to Bbc & A \to a & B \to a \end{array}$$

ha il conflitto tra $A \to a \bullet \{b\}$ e $B \to a \bullet \{b\}$. Passando a $k = 2$, la prima riduzione ha la prospezione $\{bb\}$ e la seconda ha la prospezione disgiunta $\{bc\}$. La scansione anticipata produce la grammatica equivalente $LR(1)$:

$$\begin{array}{lll} S \to \langle Ab\rangle b & \langle Ab\rangle \to a\langle Ab\rangle & \langle Bb\rangle \to a\langle Bb\rangle \\ S \to \langle Bb\rangle c & \langle Ab\rangle \to ab & \langle Bb\rangle \to ab \end{array}$$

Grammatica LR(2) con conflitto riduzione-spostamento per $k = 1$

Il conflitto più immediato da curare si presenta quando in un macrostato I stanno due candidate

$$A \to \alpha \bullet a\beta, \pi \qquad\qquad B \to \gamma\bullet, \{a\}$$

conflittuali, perché dal macrostato esce l'arco etichettato a ma a appartiene all'insieme di prospezione della riduzione.
Intervento: consiste nell'introdurre un nuovo nonterminale $\langle Ba\rangle$ che equivale a B seguito da a, sostituendo la seconda regola con la $\langle Ba\rangle \to \gamma a$. Di conseguenza vanno aggiustate le regole aventi B nella parte destra, in modo da preservare l'equivalenza della grammatica.
Essendo per ipotesi la grammatica $LR(2)$, l'insieme di prospezione associato a $\langle Ba\rangle \to \gamma a$ è sicuramente disgiunto da a, e il conflitto scompare.
Il problema si complica un poco quando il carattere a di prospezione, che provoca il conflitto tra le candidate $A \to \alpha \bullet a\beta, \pi$ e $B \to \gamma\bullet, \{a\}$, proviene da una derivazione d'un nonterminale C, il quale segue immediatamente B in una forma di frase:

$$S \stackrel{+}{\Rightarrow} \ldots BC \ldots \stackrel{+}{\Rightarrow} \ldots Ba \ldots \ldots$$

Intervento: si crea un nuovo nonterminale denominato $\langle a/_S C\rangle$, che deve generare le stesse stringhe generate da C, ma decurtate del prefisso a, garantendo la condizione:

$$\langle a/_S C\rangle \stackrel{+}{\Rightarrow} \gamma \text{ se, e solo se, } C \stackrel{+}{\Rightarrow} a\gamma \text{ nella grammatica originale.}$$

Poi si aggiusta la grammatica in modo da preservare l'equivalenza.
Questa trasformazione è detta *quoziente sinistro*, e si basa sull'operazione $/_S$ definita a p. 17.
Alla grammatica così modificata, si può infine applicare la scansione anticipata, che elimina il conflitto.

Esempio 4.70. La grammatica G_2

$$\begin{array}{llll} S \to AC & A \to a & C \to c & D \to d \\ S \to BD & B \to ab\, C & \to bC & \end{array}$$

è $LR(2)$ ma non $LR(1)$ a causa del conflitto riduzione spostamento

$$A \to a \bullet \{b, c\} \qquad B \to a \bullet b\{d\}$$

Dopo la trasformazione con il quoziente sinistro, si ottiene

$$\begin{array}{lll} S \to Ab\langle b/_S C\rangle & A \to a & \langle b/_S C\rangle \to b\langle b/_S C\rangle \\ S \to Ac\langle c/_S C\rangle & B \to ab & \langle b/_S C\rangle \to c \\ S \to BD & D \to d & \langle c/_S C\rangle \to \varepsilon \end{array}$$

Ora il conflitto diventa eliminabile con la scansione anticipata.

Grammatica non $LR(k)$

Diversamente dai casi precedenti, non si possono dare delle trasformazioni sistematiche, ma la grammatica va ristrutturata, studiando i linguaggi generati da ogni nonterminale, identificando le cause dei conflitti, e modificando le corrispondenti sottogrammatiche, in modo da renderle $LR(1)$.
Una modifica talvolta efficace è la trasformazione della ricorsione da sinistra a destra. La ragione è che gli algoritmi LR sono in grado di portare avanti più calcoli non deterministici soltanto fino al momento in cui uno di essi si conclude con una riduzione, che deve essere deterministica. Una regola (più in generale una derivazione) ricorsiva a destra permette di rinviare il momento della decisione, mentre una regola ricorsiva a sinistra lo anticipa. In altre parole, l'automa nel caso destro accumula maggiori informazioni nella pila, durante il calcolo che precede la prima riduzione da compiere.[24]

Esempio 4.71. Rovesciamento della ricorsione.
Il linguaggio $a^+b^+ \cup \{a^n b^n b^* c \mid n \geqslant 1\}$ è generato dalla grammatica G_s

$$\begin{array}{llll} S \to X & S \to Yc & & \\ X \to aX & X \to aB & Y \to Yb & Y \to Z \\ B \to bB & B \to b & Z \to aZb & Z \to ab \end{array}$$

[24] Il fatto che la pila si allunga maggiormente durante l'analisi con grammatiche ricorsive a destra aumenta l'occupazione della memoria, cosa di solito trascurabile nel contesto tecnologico moderno.

La macchina pilota $LR(1)$ ha un conflitto tra una riduzione e due spostamenti:

$$Z \to ab\bullet, \{b, c\} \qquad \text{e} \qquad B \to \bullet bB, \{\dashv\} \qquad B \to \bullet b, \{\dashv\}$$

Non servirebbe incrementare k per eliminare il conflitto; ad es. con $k = 2$, la stringa bb è compatibile sia con la riduzione, sia con lo spostamento.
Riscrivendo le regole di X e di B in forma lineare a sinistra, i linguaggi generati dai due nonterminali restano invariati, ma l'automa effettua anticipatamente gli spostamenti e soltanto alla fine le riduzioni. La pila così si allunga e può mantenere aperte tutte le vie, fino al momento in cui la presenza della c o del terminatore rende palese la scelta corretta.
La grammatica equivalente G_d

$$\begin{array}{llll} S \to X & S \to Yc & & \\ X \to Xb & X \to Ab & Y \to Yb & Y \to Z \\ A \to Aa & A \to a & Z \to aZb & Z \to ab \end{array}$$

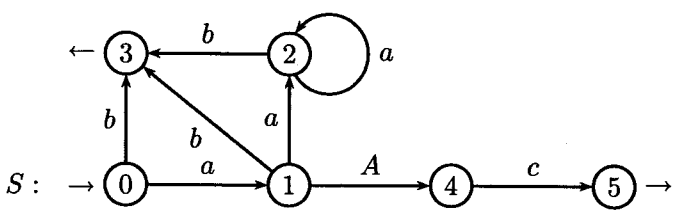
risulta LR(1).

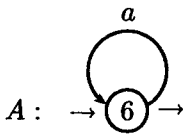
4.5.8 Analisi sintattica $LR(1)$ con grammatiche estese

Anche questi parsificatori possono operare con grammatiche BNF estese, a patto di aggiungere alcune informazioni nella pila, allo scopo di effettuare correttamente le riduzioni. Infatti per una grammatica estesa, quando la macchina pilota entra in un macrostato di riduzione, la parte destra della regola da ridurre non è sempre univocamente determinata, a causa della presenza degli iteratori e delle alternative nella grammatica. La situazione è illustrata dal prossimo esempio.

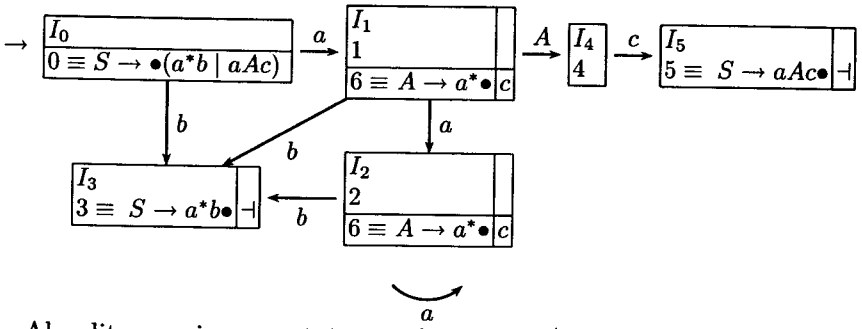
Esempio 4.72. Grammatica estesa.

$$R_1 : S \to a^*b \mid aAc$$
$$R_2 : A \to a^*$$





Si disegna la macchina pilota $LR(1)$:



Al solito, ogni macrostato contiene stati (commentati per comodità di lettura da regole marcate). Gli insiemi di prospezione, calcolati con il metodo LR(1) (basterebbe il metodo LALR(1)), sono riportati accanto alle riduzioni, nei macrostati in cui vi sono riduzioni.
Ora sia data la stringa $x_1 = aaac \dashv$. Dopo la lettura del prefisso aa, la configurazione dell'automa è:

$$\overbrace{I_0\, a\, I_1\, a\, I_2}^{\text{pila}} \mid \overbrace{ac \dashv}^{\text{suffisso}}$$

e nel macrostato I_2 si sceglie, grazie alla prospezione a, lo spostamento:

$$\overbrace{I_0\, a\, I_1\, a\, I_2\, a\, I_2}^{\text{pila}} \mid \overbrace{c \dashv}^{\text{suffisso}}$$

Nel macrostato I_2 si deve scegliere la riduzione $A \to a^*$, in accordo con la prospezione c.
Ora nasce un nuovo problema, che non esisteva per le grammatiche non estese. Quante sono le coppie di simboli da disimpilare con la mossa di riduzione?
A prima vista, osservando che la stringa letta è aa, vi sono più scelte possibili: zero (caso $A \Rightarrow \varepsilon$), una (caso a) o due (caso a^2).
Ma tale incertezza renderebbe indeterministico il funzionamento dell'automa.
Similmente l'analisi della stringa $x_2 = aaab \dashv$ porta l'automa a pila nella configurazione:

$$\overbrace{I_0\, a\, I_1\, a\, I_2\, a\, I_2\, b\, I_3}^{\text{pila}} \mid \overbrace{\dashv}^{\text{suffisso}}$$

Il macrostato corrente I_3 prescrive l'operazione di riduzione con la regola $S \to a^*b$, la quale, a causa della presenza della stella, di nuovo non dice quanti simboli siano da disimpilare.
Esaminando la pila dalla cima, la mossa di riduzione dovrebbe controllare che i simboli (terminali o nonterminali) incontrati appartengano al linguaggio definito dalla espressione regolare riflessa $(a^*b)^R$. In altre parole, la stringa di tali simboli deve essere riconosciuta dalla macchina, ottenuta dalla macchina M_A invertendo il verso delle frecce, partendo dallo stato 3, quello che corrisponde alla riduzione da eseguire.
In questo caso, il prefisso letto è *aaab*, riflesso in *baaa*, e le stringhe *b*, *ba*, *baa*, *baaa* soddisfano tale condizione. Il numero di simboli terminali disimpilabili varia da uno a quattro.
La scelta giusta è quattro, che produce la configurazione

$$\overbrace{I_0\, S}^{\text{pila}} \mid \overbrace{\dashv}^{\text{suffisso}}$$

che accetta la stringa data.

Per risolvere l'incertezza sulla lunghezza della stringa da disimpilare in fase di riduzione sono stati proposti tanti metodi diversi,[25] tra i quali si presenta ora uno dei più semplici.

Metodo di Morimoto e Sassa per il controllo delle riduzioni

L'idea è di suddividere le operazioni di spostamento del pilota in due categorie, dette di *apertura* e di *proseguimento*, a seconda che esse corrispondano a una mossa che inizia la ricerca della parte destra d'una regola, oppure a una mossa che prosegue tale ricerca.
Le operazioni di spostamento-apertura inseriscono sulla pila un'informazione aggiuntiva: l'*etichetta* della regola (ovvero il nome della macchina) che inizia la scansione della parte destra.
Invece le operazioni di spostamento-proseguimento non ne hanno bisogno e agiscono esattamente come gli spostamenti dell'algoritmo $LR(1)$ per le grammatiche non estese.
Quando si deve compiere una riduzione, le etichette presenti nella pila permetteranno di rendere deterministica la scelta del numero di simboli da disimpilare.
Passando alla realizzazione, si descrive la costruzione del pilota $LR(1)$ di Morimoto e Sassa.
Gli stati presenti in un macrostato si ripartiscono in due insiemi (al più uno dei quali può essere vuoto) il *nucleo* e il *resto*, così caratterizzati:

1. nel macrostato iniziale I_0 tutti gli stati appartengono al resto, ossia il nucleo è vuoto;

[25] Si veda la rassegna in [35].

2. se nel pilota vi è la mossa $I \xrightarrow{X} I'$, il nucleo di I' contiene gli stati q' tali che, esiste uno stato $q \in I$ per il quale, in qualche macchina della rete, vi è l'arco $\delta(q, X) = q'$:

$$nucleo(I') = \{q' \mid \exists I \text{ tale che } q \in I \wedge q' = \delta(q, X)\}$$

3. il resto d'un macrostato I (non iniziale) contiene gli stati r ottenuti mediante l'operazione di chiusura, a partire da uno stato q presente nel nucleo di I:

$$resto(I) = \{r \mid q \in Nucleo(I) \wedge r \in chius_k(q)\}$$

La chiusura (p. 201 e 212) è calcolata con la lunghezza di prospezione $k \geqslant 0$ desiderata.

Per completare la descrizione delle azioni di spostamento, che il pilota compie, passando dal macrostato I al macrostato I' alla "lettura" del simbolo (terminale o non) X, si introducono due relazioni binarie tra gli stati q di I e q' di I', scritti nella forma (I, q) e (I', q').
Una tradizionale mossa di spostamento $I \xrightarrow{X} I'$ si suddivide in due casi:

Mossa di proseguimento: siano $q \in nucleo(I)$ e $q' \in I'$, due stati per i quali esiste l'arco $\delta(q, X) = q'$;
allora nel pilota vale la relazione di proseguimento:

$$(I, q) \xrightarrow{X,\, pr} (I', q')$$

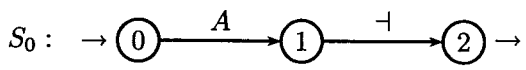
Mossa di apertura: siano $q \in resto(I)$ e $q' \in I'$, due stati per i quali esiste l'arco $\delta(q, X) = q'$;
allora nel pilota vale la relazione di apertura:

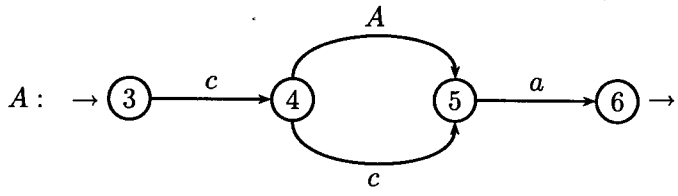
$$(I, q) \xrightarrow{X,\, ap} (I', q')$$

Si noti che è l'appartenenza dello stato d'origine al nucleo o al resto, che determina la classe della mossa, di proseguimento o di apertura.
La costruzione del pilota con le relative mosse di apertura e proseguimento è illustrata dal prossimo esempio.

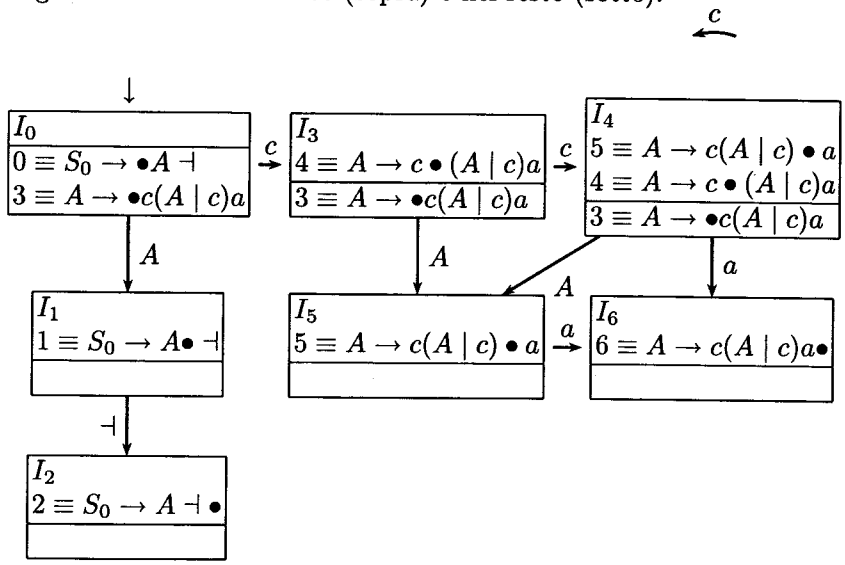
Esempio 4.73. Macchina pilota specializzata (da [35]).
La grammatica comprende due regole:

$$S_0 \to A \dashv \qquad\qquad A \to c(A \mid c)a$$

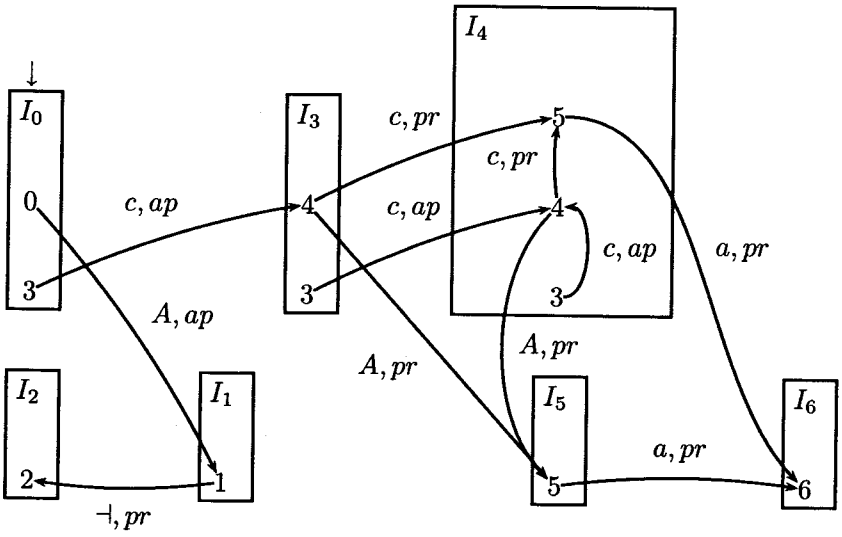




Si costruisce al solito modo la macchina pilota; la prospezione non serve poiché non vi sono conflitti nei macrostati. Nella figura, i macrostati sono divisi da una riga orizzontale nel nucleo (sopra) e nel resto (sotto).

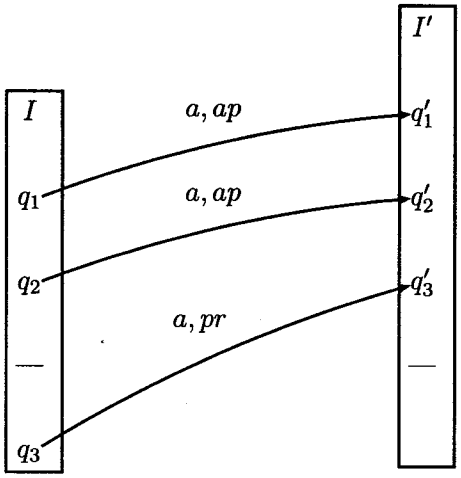


La prossima figura mostra le relazioni di apertura e proseguimento tra gli stati dei macrostati:



Ad es. l'arco da $(I_3, 3)$ a $(I_4, 4)$ è un'operazione di apertura, perché in I_3 lo stato 3 sta nel resto. Similmente, poiché è $3 \in resto(I_4)$, l'arco $(I_4, 3) \longrightarrow (I_4, 4)$ è di apertura.
Invece gli archi $(I_3, 4) \longrightarrow (I_4, 5)$ e $(I_4, 4) \longrightarrow (I_4, 5)$ sono di proseguimento poiché gli stati d'origine sono nel nucleo dei rispettivi macrostati. Anche $(I_3, 4) \longrightarrow (I_5, 5)$ è un'operazione di proseguimento poiché in I_3 lo stato 4 sta nel nucleo.
Si noti che tra i macrostati I_3 e I_4 vi sono relazioni di classi diverse, apertura e proseguimento.

Si osservi che nel pilota esiste la mossa $I \xrightarrow{X} I'$ se, e solo se, esiste almeno una relazione $(I, q) \overset{X,(pr|ap)}{\longrightarrow} (I', q')$ tra due stati rispettivamente appartenenti a I e I'.
Quando esistono due o più relazioni, esse possono essere della stessa classe o di classi diverse, come sotto schematizzato:



Quando, come in questo caso, tra due macrostati vi sono relazioni sia di proseguimento che di apertura, si dice che vi è un *conflitto d'impilamento*.
La presenza di tale tipo di conflitto nella grammatica non è di impedimento alla costruzione del parsificatore deterministico. In caso di conflitto, il parsificatore applicherà l'operazione di apertura, perché è quella che inserisce nella pila l'etichetta della macchina che potrebbe essere stata attivata.
Si sottolinea che tra due macrostati vi possono essere due o più relazioni di apertura, con la conseguenza che occorre inserire nella pila un insieme di etichette.

Algoritmo di parsificazione $ELR(1)$

Sia stata costruita la macchina pilota della grammatica, con le transizioni tra i macrostati qualificate come aperture o proseguimenti. Gli insiemi di prospezione sono calcolati come nel caso delle grammatiche non estese.
Si supponga inoltre che ogni macrostato della macchina pilota soddisfi la condizione $LR(1)$ (p. 216). Ciò significa che in ogni macrostato, il carattere corrente determina la natura della mossa, spostamento o riduzione, e nel secondo caso, determina anche quale sia lo stato finale, corrispondente al riconoscimento della parte destra della regola da applicare nella riduzione.
Resta tuttavia il fatto che la macchina, che rappresenta la regola, può contenere cammini alternativi che conducono allo stato finale prescelto.
Affinché il parsificatore sia deterministico, deve valere la seguente condizione: in ogni stato di riduzione la stringa da disimpilare dalla pila può essere univocamente determinata dal parsificatore.
Il seguente algoritmo offre una possibile soluzione.

Il parsificatore $ELR(1)$ è molto simile a quello $LR(1)$ (p. 222) da cui si distacca nelle operazioni di apertura e di riduzione. Esso è descritto da un automa a pila in cui la pila ha la forma

$$I_0\, E_0\, D_0\, I_1\, E_1\, D_1\, I_2\, \ldots\, I_{n-1}\, E_{n-1}\, D_{n-1}\, I_n$$

dove gli I sono macrostati, gli E sono insiemi di etichette delle regole, e i D sono simboli (terminali o non) della grammatica.
L'etichetta d'una regola avente il nonterminale B come parte sinistra e $q_{B,0}$ come stato iniziale della macchina M_B, sarà scritta nella forma $(M_B, q_{B,0})$.
La pila differisce da quella del caso $LR(1)$ soltanto per la presenza degli insiemi E, le etichette inserite dalle mosse di apertura. Gli insiemi E mancano invece nelle mosse di proseguimento. Conviene premettere la descrizione intuitiva dell'algoritmo.
Quando l'automa esegue la mossa d'apertura d'una regola, esso impila l'etichetta della regola, il terminale letto e il macrostato indicato dal pilota come arrivo della mossa.
Quando esegue una mossa di proseguimento, l'automa impila soltanto il simbolo letto e il macrostato d'arrivo.
Quando il macrostato corrente e il carattere di prospezione selezionano una riduzione, ossia uno stato finale q_f della macchina M_B, l'automa esegue una serie di passi, per togliere dalla pila una stringa di simboli grammaticali, $D_k, D_{k+1}, \ldots, D_{n-1}$, tale che:

- essa appartiene al linguaggio regolare $R(M_B)$, di alfabeto terminale e nonterminale, riconosciuto dalla macchina M_B;
- essa è riconosciuta nello stato finale q_f (infatti la macchina M_B potrebbe avere altri stati finali);
- nella pila tale stringa è compresa tra un punto dove sta l'etichetta $(M_B, q_{0,B})$ e la cima.

Ciò fatto, la mossa di riduzione esegue lo spostamento del nonterminale B, partendo dal macrostato affiorato sulla pila.

Algoritmo 4.74. Analizzatore sintattico $ELR(1)$.
Sia $a \in \Sigma$ il carattere corrente.
L'automa inizia nel macrostato I_0.

Mossa di *proseguimento*: si applica se nel macrostato corrente I è definita la relazione

$$(I, q) \xrightarrow{a,pr} (I', q')$$

l'automa sposta a dalla stringa sorgente alla pila e impila il macrostato I';

Mossa di *apertura*: si applica se nel macrostato corrente I sono definite una o più relazioni di apertura:

$$(I, q_1) \xrightarrow{a,ap} (I', r_1)$$
$$\ldots \xrightarrow{a,ap} \ldots$$
$$(I, q_m) \xrightarrow{a,ap} (I', r_m)$$

Si indica con E l'insieme delle etichette delle regole grammaticali corrispondenti.
L'automa impila nell'ordine:

1. l'insieme E delle etichette;
2. il simbolo a, togliendolo dal suffisso d'ingresso;
3. il macrostato I'.

Se nella configurazione corrente è definita sia una mossa di proseguimento, sia una mossa di apertura, l'automa sceglie la seconda.

Mossa di *riduzione*: Si applica se nel macrostato corrente I_n esiste una coppia $\langle q_f, \pi \rangle$, dove q_f è uno stato finale della macchina M_B, e il carattere corrente a sta nell'insieme di prospezione π.
La pila corrente sia

$$I_0\, E_0\, D_0\, I_1\, \ldots\, I_{n-k}\, \overbrace{E_{n-k}\, D_{n-k}\, I_{n-k+1}\, \ldots I_{n-1}\, E_{n-1}\, D_{n-1} I_n}^{\beta'}$$

Si disimpilano uno o più elementi (indicati come β'), ottenendo la pila

$$I_0\, E_0\, D_0\, I_1\, \ldots\, I_{n-k}$$

tale da soddisfare entrambe le seguenti condizioni:

1. E_{n-k} è l'insieme di etichette, più vicino alla cima, il quale contiene l'etichetta (M_B, q_f) della riduzione selezionata;
2. cancellando i macrostati e le etichette dalla stringa disimpilata β', la stringa ottenuta $D_{n-k+1} \ldots D_{n-1} \in (V \cup \Sigma)^*$ appartiene al linguaggio regolare accettato dalla macchina M_B con stato finale q_f.

Poi l'automa esegue la mossa spontanea che va dal macrostato I_{n-k} al macrostato I', leggendo il nonterminale B. Anche tale mossa è trattata come una mossa di apertura o di proseguimento, a seconda della relazione esistente tra i macrostati I_{n-k} e I' del pilota.

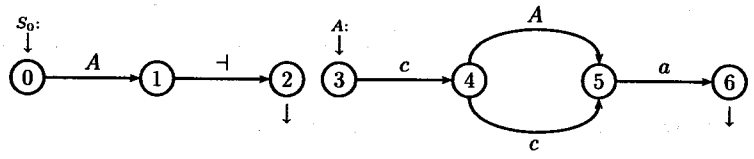
La mossa di riduzione è la più nuova e complessa. Essa può agire nel seguente modo: usando la versione specularmente riflessa $(M_B)^R$ della macchina M_B, analizza dalla cima in giù i simboli grammaticali presenti nella pila, fintanto che le due condizioni sono soddisfatte:
la macchina $(M_B)^R$ ha raggiunto lo stato $q_{B,0}$ (quello iniziale della macchina M_B); l'ultimo elemento tolto dalla pila contiene l'etichetta (M_B, q_f) della riduzione selezionata.
Esaminando il determinismo del parsificatore, si nota che, se la grammatica soddisfa la condizione $LR(1)$, la scelta tra riduzione e spostamento è deterministica. In caso di spostamento, la preferenza accordata all'operazione di

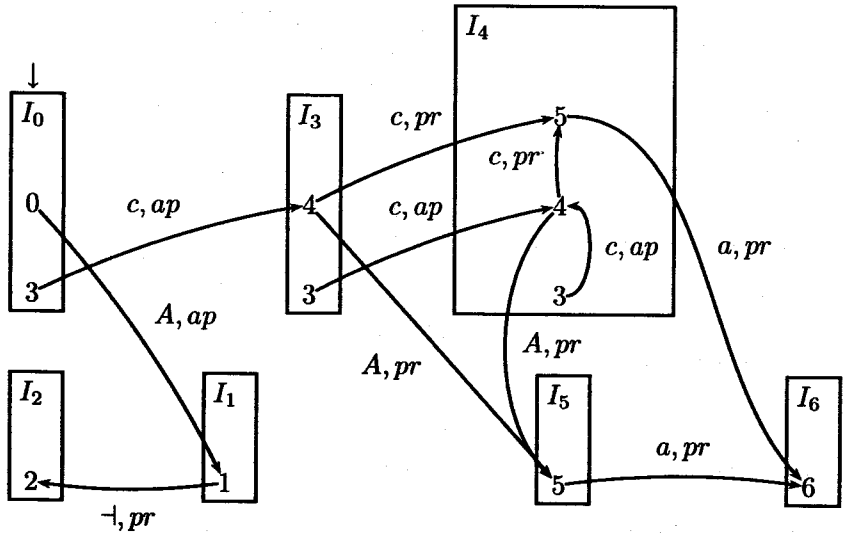
apertura, risolve il conflitto eventuale tra apertura e proseguimento. Se la mossa è di riduzione, l'insieme E_{n-k} delle etichette potrebbe contenere più regole tra cui scegliere; ma la condizione 2. della mossa di riduzione è soddisfatta da una e una sola regola, a ragione del modo in cui è stata costruita la pila.
Di conseguenza il funzionamento è deterministico.

Diversi miglioramenti sono stati proposti per accelerare le mosse di riduzione: uso di contatori o di puntatori messi nella pila in fase di apertura, costruzione d'una macchina finita per operare il riconoscimento della stringa dal fondo all'inizio, trasformazione della grammatica per eliminare le situazioni inefficienti.

Segue la traccia dell'analisi sintattica d'una stringa.

Esempio 4.75. Traccia dell'analisi (es. 4.73 continuato).
Si riproduce per comodità la rete e il pilota:

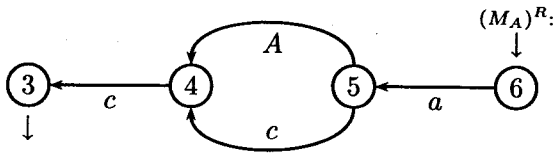




Traccia:

Pila	x	*Commento*
I_0	$c \quad c \quad c \quad a \quad a \dashv$	apertura
I_0	$(M_A,3)cI_3 \quad c \quad c \quad a \quad a \dashv$	apertura
I_0	$(M_A,3)cI_3 \quad (M_A,3)cI_4 \quad c \quad a \quad a \dashv$	apertura
I_0	$(M_A,3)cI_3 \quad (M_A,3)cI_4 \quad cI_4 \quad a \quad a \dashv$	proseguimento
I_0	$(M_A,3)cI_3 \quad \underline{(M_A,3)cI_4} \quad cI_4 \quad aI_6 \quad a \dashv$	riduzione: stato 6

Nel macrostato corrente I_6, il pilota prescrive una riduzione associata allo stato 6 della macchina M_A, corrispondente alla regola $A \rightarrow c(A \mid c)a$.
Per individuare più agevolmente la stringa da disimpilare, conviene visualizzare la macchina riflessa:



Leggendo la stringa ac compresa tra la cima della pila e la prima etichetta $(M_A,6)$, relativa alla macchina prescritta, il parsificatore svolge un calcolo con la macchina riflessa. Poiché questa raggiunge lo stato 4 non finale di $(M_A)^R$, la stringa ac non è accettabile, e occorre scavare ancora nella pila.
Il punto corretto di disimpilamento è l'etichetta sottolineata, perché la stringa acc è accettata dalla macchina riflessa. Effettuata la riduzione $cca \Rightarrow A$, ripartendo dalla nuova configurazione, l'algoritmo esegue due successive mosse di proseguimento:

$I_0 \; (M_A,3)cI_3$	$a \dashv$	proseguimento
$I_0 \; (M_A,3)cI_3 \; AI_5$	$a \dashv$	proseguimento
$I_0 \; \underline{(M_A,3)cI_3} \; AI_5 \; aI_6$	$\dashv$	riduzione: stato 6

La riduzione $cAa \Rightarrow A$, consistente con l'insieme sottolineato e con l'accettazione di aAc da parte della macchina riflessa, svuota la pila, e la stringa sorgente è accettata.

4.6 Un algoritmo generale di analisi sintattica

Per completare lo studio della parsificazione, si espone il metodo di *Earley*, che permette di trattare qualsiasi grammatica libera e costruisce tutte le derivazioni delle frasi ambigue. La sua complessità di calcolo è proporzionale al

cubo della lunghezza della stringa, ma si riduce al quadrato se la grammatica non è ambigua, e ancora più se essa è deterministica.
Questo sviluppo conclude il percorso iniziato con i metodi deterministici $LL(k)$ e proseguito con quelli $LR(k)$; lungo il percorso gli algoritmi si sono via via arricchiti della capacità di risolvere situazioni non deterministiche. Gli algoritmi $LR(k)$ portano avanti più calcoli in parallelo, ma soltanto fino al momento in cui avviene una riduzione; tale limite è insito nel modello dell'automa a pila deterministico, che non permette di gestire contemporaneamente più pile.
Il prossimo algoritmo prende le mosse da quello $LR(k)$ ma, abbandonando il modello deterministico a pila, si avvale d'una struttura dati più ricca (vettore di insiemi), che rappresenta efficientemente tante pile aventi parti condivise. Così esso permette la simulazione di un automa a pila indeterministico, senza cadere nella complessità di calcolo esponenziale (p. 150) di tale modello.
Per semplicità, le grammatiche considerate in questa parte non sono estese, ma l'algoritmo di Earley vale senza difficoltà anche per quelle estese. Inoltre, per gradualità espositiva, si impone inizialmente che la grammatica sia priva di regole vuote.
Al solito al posto della grammatica si potrà usare la rete ricorsiva di macchine.
Sia x la stringa sorgente di lunghezza $n \geqslant 1$; si denotano con x_i l'i-esimo carattere e con $x_{i..j}$ la sottostringa dei caratteri da i a j, estremi compresi; l'intera stringa sorgente è dunque $x \equiv x_{1..n}$.
Questo algoritmo non usa una macchina pilota (la quale in generale violerebbe la condizione $LR(k)$), bensì registra in un vettore $E[0..n]$, dimensionato sulla lunghezza della stringa sorgente, ogni stato in cui la rete ricorsiva potrebbe trovarsi, dopo la lettura dell' i-esimo carattere. Lo stato è affiancato da un intero (o puntatore all'indietro), che dice in quale posizione della stringa l'algoritmo ha iniziato a cercare l'istanza corrente della macchina (ossia della regola grammaticale).
Più precisamente ogni elemento o casella $E[i]$, $0 \leqslant i \leqslant n$, del vettore contiene un insieme di *coppie*

$$\langle \text{ stato, puntatore } \rangle = (s, p)$$

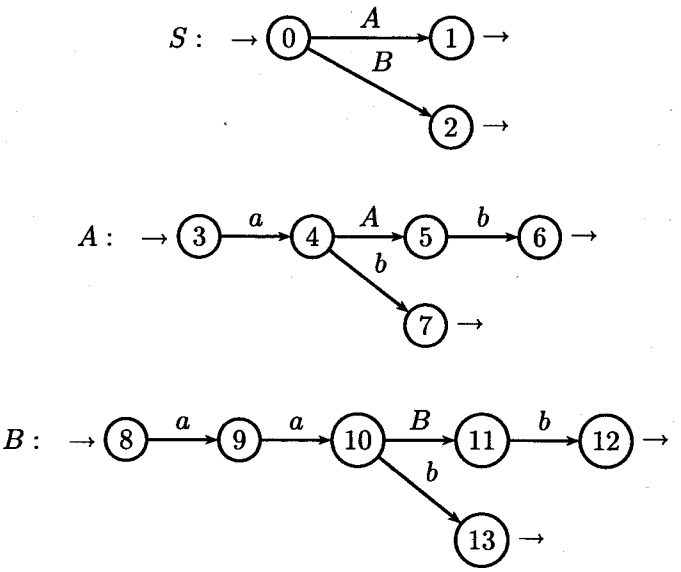
dove s è uno stato della rete e p sta nell'intervallo $(0 \ldots i)$.
Per immediatezza di lettura, accanto allo stato s, si potrà scrivere la regola sintattica marcata corrispondente.
L'algoritmo può operare anche con la prospezione, ma la sua complessità asintotica di calcolo non migliora, e conviene per semplicità presentare la versione che non fa uso della prospezione.

Esempio 4.76. Introduzione al metodo di Earley.
Il linguaggio

$$\{a^n b^n \mid n \geqslant 1\} \cup \{a^{2n} b^n \mid n \geqslant 1\}$$

non è deterministico, e quindi la condizione $LR(k)$ è violata dalla seguente grammatica:

$$S \to A \mid B \qquad A \to aAb \mid ab \qquad B \to aaBb \mid aab$$



L'analisi della stringa *aabb* costruirà un vettore $E[0] \ldots E[4]$ avente la casella iniziale $E[0]$ e una casella per ogni posizione della stringa. Ogni casella contiene un insieme di coppie.
$E[0]$ contiene inizialmente una sola coppia:

$$\langle \text{stato} = 0, \text{puntatore} = 0 \rangle$$

scritta come $(0, p = 0)$, dove il primo zero è lo stato iniziale della rete, e il puntatore punta alla posizione zero, che precede il primo carattere della stringa.

$$E[0] = \{(0 \equiv S \to \bullet(A \mid B), p = 0)\}$$

In generale l'algoritmo opera sull'insieme $E[i]$ nel modo seguente: esamina in ordine le coppie ivi presenti, eseguendo una di tre operazioni, scansione, predizione, completamento, su ciascuna coppia, a seconda della forma di essa. La *predizione* è soltanto un nuovo nome per l'operazione di chiusura $LR(0)$ (p. 201); si applica a una coppia il cui stato abbia un arco uscente con etichetta nonterminale (ossia nella regola marcata vi è un nonterminale a destra del

pallino). Essa aggiunge all'insieme $E[i]$ una nuova coppia con lo stato iniziale della macchina (ossia con il pallino all'inizio della parte destra della regola marcata).
Alla seconda componente della coppia, il puntatore, si assegna il valore i, poiché la coppia è stata creata al passo i, partendo da una coppia presente in $E[i]$.
Il senso dell'operazione è che la predizione aggiunge a $E[i]$ tutti gli stati iniziali delle macchine che potrebbero riconoscere una sottostringa che inizia da x_{i+1}.
Nell'esempio la predizione aggiunge a $E[0]$ le coppie:

$$(3 \equiv A \to \bullet aAb \mid \bullet ab, p = 0), \qquad (8 \equiv B \to \bullet aaBb \mid \bullet aab, p = 0)$$

Poiché la predizione non è applicabile alle nuove coppie, si passa ora alla seconda operazione, la *scansione*, che è applicabile quando da uno stato esce un arco con etichetta terminale. Se tale terminale eguaglia x_{i+1}, lo stato di arrivo è aggiunto all'insieme $E[i+1]$ con lo stesso puntatore dello stato di partenza. Ciò sta a indicare che il terminale è stato letto.
Nell'esempio, con $i = 0$ e $x_1 = a$, la scansione inserisce in $E[1]$ le coppie:

$$(4 \equiv A \to a \bullet Ab \mid a \bullet b, p = 0), \qquad (9 \equiv B \to a \bullet aBb \mid a \bullet ab, p = 0)$$

Applicando la predizione alle nuove coppie, si aggiunge a $E[1]$ la coppia

$$(3 \equiv A \to \bullet aAb \mid \bullet ab, p = 1)$$

dove è da notare il valore 1 del puntatore.
Se al termine dell'elaborazione di $E[i]$ l'insieme $E[i+1]$ restasse vuoto, si sarebbe scoperto un errore nella stringa sorgente.
Ora si applica a $E[1]$ la scansione di $x_2 = a$, che produce in $E[2]$ le coppie

$$(10 \equiv B \to aa \bullet Bb \mid aa \bullet b, p = 0), \qquad (4 \equiv A \to a \bullet Ab \mid a \bullet b, p = 1)$$

alle quali la predizione aggiunge le coppie

$$(8 \equiv B \to \bullet aaBb \mid \bullet aab, p = 2), \qquad (3 \equiv A \to \bullet aAb \mid \bullet ab, p = 2)$$

La scansione di $x_3 = b$ produce in $E[3]$ le coppie

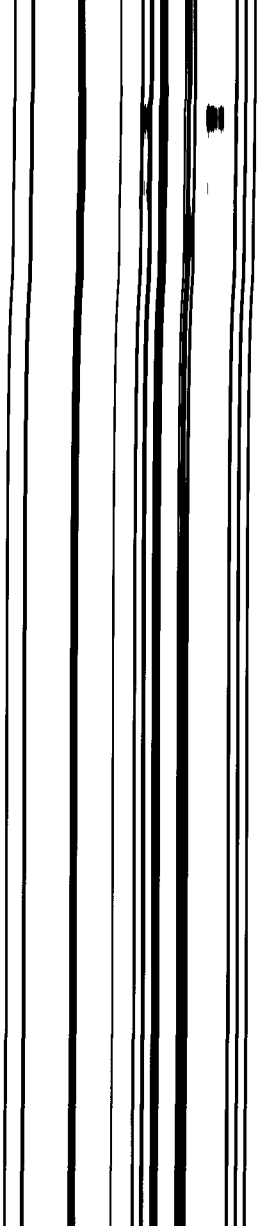
$$(13 \equiv B \to aab\bullet, p = 0), \qquad (7 \equiv A \to ab\bullet, p = 1)$$

Ora entra in gioco la terza operazione, il *completamento* (che ha un effetto analogo alla riduzione di $LR(k)$). Esso si applica a una coppia $(s, j) \in E[i]$ il cui stato s è finale per una macchina M_A, ossia corrisponde a una regola $s \equiv A \to \ldots \bullet$, con marca in fondo.
L'operazione ritorna sull'insieme $E[j]$ puntato da j (è necessariamente $j < i$ non essendovi regole vuote nella grammatica). Ricerca in $E[j]$ una coppia (q, k), tale che da q esca l'arco $q \xrightarrow{A} r$. Infine aggiunge all'insieme corrente

$E[i]$ lo stato r d'arrivo, con il valore k del puntatore. Ossia aggiunge la regola marcata ottenuta, da quella trovata in $E[j]$, spostando il pallino a destra di A.

Il completamento si applica a entrambe le coppie di $E[3]$, poiché 13 e 7 sono stati finali, rispettivamente di B e di A. Il puntatore di 13 rimanda a $E[0]$, che intuitivamente è l'insieme in cui l'analisi si trovava quando fu iniziata la ricerca di questa istanza di B.

In $E[0]$ si trova la coppia $(0 \equiv S \rightarrow \bullet A \mid \bullet B, p = 0)$, dal cui stato origina l'arco $0 \xrightarrow{B} 2$; si inserisce in $E[3]$ lo stato di arrivo 2, con lo stesso puntatore:



Descrizione precisa dell'algoritmo di Earley

L'algoritmo è in effetti un parsificatore che sviluppa simultaneamente tutte le possibili derivazioni sinistre della stringa. L'algoritmo legge la stringa $x_{1\ldots n}$ da sinistra a destra e, quando esamina il carattere x_i, produce certe strutture a due campi o coppie, della forma ⟨stato della rete, puntatore⟩, scritta come $(s, p = \ldots)$, dove il puntatore p è compreso tra 0 e i. Lo stato può essere scritto anche come *regola grammaticale marcata* $A \to \alpha \bullet \beta$.
Intuitivamente, la coppia $(s \equiv A \to \alpha \bullet \beta,\ j)$ rappresenta un'asserzione e un obiettivo:

Asserzione: è stata trovata una sottostringa $x_{j+1..i}$ $(0 \leqslant j < i)$ che deriva da α, in formula

$$\alpha \stackrel{*}{\Rightarrow} x_{j+1..i}$$

Obiettivo: trovare tutte le posizioni k $(i < k \leqslant n)$ tali che la sottostringa $x_{i+1..k}$ derivi da β, in formula

$$\beta \stackrel{*}{\Rightarrow} x_{i+1..k}$$

Se l'algoritmo troverà tale posizione k, potrà asserire che dal nonterminale A deriva la sottostringa $x_{j+1..k}$, ossia

$$A \stackrel{*}{\Rightarrow} x_{j+1..k}$$

Una coppia $(q \equiv A \to \alpha\bullet, j)$, il cui stato q è finale (ossia ha la marca in fondo alla regola), è detta *completata.*

Algoritmo 4.77. Riconoscitore di Earley.
L'algoritmo costruisce un vettore $E[0..n]$ dimensionato sulla lunghezza della stringa sorgente, le cui caselle contengono degli insiemi di coppie. $E[0]$ è l'insieme iniziale e $E[i]$ quello associato alla posizione x_i della stringa.
Pur se il compito del riconoscitore è trovare la derivazione dell'intera stringa x, l'algoritmo produce più di quanto richiesto, in quanto dice anche se ogni prefisso della stringa appartiene al linguaggio.
L'insieme iniziale è riempito con certe coppie ricavate dall'assioma; tutti gli altri insiemi sono inizialmente vuoti.

Passo 0: Inizializzazione. (Predispone gli obiettivi per trovare ogni prefisso di x derivabile dall'assioma S. Al puntatore è assegnato il valore zero.)

$E[0]$ $:= (q_{ini},\ 0)$, dove q_{ini} è lo stato iniziale della rete.
$E[i]$ $:= \emptyset$, per $i = 1, \ldots, n$
$i := 0$

Poi si applicano nell'ordine naturale $0, 1, \ldots, n$ le operazioni di predizione, completamento e scansione descritte nel seguito, per calcolare tutti gli insiemi

$E[i]$. L'algoritmo al passo i può aggiungere coppie soltanto all'insieme corrente $E[i]$ e a quello successivo $E[i+1]$. Se nessuna delle operazioni ha aggiunto nuove coppie a $E[i]$, si passa al calcolo di $E[i+1]$. Se $E[i+1]$ è vuoto e $i < n$ la stringa è rifiutata.

Predizione. (Ogni obiettivo presente nell'insieme $E[i]$ può aggiungere altri sotto obiettivi all'insieme stesso; al puntatore è assegnato l'indice corrente.)

per ogni coppia in $E[i]$ della forma $(q \equiv A \to \alpha \bullet B\gamma,\ j)$,
dove da q esce l'arco $q \xrightarrow{B} s$,
aggiungi all'insieme $E[i]$ la coppia (r, i),
dove r è lo stato iniziale della macchina M_B

Completamento. (Una coppia completata $(q \equiv A \to \alpha\bullet,\ j)$ asserisce che è stata trovata la derivazione della stringa $x_{j+1..i}$ dal nonterminale A. Occorre aggiornare l'insieme $E[i]$ con tale asserzione.)

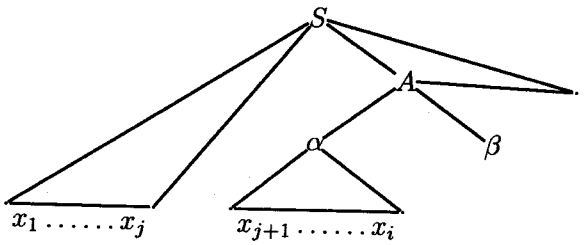
per ogni coppia completata $(q \equiv A \to \alpha\bullet, j)$ in $E[i]$,
per ogni coppia in $E[j]$ della forma $(r \equiv B \to \beta \bullet A\gamma, k)$,
tale che da r esce l'arco $r \xrightarrow{A} s$,
aggiungi a $E[i]$ la coppia $(s \equiv B \to \beta A \bullet \gamma, k)$

Scansione. (Aggiorna gli obiettivi dell'insieme $E[i+1]$ in accordo con il carattere corrente. Il puntatore è posto eguale a quello della coppia esaminata.)

per ogni coppia in $E[i]$ della forma $(q \equiv A \to \alpha \bullet a\gamma, j)$, se $a = x_{i+1}$:
aggiungi all'insieme $E[i+1]$ la coppia $(r \equiv A \to \alpha a \bullet \gamma, j)$
dove r è l'arrivo dell'arco $q \xrightarrow{a} r$

L'algoritmo termina quando la costruzione dell'insieme $E[n]$ è stata finita; termina prematuramente con insuccesso, se la scansione non ha trovato una coppia da aggiungere a $E[i+1], i < n$.
Se l'insieme finale $E[n]$ contiene (almeno) una coppia completata $(q_{term} \equiv S \to \alpha\bullet, 0)$, dove q_{term} è lo stato finale della macchina dell'assioma, la stringa sorgente è accettata.

È significativo ripensare le tre operazioni come costruzione progressiva di alberi sintattici. Allora a ogni insieme corrisponde un insieme di alberi. In particolare in $E[i]$ vi è la coppia $(A \to \alpha\bullet\beta, p = j)$ se, e solo se, per la grammatica data, esiste un albero sintattico della forma:



Al termine dell'algoritmo, la stringa è accettata se, e solo se, tra gli alberi associati all'ultimo insieme vi è un albero sintattico completo con radice nell'assioma.

Note:

Ciascun carattere della stringa è esaminato dall'algoritmo una sola volta nella scansione.

La predizione esamina ripetutamente le coppie dell'insieme corrente, il quale può crescere. Rappresendando l'insieme in una coda FIFO, è facile evitare di esaminare più volte la stessa coppia.

Nel completamento: se nella coppia $(r \equiv B \to \beta \bullet A\gamma, k) \in E[j]$ la stringa γ è vuota, la nuova coppia $(s \equiv B \to \beta A \bullet \gamma, k)$ aggiunta a $E[i]$ è completata, quindi occorre iterare di nuovo.

Sarebbe facile dimostrare che l'algoritmo accetta una stringa soltanto se appartenente al linguaggio $L(G)$: infatti una coppia viene aggiunta all'insieme, soltanto se la derivazione da essa asserita è possibile; d'altra parte, la dimostrazione che ogni frase del linguaggio è riconosciuta dall'algoritmo presenta qualche difficoltà ed è omessa.[26]

Costruzione dell'albero sintattico

Si è finora considerato il riconoscimento d'una stringa, ma l'interpretazione delle coppie come alberi permette di costruire l'albero sintattico della frase.

Esempio 4.78. Costruzione dell'albero per l'es. 4.76 (p. 240).
Per comodità di lettura, sono riprodotti gli insiemi di coppie per la stringa *aabb*. A destra si vedono tre tappe nella costruzione dell'albero:

[26] Si veda l'articolo originale di Earley [17] o un testo quale [19, 41, 23].

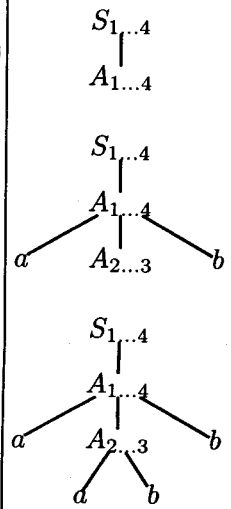
$E[0]$ $(S \to \bullet(A \mid B), p = 0)$ $(A \to \bullet aAb \mid \bullet ab, p = 0)$ $(B \to \bullet aaBb \mid \bullet aab, p = 0)$

a $E[1]$ $(A \to a \bullet Ab \mid a \bullet b, p = 0)$ $(B \to a \bullet aBb \mid a \bullet ab, p = 0)$ $(A \to \bullet aAb \mid \bullet ab, p = 1)$

a $E[2]$ $(B \to aa \bullet Bb \mid aa \bullet b, p = 0)$ $(A \to a \bullet Ab \mid a \bullet b, p = 1)$ $(B \to \bullet aaBb \mid \bullet aab, p = 2)$ $(A \to \bullet aAb \mid \bullet ab, p = 2)$

b $E[3]$ $(B \to aab \bullet, p = 0)$ $(A \to ab \bullet, p = 1)$ $(S \to B \bullet, p = 0)$ $(A \to aA \bullet b, p = 0)$

b $E[4]$ $(A \to aAb \bullet, p = 0)$ $(S \to A \bullet, p = 0)$



Si inizia dall'ultima casella. Poiché essa contiene la coppia completata $(S \to A\bullet, p = 0)$, deve esistere la derivazione $S \Rightarrow A \overset{*}{\Rightarrow} x_{1...4}$, che è rappresentata dall'albero superiore.
Per l'ultimo indice 4, si esamina l'insieme $E[4]$, alla ricerca di una coppia completata del nonterminale A; si trova $(A \to aAb\bullet, p = 0)$, che fa disegnare l'albero di mezzo.
Il sottoalbero centrale $A_{2..3}$ da espandere copre la stringa tra le posizioni 2 e 3; la seconda di esse porta a esaminare l'insieme $E[3]$. Al fine di sviluppare l'albero innestato sotto $A_{2...3}$, si cerca dunque in $E[3]$ una coppia completata di A, che punti alla casella precedente 2, ossia con puntatore eguale a $(2-1) = 1$; si trova $(A \to ab\bullet, p = 1)$. L'albero così cresce e si completa nella figura inferiore.

Analisi d'una frase ambigua

Se la grammatica è ambigua, l'analisi d'una frase produce tutti gli alberi possibili, rappresentati in modo fattorizzato, come si vedrà nel prossimo esempio.

Esempio 4.79. Parsificazione d'un linguaggio ambiguo
La grammatica, ricorsiva bilateralmente quindi ambigua, è

$$S \to E \qquad E \to E + E \qquad E \to a$$

Gli insiemi per la stringa sorgente $a + a + a$ sono sotto mostrati; una linea orizzontale separa le coppie create da successive iterazioni del completamento.

$$
\begin{array}{cc}
 & E[0] \\
 & \hline
 & S \to \bullet E, 0 \\
 & E \to \bullet E + E \mid \bullet a, 0
\end{array}
$$

$$
\begin{array}{cc}
a & E[1] \\
 & \hline
 & E \to a\bullet, 0 \\
 & \hline
 & S \to E\bullet, 0 \\
 & E \to E \bullet + E, 0
\end{array}
$$

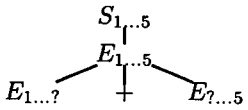
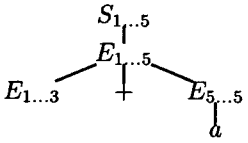
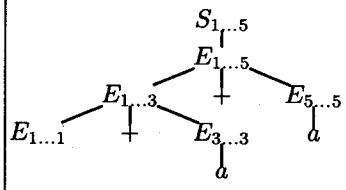
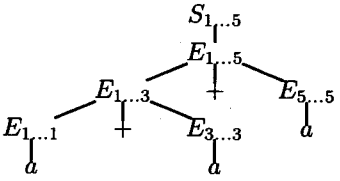
$$
\begin{array}{cc}
+ & E[2] \\
 & \hline
 & E \to E + \bullet E, 0 \\
 & E \to \bullet E + E \mid \bullet a, 2
\end{array}
$$

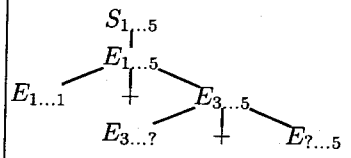
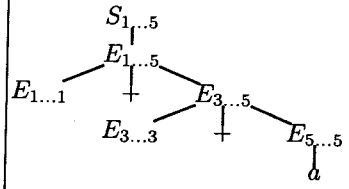
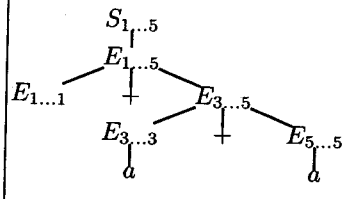
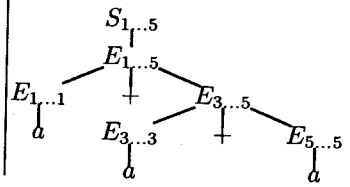
$$
\begin{array}{cc}
a & E[3] \\
 & \hline
 & E \to a\bullet, 2 \\
 & \hline
 & E \to E + E\bullet, 0 \\
 & E \to E \bullet + E, 2 \\
 & \hline
 & S \to E\bullet, 0 \\
 & E \to E \bullet + E, 0
\end{array}
$$

$$
\begin{array}{cc}
+ & E[4] \\
 & \hline
 & E \to E + \bullet E, 0 \\
 & E \to E + \bullet E, 2 \\
 & E \to \bullet E + E \mid \bullet a, 4
\end{array}
$$

$$
\begin{array}{cc}
a & E[5] \\
 & \hline
 & E \to a\bullet, 4 \\
 & \hline
 & E \to E + E\bullet, 2 \\
 & E \to E \bullet + E, 4 \\
 & E \to E + E\bullet, 0 \\
 & \hline
 & E \to E \bullet + E, 2 \\
 & S \to E\bullet, 0 \\
 & E \to E \bullet + E, 0
\end{array}
$$

Si noti la ripetizione d'una stessa coppia in insiemi diversi. Nell'ultimo insieme, $E[5]$, la coppia completa $(S \to E\bullet, 0)$ manifesta la validità della stringa. Si mostra la costruzione dei due alberi della frase considerata.

Coppie	*Primo albero*
$E[5]$ $S \rightarrow E\bullet, 0$ $E \rightarrow E + E\bullet, 0$	
$E \rightarrow a\bullet, 4$	
$E[3]$ $E \rightarrow E + E\bullet, 0$ $E \rightarrow a\bullet, 2$	
$E[1]$ $E \rightarrow a\bullet, 0$	

Coppie	*Secondo albero*
$E[5]$ $S \to E\bullet, 0$ $E \to E + E\bullet, 0$ $E \to E + E\bullet, 2$	
$E \to a\bullet, 4$	
$E[3]$ $E \to a\bullet, 2$	
$E[1]$ $E \to a\bullet, 0$	

Complessità di calcolo del riconoscitore

Non è difficile calcolare la complessità asintotica di calcolo dell'algoritmo di Earley, nel caso peggiore.
Per una stringa di lunghezza n, si conteggiano le coppie e le operazioni eseguite su di esse.

1. Ciascun insieme $E[i]$ può contenere un numero di coppie che cresce linearmente con i, dunque è maggiorato da n.
2. Le operazioni di scansione e predizione eseguono, su ogni coppia dell'insieme $E[i]$, un numero di passi indipendente da n.
3. L'operazione di completamento esegue $\mathcal{O}(i)$ passi per ogni coppia trattata, perché potrebbe dover aggiungere a $E[i]$ un numero di coppie dell'ordine di $\mathcal{O}(j)$, dove $E[j]$, $0 \leqslant j < i$ è un insieme precedente. In complesso il completamento richiede $\mathcal{O}(n^2)$ passi.

4. Sommando i passi eseguiti per ogni i da 0 a n, si ottiene il limite $\mathcal{O}(n^3)$.

Proprietà 4.80. La complessità asintotica dell'algoritmo di Earley nel caso peggiore è $\mathcal{O}(n^3)$, dove n è la lunghezza della stringa sorgente.

In pratica l'algoritmo è più veloce: per molte grammatiche è $\mathcal{O}(n)$ e per ogni grammatica inambigua è $\mathcal{O}(n^2)$.
L'aggiunta delle operazioni sui puntatori, necessarie per costruire gli alberi sintattici, non modifica la classe di complessità asintotica di calcolo del riconoscitore.

Trattamento delle regole vuote

Nella presentazione dell'algoritmo di Earley sono state finora evitate le grammatiche con regole vuote, ma anch'esse possono essere ammesse, al costo di alcune modifiche.
Sia $E[i]$ l'insieme corrente. L'algoritmo (p. 245) sta costruendo due insiemi di coppie: l'insieme $E[i]$ può ricevere coppie dai passi di *predizione* e *completamento*, mentre l'insieme $E[i+1]$ riceve coppie a causa della *scansione*. Il completamento richiede un ripensamento, a causa delle ε-regole.
Quando in $E[i]$ il completamento esamina una coppia completata

$$(q \equiv A \rightarrow \varepsilon\bullet,\ j)$$

dove lo stato q della macchina M_A è iniziale e finale, deve cercare nell'insieme $E[j]$ le coppie aventi la marca prima della A. Ma, per le ε-regole, il puntatore j è sempre eguale a i: infatti tali coppie possono essere soltanto aggiunte dalla *predizione*, la quale assegna al puntatore il valore corrente dell'indice.
Al fine di trattare correttamente questi casi, il *completamento* deve esaminare l'insieme $E[i]$ parzialmente costruito, poi invocare la *predizione*, la quale deve riattivare il *completamento*, e così via, terminando quando nessuna delle due operazioni ha più nulla da aggiungere all'insieme. Questo modo di procedere è corretto ma inefficiente.
Migliore è il *metodo di Aycock e Horspool*[27], di cui si indica (in grassetto) la modifica, concernente la sola operazione di predizione. Si ricorda che un nonterminale A è annullabile (p.60) se da esso può derivare in uno o più passi la stringa vuota.

Predizione con ϵ-regole
per ogni coppia in $E[i]$ della forma $(q \equiv A \rightarrow \alpha \bullet B\gamma, j)$,
dove da q esce l'arco $q \xrightarrow{B} s$,
aggiungi all'insieme $E[i]$ la coppia $(r \equiv B \rightarrow \bullet\delta, i)$,
dove r è lo stato iniziale della macchina M_B.
Se B è annullabile, aggiungi a $E[i]$ anche le coppie
$(s \equiv A \rightarrow \alpha B \bullet \gamma, j)$

[27] Vedasi [5].

Detto diversamente, l'azione di predizione sposta il pallino dalla sinistra alla destra di un nonterminale, se da esso può derivare la stringa vuota, in accordo con il fatto che la derivazione farebbe sparire il nonterminale stesso.

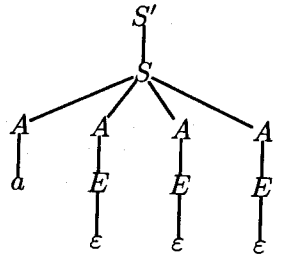
Esempio 4.81. Nella grammatica

$$S' \to S \quad S \to AAAA \quad A \to a \quad A \to E \quad E \to \varepsilon$$

tutti i nonterminali sono annullabili.
Si mostra la traccia del riconoscimento della stringa a e due tra le possibili derivazioni:

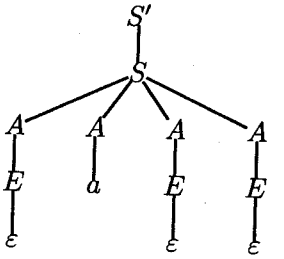
$E[0]$

$S' \to \bullet S, 0$
$S \to \bullet AAAA, 0$
$S' \to S\bullet, 0$
$A \to \bullet a, 0$
$A \to \bullet E, 0$
$S \to A \bullet AAA, 0$
$E \to \bullet, 0$
$A \to E\bullet, 0$
$S \to AA \bullet AA, 0$
$S \to AAA \bullet A, 0$
$S \to AAAA\bullet, 0$



$a\ E[1]$

$A \to a\bullet, 0$
$S \to A \bullet AAA, 0$
$S \to AA \bullet AA, 0$
$S \to AAA \bullet A, 0$
$S \to AAAA\bullet, 0$
$A \to \bullet a, 1$
$A \to \bullet E, 1$
$S' \to S\bullet, 0$
$E \to \bullet, 1$
$A \to E\bullet, 1$



Ulteriori sviluppi del metodo di Earley

Si è già detto che le coppie presenti negli insiemi possono essere arricchite con la prospezione, calcolata esattamente come nel metodo $LR(1)$. Affiancando a ogni coppia un insieme di prospezione l'algoritmo evita di inserire negli insiemi certe coppie corrispondenti a scelte destinate al fallimento. A prima vista questo può apparire come un perfezionamento da cui ci si potrebbe attendere migliore efficienza, quanto meno per le grammatiche che soddisfano

la condizione $LR(1)$; ma a un esame più approfondito si trova che il numero di coppie da elaborare per molte grammatiche può aumentare anziché diminuire, a causa della differenzazione portata dai caratteri di prospezione. In definitiva, i vantaggi della prospezione sono controversi, e le implementazioni più efficienti del metodo di Earley non ne fanno uso.
Con lievi ritocchi, l'algoritmo di Earley continua a funzionare anche per le grammatiche con regole BNF estese. [28]

4.7 Scelta del parsificatore

Per molti linguaggi tecnici esistono compilatori realizzati con entrambi i metodi deterministici $LL(1)$ e $LR(1)$ (spesso nella variante $LALR(1)$); ciò attesta che nella maggior parte dei casi la scelta del metodo non è critica.
Se la grammatica di riferimento non soddisfa né le condizioni LL né quelle LR, e non si desidera modificarla, è possibile usare una parsificatore di tipo generale, come quello di Earley, certamente più lento e ingombrante d'uno deterministico.
Esistono poi altri approcci, di complessità intermedia tra quelli deterministici e quelli validi per ogni grammatica libera. Partendo dagli algoritmi $LL(k)$, vi sono i già ricordati parsificatori che, per decidere la mossa, usano una prospezione di lunghezza illimitata.
Per l'analisi ascendente, un algoritmo abbastanza diffuso in diverse varianti è quello di Tomita[29] che tratta le grammatiche in cui il numero di scelte non deterministiche è limitato.
Una strategia di parsificazione molto diversa è talvolta applicata quando la grammatica è fortemente ambigua, per evitare di costruire tutti i numerosi alberi sintattici. Infatti molti di essi non corrispondono a interpretazioni sensate del testo, e possono essere eliminati, già in fase di parsificazione, applicando dei controlli semantici. Si parla allora di parsificazione guidata dalla semantica, un concetto che sarà esposto nel prossimo capitolo.
Ritornando al più comune caso deterministico, alcune considerazioni possono guidare il progettista nella scelta del metodo di analisi, $LL(k)$ o $LR(1)$.
Dal punto di vista della velocità di esecuzione, le differenze di prestazioni tra gli analizzatori dei due tipi sono trascurabili.
La famiglia dei linguaggi $LR(1)$ (anche nella variante $LALR(1)$) è più ampia della famiglia $LL(1)$. Di fatto per molti linguaggi tecnici la grammatica di riferimento non è $LL(1)$, vuoi per la presenza di ricorsioni sinistre, vuoi per la sovrapposizione tra gli insiemi guida delle alternative. Di conseguenza, per attuare l'analisi discendente, il progettista deve trasformare la grammatica. Di solito le semplici trasformazioni studiate (come la fattorizzazione sinistra p. 4.35) permettono d'ottenere una grammatica equivalente $LL(k), k \geqslant 1$, perché

[28] Il modo di fare ciò è delineato da Earley [17].
[29] Si veda [50].

è raro che il linguaggio sorgente non sia $LL(k)$. Ma il risultato è abbastanza diverso dalla grammatica di riferimento, spesso è meno leggibile, e comporta l'onere del mantenimento di due versioni della grammatica, se il linguaggio è soggetto a cambiamenti.

Per costruire un analizzatore ascendente è indispensabile l'impiego d'uno strumento generatore (ad es. yacc o bison), mentre gli analizzatori a discesa ricorsiva possono anche essere progettati a mano, visto che il loro codice è sostanzialmente allineato alla struttura delle regole. Il codice dei compilatori a discesa ricorsiva è più facilmente comprensibile, anche da parte di tecnici non specializzati nel progetto dei compilatori.

Anche per il caso LL vi sono strumenti per il calcolo degli insiemi guida e la generazione delle procedure a discesa ricorsiva, che agevolano il lavoro del progettista. Di solito tali strumenti trattano anche le grammatiche BNF estese con espressioni regolari.

Al contrario gli strumenti LR o $LALR$ più comuni non accettano le grammatiche BNF estese, e impongono al progettista una fastidiosa, anche se ovvia conversione, della grammatica originale.

Un analizzatore non è mai un programma isolato, ma deve interfacciarsi con altri algoritmi di traduzione guidata dalla sintassi (trattati nel prossimo capitolo). Anticipando l'esposizione, una proprietà teorica, relativa alle traduzioni calcolabili durante l'analisi sintattica, rende gli analizzatori discendenti un po' più potenti degli altri; in breve, certe traduzioni dal linguaggio sorgente al linguaggio pozzo possono essere svolte aggiungendo delle azioni semantiche nel corpo delle procedure a discesa ricorsiva; mentre ciò non è sempre possibile nell'analisi ascendente. Ciò favorisce la realizzazione di compilatori discendenti capaci di operare in una sola passata.

Una capacità talvolta richiesta a un parsificatore (più in generale a un compilatore) è l'incrementalità, un concetto che si manifesta in due specie assai diverse.

Incrementalità della grammatica. Talvolta, in circostanze molto particolari, si deve modificare di continuo la grammatica del linguaggio sorgente, e di conseguenza si deve aggiornare il corrispondente parsificatore. Una situazione del genere si presenta nei cosiddetti linguaggi estensibili, che offrono all'utente la libertà di inventare nuove istruzioni. Diventa allora indispensabile rigenerare automaticamente il parsificatore, dopo ogni modifica della sintassi.[30]

Parsificazione incrementale. Più comune è l'esigenza di ricalcolare rapidamente l'albero sintattico, dopo una modifica del testo sorgente.

Un buon ambiente di lavoro per la scrittura di programmi o di documenti strutturati dovrebbe essere interattivo, e permettere la modifica del testo sorgente, senza imporre attese per ottenere la traduzione (compilazione) del testo modificato. Per ridurre il tempo di ritraduzione dopo una modifica, sono stati studiati gli algoritmi di compilazione incrementale. Poiché la compilazione richiede l'analisi sintattica e poi quella semantica, per entrambi i passi sono

[30] Per questo problema si veda [25].

stati studiati e utilizzati dei metodi incrementali.
Si supponga che il parsificatore abbia analizzato un testo, accettandolo o diagnosticando degli errori; l'autore lo abbia poi corretto, producendo un nuovo testo, che differisce in pochi punti. Il parsificatore è detto *incrementale* se, per analizzare il nuovo testo, impiega un tempo inferiore a quello che spenderebbe per analizzarlo la prima volta. Affinché ciò sia possibile, l'algoritmo deve conservare il risultato dell'analisi precedente, e modificare l'albero soltanto dove necessario. L'algoritmo incrementale mantiene un'opportuna rappresentazione delle configurazioni attraversate dall'automa a pila, organizzata in modo che, dopo una modifica del testo, sia facile individuare e aggiornare la parte che risente del cambiamento.[31]

[31] Sull'analisi sintattica incrementale si veda [21, 31].