Esercizi di Linguaggi e Traduttori

Stefano Paraboschi, Pierluigi San Pietro
Dipartimento di Elettronica e Informazione
Politecnico di Milano

Contents

1 Definizione di grammatiche **2**
1.1 Esercizio 2
1.2 Esercizio 3
1.3 Esercizio 4
1.4 Esercizio 5
1.5 Esercizio 6
1.6 Esercizio 9
1.7 Esercizio 9
1.8 Esercizio 11
1.9 Esercizio 11
1.10 Esercizi proposti 12

2 Automi, linguaggi, espressioni regolari **13**
2.1 Esercizio 13
2.2 Esercizio 14
2.3 Esercizio 14
2.4 Esercizio 17
2.5 Esercizio 19
2.6 Esercizio 21
2.7 Esercizio 24
2.8 Esercizi proposti 24

3 Eliminazione ambiguità **25**
3.1 Esercizio 25
3.2 Esercizio 27
3.3 Esercizio 28
3.4 Esercizio 29
3.5 Esercizio 33
3.6 Esercizi proposti 33

4 Analisi discendente deterministica **34**
4.1 Esercizio 34
4.2 Esercizio 37
4.3 Esercizio 39
4.4 Esercizio 43
4.5 Esercizio 45
4.6 Esercizio 46
4.7 Esercizi proposti 49

5 Analisi sintattica ascendente **50**
5.1 Esercizio 50
5.2 Esercizio 55
5.3 Esercizio 59
5.4 Esercizio 62
5.5 Esercizi proposti 64

6 Traduzioni sintattiche **65**
6.1 Esercizio 65
6.2 Esercizio 67
6.3 Esercizio 70
6.4 Esercizio 71
6.5 Esercizi proposti 72

7 Grammatiche ad attributi **72**
7.1 Esercizio 72
7.2 Esercizio 74
7.3 Esercizio 75
7.4 Esercizio 79
7.5 Esercizio 79
7.6 Esercizio 81
7.7 Esercizio 83
7.8 Esercizio 85
7.9 Esercizio 86
7.10 Esercizio 92
7.11 Esercizi proposti 94

1 Definizione di grammatiche

1.1 Esercizio

Scrivere la grammatica che definisce il linguaggio L delle stringhe di alfabeto $\{a, b\}$ in cui il numero di a è uguale al numero di b.

Soluzione 1.1

Sia $x \in L$, con $x \neq \epsilon$. Allora x può avere come primo carattere una a (cioè $x = ay$) oppure una b ($x = by$). Nel caso in cui $x = ay$, nel suffisso y ci deve essere una b in più delle a. Certamente deve esistere una b in y che divide y in due parti, u e v, ciascuna delle quali contiene lo stesso numero di a e di b, cioè: $x = ay = aubv$, con $u, v \in L$. Se $x = by$, allo stesso modo si ha che $x = buav$. Da queste considerazioni si deriva facilmente la seguente grammatica:

$$G : S \to aSbS \mid bSaS \mid \epsilon$$

G è ambigua, come si comprende esaminado gli alberi di derivazione possibili per la stringa $abab$.

La grammatica è già molto semplice e sembra difficilmente semplificabile in modo ulteriore, perlomeno rispetto al numero delle produzioni e dei simboli non terminali utilizzati.

La grammatica non può essere semplificata nemmeno rispetto al numero massimo di non terminali presenti nelle parti destre delle produzioni: si può dimostrare infatti che nessuna grammatica lineare è in grado di descrivere L (ricordiamo che una grammatica è detta lineare quando vi è al più un solo non terminale nelle parti destre delle produzioni).

La grammatica data ha il vantaggio di essere compatta. Per costruire una grammatica $LL(1)$ per lo stesso linguaggio si può far riferimento al modo di operare di un riconoscitore a pila per il medesimo linguaggio. Il riconoscitore opererà confrontando il carattere letto col carattere presente in cima alla pila. Se il carattere in cima alla pila è diverso da quello letto, sulla pila verrà eseguita un'azione di `pop`, altrimenti (se la pila è vuota o il carattere in cima alla pila è lo stesso) il carattere verrà posto in cima alla pila. Il riconoscitore è un riconoscitore a pila vuota.

Dalla descrizione del riconoscitore è possibile costruire la seguente grammatica in cui vengono rappresentati anche gli insiemi guida:

$$G' = \begin{cases} S \to aBS\{a\} \mid bAS\{b\} \mid \epsilon\{\swarrow\} \\ A \to a\{a\} \mid bAA\{b\} \\ B \to b\{b\} \mid aBB\{a\} \end{cases}$$

1.2 Esercizio

Si definisca una grammatica per il linguaggio D delle parentesi in cui il numero totale delle parentesi aperte è dispari.

Esempi: $()$, $()()()$, $(()())()()$ sono in D, mentre $()()$, $(())()()$ non sono in D.

Soluzione 1.2

Il linguaggio D è noncontestuale, poichè si può ottenere per intersezione del linguaggio delle parentesi con il linguaggio regolare costituito da qualunque stringa in $\{(,)\}^*$ in cui il numero di (è dispari. Si potrebbe ottenere D applicando la classica costruzione usata per dimostrare la chiusura dei noncontestuali per intersezione con i regolari. La costruzione è tuttavia assai complessa e conviene quindi analizzare il problema nel modo seguente.

Sia P il linguaggio, analogo a D, in cui il numero totale delle parentesi aperte è pari. Le frasi di D possono allora essere caratterizzate induttivamente nel modo seguente:

1. $() \in D$;
2. se $p \in P$ allora aggiungendo a p una coppia di parentesi esterne si ottiene una frase di D: $(p) \in D$;
3. se $p \in P$ e $d \in D$, allora dp e pd sono in D, poichè il numero delle parentesi aperte in entrambi i casi è dispari.

Analogamente, si possono caratterizzare le frasi di P:

1. $\epsilon \in P$;
2. se $d \in D$ allora aggiungendo a d una coppia di parentesi esterne si ottiene una frase di P: $(d) \in P$;
3. se $p_1, p_2 \in P$ e $d_1, d_2 \in D$, allora p_1p_2 e d_1d_2 sono in P, poichè il numero delle parentesi aperte in entrambi i casi è pari.

Si può convincersi (oppure dimostrare) che queste caratterizzazioni esauriscono tutte le possibili stringhe di D e P. È a questo punto immediato ricavare una grammatica per D, in cui il nonterminale D rappresenta l'assioma:

$$G = \begin{cases} D \to (P) \mid PD \mid DP \\ P \to \epsilon \mid (D) \mid PP \mid DD \end{cases}$$

La produzione $D \to ()$ è stata eliminata perchè inutile.

Si osservi che la seguente grammatica G', più semplice di G, *non* genera il linguaggio corretto:

$$G' = \Big\{ \; S \rightarrow () \mid ((S)) \mid (S)S \mid S(S)$$

Infatti, G' non riesce a generare stringhe, come ad esempio ()()(), costituite da un numero dispari ≥ 3 di sottostringhe di D.

1.3 Esercizio

Si scriva la grammatica che caratterizza il linguaggio delle stringhe di alfabeto $\{a, b, c\}$ che non sono costituite da una stringa di a e b separata da una marca di centro c dalla sua copia riflessa :

$$L = \neg\{ucu^R : u \in \{a, b\}^*\}$$

Soluzione 1.3

Costruiamo dapprima la grammatica che riconosce il complemento del linguaggio cercato, ovvero il linguaggio delle palindromi con marca di centro:

$$G = \Big\{ \; S \rightarrow aSa \mid bSb \mid c$$

Il complemento di un linguaggio libero non è in generale libero. In questo caso però il passaggio dal linguaggio al suo complemento risulta possibile. Osserviamo che vi sono tre casi possibili in cui una stringa $x \in L$, ciascuno corrispondente a una differente "violazione" rispetto al linguaggio delle palindromi:

1. in x non vi è nessuna c in posizione mediana (questo include anche il caso in cui in x non vi sono c)
2. in x vi è almeno una c in posizione non mediana (questo include anche il caso in cui vi siano due o più c)
3. in x vi sono almeno due caratteri distinti in posizione simmetrica.

La seguente grammatica G' genera L.

$$G' = \begin{cases} S \rightarrow QSQ \mid \epsilon \mid a \mid b \mid QRc \mid cRQ \mid aRb \mid bRa \\ R \rightarrow QR \mid \epsilon \\ Q \rightarrow a \mid b \mid c \end{cases}$$

Dal nonterminale Q si generano i caratteri a, b, e c, mentre da R si genera qualunque stringa in $\{a, b, c\}^*$. Per generare una frase a partire da S, occorre dapprima applicare zero, una o più volte la produzione $S \rightarrow QSQ$, e poi applicare una delle altre produzioni sulla forma di frase del tipo uSv, con $\mid u \mid = \mid v \mid$. Le produzioni $S \rightarrow \epsilon \mid a \mid b$ permettono di generare le frasi corrispondenti al caso (1), le produzioni $S \rightarrow QRc \mid cRQ$ considerano il caso (2) (inserisono una c in posizione non centrale, e completano la frase con una stringa di lunghezza qualunque, ≥ 1, a sinistra o destra della c), mentre $S \to aRb \mid bRa$ corrispondono al caso (3).

La grammatica è ambigua, in quanto una stringa in cui sono verificate due o più condizioni può essere generata in vari modi.

La seguente grammatica G" è invece non ambigua, in quanto riconosce la prima condizione che si verifica nella stringa (procedendo dall'esterno verso l'interno)

$$G" = \begin{cases} 1: \ S \to aSa \mid bSb \mid A \\ 2: \ A \to a \mid b \mid aBb \mid aBc \mid bBa \mid bBc \mid cBa \mid cBb \mid cBc \mid \epsilon \\ 3: \ B \to aB \mid bB \mid cB \mid \epsilon \end{cases}$$

L'alternativa (1) genera il linguaggio delle stringhe simmetriche di a e b con centro A. La grammatica impone che venga utilizzata la produzione $S \to A$, la quale introduce attraverso la (2) una "violazione" rispetto al linguaggio delle palindromi con marca di centro. Le alternative (2) contengono tutte le possibili violazioni rispetto al linguaggio delle palindromi. Dopo che il non terminale A è stato utilizzato, una qualsiasi stringa composta di a, b e c può essere generata dalle alternative (3).

1.4 Esercizio

Scrivere una grammatica che individui il seguente linguaggio:

$$L_1 = \neg\{ucu : u \in \{a, b\}^*\}$$

Soluzione 1.4

Il linguaggio che si vuole definire è rappresentato dall'insieme di stringhe che verificano almeno una delle seguenti tre condizioni:

- non vi è nessuna c in posizione mediana;
- vi è almeno una c in posizione non mediana;
- esiste almeno una posizione i della sottostringa prima della marca di centro in cui vi sia il carattere a e nella posizione i dopo la marca di centro vi sia il carattere b (o viceversa). La figura 1 rappresenta la situazione.

La seguente grammatica ambigua risolve il problema.

$$G = \begin{cases} S \to W \mid X \mid Z \\ W \to QWQ \mid a \mid b \mid \epsilon \\ X \to QXQ \mid Rc \mid cR \\ Z \to AbR \mid BaR \\ A \to QAQ \mid aRc \\ B \to QBQ \mid bRc \\ R \to QR \mid Q \\ Q \to a \mid b \mid c \end{cases}$$

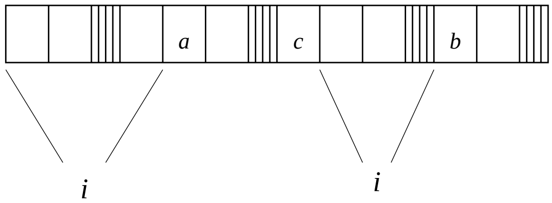

Figure 1: Stringhe appartenenti a L_1

Analizziamo il significato dei vari non terminali. Il non terminale Q può essere sostituito dall'espressione regolare $(a \mid b \mid c)$, mentre il non-terminale R rappresenta l'espressione regolare $(a \mid b \mid c)^+$. Da W si genera l'insieme delle stringhe in cui in posizione mediana non compare il carattere c. Da X si genera l'insieme delle stringhe in cui compare almeno una c in posizione non mediana. Il caso rimasto (un solo carattere c che divide in due parti di lunghezza identica la stringa) è trattato dal non terminale Z. Visto che interessa che le due stringhe siano differenti, basta imporre che in una posizione i vi sia un carattere a da una parte e un carattere b dall'altra. Queste stringhe sono in effetti generate dal non terminale Z, il quale distingue i due sottocasi in cui si ha un a nella prima parte e un b nella seconda (prima produzione) e il caso contrario gestito dalla seconda produzione. La figura 2 rappresenta sinteticamente l'albero sintattico corrispondente ad un esempio del primo caso. La grammatica è ambigua, in quanto ad esempio da Z si possono derivare anche stringhe generabili a partire da X.

Si può osservare come il linguaggio $\neg L_1$ non risulti libero: questo dimostra come la classe dei linguaggi liberi non sia chiusa rispetto all'operazione di complemento.

1.5 Esercizio

Scrivere una grammatica per il seguente linguaggio:

$$L_2 = \neg\{uu : u \in \{a, b\}^*\}$$

Soluzione 1.5

Il linguaggio da definire è quello delle stringhe di alfabeto $\{a, b\}$ che non possono essere decomposte in due stringhe identiche che si ripetono. Le stringhe di lunghezza dispari faranno parte di questo linguaggio. Possiamo associare all'assioma una prima produzione che genera le stringhe di alfabeto $\{a, b\}$ di lunghezza dispari:

$$G = \begin{cases} S \to D \\ D \to QQD \mid Q \\ Q \to a \mid b \end{cases}$$

Si tratta ora di generare le stringhe di lunghezza pari in cui la sequenza dei primi $n/2$ caratteri è diversa dalla sequenza degli ultimi $n/2$ caratteri. Questo corrisponde

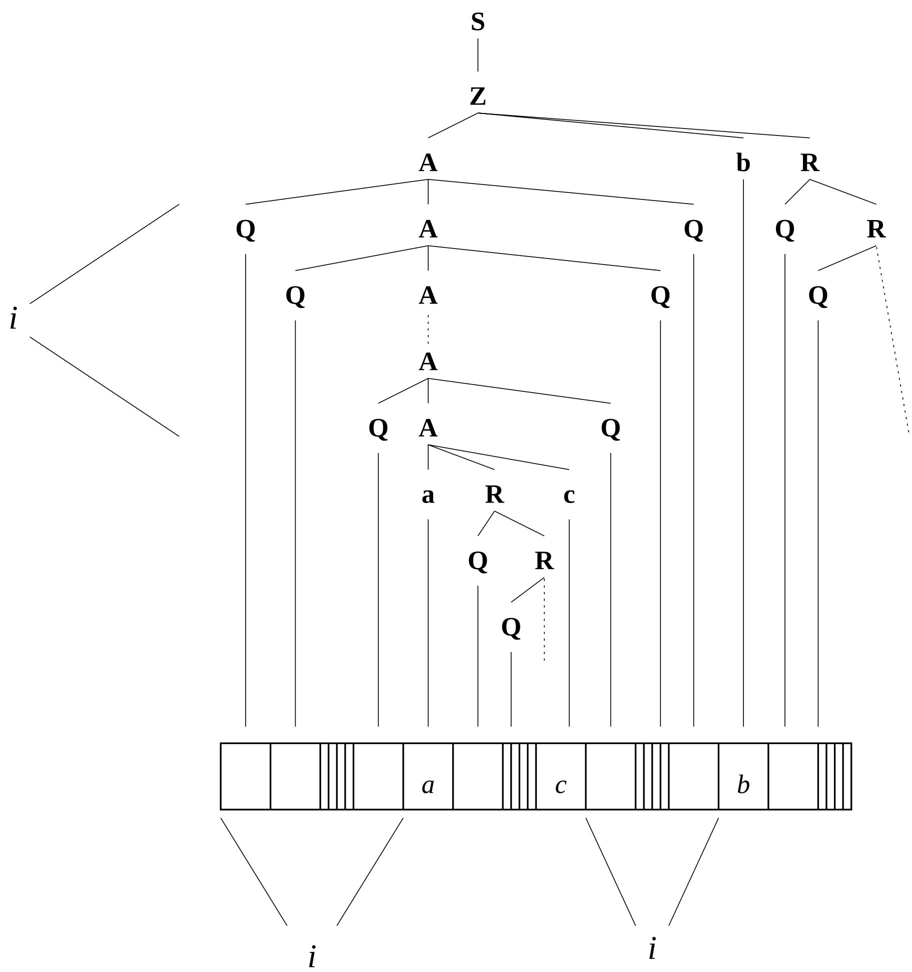

Figure 2: Albero sintattico per una stringa di L_1

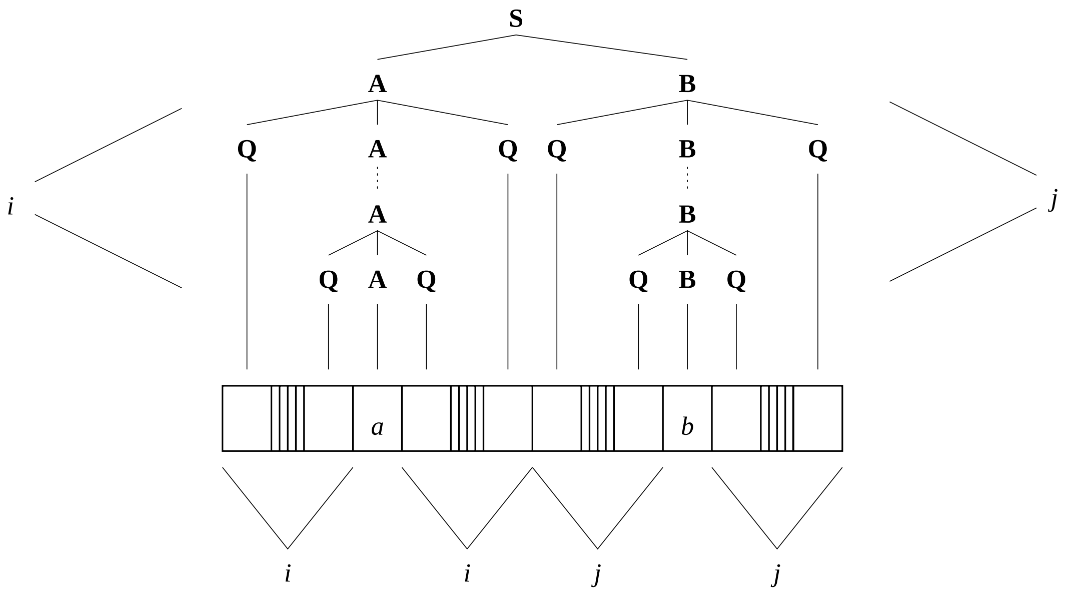

Figure 3: Albero sintattico per una stringa di L_2

ad imporre che esista almeno un valore $i, 1 \leq i \leq n/2$, per cui il carattere in posizione iΦi sia diverso dal carattere in posizione $i + n/2$, cioé $c(i) \neq c(i + n/2)$.

L'espediente che si usa consiste nell'osservare che se si prendono due stringhe S_1 di lunghezza dispari n_1 ed S_2 di lunghezza dispari n_2 e si costruisce la stringa $S = S_1 \cdot S_2$ di lunghezza $n = n_1 + n_2$, la distanza tra i due elementi centrali è pari a $d = (n_1 - 1)/2 + (n_2 - 1)/2 + 1 = n/2$.

Le stringhe di lunghezza pari del linguaggio cercato si potranno perciò costruire accostando una stringa di lunghezza $2i - 1$ con al centro il carattere a ad una stringa di lunghezza $n-2i+1$ con al centro il carattere b. Scriviamo la grammatica completa:

$$G = \begin{cases} S \to AB \mid BA \mid D \\ D \to QQD \mid Q \\ A \to QAQ \mid a \\ B \to QBQ \mid b \\ Q \to a \mid b \end{cases}$$

In figura 3 è rappresentata una stringa di lunghezza pari del linguaggio con il corrispondente albero sintattico. La stringa ϵ, correttamente, non è inclusa in $L(G)$. Il metodo qui descritto per generare L può essere applicato anche al linguaggio L_1 dell'esercizio 1.4, permettendo di ottenere una grammatica di dimensioni inferiori a quella presentata nella sua soluzione. Si lascia la costruzione di questa grammatica per esercizio al lettore.

1.6 Esercizio

Si definisca una grammatica per il seguente linguaggio:

$$L = \neg\{uu^R \mid u \in \{a,b\}^*\}$$

Soluzione 1.6

Si osservi dapprima che tutte le stringhe di lunghezza dispari sono in L. Le stringhe di lunghezza pari che sono in L contengono almeno una violazione rispetto al linguaggio delle palindromi senza marca di centro: sono del tipo $uaw_1w_2bu^R$ o del tipo $ubw_1w_2au^R$, dove $u, w_1, w_2 \in \{a,b\}^*$ e $\mid w_1 \mid = \mid w_2 \mid$. Di fatto, la sottostringa w_1w_2 è una qualunque stringa di lunghezza pari. Se denotiamo allora con P un nonterminale da cui si genera qualunque stringa di lunghezza pari, e con D un nonterminale da cui si genera qualunque stringa di lunghezza dispari, una grammatica (in forma b.n.f. estesa) per L può essere ricavata immediatamente:

$$G = \begin{cases} S \rightarrow aSa \mid bSb \mid aPb \mid bPa \mid D \\ P \rightarrow ((a \cup b)(a \cup b))^* \\ D \rightarrow (a \cup b)P \end{cases}$$

Le due produzioni $S \rightarrow aSa \mid bSb$ permettono di generare uSu^R, mentre le produzioni $S \rightarrow \mid aPb \mid bPa$ introducono una violazione rispetto al linguaggio delle palindromi, consentendo di generare ad esempio $uaPbu^R$, da cui si ricava una qualunque frase del tipo uaw_1w_2bu.

La grammatica, scritta in forma non BNF estesa, non è ambigua.

Alla grammatica può essere data una forma ancora più compatta eliminando la differente gestione delle stringhe di lunghezza pari e lunghezza dispari: una stringa di lunghezza dispari è infatti rappresentabile come uau^R, ubu^R, $uawbu^R$ o $ubwau^R$ (w è una qualunque stringa di lunghezza dispari). Il risultato, in forma b.n.f. estesa, è:

$$G = \left\{ S \rightarrow aSa \mid bSb \mid a(a \cup b)^*b \mid b(a \cup b)^*a \mid a \mid b \right.$$

1.7 Esercizio

Si descriva una grammatica lineare a destra che genera il linguaggio

$$L = (a(ab \cup bb)^*(aa \cup c))^+$$

Soluzione 1.7

Per definire una grammatica conviene costruire prima il riconoscitore a stati finiti non-deterministico (rappresentato in figura 4, da cui è ricavabile l'insieme delle regole della grammatica BNF.

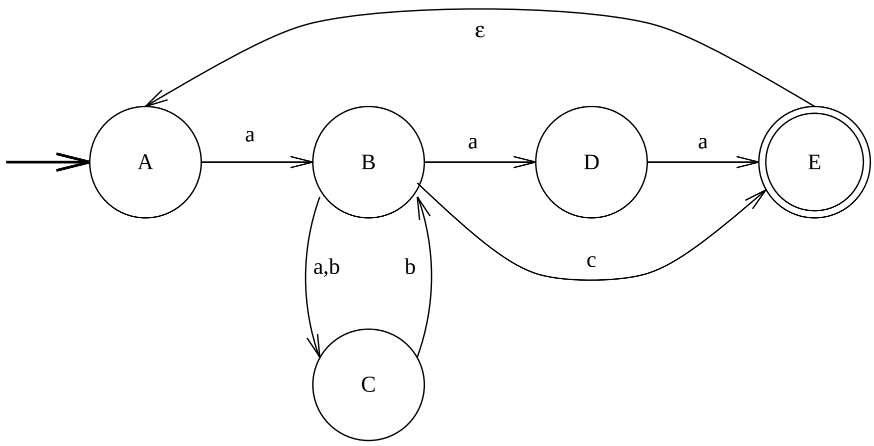

Figure 4: Automa riconoscitore di L

Dall'analisi del riconoscitore si ricava la grammatica. Il linguaggio è rappresentato dalle stringhe di caratteri che si possono ottenere partendo dallo stato A. Nella grammatica comparirà la seguente produzione:

$$S \to A$$

Le stringhe che vengono riconosciute nello stato A sono uguali alle stringhe che vengono ottenute partendo dallo stato B precedute dal terminale a. Questa osservazione ci porta a scrivere la seconda regola della grammatica:

$$A \to aB$$

A sua volta le stringhe riconosciute a partire dallo stato B corrispondono alle stringhe ottenute partendo dallo stato C precedute da a o b, oltre che le stringhe accettate partendo dallo stato D precedute da a e le stringhe accettate partendo da E precedute da c. Così:

$$B \to aC \mid bC \mid aD \mid cE$$

Si può applicare lo stesso procedimento anche agli stati C e D ottenendo altre due produzioni (in corrispondenza degli altri due non terminali C e D). E è invece uno stato finale dell'automa riconoscitore. Una delle possibile stringhe riconosciute a partire da E è la stringa vuota, che dovrà entrare a far parte delle alternative.

In conclusione si ottiene la seguente grammatica:

$$G = \begin{cases} S \to A \\ A \to aB \\ B \to aC \mid bC \mid aD \mid cE \\ C \to bB \\ D \to aE \\ E \to A \mid \epsilon \end{cases}$$

La grammatica risulta del tipo 3 della classificazione di Chomsky, ovvero lineare a destra.

1.8 Esercizio

Rappresentare mediante una grammatica BNF-estesa le espressioni aritmetiche con parentesi graffe, quadre e tonde, ricordando che le parentesi graffe possono contenere sia espressioni con parentesi quadre che tonde, mentre le parentesi quadre e tonde possono contenere solo espressioni con parentesi tonde. Si rappresenti anche la tradizionale gerachia tra gli operatori di addizione, sottrazione, moltiplicazione e divisione.

Soluzione 1.8

Una soluzione possibile é la seguente:

Espr$_0$ → Add$_0$ ('+' Add$_0$ | '-' Add$_0$)*
Add$_0$ → Fatt$_0$ ('*' Fatt$_0$ | '/' Fatt$_0$)*
Fatt$_0$ → Term | '{' Espr$_1$ '}' | '[' Espr$_2$ ']' | '(' Espr$_3$ ')'

Espr$_1$ → Add$_1$ ('+' Add$_1$ | '-' Add$_1$)*
Add$_1$ → Fatt$_1$ ('*' Fatt$_1$ | '/' Fatt$_1$)*
Fatt$_1$ → Term | '[' Espr$_2$ ']' | '(' Espr$_3$ ')'

Espr$_2$ → Add$_2$ ('+' Add$_2$ | '-' Add$_2$)*
Add$_2$ → Fatt$_2$ ('*' Fatt$_2$ | '/' Fatt$_2$)*
Fatt$_2$ → Term | '(' Espr$_3$ ')'

Espr$_3$ → Add$_3$ ('+' Add$_3$ | '-' Add$_3$)*
Add$_3$ → Fatt$_3$ ('*' Fatt$_3$ | '/' Fatt$_3$)*
Fatt$_3$ → Term | '(' Espr$_3$ ')'

1.9 Esercizio

Il linguaggio $L_1 = \{a^n b^m a^n b^m\}$ non è noncontestuale. Provare che anche $L_2 = \{uu \mid u \in \{a,b\}^*\}$ non è noncontestuale.

Soluzione 1.9

Se L_2 fosse noncontestuale, anche $L_2 \cap a^*b^*a^*b^*$ sarebbe noncontestuale, grazie alla nota proprietà di chiusura dei linguaggi noncontestuali per intersezione con i linguaggi regolari. Ma questo linguaggio è proprio L_1: assurdo.

1.10 Esercizi proposti

1.10.1 Esercizio

Si progetti una sintassi G per il linguaggio L di alfabeto $\Sigma = \{a, d,' (',')'\}$ delle liste a più livelli così definite:

- una lista è sempre racchiusa tra le parentesi e contiene un numero pari non nullo di componenti, separati dal delimitatore 'd';
- un componente è 'a' oppure una lista.

Ad es.:

$$(ada)$$

$$((ad(ada)dada)dadada)$$

Sono scorrette:

$$(\underline{(ad(adada)da)}dadada)$$

perchè la lista sottolineata ha un numero dispari di componenti.

1.10.2 Esercizio

È noto il linguaggio di Dyck di alfabeto $\{a, c\}$, ovvero il linguaggio delle stringhe ben parentetizzate, intendendo a come parentesi aperta e c come chiusa. Si consideri il linguaggio L ottenuto da Dyck cancellando *una sola* lettera c in qualsiasi punto della stringa.

Ad esempio ad L appartengono le frasi

$$aaacc \swarrow, \; aacac \swarrow$$

ottenute dalla frase ben parentetizzata $aacacc \swarrow$.

Si scriva una sintassi G per L e se ne esamini l'ambiguità.

1.10.3 Esercizio

L'operazione di *mischia* ($||$) mescola due stringhe x e y di alfabeto Σ in tutti i modi possibili, sempre rispettando l'ordinamento dei caratteri di ognuna delle stringhe. Essa è così definita:

$$x \parallel y = \{x_1y_1x_2y_2\dots x_ny_n \mid x = x_1x_2\dots x_n \ \wedge\ y = y_1y_2\dots y_n, n \geq 1,\ x_i, y_i \in \Sigma^*\}$$

Ad es.: $ab \parallel c = \{abc, acb, cab\}$, $ab \parallel cd = \{abcd, acdb, cdab, cadb, \dots\}$
La mischia di due linguaggi è l'insieme delle mischie delle loro stringhe:

$$L = L' \parallel L'' = \{x \parallel y \mid x \in L', y \in L''\}$$

Si costruisca una grammatica G per il linguaggio $L = L' \parallel L''$ dove $L' = cd^*$ e $L'' = \{a^nb^n \mid n \geq 0\}$.
Ad es. si hanno le frasi: c, cd, $cdab$, $cadb$, $cadadbdbddd, \dots$

2 Automi, linguaggi, espressioni regolari

2.1 Esercizio

Per ciascuno dei seguenti linguaggi, definiti a parole, si trovi una espressione regolare corrispondente:

1. Tutte le stringhe in $\{0,1\}^*$ in cui ogni 0 ha un 1 immediatamente a destra.
2. Tutte le stringhe in $\{0,1\}^*$ costituite esclusivamente da un numero pari (≥ 0) di caratteri 0.
3. Tutte le stringhe in $\{0,1\}^*$ che non hanno tre 0 consecutivi.
4. Tutte le stringhe in $\{0,1\}^*$ in cui ogni coppia di 0 adiacenti compare prima di una qualsiasi coppia di 1 adiacenti.
5. Tutte le stringhe in $\{0,1\}^*$ in cui il numero di 0 é uguale al numero degli 1, e nessun prefisso ha due 0 in piú degli 1 o due 1 in piú degli 0.

Soluzione 2.1

1. $(1 \cup 01)^*$
2. $(00)^+$
3. $(1 \cup 01 \cup 001)^*(0 \cup 00 \cup \epsilon)$
4. $(1 \cup \epsilon)(0^+1)^*0^*(1^+0)^*1^*$
5. $(10 \cup 01)^+$

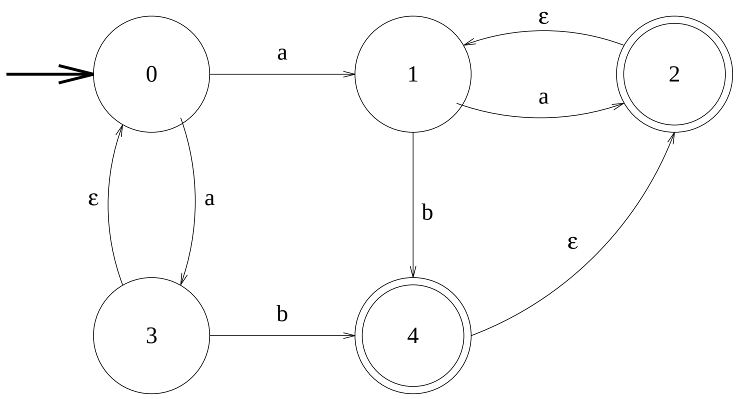

Figure 5: Automa non-deterministico A_1

2.2 Esercizio

É noto che il linguaggio $L_1 = \{a^n b^n \mid n \geq 0\}$ non é regolare. Dimostrare che anche $L_2 = \{a^n b^n c^m d^m \mid n, m \geq 0\}$ non é regolare.

Soluzione 2.2

Per assurdo, sia L_2 regolare. Allora, poiché i linguaggi regolari sono chiusi rispetto all'intersezione, anche $L_2 \cap (a^* b^*)$ sarebbe regolare. Ma questo linguaggio é proprio L_1. L'ipotesi che L_2 sia regolare é pertanto scorretta.

2.3 Esercizio

Si calcoli l'espressione regolare del linguaggio definito dall'automa A_1 in figura 5, e si trasformi l'automa in uno equivalente deterministico. Si minimizzi l'automa ottenuto.

Soluzione 2.3

Determinazione dell'espressione regolare

Per la determinazione dell'espressione regolare operiamo costruendo dapprima la grammatica lineare a destra corrispondente all'automa, quindi calcolando l'espressione regolare a partire dalla grammatica risolvendo le equazioni corrispondenti.

La grammatica corrispondente all'automa è la seguente:

$$\begin{cases} S \to aA \mid aC \\ A \to aB \mid bD \\ B \to A \mid \epsilon \\ C \to S \mid bD \\ D \to B \mid \epsilon \end{cases}$$

Da questa grammatica è possibile derivare il seguente insieme di equazioni nelle variabili A, B, C, D ed S. Ogni variabile rappresenta un linguaggio regolare.

$$\begin{cases} S = aA \cup aC \\ A = aB \cup bD \\ B = A \cup \epsilon \\ C = S \cup bD \\ D = B \cup \epsilon \end{cases}$$

Si dimostra che questo tipo di equazioni, ricavati da una grammatica lineare destra, ammettono sempre una e una sola soluzione nello spazio dei linguaggi regolari. La soluzione per il non terminale S rappresenta proprio il linguaggio cercato. Il procedimento da seguire per risolvere questi sistemi consiste nel ridurre, tramite sostituzioni successive, il sistema a una o più equazioni del tipo:

$$X = \alpha \cup \beta X$$

dove α è una espressione regolare nei simboli terminali e nonterminali (diversi da X stesso) e β è una espressione regolare nei soli simboli terminali. Questo tipo di equazioni, infatti, è immediatamente risolubile ponendo

$$X := \beta^* \alpha$$

. Gli eventuali simboli nonterminali in α sono facilmente eliminabili con ulteriori sostituzioni successive.

Risolvendo il sistema dato nell'ordine D, B, A, C ed infine S si ottiene:

$$\begin{aligned}
&D = B \\
&B = A \cup \epsilon \\
&A = (a \cup b)A \cup a \cup b \Rightarrow A = (a \cup b)^*(a \cup b) = (a \cup b)^+ \\
&C = S \cup b(A \cup \epsilon) = S \cup b(a \cup b)^+ \cup \epsilon) = S \cup b(a \cup b)^* \\
&S = a(a \cup b)^+ \cup aS \cup b(a \cup b)^* \\
&\Rightarrow S = a^*(a(a \cup b)^+ \cup ab(a \cup b)^*) = a^+(a \cup b)^*((a \cup b) \cup b) = \\
&= a^+(a \cup b)^+ = a(a \cup b)^+
\end{aligned}$$

L'espressione regolare cercata é quella ottenuta in corrispondenza dell'assioma del linguaggio S, ovvero $a(a \cup b)^+$. Si può ora verificare la correttezza della soluzione analizzando l'automa originale.

Trasformazione nell'automa deterministico

Il risultato della trasformazione nell'automa deterministico é rappresentato in figura 6.

Minimizzazione dell'automa

L'automa può essere minimizzato applicando il classico procedimento di calcolo delle classi di equivalenza degli stati.

Costruiamo la tabella di equivalenza:

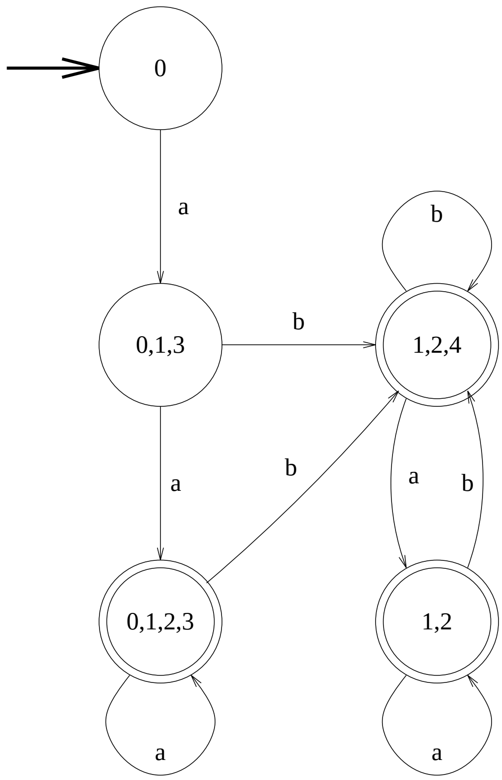

Figure 6: Automa deterministico corrispondente

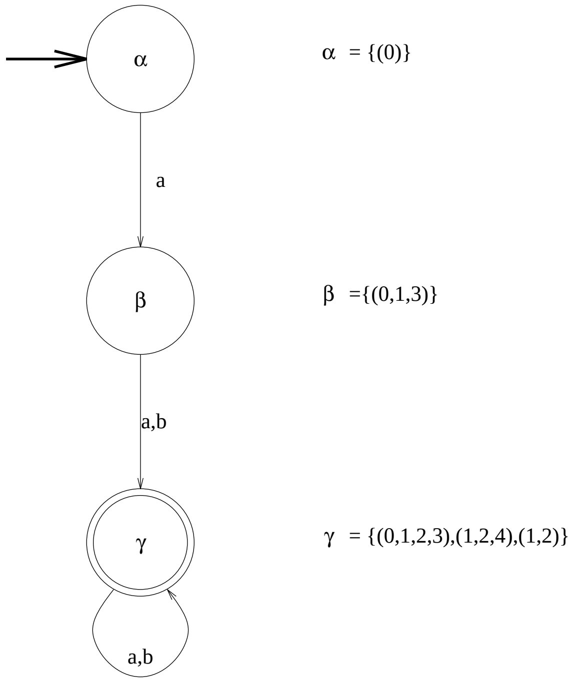

Figure 7: Automa deterministico minimo

0,1,3	X			
0,1,2,3	X	X		
1,2,4	X	X		
1,2	X	X		
	0	0,1,3	0,1,2,3	1,2,4

Quasi tutta la tabella viene riempita al primo passo, quando gli stati finali vengono distinti dagli stati non finali. Gli stati $\{0\}$ e $\{0, 1, 3\}$ vengono poi distinti al passo successivo, dopodiché la tabella non cambia.

Si distinguono perciò 3 classi di equivalenza negli stati. Si potrà derivare l'automa minimo costituito da 3 stati, rappresentato in figura 7.

Si poteva facilmente ricavare l'automa deterministico minimo dall'espressione regolare ottenuta per S, $a(a \cup b)^+$.

2.4 Esercizio

Scrivere una espressione regolare per il linguaggio L:

- $\Sigma = \{a, b\}$
- L = linguaggio riconosciuto dall'automa nella figura 8, di cui si riporta qui anche una rappresentazione tabellare:

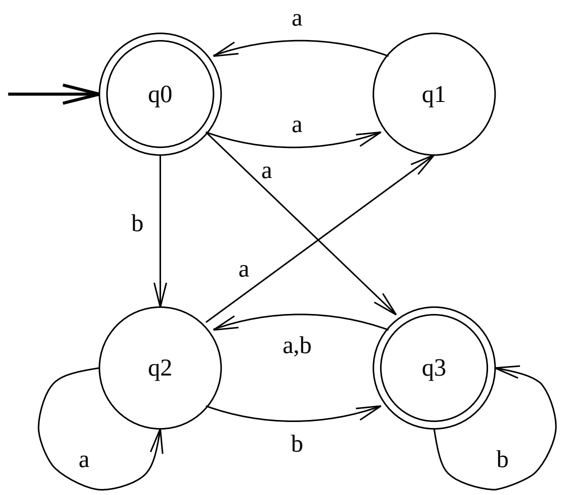

Figure 8: Automa riconoscitore di L

Stato	a	b
q_0	q_1, q_3	q_2
q_1	q_0	
q_2	q_2, q_1	q_3
q_3	q_2	q_2, q_3

Stati finali q_0 e q_3.

Soluzione 2.4

Dall'automa di può derivare la grammatica BNF lineare a destra equivalente. Si introduce un simbolo per ogni stato dell'automa riconoscitore, e l'assioma della grammatica corrisponderà al simbolo dello stato iniziale. Ogni stato finale potrà trasformarsi nella stringa ϵ.

Otteniamo:

$$\begin{cases} S \rightarrow aA \mid aC \mid bB \mid \epsilon \\ A \rightarrow aS \\ B \rightarrow aA \mid aB \mid bC \\ C \rightarrow aB \mid bB \mid bC \mid \epsilon \end{cases}$$

Questo insieme può essere trasformato nel seguente insieme di equazioni su variabili che rappresentano linguaggi regolari:

$$\begin{cases} S = aA \cup aC \cup bB \cup \epsilon \\ A = aS \\ B = aA \cup aB \cup bC \\ C = (a \cup b)B \cup bC \cup \epsilon \end{cases}$$

Si sostituisce A:

$$\begin{cases} A = aS \\ B = a^2S \cup aB \cup bC \\ C = (a \cup b)B \cup bC \cup \epsilon \\ S = a^2S \cup aC \cup bB \cup \epsilon \end{cases}$$

Si sostituisce B:

$$\begin{cases} A = aS \\ B = a^*(a^2S \cup bC) \\ C = (a \cup b)a^*a^2S \cup (a \cup b)a^*bC \cup bC \cup \epsilon \\ S = a^2S \cup aC \cup ba^*a^2S \cup ba^*bC \cup \epsilon \end{cases}$$

Si sostituisce C:

$$\begin{cases} A = aS \\ B = a^*(a^2S \cup bC) \\ C = ((a \cup b)a^*b \cup b)^*((a \cup b)a^*a^2S \cup \epsilon) \\ S = (a^2 \cup ba^*a^2)S \cup (a \cup ba^*b)((a \cup b)a^*b \cup b)^*((a \cup b)a^*a^2S \cup ((a \cup b)a^*b \cup b)^* \cup \epsilon) \end{cases}$$

Si ottiene come risultato l'espressione regolare:

$$L = S = ((a^2 \cup ba^*a^2) \cup (a \cup ba^*b)((a \cup b)a^*b \cup b)^*((a \cup b)a^*a^2)^*(((a \cup b)a^*b \cup b)^* \cup \epsilon)$$

2.5 Esercizio

Costruire il riconoscitore a stati finiti del complemento di

$$L = (a(ab \cup bb)^*(aa \cup c))^+$$

Soluzione 2.5

Il riconoscitore non deterministico del linguaggio L è rappresentato in figura 4. Costruiamo dapprima il riconoscitore deterministico del linguaggio L, rappresentato in figura 9.

Il riconoscitore del complemento si ottiene aggiungendo uno stato pozzo finale (P in figura) con un autoanello etichettato con tutti i simboli dell'alfabeto, tramutando tutti gli stati finali del riconoscitore in stati intermedi e gli stati intermedi in stati finali, aggiungendo ad ogni stato un arco uscente diretto verso lo stato pozzo etichettato con i simboli non accettati dallo stato. Il risultato dell'applicazione di questi passi al riconoscitore di figura 4 restituisce il riconoscitore del complemento rappresentato in figura 10.

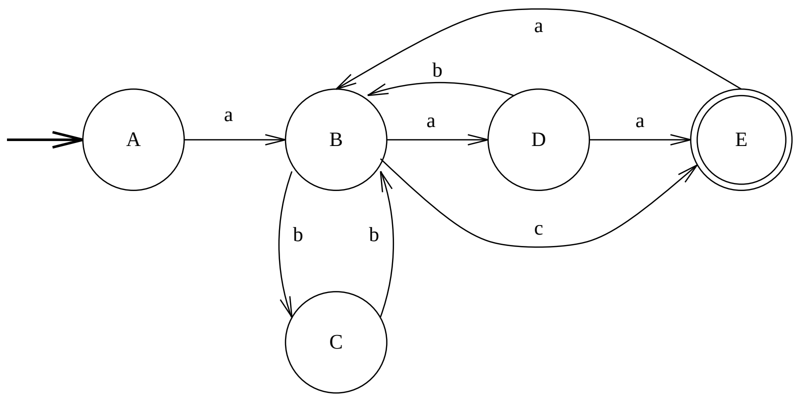

Figure 9: Automa deterministico riconoscitore di L

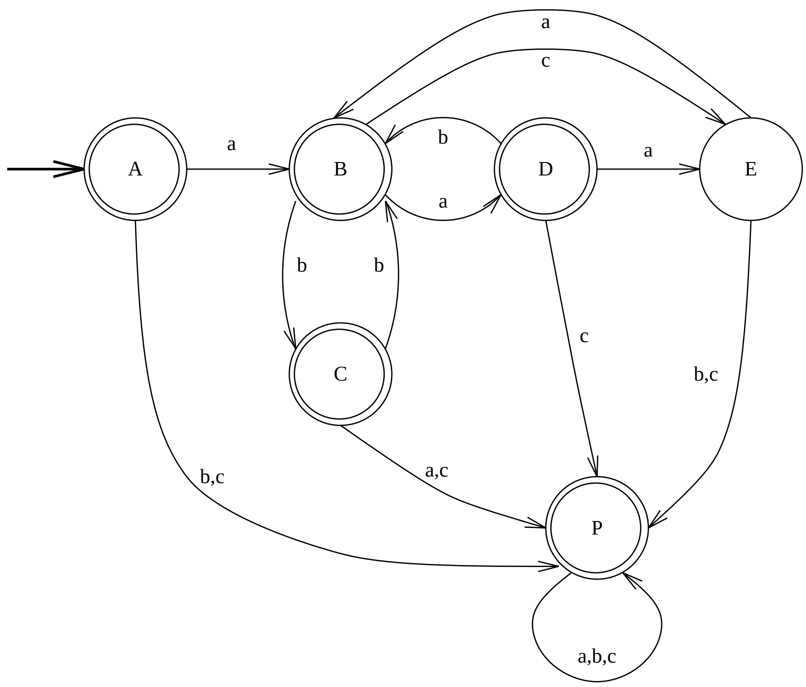

Figure 10: Automa deterministico riconoscitore del complemento di L

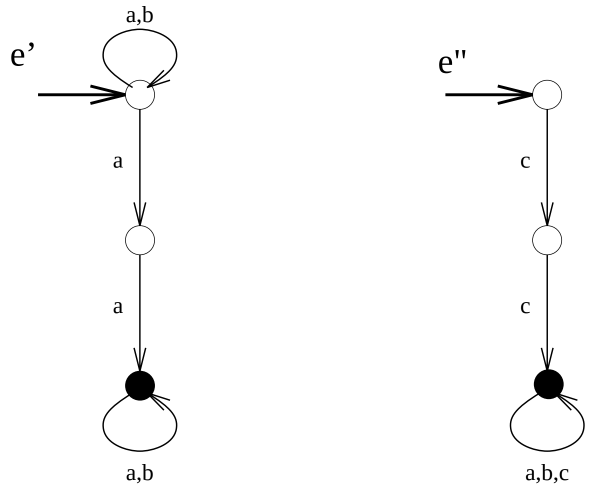

Figure 11: Automi non-deterministici per e' ed e''

2.6 Esercizio

Si costruisca, spiegando il procedimento seguito, l'automa deterministico minimo che riconosce il linguaggio definito dalla espressione regolare estesa con il complemento:

$$e = \neg((a \cup b)^* aa(a \cup b)^*)(cc(a \cup b \cup c)^*)$$

Soluzione 2.6

Identifichiamo i due componenti $e' = ((a \cup b)^* aa(a \cup b)^*)$ ed $e'' = (cc(a \cup b \cup c)^*)$ dell'espressione regolare $e = \neg e' \cdot e''$ (si ricorda che l'operatore di negazione ($\neg$) ha la precedenza rispetto all'operatore di concatenazione ($\cdot$)).

Si possono dapprima definire gli automi per il riconoscimento delle espressioni regolari e' ed e'' (figura 11).

Si osserva che l'automa riconoscitore di e'' é già deterministico. Si può rendere deterministico anche l'automa riconoscitore di e' con una semplice trasformazione, ottenendo l'automa rappresentato in figura 12.

Otteniamo quindi l'automa per riconoscere il complemento di e'. Per questo, tutti gli stati non-finali divengono finali e viceversa, e per ogni simbolo non riconosciuto in uno stato del riconoscitore si crea un arco etichettato con quel simbolo che porta dallo stato ad uno stato *pozzo* finale. Il risultato dell'applicazione di questo metodo all'automa e' é rappresentato in figura 13.

La concatenazione tra due linguaggi si ottiene aggiungendo un arco ϵ che collega tutti gli stati finali del riconoscitore del primo linguaggio con lo stato iniziale del riconoscitore del secondo linguaggio (figura 14). Riducendo poi il non-determinismo si ottiene l'automa deterministico cercato (figura 15).

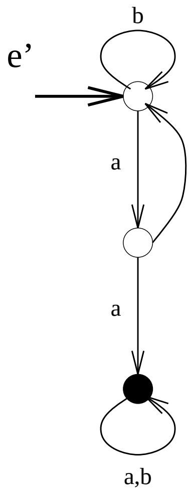

Figure 12: Automa deterministico per e'

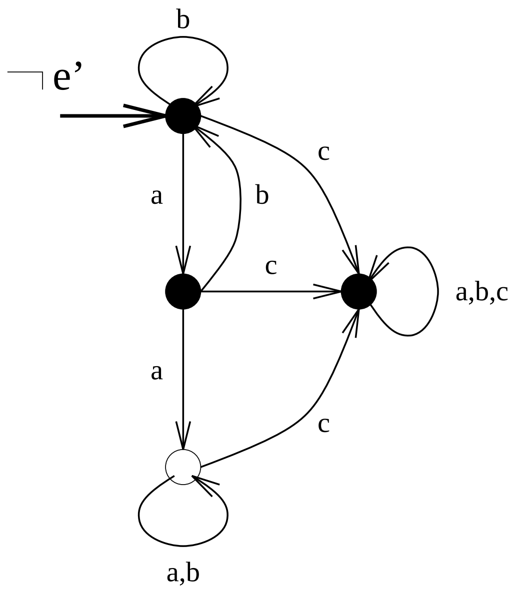

Figure 13: Automa deterministico per $\neg e'$

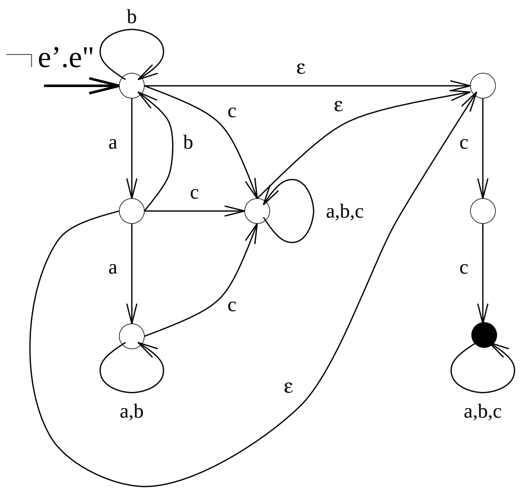

Figure 14: Automa non-deterministico per $\neg e' \cdot e''$

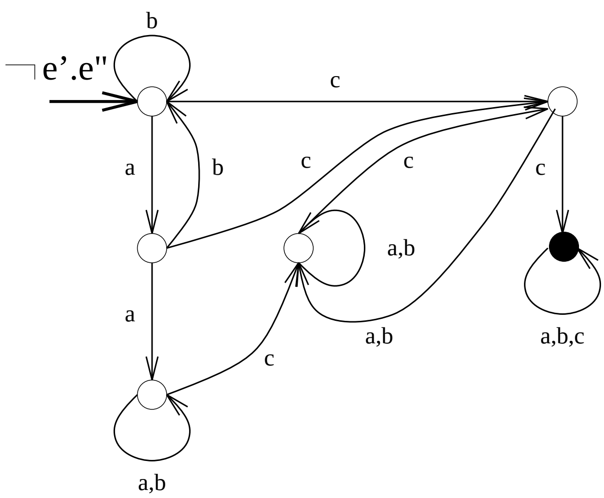

Figure 15: Automa deterministico per $\neg e' \cdot e''$

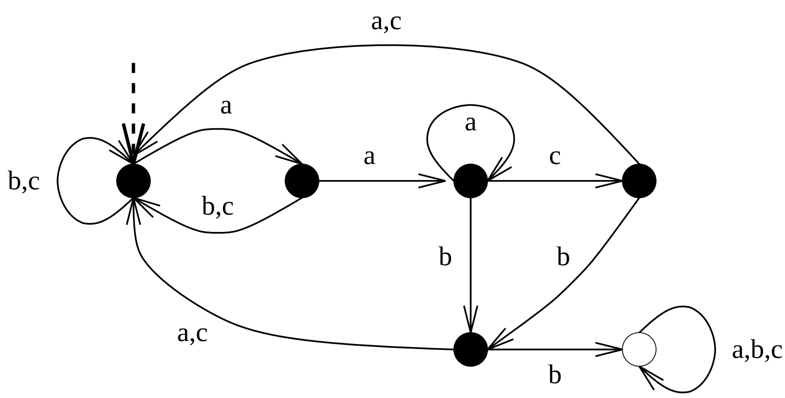

Figure 16: Automa deterministico per L

2.7 Esercizio

Scrivere una espressione regolare per il linguaggio L:

$$\Sigma = \{a, b, c\}$$

L é l'insieme delle stringhe che contengono (almeno) una sottostringa cc tra ogni coppia di aa e di bb (aa e bb sono consecutive).

Ad es. *baacccbbb*, *aaccbbcbb*, *aaccbbba*, *bbaaacccbbaa* sono corrette, mentre *aacbb*, *aaccbbaacbb* non lo sono.

Soluzione 2.7

Potendo utilizzare l'operatore di complemento, l'espressione regolare risulta la seguente:

$$L = \neg((a \cup b \cup c)^* aa(c \cup \epsilon)bb(a \cup b \cup c)^*)$$

Disegnamo l'automa riconoscitore del linguaggio L (figura 16).

L'espressione regolare che descriva l'automa senza complemento si può ottenere utilizzando il procedimento già visto che a partire dall'automa riconoscitore crea dapprima la grammatica, e poi traduce la grammatica in un sistema di equazioni su variabili che rappresentano espressioni regolari. La soluzione per l'espressione regolare corrispondente all'assioma della grammatica definisce il linguaggio cercato.

2.8 Esercizi proposti

2.8.1 Esercizio

Si scriva una espressione regolare per il linguaggio L di alfabeto $\{a, b, c, d\}$ così definito. Ogni frase è una lista di uno o più termini, separati da un separatore.

Un termine è qualsiasi parola (non nulla) di alfabeto $\{a, b\}$ in cui non appaia la sottostringa aa. Un separatore è qualsiasi parola (non nulla) di alfabeto $\{c, d\}$, iniziante e terminante per d, in cui il numero di c interposti tra due d consecutivi sia dispari.

Es. validi: $ababb$, $ababdabb$, $bbdcccdabdcdcdbb$
Non validi: cdd, abc, $aabca$, $abdccdb$

Si costruisca inoltre l'automa riconoscitore di L.

2.8.2 Esercizio

Si costruiscano gli automi riconoscitori minimi dei linguaggi:

$$L_1 = ((\epsilon \mid 00)1^*)^*$$

$$L_2 = ((\neg L_1)11)^+$$

2.8.3 Esercizio

Si consideri il linguaggio L di alfabeto $\Sigma = \{a, b, \#\}$ in cui ogni frase è una lista di parole di alfabeto $\{a, b\}$ separate dal $\#$. Ogni parola deve soddisfare a una delle seguenti condizioni:

- il numero delle a è pari e quello delle b è dispari
- il numero delle a è dispari e quello delle b è pari

Esempi di frasi: $abbaa$, $abbba\#bab\#a$ mentre sono scorrette $abbaa\#$, $bab\#ab$.
Si costruisca un automa riconoscitore deterministico per L.

3 Eliminazione ambiguità

3.1 Esercizio

Costruire una grammatica non ambigua per il linguaggio L delle espressioni ben parentetizzate su un alfabeto costituito da una sola coppia di parentesi e dal simbolo terminale b. Esempio di espressione corretta: $b()((b)(bbb)())$.

Soluzione 3.1

La grammatica più ovvia per L è la seguente:

$G = \Big\{ \; S \rightarrow SS \mid (S) \mid b \mid \epsilon$

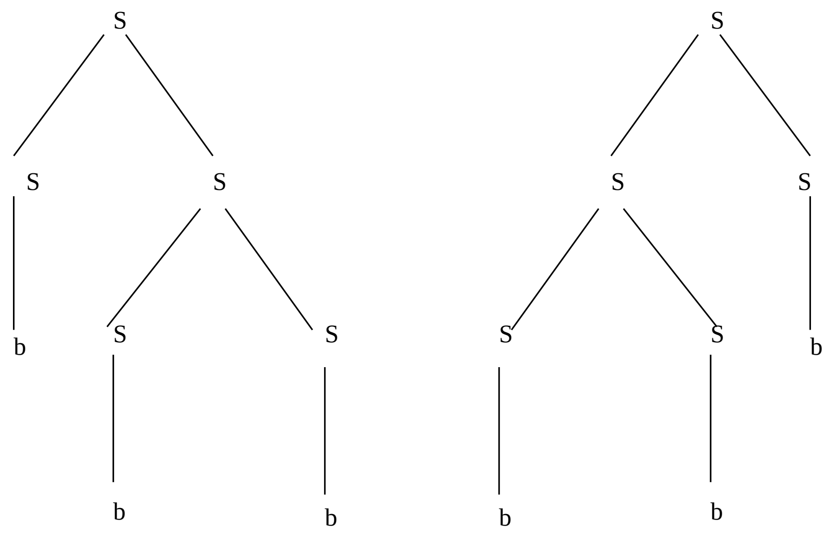

Figure 17: Due alberi di derivazione distinti per la frase bbb della grammatica G dell'esercizio 3.1.

La grammatica è però ambigua. Ad esempio, la stringa bbb può essere generata con le due seguenti derivazioni destre distinte:

$$S \Rightarrow SS \Rightarrow Sb \Rightarrow SSb \Rightarrow Sbb \Rightarrow bbb$$

$$S \Rightarrow SS \Rightarrow SSS \Rightarrow SSb \Rightarrow Sbb \Rightarrow bbb$$

che corrispondono ai due alberi in figura 17.

L'ambiguità è determinata dalla presenza della regola $S \to SS$, che è ricorsiva sia a sinistra che a destra. Si può dimostrare che ogni grammatica in cui compare una regola del tipo $X \to X\alpha X$, con $\alpha \in (\Sigma \cup V)^*$, in cui X è definito (vale a dire che X può generare almeno una stringa di terminali) e raggiungibile da S, è ambigua. Una possibile soluzione in questo caso è quella di modificare la grammatica G in questo modo:

$$G_1 = \Big\{ \; S \to bS \mid (S)S \mid \epsilon$$

Dimostriamo per induzione sulla lunghezza n delle frasi di L che nessuna frase generata dalla grammatica G_1 è ambigua, nell'ipotesi che G_1 generi correttamente il linguaggio L.

Il passo base ($n = 0$) è ovvio in quanto l'unico modo di generare la stringa nulla è applicare la produzione $S \to \epsilon$. Il passo induttivo si dimostra in base all'ipotesi induttiva, per la quale ogni frase di lunghezza inferiore a n è non ambigua. Sia x una frase di lunghezza n. Le frasi possono cominciare o con il carattere "b" o con il carattere "(". Se x è del tipo "by", dove y è una frase, allora per ipotesi induttiva y

è una frase non ambigua, in quanto di lunghezza minore di n, cioè esiste una unica derivazione sinistra $S \Rightarrow^* y$. Quindi anche x non è ambigua poiché l'unico modo per derivarla è derivare bS con il passo $S \Rightarrow bS$ e poi continuare con la derivazione precedente da S, ottenendo $S \Rightarrow bS \Rightarrow^* by$. Se invece x è del tipo "$(z$", il primo passo di derivazione deve applicare la regola $S \Rightarrow (S)S$. Allora $z = v)w$, con v e w derivate da S. Ma per ipotesi induttiva sia v che w non sono frasi ambigue e perciò non lo è anche $x = (z$. Non ci sono altri casi possibili e la dimostrazione è quindi conclusa.

Sarebbe ora necessario dimostrare che la nuova grammatica G_1 è equivalente a G, ma in questo caso ci si può accontentare di una "evidenza intuitiva".

3.2 Esercizio

Trovare una forma non ambigua per la seguente grammatica:

$$G_1 = \begin{cases} S \rightarrow EF \\ E \rightarrow bE \mid cE \mid \epsilon \\ F \rightarrow cF \mid dF \mid \epsilon \end{cases}$$

Soluzione 3.2

Il linguaggio generato dalla grammatica G_1 è costituito dall'insieme delle stringhe di $\{b, c, d\}^*$ in cui nessuna d è a sinistra di una b, che può essere anche rappresentato con l'espressione regolare: $(b \cup c)^* \cdot (c \cup d)^*$.

G_1 è ambigua in quanto le c che seguono una b e precedono la prima d possono essere generate sia dal nonterminale E che dal nonterminale F. Ad esempio, la frase cc può essere generata con due derivazioni sinistre distinte:

$$S \Rightarrow EF \Rightarrow cEF \Rightarrow ccEF \Rightarrow ccE \Rightarrow cc$$

$$S \Rightarrow EF \Rightarrow F \Rightarrow cF \Rightarrow ccF \Rightarrow cc$$

Una soluzione per risolvere l'ambiguità può essere quella di considerare due insiemi di produzioni distinti, il primo per generare solo stringhe di b e di c, il secondo per generare una stringa di b e di c seguita da una d e da un stringa di c e di d. Una grammatica non ambigua per $L(G_1)$ è:

$$G_2 = \begin{cases} S \rightarrow E \mid EdF \\ E \rightarrow bE \mid cE \mid \epsilon \\ F \rightarrow cF \mid dF \mid \epsilon \end{cases}$$

3.3 Esercizio

Si considerino i due linguaggi $L_1 = b^*d^*$ e $L_2 = \{b^n d^n \mid n \geq 0\}$. Si trovi una grammatica non ambigua in forma b.n.f. estesa per $L = L_1 \cdot L_2$ e una per $L_1 \bigcup L_2$.

Soluzione 3.3

Si può facilmente verificare che, date due grammatiche G_1 e G_2, se $L_1 = L(G_1)$ e $L_2 = L(G_2)$, con $L_1 \cap L_2 \neq \emptyset$, e si applica a G_1 e a G_2 il procedimento usuale per la costruzione dell'unione di due grammatiche, la grammatica risultante è sempre ambigua, anche quando il linguaggio non lo è. Qualora invece G_1 e G_2 non siano ambigue e $L_1 \cap L_2 = \emptyset$, la grammatica unione non é mai ambigua. Un problema analogo accade con la concatenazione, cioè per il linguaggio $L_1 \cdot L_2$, con $L_1 \cap L_2 \neq \emptyset$, quando $\epsilon \in L_1$ o $\epsilon \in L_2$. In questo caso, tuttavia, la condizione che i due linguaggi siano disgiunti é necessaria ma non sufficiente per garantire la non ambiguitá della grammatica ottenuta per concatenazione.

Siano G_1 e G_2 due grammatiche in forma b.n.f. estesa per L_1 e L_2 rispettivamente:

$$G_1 = \left\{ \ S \rightarrow b^*d^* \right.$$

$$G_2 = \left\{ \ S \rightarrow bSd \mid \epsilon \right.$$

Si può ricavare immediatamente per concatenazione una grammatica in forma b.n.f. estesa per $L = L_1 \cdot L_2$, rinominando con E l'assioma di G_2:

$$G_3 = \left\{ \begin{array}{l} S \rightarrow b^*d^*E \\ E \rightarrow bEd \mid \epsilon \end{array} \right.$$

G_3 è ambigua, in quanto una frase del tipo $b^m d^m$ può essere ricavata con due alberi sintattici distinti, corrispondenti alle due derivazioni seguenti:

$$S \Rightarrow b^m d^m E \Rightarrow b^m d^m$$

$$S \Rightarrow E \Rightarrow bEd \Rightarrow \ldots \Rightarrow b^m E d^m \Rightarrow b^m d^m$$

Per eliminare l'ambiguità occorre distinguere le derivazioni di $b^m d^m$ da quelle di $b^n d^m$, con $n \neq m$. Una grammatica non ambigua in forma b.n.f. estesa è la seguente, in cui dal nonterminale E si derivano solo stringhe del tipo $b^m d^m$, con $m > 0$, mentre dal nonterminale D si derivano solo stringhe del tipo $b^n d^m$, con $n \neq m$.

$$G_4 = \left\{ \begin{array}{l} S \rightarrow DE \mid E \mid EE \mid D \mid \epsilon \mid b^+ \mid d^+ \\ D \rightarrow b^+E \mid Ed^+ \\ E \rightarrow bEd \mid bd \end{array} \right.$$

Le produzioni per il nonterminale S si giustificano subito notando che da S occorre potere derivare stringhe del tipo $b^n d^n b^m d^m$ $(S \rightarrow EE)$, del tipo $b^n d^m b^k d^k$ con $n \neq m$

($S \rightarrow DE$), del tipo $b^m d^m$, con $m > 0$ ($S \rightarrow E$), del tipo $b^n d^m$ con $n \neq m$ ($S \rightarrow D$), e infine dei tipi b^* e d^*.

La definizione di una grammatica non ambigua per $L_1 \bigcup L_2$ è ora immediata:

$$G_5 = \begin{cases} S \rightarrow E \mid D \mid \epsilon \mid b^+ \mid d^+ \\ D \rightarrow b^+E \mid Ed^+ \\ E \rightarrow bEd \mid bd \end{cases}$$

Tuttavia, essendo L_2 incluso in L_1, in questo caso $L_1 \bigcup L_2$ è semplicemente L_1, cioè la grammatica G_1 va bene anche per l'unione.

3.4 Esercizio

Costruire una grammatica non ambigua per le espressioni booleane, con la possibilità di parentesi, nelle variabili p, q, r e negli operatori **not**, **or**, **and**. La grammatica deve rispettare la precedenza degli operatori, che hanno le seguenti priorità: 3 per **not**, 2 per **and**, 1 per **or**. Si vuole inoltre che **or** sia associativo a destra, **and** sia associativo a sinistra. Si dia anche una versione della grammatica in forma bnf estesa, che mantenga, qualora possibile, la priorità e l'associatività degli operatori.

Soluzione 3.4

Consideriamo per ora solo la generazione di espressioni senza parentesi, che sono comunque frasi del linguaggio in quanto le parentesi sono opzionali.

Una grammatica ambigua e che non rispetta le precedenze e l'associatività può essere ricavata immediatamente:

$$G_0 = \left\{ S \rightarrow S \textbf{ and } S \mid S \textbf{ or } S \mid \textbf{not } S \mid p \mid q \mid r \right.$$

L'ambiguità deriva dalle regole ricorsive a sinistra e a destra, che devono essere eliminate. In questo caso, il modo più semplice per costruire una grammatica non ambigua è quello di eliminare, ad esempio, le ricorsioni sinistre dalle regole ricorsive a sinistra e a destra, introducendo opportuni simboli non terminali. Il risultato è la seguente grammatica non ambigua (ma che ancora non rispetta le precedenze):

$$G_1 = \begin{cases} S \rightarrow A \textbf{ or } S \mid A \textbf{ and } S \mid \textbf{not } S \mid A \\ A \rightarrow p \mid q \mid r \end{cases}$$

Le grammatiche per generare espressioni sono di solito chiamate "grammatiche ad operatori", e possono essere così caratterizzate:

- alcuni terminali sono detti *operatori*;
- alcuni non terminali sono detti *oggetti*;
- ci può essere al più un operatore nella parte destra di ogni regola;

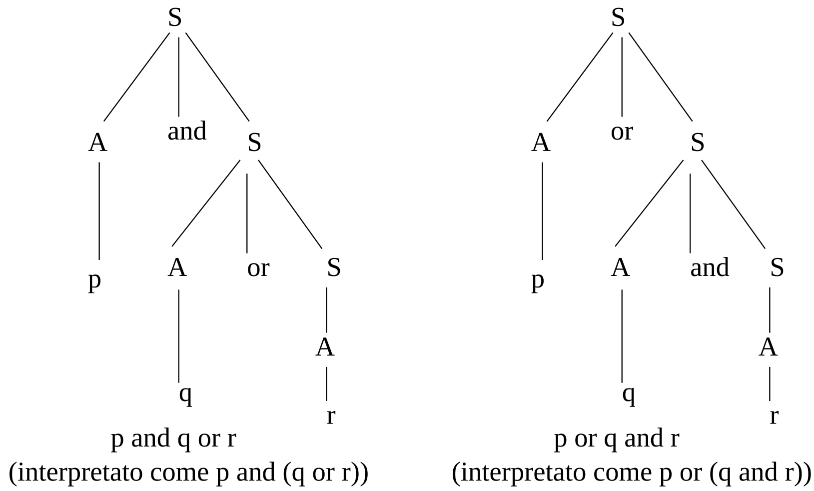

Figure 18: Due alberi di derivazione per la grammatica G_1 dell'esercizio 3.4. Nell'albero di sinistra **and** cede la precedenza a **or**, mentre in quello di destra **or** cede la precedenza a **and**.

- gli operatori possono comparire solo in regole la cui parte sinistra è un oggetto e in cui vi è almeno un oggetto nella parte destra.

Gli operatori di G_1 sono i simboli **not**, **and**, **or**, mentre gli oggetti sono i nonterminali S e A.

La precedenza tra operatori può essere definita, per un albero di derivazione di una frase, in questo modo: l'operatore b cede la *precedenza* all'operatore a se c'è un cammino verso l'alto nell'albero da a fino al nodo che è immediatamente al di sopra di b (cioè contiene l'oggetto da cui si è derivato b).

Intuitivamente, se un operatore ha la precedenza sull'altro, agisce sui propri operandi prima che il secondo operatore agisca sui propri operandi, in quanto si trova a un livello piú basso nell'albero di derivazione.

L'ordine di precedenza degli operatori è un concetto di tipo semantico, in quanto riguarda la valutazione delle espressioni. Poiché però lo scopo delle grammatiche a operatori è quello di fornire un ausilio agli analizzatori, tramite la costruzione dell'albero sintattico di una frase, è conveniente che la precedenza abbia un corrispondente nella sintassi, in modo da semplificare l'elaborazione di tipo semantico.

La grammatica G_1 implementa una precedenza da sinistra a destra: gli operatori in una frase cedono la precedenza agli operatori che stanno alla loro destra. Si vedano ad esempio gli alberi di figura 18.

Per implementare le priorità *prefissate* per ciascun operatore, richieste dal testo dell'esercizio, occorre modificare la grammatica precedente, distinguendo un oggetto per ogni operatore e definendo le produzioni in ordine inverso rispetto alla priorità. L'operatore a priorità inferiore, cioé l'**or**, è così generato sempre ai livelli più alti dell'albero. In questo modo non può mai succedere che un **or** sia generato in un nodo

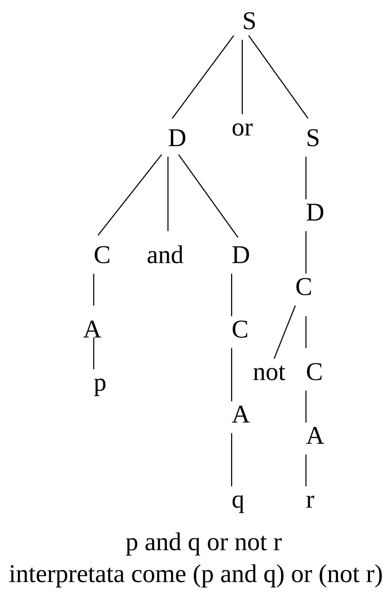

Figure 19: Albero di derivazione per la grammatica G_3 dell'esercizio 3.4.

di un albero di derivazione che sta al di sotto di un qualunque nodo con un **and**.

$$G_2 = \begin{cases} S \to D \textbf{ or } S \mid D \\ D \to C \textbf{ and } D \mid C \\ C \to \textbf{not } C \mid A \\ A \to p \mid q \mid r \end{cases}$$

Ad esempio, la frase p **and** q **or not** r ha l'albero di derivazione di figura 19.

Si dice che l'operatore a è *associativo a sinistra* in un albero di derivazione se ogni a cede sempre la precedenza agli a che stanno alla sua sinistra. L'operatore a è *associativo a destra* se ogni a cede sempre la precedenza agli a che stanno alla sua destra.

La grammatica G_2 non implementa la richiesta associatività degli operatori. L'associatività è infatti destra per tutti gli operatori, cioè la frase p **and** q **and** r è generata come se fosse p **and** $(q$ **and** $r)$.

Per avere l'associatività a sinistra per **and** e a destra per **or** basta introdurre la regola $D \to D$ **and** C al posto della regola $D \to C$ **and** D.

Introduciamo ora le parentesi nelle espressioni, facendo attenzione a non rendere ambigua la grammatica. Il metodo corretto è quello di aggiungere la regola $A \to (S)$, che consente di espandere i nodi a livello più basso (marcati con A) con una espressione tra parentesi oltre che con le variabili p, q, r. Il risultato finale è la seguente grammatica G_3, di cui alcuni esempi di derivazioni sono mostrati in figura 20.

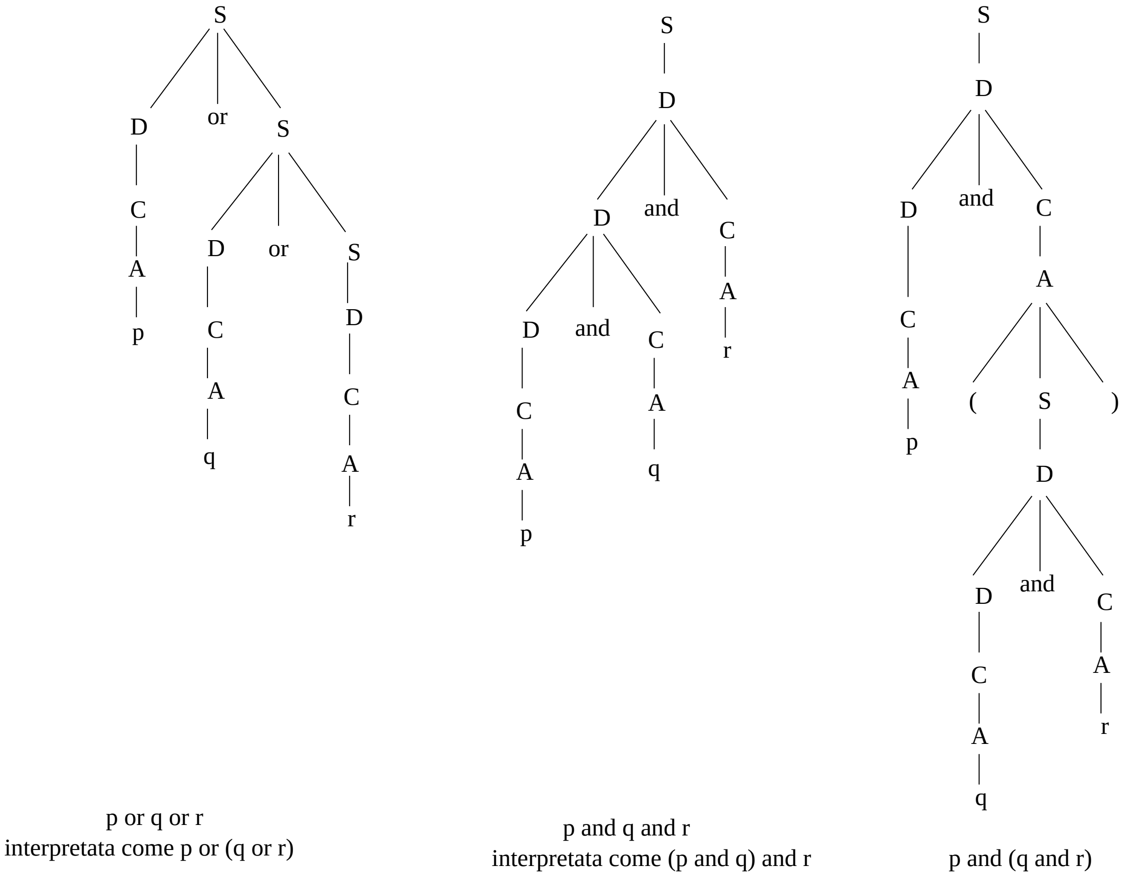

Figure 20: Alberi di derivazione per la grammatica G_3 dell'esercizio 3.4.

$$G_3 = \begin{cases} S \rightarrow D\,\mathbf{or}\,S \mid D \\ D \rightarrow D\,\mathbf{and}\,C \mid C \\ C \rightarrow \mathbf{not}\,C \mid A \\ A \rightarrow p \mid q \mid r \mid (S) \end{cases}$$

La grammatica può essere riscritta nella seguente forma b.n.f. estesa:

$$G_4 = \begin{cases} S \rightarrow D\,(\mathbf{or}\,D)^* \\ D \rightarrow C\,(\mathbf{and}\,C)^* \\ C \rightarrow \mathbf{not}\,C \mid A \\ A \rightarrow p \mid q \mid r \mid\ '('\,S\,')' \end{cases}$$

La precedenza fra operatori è mantenuta dalla G_4. Si può ritenere che l'associatività non abbia più molto senso, in quanto ad esempio una frase come $p\mathbf{or}q\mathbf{or}r$ viene generata in questo modo: $S \Rightarrow D\mathbf{or}D\mathbf{or} \Rightarrow^+ p\mathbf{or}q\mathbf{or}r$: gli **or** sono generati di fatto alla stessa altezza nell'albero sintattico. Tuttavia, nella costruzione di analizzatori semantici di tipo discendente per grammatiche di questo tipo, la generazione e la valutazione avvengono da sinistra verso destra, cioè in un modo che realizza naturalmente una associatività sinistra. In realtà, non è difficile modificare l'analizzatore per ottenere una associatività destra, ad esempio usando una pila per memorizzare i valori associati ai terminali.

Si osservi infine che per questo esercizio, dal punto di vista semantico, è irrilevante quale sia la associatività dell'operatore **or**, che era stata richiesta solo allo scopo di definire il concetto relativo per le grammatiche a operatori.

3.5 Esercizio

Rendere non ambigua la grammatica G:

$$G = \begin{cases} S \to aSb \mid C \\ C \to aD \mid \epsilon \\ D \to bC \end{cases}$$

Soluzione 3.5

La grammatica individua il linguaggio (non regolare):

$$\{a^n(ab)^*b^n \mid n \geq 0\}$$

Per frasi del tipo a^nabb^n vi è un'ambiguità su quale derivazione è stata effettivamente seguita. Così alla stringa $aabb$ corrispondono le due derivazioni $S \to aSb \to aaSbb \to aaCbb \to aabb$ e $S \to aSb \to aCb \to aaDb \to aabCb \to aabb$.

Per evitare questo caso bisogna modificare la grammatica di partenza accedendo alle produzioni dei non-terminali C e D solo quando si presenta effettivamente una sequenza di ab innestati di lunghezza pari almeno a 2. Una soluzione è offerta dalla seguente grammatica, in cui si sono anche semplificate le produzioni da C e D:

$$G' = \begin{cases} S \to aSb \mid aababCb \mid \epsilon \\ C \to abC \mid \epsilon \end{cases}$$

3.6 Esercizi proposti

3.6.1 Esercizio

Data la grammatica:

$$G = \begin{cases} S \to aA \mid A \mid Bb \mid B \\ B \to C \mid Bb \\ C \to aCb \mid \epsilon \\ A \to aA \mid C \end{cases}$$

si determini se G è ambigua e si descriva in termini insiemistici il linguaggio $L(G)$.

3.6.2 Esercizio

Mostrare che le seguenti grammatiche sono ambigue e trovare per ciascuna una forma non ambigua equivalente.

$$G_1 = \begin{cases} S \to aST \mid \epsilon \\ T \to aT \mid a \end{cases}$$

$$G_2 = \begin{cases} S \to aS \mid aSb \mid T \\ T \to aTa \mid a \end{cases}$$

$$G_3 = \begin{cases} S \to TUTU \\ T \to aT \mid bT \mid \epsilon \\ U \to bU \mid cU\epsilon \end{cases}$$

$$G_4 = \begin{cases} S \to bS \mid Sd \mid EF \\ E \to bEc \mid bc \\ F \to cFd \mid cd \end{cases}$$

4 Analisi discendente deterministica

4.1 Esercizio

Si definiscano mediante una grammatica il linguaggio $L = \{a^n b^p c^q \mid n \geq 0 \wedge n = p+q\}$ e il linguaggio $L_1 = L^R$ (linguaggio speculare).

Si verifichi se tale grammatiche risultino LL(1), e in caso affermativo se ne costruisca l'analizzatore a discesa ricorsiva e l'automa a pila riconoscitore.

Soluzione 4.1

Una grammatica non ambigua G per L è la seguente:

$$G = \begin{cases} S \to aSc \mid T \\ T \to aTb \mid \epsilon \end{cases}$$

Una derivazione è ad esempio:

$$S \Rightarrow aSc \Rightarrow aaSc \Rightarrow aaaTbcc \Rightarrow aaabcc$$

.

Calcolo insiemi guida per L: $\mathcal{G}(A \to \alpha) = \begin{cases} \mathcal{I}(\alpha) \text{ se } \alpha \text{ non genera } \epsilon \\ \mathcal{I}(\alpha) \bigcup \mathcal{S}(A) \text{ se } \alpha \Rightarrow^* \epsilon \end{cases}$

con

$$\mathcal{I}(\alpha) = \{a \in \Sigma \mid \alpha \Rightarrow^* a\beta\}$$

$$\mathcal{S}(A) = \{a \in \Sigma \mid S \Rightarrow^* \gamma A a \beta\}$$

Pertanto:

$$\mathcal{G}(S \to aSc) = \mathcal{I}(aSc) = \{a\}$$

$$\mathcal{G}(S \to T) = \mathcal{I}(T) \bigcup \mathcal{S}(S) = \{a\} \cup \{c\}$$

I due insiemi guida non sono disgiunti e quindi la grammatica non è LL(1).

Ci si potrebbe domandare se una grammatica LL(1) possa esistere per L. La risposta è negativa, anche se L è certamente un linguaggio deterministico (è facile costruire un automa a pila deterministico con tre soli stati che riconosce L). Infatti, quando si legge una a la conoscenza del carattere successivo non basta per stabilire la ripartizione delle b e delle c. È anzi facile intuire che la grammatica non è LL(k) per nessun k.

Una grammatica per $L_1 = L^R$ si ottiene immmediatamente da quella di L:

$$G_1 = \begin{cases} S \to cSa \mid T \\ T \to bTa \mid \epsilon \end{cases}$$

Verichiamo che G_1 è LL(1).

Calcolo insiemi guida per L_1:

$$\mathcal{G}(S \to cSa) = \mathcal{I}(cSa) = \{c\}$$

$$\mathcal{G}(S \to T) = \mathcal{I}(T) \bigcup \mathcal{S}(S) = \{b\} \cup \{a\} = \{a, b\}$$

$$\mathcal{G}(T \to bTa) = \mathcal{I}(bTa) = \{b\}$$

$$\mathcal{G}(T \to \epsilon) = \mathcal{S}(T) = \{a\}$$

G_1 è LL(1) in quanto gli insiemi guida delle produzioni con la stessa parte sinistra sono disgiunti.

Costruiamo ora l'analizzatore a discesa ricorsiva per G_1.

```
procedure S;
if cc ∈ {c} then cc:= PROSSIMO; /* S → cSa */
                 S;
                 if cc ≠' a' then ERRORE else cc := PROSSIMO endif
           elsif cc ∈ {a, b} then T /* S → T */
           else ERRORE
endif
end S;
```

```
procedure T;
if cc ∈ {b} then cc:= PROSSIMO; /* T → bTa */
                 T;
                 if cc ≠' a' then ERRORE else cc := PROSSIMO endif
           elsif cc ∈ {a} then return /* T → ε */
           else ERRORE
endif
end T;
```

Automa a pila deterministico LL(1), riconoscitore per L_1

L'*alfabeto* della pila è costituito dall'unione dell'alfabeto terminale e nonterminale della grammatica. S è il simbolo iniziale della pila. L'automa ad ogni mossa estrae il simbolo in cima alla pila e, se esplicitamente specificato, avanza la testina di lettura del nastro di ingresso. L'automa riconosce a pila vuota, vale a dire quando non ci sono più simboli sulla pila e sul nastro di ingresso. L'automa si arresta senza accettare quando si trova in una configurazione da cui nessuna mossa è possibile (ad esempio, T sulla pila e c in ingresso). La mossa *push*(ϵ) corrisponde a eseguire una *pop* dalla pila non seguita da una *push*. La mossa *avanza* esegue una *pop* e consuma un simbolo in ingresso, senza eseguire una *push*.

Cima Pila	Carattere di ingresso a	b	c
S	push(T)	push(T)	push(aSc)
T	push(ϵ)	push(aTb)	
a	avanza		
b		avanza	
c			avanza

Esempio di esecuzione sulla stringa ccbbaaaa:

$(S, ccbbaaaa) \vdash (aSc, ccbbaaaa) \vdash (aS, cbbaaaa) \vdash (aaSc, cbbaaaa) \vdash$
$(aaS, bbaaaa) \vdash (aaT, bbaaaa) \vdash (aaaTb, bbaaaa) \vdash (aaaT, baaaa) \vdash$
$(aaaaTb, baaaa) \vdash (aaaaT, aaaa) \vdash (aaaa, aaaa) \vdash (aaa, aaa) \vdash$
$(aa, aa) \vdash (a, a) \vdash (\epsilon, \epsilon)$ accetta

4.2 Esercizio

Si calcolino gli insiemi guida per le produzioni della seguente grammatica:

$$G = \begin{cases} S \to XY \\ X \to aT \mid TT \\ T \to b \mid TX \mid \epsilon \\ Y \to c \mid XY \mid ZX \\ Z \to dZ \mid e \end{cases}$$

Soluzione 4.2

Per potere effettuare il calcolo in modo sistematico sulla grammatica data, conviene dapprima definire un insieme di regole generali, applicabili a qualunque grammatica libera.

Sia G una grammatica con tutti i nonterminali definiti e raggiungibili. Definiamo l'insieme degli inizi $\mathcal{I}(x)$ per un qualunque $\alpha \in (V_N \cup V_T)^*$.

Se $x \in V_T$, allora

$$\mathcal{I}(x) = \{x\}$$

Se $x = X$, $X \in V_N$ e $X \to \alpha \mid \beta \mid \ldots \mid \omega$ sono tutte e sole le produzioni di x, allora

$$\mathcal{I}(X) = \mathcal{I}(\alpha) \cup \ldots \cup \ \mathcal{I}(\omega)$$

Se $x = X_1 \ldots X_n$, $X_i \in (V_N \cup V_T)$, con $1 \leq i \leq n$, allora

$$\mathcal{I}(X_1 \ldots X_n) = \bigcup \{\mathcal{I}(X_i) \mid 1 \leq i \leq n, X_1 \ldots X_{i-1} \Rightarrow^* \epsilon\}$$

In questo modo è possibile costruire un sistema di equazioni, le cui incognite sono gli insiemi degli inizi per ogni nonterminale.

Per la grammatica dell'esercizio, una volta notato che S, X, Y, T sono tutti e soli i nonterminali da cui si può derivare ϵ, si ottiene facilmente il seguente sistema, in cui X sta per $\mathcal{I}(X)$.

$$\begin{cases} S = X \cup Y \\ X = a \cup T \\ T = b \cup T \cup X \\ Y = c \cup X \cup Y \cup Z \\ Z = d \cup e \end{cases}$$

La soluzione di un sistema di questo tipo si può facilmente ottenere per sostituzioni successive e applicando la legge di *assorbimento*: se vale l'equazione:

$$W = W \cup \mathcal{E}$$

dove $\mathcal{E}$ è una qualunque espressione di unione di variabili e insiemi regolari, allora vale l'equazione:

$$W = \mathcal{E}$$

Si noti che in particolare se l'equazione è ridotta al caso degenere $W = W$, allora vale l'equazione $W = \emptyset$.

Ad esempio, nel sistema precedente si possono sostituire le espressioni $X = a \cup T$ e $Z = d \cup e$, ottenendo:

$$\begin{cases} S = X \cup Y \\ X = a \cup T \\ T = b \cup T \cup a \cup T = b \cup a \\ Y = c \cup X \cup Y \cup d \cup e \\ Z = d \cup e \end{cases}$$

da cui si ricava:

$$\begin{cases} S = X \cup Y \\ X = a \cup T = a \cup b \\ T = b \cup a \\ Y = c \cup X \cup Y \cup d \cup e = a \cup b \cup c \cup d \cup e \\ Z = d \cup e \end{cases}$$

da cui anche $S = a \cup b \cup c \cup d \cup e$.

Dal punto di vista teorico, l'esistenza della soluzione cercata è garantita dai classici teoremi di punto fisso. La trasformazione individuata dal sistema è monotona e continua nell'alfabeto terminale e quindi il sistema ammette un unico minimo punto fisso, che costituisce proprio la soluzione cercata. Dalla monotonia della trasformazione deriva immediatamente la validità della legge di assorbimento e la garanzia dell'esistenza della soluzione procedendo opportunamente per applicazioni successive della sostituzione e dell'assorbimento.

Relazioni analoghe a quelle per gli insiemi degli inizi esistono anche per gli insiemi dei seguiti. In particolare:

Se $X \in V_N$ e $Y_1 \to \alpha_1 X \beta_1$, $Y_2 \to \alpha_2 X \beta_2$, …, $Y_n \to \alpha_n X \beta_n$ sono tutte e sole le produzioni di G in cui X compare nella parte destra, allora l'insieme $\mathcal{S}(X)$ dei seguiti è:

$$\mathcal{S}(X) = \bigcup \{\mathcal{I}(\beta_i)\} \cup \bigcup \{\mathcal{S}(Y_i) \mid \beta_i \Rightarrow^* \epsilon\}$$

Il risultato dell'applicazione di queste regole è un sistema di equazioni, dello stesso tipo di quelle viste per calcolare l'insieme degli inizi. Il sistema può essere risolto in base al metodo appena visto. L'insieme guida si può calcolare come al solito, una volta verificato anche quali nonterminali sono annullabili.

Ad esempio, per la grammatica data nel testo dell'esercizio il sistema è:

$$\begin{cases} \mathcal{S}(S) = \swarrow \\ \mathcal{S}(X) = \mathcal{S}(T) \cup \mathcal{S}(Y) \cup \mathcal{I}(Y) \\ \mathcal{S}(T) = \mathcal{I}(X) \cup \mathcal{I}(T) \cup \mathcal{S}(X) \cup \mathcal{S}(T) \\ \mathcal{S}(Y) = \mathcal{S}(S) \\ \mathcal{S}(Z) = \mathcal{I}(X) \cup \mathcal{S}(Y) \end{cases}$$

La soluzione del sistema può essere realizzata sfruttando i valori già calcolati per gli insiemi degli inizi e sfruttando il fatto che $\mathcal{S}(S) = \swarrow$ per ottenere che $\mathcal{S}(Y) = \swarrow$, $\mathcal{S}(Z) = a \cup b \cup \swarrow$, da cui, sostituendo $\mathcal{S}(X)$ in $\mathcal{S}(T)$ si ricava:

$$\mathcal{S}(T) = \mathcal{I}(X) \cup \mathcal{I}(T) \cup \mathcal{S}(T) \cup \mathcal{S}(Y) \cup \mathcal{I}(Y) \cup \mathcal{S}(T)$$

da cui per assorbimento si deduce che:

$$\mathcal{S}(T) = a \cup b \cup c \cup d \cup e \cup \swarrow = \mathcal{S}(X)$$

È ora possibile ricavare gli insiemi guida delle varie produzioni, che riportiamo di seguito.

$$G = \begin{cases} S \to XY\{a \cup b \cup c \cup d \cup e \cup \swarrow\} \\ X \to aT\{a\} \mid TT\{a \cup b \cup c \cup d \cup e \cup \swarrow\} \\ T \to b\{b\} \mid TX\{a \cup b \cup c \cup d \cup e \cup \swarrow\} \mid \epsilon\{a \cup b \cup c \cup d \cup e \cup \swarrow\} \\ Y \to c\{c\} \mid XY\{a \cup b \cup c \cup d \cup e \cup \swarrow\} \mid ZX\{d, e\} \\ Z \to dZ\{d\} \mid e\{e\} \end{cases}$$

La grammatica non è pertanto LL(1).

4.3 Esercizio

Si definisca mediante una grammatica BNF estesa un minilinguaggio di programmazione. Le caratteristiche del linguaggio sono le seguenti:

- Un programma inizia con la parola chiave *prog* e termina con la parola chiave *endprog*;
- un programma è costituito da una o più dichiarazioni di procedura;
- ogni procedura è preceduta dalla parola chiave *procedure*, seguita da un nome e da una dichiarazione di zero, uno o più parametri;
- ogni procedura contiene anche un corpo, individuato dalla coppia *begin* ... *end*;
- non si possono annidare le dichiarazioni di procedura;
- il corpo di una procedura può essere costituito da vari blocchi annidati;

- un blocco inizia con un *begin* e termina con un *end* e può contenere, subito dopo il *begin*, zero o più dichiarazioni di variabili;
- ogni blocco può contenere delle istruzioni *call*, seguite dal nome di una procedura e da una lista di parametri attuali;
- le variabili e i parametri formali possono essere solo di tipo *integer* o *real*;
- si possono usare costanti numeriche come parametri attuali.

Esempio:

```
prog
procedure P1 (I1, I2: integer; F3:real);
begin
XYZ: real;
w,v:integer;
call P2;
begin
call P1(3, 5, 7.4);
end;
end;
procedure P2;
begin
end;
endprog
```

Si progetti almeno in parte l'analizzatore sintattico a discesa ricorsiva.

Soluzione 4.3

Una grammatica BNF estesa per il linguaggio è:

$$\left\{\begin{array}{l}
program \rightarrow \mathbf{prog}(procedureDeclaration";")^{+}\mathbf{endprog} \\
procedureDeclaration \rightarrow \mathbf{procedure}pId(formalParameters \mid \epsilon)";"block \\
formalParameters \rightarrow "("(fId(","fId)^{*}\ ":"(\mathbf{integer} \mid \mathbf{real})";")^{+}\ ")" \\
block \rightarrow \mathbf{begin}(varDeclaration \mid \epsilon)(phrase";")^{*}\mathbf{end} \\
varDeclaration \rightarrow (vId(","vId)^{*}\ ":"(\mathbf{integer} \mid \mathbf{real})";")^{+} \\
phrase \rightarrow (\mathbf{call}pId(actualParameters \mid \epsilon)) \mid block \\
actualParameters \rightarrow "("(aId(","aId)^{*})")" \\
aId \rightarrow identifier \mid constant \\
fId \rightarrow identifier \\
pId \rightarrow identifier
\end{array}\right.$$

I simboli $identifier$ e $constant$ individuano rispettivamente i generici identificatori alfanumerici e le costanti numeriche, ricavati da un opportuno analizzatore lessicale.

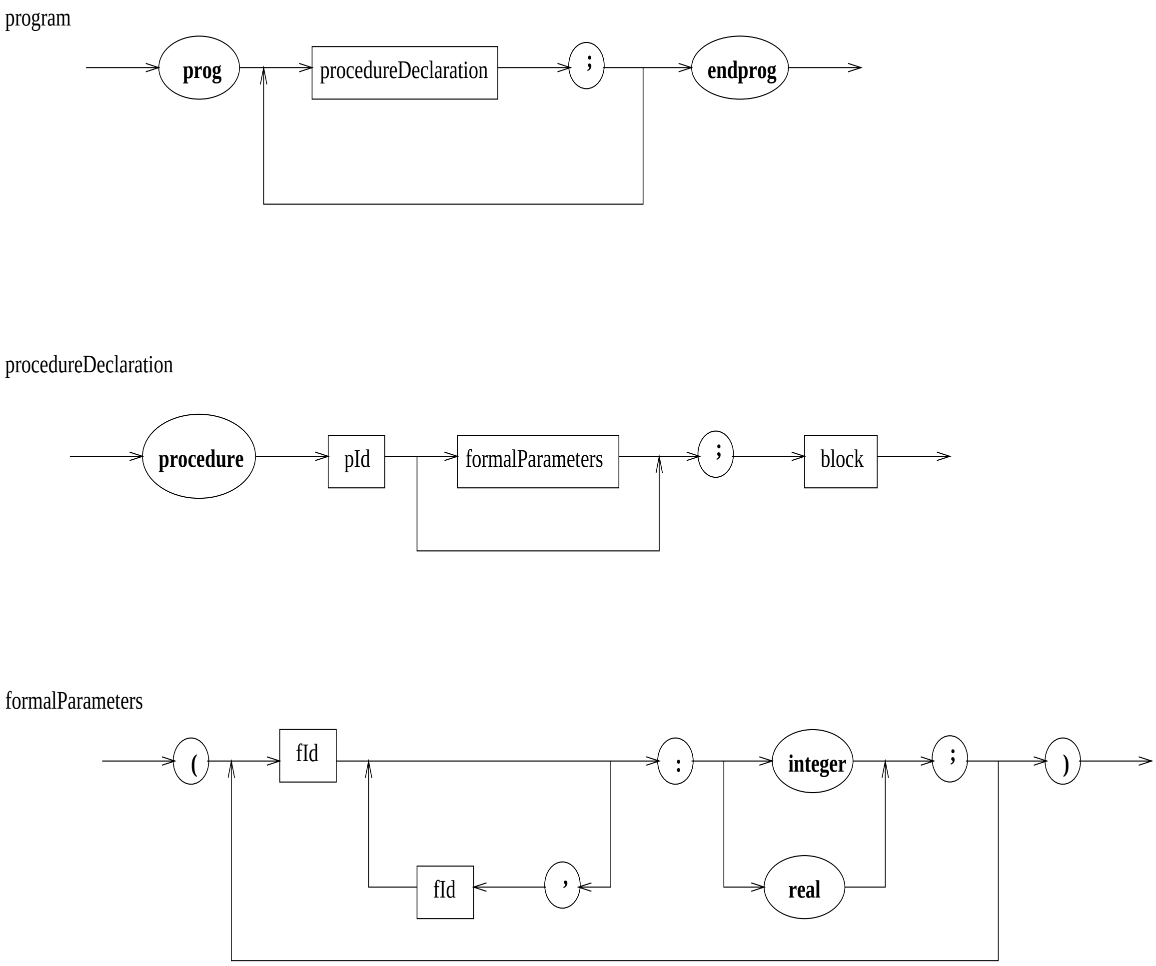

Figure 21: Alcuni diagrammi sintattici corrispondenti alla grammatica dell'esercizio 4.3.

La figura 21 mostra i diagrammi sintattici corrispondenti ad alcune delle dichiarazioni BNF estese della grammatica.

Costruiamo ora le procedure sintattiche relative ai non terminali *program* e *block*.

La regola per *program* soddisfa la condizione di determinismo LL(1) per la sintassi BNF estesa, in quanto $\mathcal{G}(procedureDeclaration';')\bigcap\mathcal{S}((procedureDeclaration';')^{+}) = \{procedure\}\bigcap\{endprog\} = \emptyset$.

Se si suppone che la variabile *cc* contenga identificatori invece di un singolo carattere (come lecito aspettarsi se si usa un opportuno analizzatore lessicale), la procedura sintattica per *program* è:

```
procedure program;
if cc =' prog' then cc:=PROSSIMO else ERRORE endif;
if cc ∈ {procedure} then repeat
                              procedureDeclaration;
                              if cc ≠';' then ERRORE
                                     else cc := PROSSIMO
                              endif
                         until cc ∉ {procedure}
                         if cc ≠' endprog' then ERRORE
                                     else cc := PROSSIMO
                         endif
                    else ERRORE
endif
end program;
```

La regola per *block* soddisfa la condizione di determinismo LL(1) per la sintassi BNF estesa, in quanto:

- per la sottoespressione $(varDeclaration \mid \epsilon)$ si ha:

  $\mathcal{G}(varDeclaration)\bigcap\mathcal{G}(\epsilon) = vId\bigcap\{end, call, begin\} = \emptyset$, poiché $end, cal, begin$ sono parole riservate che non possono essere usate per gli identificatori (in caso contrario, verrebbe segnalato un errore in quanto l'analizzatore lessicale non restituirebbe una variabile);

- per la sottoespressione $(phrase";")^{*}$ si ha:

  $\mathcal{G}(phrase";")\bigcap\mathcal{S}((phrase";")^{*}) = \{call, begin\}\bigcap\{end\} = \emptyset$.

La procedura sintattica è:

```
procedure block;
if cc =' begin' then cc := PROSSIMO;
                    if cc ∈ vId then varDeclaration
```

elsif $cc \notin \{call, begin, end\}$ **then ERRORE**
endif
while $cc \in \{call, begin\}$ **do**
    phrase
enddo;
if $cc =' end'$ **then** cc**:= PROSSIMO else ERRORE endif**
else ERRORE
endif
end program;

4.4 Esercizio

Data la grammatica soluzione dell'esercizio 3.4, si verifichi se la grammatica è di tipo LL(1). Si consideri poi la versione in forma b.n.f. estesa, si verifichi se è di tipo LL(1) e in caso affermativo se ne costruisca l'analizzatore a discesa ricorsiva.

Soluzione 4.4

È immediato verificare che la grammatica non è di tipo LL(k) per nessun k. È possibile trasformarla in una grammatica LL(1), come la seguente, in cui sono riportati anche gli insiemi guida delle produzioni:

$$G = \begin{cases} S \to XY\{p, q, r, or, and, \swarrow\} \\ Y \to \epsilon\{\swarrow\} \mid \textbf{or}\; XY\{or\} \\ X \to AZ\{p, q, r, (, not\} \\ Z \to \epsilon\{or, \swarrow\} \mid \textbf{and}\; AZ\{and\} \\ A \to p \mid q \mid r \mid (S) \mid \textbf{not}\; A \end{cases}$$

La grammatica è scomoda per la costruzione di analizzatori sintattici e semantici integrati (una grammatica ad attributi per valutare il valore delle espressioni o per tradurre in un linguaggio macchina sarebbe notevolmente più complicata del necessario).

Per costruire analizzatori sintattico-semantici integrati a discesa ricorsiva si preferisce allora usare, per questo tipo di linguaggi, grammatiche in forma b.n.f. estesa, come ad esempio la seguente grammatica G'.

$$G' = \begin{cases} S \to D\; (\textbf{or}\; D)^* \\ D \to C\; (\textbf{and}\; C)^* \\ C \to\; \textbf{not}^*\; A \\ A \to p \mid q \mid r \mid\; '('\; S\; ')' \end{cases}$$

G' è LL(1). Per la produzione $S \to D\; (\textbf{or}\; D)^*$:
$\mathcal{G}(\textbf{or} D) \bigcap \mathcal{S}((\textbf{or}\; D)^*) = \{or\} \bigcap \{\swarrow,)\} = \emptyset$.
Per la produzione $D \to C\; (\textbf{and}\; C)^*$:

$\mathcal{G}(\textbf{ and } C) \bigcap \mathcal{S}((\textbf{and} D)^*) = \{and\} \bigcap \{\swarrow,)\} = \emptyset$.

La condizione di determinismo LL(1) è banalmente verificata dalle altre produzioni.

La procedura sintattica per S è:

procedure S;
D;
while $cc = ' or'$ **do**
    cc **:= PROSSIMO;**
    D;
enddo;
if $cc \notin \{\swarrow,)\}$ **then ERRORE endif**
end S;

La procedura sintattica per D è:

procedure D;
C;
while $cc = ' and'$ **do**
    cc **:= PROSSIMO;**
    C;
enddo;
if $cc \notin \{\swarrow,)\}$ **then ERRORE endif**
end D;

La procedura sintattica per C è:

procedure C;
while $cc = ' not'$ **do**
    cc **:= PROSSIMO;**
enddo;
A;
end C;

Lasciamo al lettore il completamento dell'analizzatore discendente con la procedura per il nonterminale A.

4.5 Esercizio

Data la grammatica:

$$G = \begin{cases} S \to SB \mid bB \mid \epsilon \\ B \to dS \mid SBe \end{cases}$$

1. Si eliminino le ricorsioni sinistre trasformandole in ricorsioni destre.
2. Detta G' la grammatica equivalente a G così ottenuta, se ne calcolino gli insiemi guida verificando se essa è LL.

Soluzione 4.5

Nella grammatica vi sono due ricorsioni sinistre, $S \to SB$ e $B \stackrel{2}{\to} Be$. Per eliminare le ricorsioni sinistre bisogna ricorrere all'algoritmo descritto in [Hopcroft 1969].

Conviene evitare di avere ϵ-regole associate al non-terminale di cui si deve eliminare la ricorsione sinistra, allo scopo di ridurre il numero di passi richiesto dall'algoritmo. Per questo si introduce un nuovo simbolo S_0 che manterrà inalterato il linguaggio eliminando la produzione critica. La grammatica diventerà:

$$G = \begin{cases} S_0 \to S \mid \epsilon \\ S \to SB \mid bB \mid B \\ B \to dS \mid d \mid SBe \mid Be \end{cases}$$

L'algoritmo opera nella maniera seguente: Siano $A \to A\beta_1 \mid A\beta_2 \mid \ldots \mid A\beta_n$, dove nessun β_i o γ_jè annullabile, le alternative immediatamente ricorsive a sinistra di A, e siano $A \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m$ le rimanenti alternative per A.

Introducendo un nuovo non-terminale A' e scrivendo le seguenti regole:

$$\begin{array}{l} A \to \gamma_1 A' \mid \gamma_2 A' \mid \ldots \mid \gamma_m A' \\ A' \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A' \mid \epsilon \end{array}$$

si ottiene la grammatica equivalente ricorsiva a destra nel non-terminale A' (la ϵ-produzione non crea problemi perchè A' non può essere ricorsivo a sinistra). Si possono quindi eliminare tutte le ricorsioni sinistre immediate. Le ricorsioni sinistre non immediate vengono poi eliminate con lo stesso procedimento, trasformandole dapprima in immediate.

Applichiamo separatamente l'algoritmo sopra illustrato ai due non-terminali S e B, aggiungendo due non-terminali S' e B'. Otteniamo G':

$$G' = \begin{cases} S_0 \to S \mid \epsilon \\ S \to bBS' \mid BS' \\ S' \to BS' \mid \epsilon \\ B \to dSB' \mid dB' \mid SBeB' \\ B' \to eB' \mid \epsilon \end{cases}$$

In G' compare ancora una ricorsione sinistra nel termine B: $B \to SBeB' \to BS'BeB'$. Per eliminare anche questa ricorsione si può sostituire il termine S nella produzione critica di B, eliminandola senza alterare il linguaggio. Si ottiene in questo modo una ulteriore produzione immediatamente ricorsiva a sinistra che si può gestire con il precedente algoritmo.

La produzione del non-terminale B sarà:

$$B \to dSB' \mid dB' \mid bBS'BeB' \mid BS'BeB'$$

Applicando l'algoritmo introduciamo un nuovo simbolo B'' ottenendo la grammatica:

$$G'' = \begin{cases} S_0 \to S \mid \epsilon \\ S \to bBS' \mid BS' \\ S' \to BS' \mid \epsilon \\ B \to dSB'B'' \mid dB'B'' \mid bBS'BeB'B'' \\ B' \to eB' \mid \epsilon \\ B'' \to S'BeB'B'' \cup \epsilon \end{cases}$$

Tutte le ricorsioni sinistre sono state eliminate.

Per determinare se la grammatica è $LL(1)$ costruiamo gli insiemi guida dei non-terminali.

$$G'' = \begin{cases} S_0 \to S\{b, d\} \mid \epsilon\{\swarrow\} \\ S \to bBS'\{b\} \mid BS'\{b, d\} \\ S' \to BS'\{b, d\} \mid \epsilon\{b, d, e, \swarrow\} \\ B \to dSB'B''\{d\} \mid dB'B''\{d\} \mid bBS'BeB'B''\{b\} \\ B' \to eB'\{e\} \mid \epsilon\{b, d, e\} \\ B'' \to S'BeB'B'' \cup \epsilon \end{cases}$$

Gli insiemi guida delle regole di produzione associate al non-terminale S (come per i non-terminali S', B e B') non hanno un'intersezione nulla, quindi la grammatica non è di tipo $LL(1)$.

4.6 Esercizio

Si definisca mediante una sintassi il linguaggio delle spezzate mostrate nella figura 22.

L'alfabeto terminale è costituito dai simboli: $\Sigma = \{\nearrow, \to, \searrow\}$

La spezzata si compone di segmenti crescenti, orizzontali e decrescenti; ogni segmento è costituito da un numero intero di vettori unitari. Il primo segmento è crescente; l'ultimo è qualsiasi.

Tra un segmento crescente e uno decrescente vi può essere un segmento orizzontale di lunghezza arbitraria. La lunghezza di un segmento crescente è maggiore o eguale a quella del segmento decrescente successivo. Un segmento decrescente può mancare.

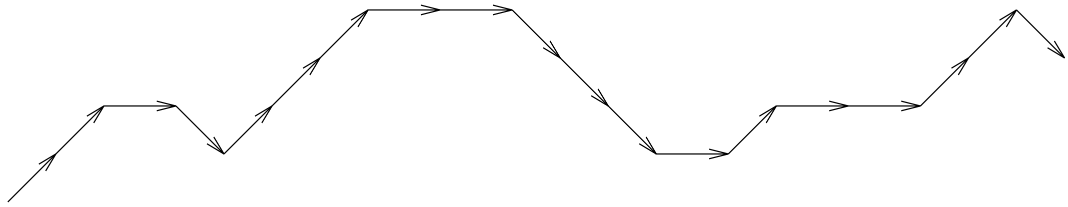

Figure 22: Le spezzate del linguaggio L

1. Si verifichi se la sintassi è $LL(1)$ calcolando gli insiemi guida
2. Si cerchi di costruire una grammatica $LL(1)$ per il linguaggio. Qualora questo non sia possibile si modifichi il linguaggio in modo tale da poter definire una grammatica $LL(1)$

Soluzione 4.6

La grammatica si può costruire passo passo: la spezzata si può scomporre in un insieme di segmenti, ognuno caratterizzato da un tratto crescente seguito da un tratto decrescente, con in mezzo eventualmente un tratto pianeggiante.

Il singolo segmento può essere rappresentato dalle seguenti produzioni:

$$\begin{cases} A \Longrightarrow \nearrow A \mid \nearrow A \searrow \mid B \\ B \Longrightarrow \rightarrow B \mid \epsilon \end{cases}$$

La sequenza di segmenti potrà essere rappresentata dalla produzione:

$$S \Longrightarrow AS \mid \epsilon$$

Inoltre, il primo segmento deve iniziare con un tratto crescente, per cui l'assioma del linguaggio sarà il non-terminale S_0 cui corrisponderà la produzione:

$$S_0 \Longrightarrow \nearrow S \mid \nearrow S \searrow$$

La grammatica che descrive il linguaggio sarà la seguente G:

$$G = \begin{cases} S_0 \Longrightarrow \nearrow S \mid \nearrow S \searrow \\ S \Longrightarrow AS \mid \epsilon \\ A \Longrightarrow \nearrow A \mid \nearrow A \searrow \mid B \\ B \Longrightarrow \rightarrow B \mid \epsilon \end{cases}$$

La grammatica risulta ambigua. Infatti la stringa $\nearrow\rightarrow\rightarrow$ ammette le due derivazioni $S_0 \Longrightarrow \nearrow S \Longrightarrow \nearrow AS \Longrightarrow \nearrow AAS \Longrightarrow \nearrow AA \Longrightarrow \nearrow BA \Longrightarrow \nearrow BB \Longrightarrow \nearrow\rightarrow B \Longrightarrow \nearrow\rightarrow\rightarrow$ e $S_0 \Longrightarrow \nearrow S \Longrightarrow \nearrow A \Longrightarrow \nearrow B \Longrightarrow \nearrow\rightarrow B \Longrightarrow \nearrow\rightarrow\rightarrow B \Longrightarrow \nearrow\rightarrow\rightarrow$

Un'altra sorgente di ambiguità è generata dal fatto che se il numero di tratti decrescenti è inferiore al numero di tratti crescenti, vi sono diversi possibili modi in cui le diverse produzioni di A possono essere utilizzate. Per eliminare la prima ambiguità è necessario distinguere il caso in cui una sequenza di tratti orizzontali compare tra un tratto crescente e un tratto decrescente dal caso in cui non è seguita da un tratto decrescente. Nel primo caso la sequenza verrà generata dal non-terminale B, mentre nel secondo verrà generata dal non-terminale S. La seconda ambiguità si risolve introducendo un nuovo simbolo per A. La grammatica non ambigua che si ottiene è la seguente:

$$G' = \begin{cases} S_0 \Longrightarrow \nearrow S \mid \nearrow S \searrow \\ S \Longrightarrow A_1 S \mid \rightarrow S \mid \epsilon \\ A_1 \Longrightarrow \nearrow A_1 \mid \nearrow A_2 \searrow \\ A_2 \Longrightarrow \nearrow A_2 \searrow \mid B \\ B \Longrightarrow \rightarrow B \mid \epsilon \end{cases}$$

Calcolando gli insiemi guida associati ai vari non terminali, si osserva come la grammatica non sia $LL(1)$. Un tentativo di renderla $LL(1)$ applicando una fattorizzazione sinistra produce il seguente risultato:

$$G'' = \begin{cases} S_0 \Longrightarrow \nearrow S\{\nearrow\}(\searrow \mid \epsilon)\{\searrow, \swarrow\} \\ S \Longrightarrow A_1 S\{\nearrow\} \mid \rightarrow S\{\rightarrow\} \mid \epsilon\{\swarrow\} \\ A_1 \Longrightarrow \nearrow A_1\{\nearrow\} \mid \nearrow A_2 \searrow \{\nearrow\} \\ A_2 \Longrightarrow \nearrow A_2 \searrow \{\nearrow\} \mid B\{\rightarrow, \searrow\} \\ B \Longrightarrow \rightarrow B\{\rightarrow\} \mid \epsilon\{\searrow\} \end{cases}$$

Tutti i non terminali tranne A_1 sono $LL(1)$. Il problema consiste nel valutare se è possibile rendere A_1 $LL(1)$. Ciò non è possibile. Infatti se si analizza il linguaggio si vede come sia impossibile per il riconoscitore a discesa ricorsiva sapere se è corretto scegliere la prima o la seconda produzione guardando solo l'elemento successivo. Pure un riconoscitore $LL(k)$ non sarà in grado di riconoscere il linguaggio: basterà infatti prendere una stringa che contenga un segmento di lunghezza $k+1$ per non essere in grado di decidere quale delle due produzioni si debba scegliere.

Si può rendere il linguaggio $LL(1)$ ad esempio imponendo che il numero di tratti crescenti sia pari al numero di tratti decrescenti. Questo corrisponde alla grammatica $LL(1)$ seguente:

$$G''' = \begin{cases} S_0 \Longrightarrow \nearrow SS'\{\nearrow\} \\ S' \Longrightarrow \searrow \{\searrow\} \mid \epsilon\{\swarrow\} \\ S \Longrightarrow \nearrow A \searrow S\{\nearrow\} \mid \rightarrow S\{\rightarrow\} \mid \epsilon\{\swarrow\} \\ A \Longrightarrow \nearrow A \searrow \{\nearrow\} \mid B\{\rightarrow, \searrow\} \\ B \Longrightarrow \rightarrow B\{\rightarrow\} \mid \epsilon\{\searrow\} \end{cases}$$

Il riconoscitore a pila della grammatica $LL(1)$ è caratterizzato dal seguente comportamento (nella tabella sono sottointese le mosse di avanzamento testina quando il terminale in ingresso è identico al terminale sulla cima della pila):

Cima Pila	Carattere di ingresso			
	$\nearrow$	$\rightarrow$	$\searrow$	$\swarrow$
S_0	push $S'S \nearrow$			
S'			push $\searrow$	push ϵ
S	push $S \searrow A$	push $S \rightarrow$		push ϵ
A	push $\searrow A \nearrow$	push B	push B	
B		push $B \rightarrow$	push ϵ	

4.7 Esercizi proposti

4.7.1 Esercizio

Verificare se la grammatica G:

$$G = \begin{cases} S \rightarrow aSb \mid C \\ C \rightarrow aD \mid \epsilon \\ D \rightarrow cC \end{cases}$$

è LL(1). Nel caso negativo, cercare di trasformarla in una grammatica LL equivalente.

4.7.2 Esercizio

Si consideri l'insieme L delle sequenze degli eventi associati ad un insieme di operazioni di lettura e di scrittura, secondo il modello di concorrenza tra processi noto come modello dei lettori-scrittori. Più lettori possono operare contemporaneamente, ma un solo scrittore può essere attivo, e soltanto se non vi sono lettori attivi. Inoltre ogni lettura deve finire, e così ogni scrittura. Infine una lettura o scrittura non può finire se non è iniziata. Si indichino gli eventi con i seguenti simboli terminali:

il inizio lettura

fl fine lettura

is inizio scrittura

fs fine scrittura

Ad. es. sono valide le stringhe:

$$il\, il\, fl\, il\, fl\, fl, \quad il\, il\, fl\, fl\, is\, fs\, is\, fs\, il\, fl$$

mentre sono scorrette le stringhe:

$$il\, is\, fs\, il\, fl\, fl, \quad is\, fs\, is\, il\, fl\, fs, \quad il\, fl\, il, \quad fs\, il\, fl$$

1. Si costruisca una grammatica G BNF-estesa per L e si traccino i diagrammi sintattici.

2. Si verifichi se G è LL(1) calcolando gli insiemi guida.

3. Se necessario si modifichi G in modo da renderla LL(1). Si scrivano almeno in parte le procedure sintattiche dell'analizzatore a discesa ricorsiva.

5 Analisi sintattica ascendente

5.1 Esercizio

Costruire un automa a spostamento e riduzione LR(0) per la seguente grammatica:

$$G = \begin{cases} S \to aTb \mid c \\ T \to DSb \mid c \\ D \to a \end{cases}$$

Si illustri il funzionamento del riconoscitore sulla stringa acb.

Soluzione 5.1

Costruiamo direttamente il riconoscitore dei prefissi ascendenti LR(0) in forma *deterministica.* Si aggiunge alla grammatica G la produzione $S_0 \to S \swarrow$. Lo stato I_0 in questo caso contiene una candidata di spostamento per ogni produzione, ma nessuna candidata di riduzione, non essendoci ϵ-produzioni.

I_0 contiene pertanto le candidate:

$$I_0 \begin{cases} S_0 \to \bullet S \swarrow \\ S \to \bullet aTb \\ S \to \bullet c \end{cases}$$

L'automa viene costruito a partire da I_0, considerando tutte le possibili mosse, costituite da terminali o non terminali, che possono provocare uno spostamento. Iterando il procedimento ai nuovi stati così ottenuti, si costruisce l'automa costituito da tutti e soli gli stati raggiungibili da I_0.

Ad esempio, da I_0 ci possono essere solo tre mosse che possono provocare uno spostamento, in quanto ci sono solo tre caratteri distinti immediatamente a destra di un $\bullet$: a, c, S.

Con un arco etichettato S possiamo perciò andare in uno stato I_1 che contiene solo la candidata di spostamento $S_0 \to S\bullet \swarrow$. Con un arco etichettato a (analogamente per c) possiamo andare in uno stato I_2 con le seguenti candidate (tutte di spostamento):

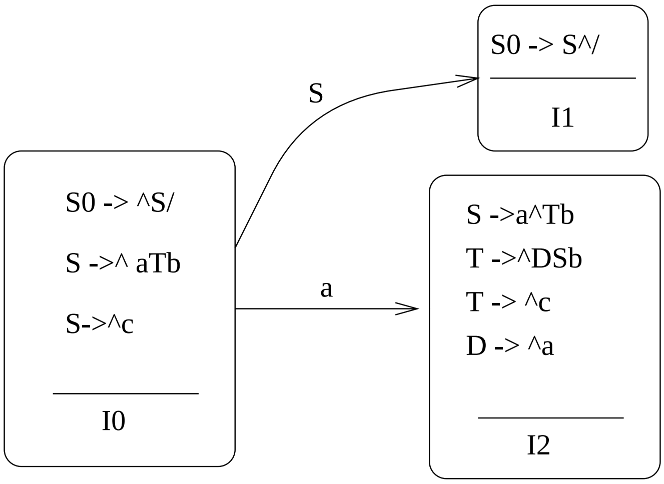

Figure 23: Costruzione dell'automa riconoscitore dei prefissi ascendenti per l'esercizio 5.1.

$$I_2 \begin{cases} S \to a \bullet Tb \\ T \to \bullet DSb \\ T \to \bullet c \\ D \to \bullet a \end{cases}$$

Le candidate $T \to \bullet DSb$, $T \to \bullet c$ e $D \to \bullet a$ devono essere riportate in I_2 in quanto la candidata $S \to a \bullet Tb$, ottenuta da I_0, ha un T immediatamente a destra di un $\bullet$. Inoltre la candidata, in I_2, $T \to \bullet DSb$ ha un D immediatamente a destra di un $\bullet$. Si veda la figura 23.

Iterando il procedimento si ottiene l'automa riportato in figura 24. Quando uno stato è di riduzione, non si deve più continuare a espanderlo e lo si marca opportunamente.

Automa LR(0) a spostamento e riduzione

La *riduzione* di una produzione $X \to x$ per il riconoscitore LR(0) consiste nelle seguenti mosse:

- Non leggere l'ingresso (ϵ-mossa);
- Togli i primi $2\times \mid x \mid$ simboli in cima alla pila;
- Se I è lo stato in cima alla pila, allora $push(X)$ e $push(\delta(I, X))$, dove δ è la funzione di transizione dell'automa riconoscitore dei prefissi ascendenti LR(0).

Lo *spostamento* di un simbolo I consiste nelle seguenti mosse:

- Consuma il simbolo z in ingresso;
- Non togliere simboli dalla pila;

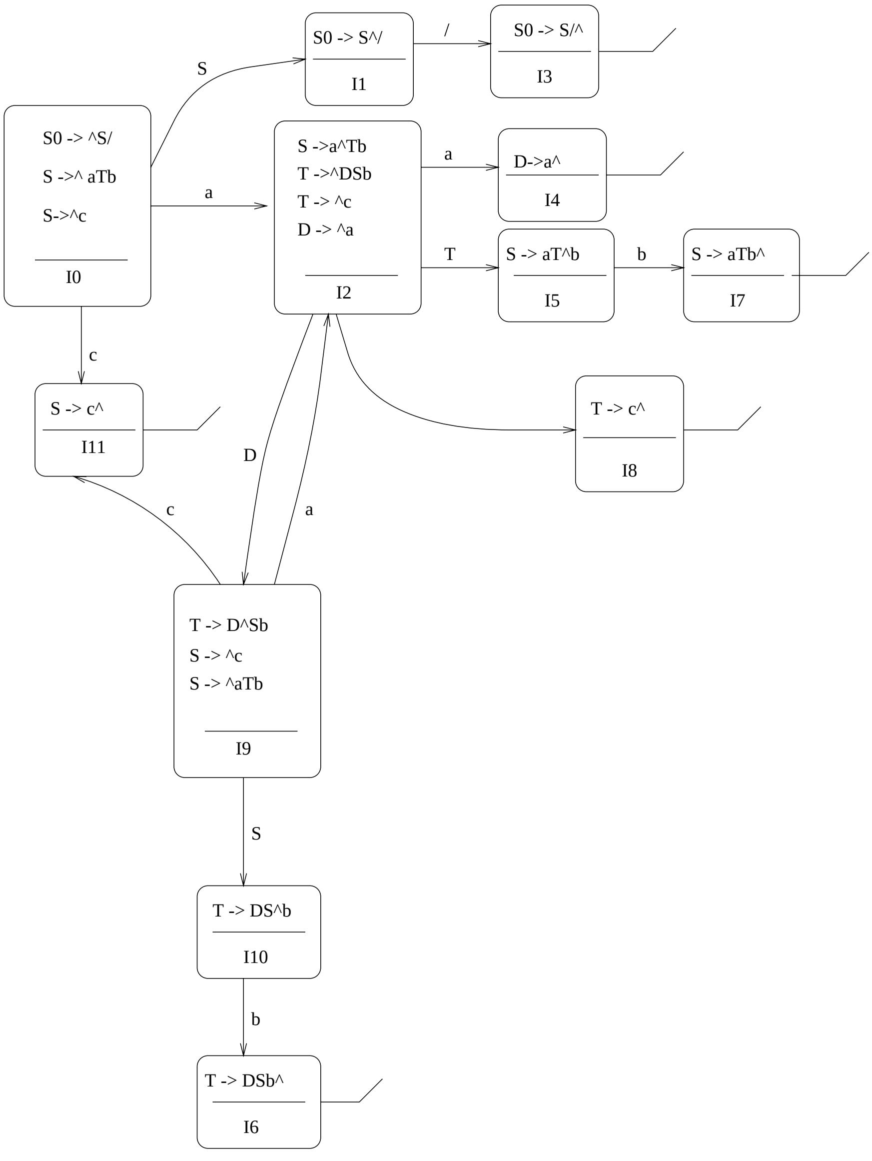

Figure 24: L'automa riconoscitore dei prefissi ascendenti LR(0) per l'esercizio 5.1.

Cima Pila	Carattere di ingresso				
	a	b	c	$\swarrow$	ϵ
I_0	Sposta I_2		Sposta I_{11}		
I_1				Sposta I_3	
I_2	Sposta I_4		Sposta I_8		
I_3					accetta
I_4					riduci $D \to a$
I_5		Sposta I_7			
I_6					riduci $T \to DSb$
I_7					riduci $S \to aTb$
I_8					riduci $T \to c$
I_9	Sposta I_2	Sposta I_{11}			
I_{10}		Sposta I_6			
I_{11}					riduci $S \to c$

Table 1: L'automa a spostamento e riduzione per l'esercizio 5.1. Per l'esecuzione delle mosse di riduzione l'automa considera la funzione di transizione δ definita dall'automa di figura 24.

- $push(z)$ e $push(I)$.

La *accettazione* avviene quando non ci sono più simboli sul nastro di ingresso e sulla pila c'è il solo stato corrispondente alla candidata di riduzione $S_0 \to S \swarrow \bullet$.

L'analizzatore parte con I_0 sulla pila e la frase da riconoscere sul nastro di ingresso. Durante il funzionamento l'analizzatore esegue azioni di spostamento e di riduzione, salvando o eliminando dalla cima della pila l'appropriata sequenza di identificatori dello stato dell'automa riconoscitore dei prefissi ascendenti, intervallati con gli opportuni simboli terminali e non terminali che costituiscono il prefisso corrente. L'automa a pila a spostamento e riduzione LR(0) è descritto dalla Tabella 1.

Il funzionamento dell'automa sulla frase *acb* è riportato nella Tabella 2.

Tradizionalmente, la tabella per l'analizzatore a spostamento e riduzione è completata da una seconda sezione, detta parte "goto". In essa vengono rappresentate le transizioni del riconoscitore dei prefissi ascendenti che sono marcate con un non terminale (e che quindi non corrispondono a un carattere in ingresso), allo scopo di dare una rappresentazione completa in una sola tabella dell'automa a spostamento e riduzione e del riconoscitore dei prefissi ascendenti. L'analizzatore ha infatti bisogno di conoscere la funzione di transizione del riconoscitore per potere eseguire correttamente le mosse di riduzione. Riportiamo in Tabella 3 questa versione, a titolo di esempio.

Pila	Nastro di ingresso	azione
I_0	$a \quad c \quad b \quad \swarrow$	sposta I_2
$I_0 \quad a \quad I_2$	$c \quad b \quad \swarrow$	sposta I_8
$I_0 \quad a \quad I_2 \quad c \quad I_8$	$b \quad \swarrow$	riduci $T \rightarrow c$
$I_0 \quad a \quad I_2 \quad T \quad I_5$	$b \quad \swarrow$	sposta I_7
$I_0 \quad a \quad I_2 \quad T \quad I_5 \quad b \quad I_7$	$\swarrow$	riduci $S \rightarrow aTb$
$I_0 \quad S \quad I_1 \quad \swarrow \quad I_3$	ϵ	riduci $S_0 \rightarrow S \swarrow$ (accetta)

Table 2: Traccia del funzionamento dell'automa di Tabella 1 sulla stringa acb.

Cima Pila	Carattere di ingresso					Parte Goto		
	a	b	c	$\swarrow$	ϵ	S	T	D
I_0	I_2		I_{11}			I_1		
I_1				I_3				
I_2	I_4		I_8				I_5	I_9
I_3					accetta			
I_4					riduci $D \rightarrow a$			
I_5		I_7						
I_6					riduci $T \rightarrow DSb$			
I_7					riduci $S \rightarrow aTb$			
I_8					riduci $T \rightarrow c$			
I_9	I_2		I_{11}			I_{10}		
I_{10}		I_6						
I_{11}					riduci $S \rightarrow c$			

Table 3: Rappresentazione sintetica dell'analizzatore ascendente per l'esercizio 5.1.

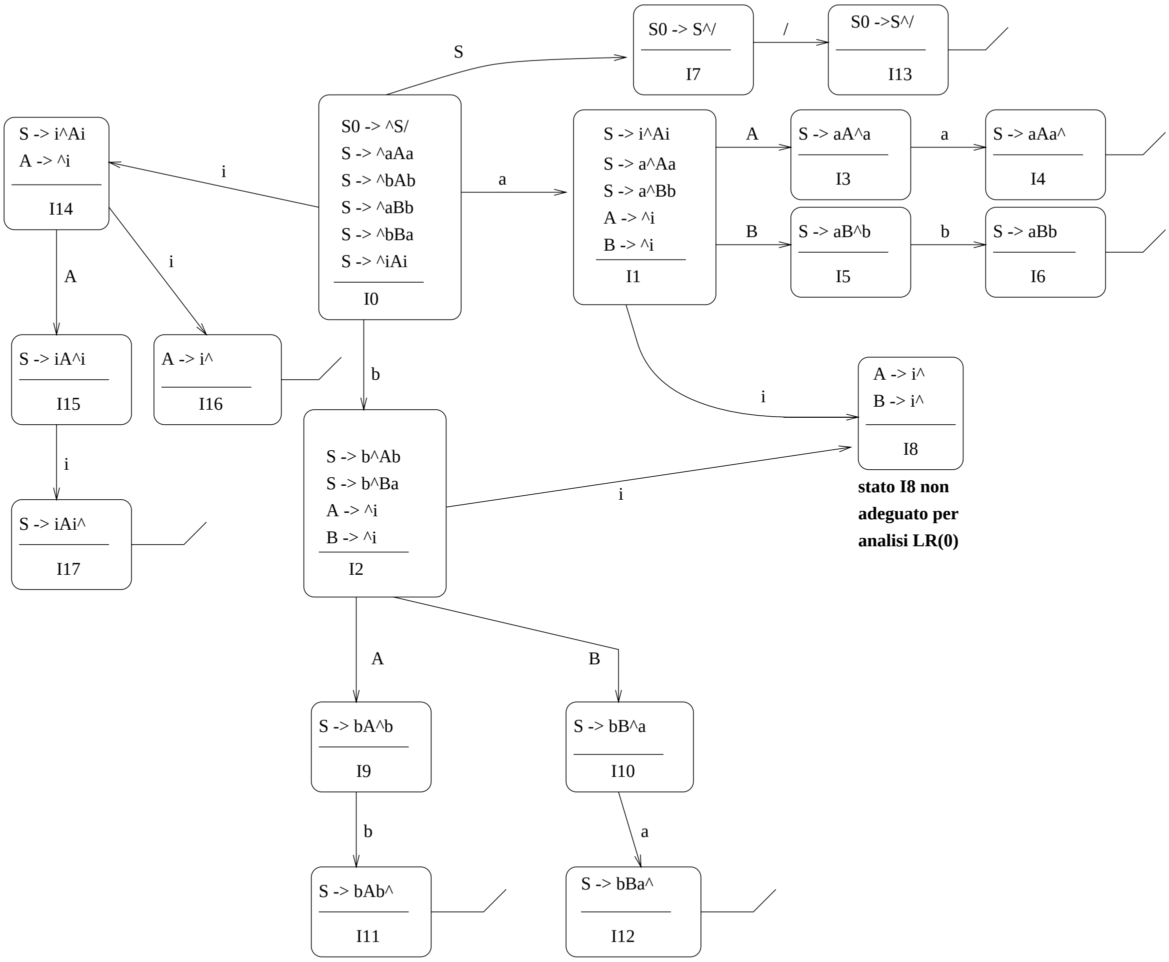

Figure 25: L'automa riconoscitore dei prefissi ascendenti LR(0) dell'esercizio 5.2.

5.2 Esercizio

Verificare se la seguente grammatica

$$G = \begin{cases} S \to iAi \mid aAa \mid bAb \mid aBb \mid bBa \\ B \to i \\ A \to i \end{cases}$$

è LR(0), SLR(1), LALR(1), LR(1). Si costruisca inoltre l'automa a spostamento e riduzione per $L(G)$, guidato dal riconoscitore dei prefissi ascendenti LR(1), e se ne illustri il funzionamento sulla frase $aib \swarrow$.

Soluzione 5.2

L'automa riconoscitore dei prefissi ascendenti LR(0) è riportato in figura 25.

L'automa risultante è però inadeguato per l'analisi LR(0): infatti, nello stato I_8 ci sono *due* candidate di riduzione (un automa a spostamento e riduzione guidato dal riconoscitore dei prefissi ascendenti non potrebbe essere deterministico, in quanto altrimenti, giunto nella situazione I_8, non saprebbe quale mossa di riduzione scegliere).

Analisi SLR(1).

Si considera ancora il riconoscitore LR(0), ma si calcola l'insieme dei seguiti di X per ogni produzione $X \to \alpha$ per le candidate di riduzione $X \to \alpha\bullet$ negli stati inadeguati. L'insieme così calcolato è usato dal riconoscitore SLR(1) come insieme di prospezione per ciascuna candidata di riduzione.

Per lo stato I_8 dobbiamo calcolare i due insiemi: $SLR(A \to i) = \mathcal{S}(A) = \{x \in \Sigma \mid S \Rightarrow^*_{destra} \gamma Axz$, con $z \in \Sigma^*$ e $\gamma \in (\Sigma \bigcup V_N)^*\} = \{a, b, i\}$ (A compare nella parte destra solo per le produzioni $S \to iAi \mid aAa \mid bAb$).

$SLR(B \to i) = \{a, b\}$

La marca finale $\swarrow$ in questo caso non appartiene a nessuno dei due insiemi (in generale, va esplicitamente considerata, inserendola nell'insieme di prospezione qualora $xz = \epsilon$).

La prima condizione di adeguatezza SLR(1) è che non ci siano candidate di spostamento del tipo $X \to \alpha\bullet c\theta$, con $c \in SLR(A \to i) \bigcup SLR(B \to i)$. Questa è banalmente verificata in quanto non ci sono candidate di spostamento in I_8. La seconda condizione, vale a dire che gli insiemi di prospezione delle varie candidate di riduzione siano disgiunti, non è però verificata, e pertanto la grammatica non è adeguata per l'analisi SLR(1).

Analisi LALR(1). Nell'analisi LALR(1) si considera ancora il riconoscitore dei prefissi ascendenti LR(0) e si esaminano solo gli stati inadeguati, allo stesso modo dell'analisi SLR(1), ma l'insieme dei seguiti per ogni candidata di riduzione viene calcolato tenendo conto anche del cammino dallo stato iniziale, che deve portare proprio allo stato inadeguato. L'insieme di prospezione in generale non dipende solo dalla candidata o dalla sua parte sinistra, ma anche dal modo in cui si è arrivati a considerare la candidata. In formule, l'insieme di prospezione per la candidata $A \to i\bullet$ nello stato I_8 è definito come:

$LALR(I_8, A \to i) = \{x \in \Sigma \mid S \Rightarrow^*_{destra} \gamma Axz \Rightarrow \gamma ixz$, con $\delta(I_0, \gamma i) = I_8\} = \{a, b\}$.

In questo caso semplice possiamo utilizzare direttamente la definizione. Infatti, ci sono le derivazioni:

$S \Rightarrow aAa \Rightarrow aia$, con $\delta(I_0, ai) = I_8$

$S \Rightarrow bAb \Rightarrow bib$, con $\delta(I_0, bi) = I_8$

mentre la derivazione $S \Rightarrow iAi \Rightarrow iii$ ha il prefisso i che non porta in I_8: $\delta(I_0, ai) = I_{14}$.

Analogamente, $LALR(I_8, B \to i) = \{a, b\}$.

Nonostante l'analisi LALR sia risultata più raffinata dell'analisi SLR, poichè il terminale i fa parte dell'insieme $SLR(A \to i)$ ma non di $LALR(I_8, A \to i)$, la grammatica non è nemmeno LALR. La condizione di adeguatezza per $LALR(1)$ è che gli insiemi di prospezione delle varie candidate di riduzione siano disgiunti, e che inoltre $LALR(I_8, A \to i) \bigcap \{c \mid \delta(I_8, c)$ è definito $\} = \emptyset$. Quest'ultima parte è banalmente verificata in quanto non ci sono candidate di spostamento in I_8 e pertanto non ci sono nemmeno archi uscenti da I_8, ma la prima parte non lo è, e pertanto la grammatica non è adeguata per l'analisi LALR(1).

Analisi LR(1).

Nell'analisi LR(1) si deve tenere conto, nella *costruzione* dell'automa riconoscitore dei prefissi ascendenti, anche degli insiemi di prospezione associati a ciascuna candidata. La stessa candidata può presentarsi in uno stesso stato con insiemi di prospezione diversi: in generale, l'automa a stati finiti così ottenuto ha un numero di stati molto maggiore di quello dell'automa LR(0) per la stessa grammatica.

L'insieme di prospezione è definito come per LALR(1), ma il metodo LR(1) si differenzia da LALR(1) poiché gli insiemi di prospezione sono usati anche per costruire l'automa riconoscitore dei prefissi ascendenti.

La costruzione dell'automa procede dallo stato I_0 che contiene le candidate:

$$I_0 \begin{cases} S_0 \to S \swarrow \\ S \to \bullet aAa\{\swarrow\} \\ S \to \bullet bAb\{\swarrow\} \\ S \to \bullet iAi\{\swarrow\} \\ S \to \bullet aBb\{\swarrow\} \\ S \to \bullet bBa\{\swarrow\} \end{cases}$$

Si procede come per l'automa LR(0), ma differenziando gli stati anche a seconda degli insiemi di prospezione delle candidate. Non conviene utilizzare la definizione dell'insieme di prospezione, ma le usuali regole di calcolo, in forma deterministica.

Ad esempio, un arco uscente da I_0 marcato a deve portare in uno stato I_1 (quindi $\delta(I_0, a) = I_1$), in cui ci sono le due candidate:

$$I_1 \begin{cases} S \to a \bullet Aa\{\swarrow\} \\ S \to a \bullet Bb\{\swarrow\} \end{cases}$$

In I_1 dobbiamo però considerare anche le candidate $A \to \bullet i$ e $B \to \bullet i$. Poichè la candidata $A \to \bullet i$ deriva dalla candidata $S \to a \bullet Aa\{\swarrow\}$, l'insieme di prospezione è costituito dal simbolo a destra della A nella candidata, cioè il solo a (la marca $\swarrow$ non fa parte dell'insieme per $A \to \bullet i$ in quanto a non può andare in ϵ).

Analogamente, l'insieme di prospezione per $B \to \bullet i$ è costituito dal solo simbolo terminale b, in quanto la candidata è stata ricavata da $S \to a \bullet Bb\{\swarrow\}$.

L'automa risultante è riportato in figura 26.

Si osservi che ora lo stato I_8 del riconoscitore LR(0) viene distinto, nell'automa LR(1), nei due stati I_8 e I_7, che sono entrambi adeguati per LR(1), in quanto in entrambi i casi gli insiemi di prospezione delle candidate di riduzione sono disgiunti. L'analisi LALR(1) non aveva permesso questa distinzione in quanto considerava il riconoscitore LR(0).

Costruzione automa a spostamento e riduzione LR(1). Costruiamo ora l'automa a spostamento e riduzione guidato dal riconoscitore dei prefissi ascendenti LR(1).

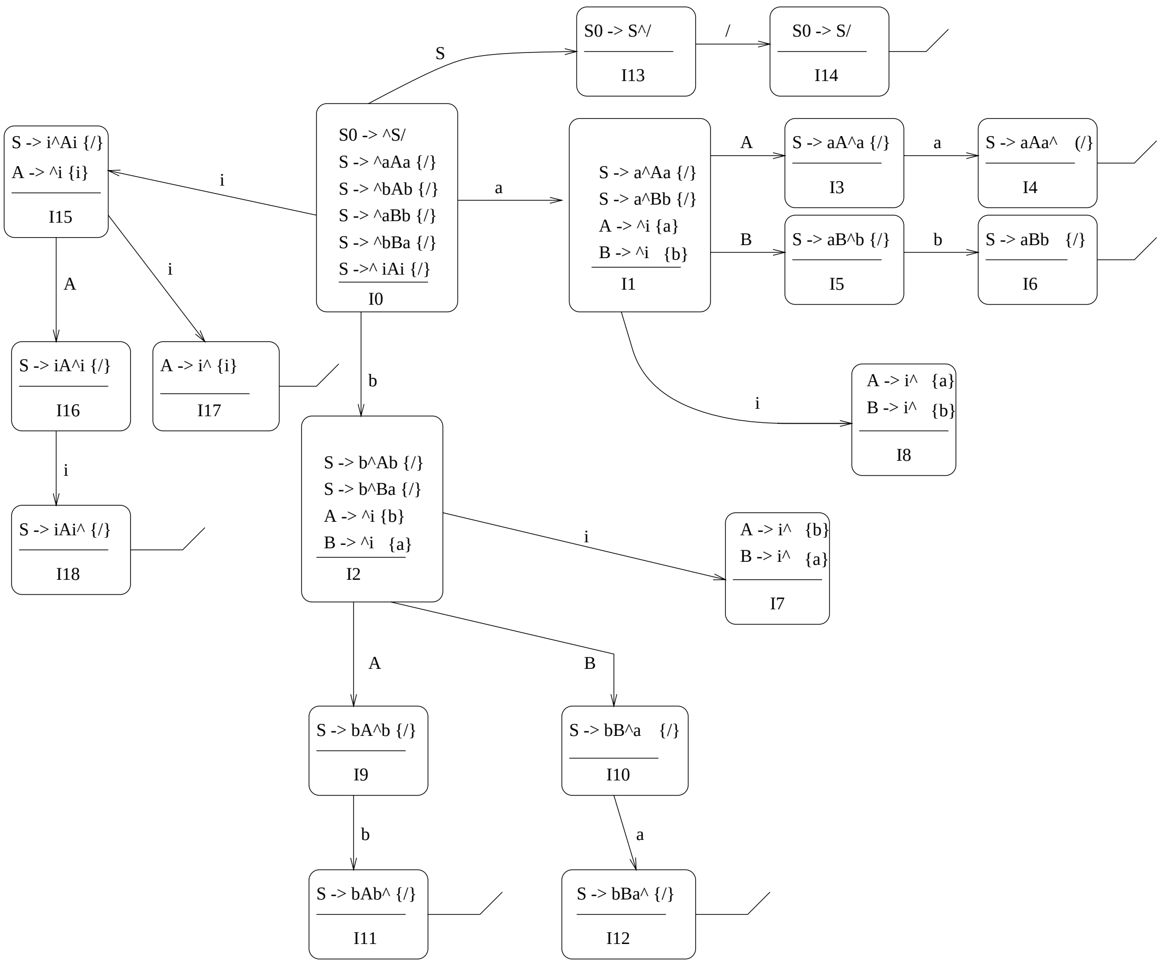

Figure 26: L'automa riconoscitore dei prefissi ascendenti LR(1) dell'esercizio 5.2.

L'*alfabeto* della pila è costituito dai simboli $I_0 \dots I_{14}$ e dall'alfabeto terminale e nonterminale della grammatica. L'automa a pila parte con il solo I_0 sulla pila e la frase da riconoscere sul nastro di ingresso. Durante il funzionamento l'automa esegue azioni di spostamento e di riduzione, salvando o eliminando dalla cima della pila l'appropriato identificatore dello stato dell'automa riconoscitore dei prefissi ascendenti. L'automa è in grado di leggere un simbolo in ingresso sensa consumarlo, cioè senza avanzare la testina di lettura.

La mossa di *riduzione* di una produzione $X \to x$ consiste nelle seguenti mosse elementari:

- *Non* consumare l'ingresso (non è una vera ϵ-mossa, poiché l'ingresso viene comunque letto, anche se non consumato, per potere decidere quale candidata ridurre negli stati inadeguati);
- Togli i primi $2\times \mid x \mid$ simboli in cima alla pila;
- Se I è lo stato in cima alla pila, allora $push(X)$ e $push(\delta(I, X))$.

Lo *spostamento* di un simbolo I consiste nelle seguenti mosse:

- Non togliere simboli dalla pila;
- Consuma un simbolo z dall'ingresso;
- $push(z)$ e $push(I)$.

La *accettazione* avviene quando non ci sono più simboli sul nastro di ingresso e sulla pila c'è il solo stato corrispondente alla candidata di riduzione $S_0 \to S \swarrow \bullet$. A questo fine, quando l'automa trova I_{14} in cima alla pila, effettua una ϵ-transizione, in modo da svuotare la pila.

È anche possibile accettare quando il prossimo simbolo in ingresso è la marca $\swarrow$ e ci si trova nello stato corrispondente alla candidata di spostamento $S_0 \to S^\bullet \swarrow$.

L'automa a pila a spostamento e riduzione LR(1) è descritto dalla Tabella 4.

Una traccia di funzionamento dell'automa sull'ingresso $aib \swarrow$ è riportato in Tabella 5.

5.3 Esercizio

Si verifichi se la grammatica G_2 dell'esercizio 3.4 è adeguata per l'analisi $LR(0)$ o $SLR(1)$:

$$G_2 = \begin{cases} S \to D\, \textbf{or}\, S \mid D \\ D \to C\, \textbf{and}\, D \mid C \\ C \to \textbf{not}\, C \mid A \\ A \to p \mid q \mid r \end{cases}$$

Cima Pila	Carattere di ingresso				
	a	b	i	$\swarrow$	ϵ
I_0	Sposta I_1	Sposta I_2			
I_1			Sposta I_8		
I_2			Sposta I_7		
I_3	Sposta I_4				
I_4				riduci $S \to aAa$	
I_5		Sposta I_6			
I_6				riduci $S \to aBb$	
I_7	riduci $B \to i$	riduci $A \to i$			
I_8	riduci $A \to i$	riduci $B \to i$			
I_9		Sposta I_1			
I_{10}	Sposta I_2				
I_{11}				riduci $S \to bAb$	
I_{12}				riduci $S \to bBa$	
I_{13}				Sposta I_{14}	
I_{14}					accetta
I_{15}			Sposta I_{17}		
I_{16}			Sposta I_{18}		
I_{17}			Riduci $A \to i$		
I_{18}				Riduci $S \to iAi$	

Table 4: L'automa a spostamento e riduzione LR(1) dell'esercizio 5.2.

Pila	Nastro di ingresso	azione
I_0	$a \quad i \quad b \quad \swarrow$	sposta I_1
$I_0 \quad a \quad I_1$	$i \quad b \quad \swarrow$	sposta I_8
$I_0 \quad a \quad I_1 \quad i \quad I_8$	$b \quad \swarrow$	riduci $B \to i$
$I_0 \quad a \quad I_1 \quad B \quad I_5$	$b \quad \swarrow$	sposta I_6
$I_0 \quad a \quad I_1 \quad B \quad I_5 \quad b \quad I_6$	$\swarrow$	riduci $S \to aBb$
$I_0 \quad S \quad I_{13} \quad \swarrow \quad I_{14}$	ϵ	accetta

Table 5: Traccia del funzionamento dell'automa di Tabella 4 sulla stringa *aib*.

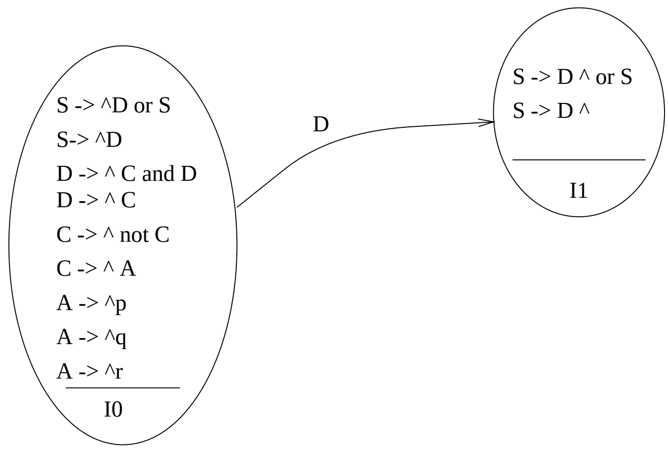

Figure 27: Un frammento dell'automa riconoscitore dei prefissi ascendenti per l'esercizio 5.3.

Soluzione 5.3

La grammatica non è adeguata per l'analisi $LR(0)$. Per verificarlo non è necessario costruire l'intero automa riconoscitore dei prefissi ascendenti, ma basta semplicemente notare la presenza delle due produzioni $S \to D\,\mathbf{or}\,S$ e $S \to D$. Il riconoscitore dei prefissi ascendenti sarà costituito fra l'altro di uno stato iniziale I_0 e uno stato I_1, con I_1 raggiungibile da I_0 tramite un arco marcato D. Lo stato I_1 non è adeguato per l'analisi LR(0) in quanto contiene la candidata di spostamento $S \to D^{\bullet}\ \mathbf{or}\ S$ e la candidata di riduzione $S \to D^{\bullet}$. Si veda la figura 27. Per ovviare a questo inconveniente, le grammatiche del tipo di G_2 sono di solito analizzate con metodi SLR o $LALR$.

La verifica se la grammatica è $SLR(1)$ è molto facile. Lasciamo al lettore la costruzione del riconoscitore completo $LR(0)$, dopo avere aggiunto come di consueto la produzione $S_0 \to S \swarrow$. Dal riconoscitore si può facilmente vedere che gli unici stati inadeguati per LR(0) sono i seguenti:

1. Stato con la candidata di spostamento $S \to D^{\bullet}\,\mathbf{or}\,S$ e la candidata di riduzione $S \to D^{\bullet}$;

2. Stato con la candidata di spostamento $D \to C^{\bullet}\mathbf{and}D$ e la candidata di riduzione $D \to C^{\bullet}$;

Nel caso (1) l'insieme dei seguiti della candidata di riduzione è $\{\swarrow\}$: la condizione $SLR(1)$ è verificata in quanto la candidata di spostamento è $S \to D^{\bullet}\ \mathbf{or}\ S$, con $\mathbf{or} \notin \{\swarrow\}$. Nel caso (2) l'insieme dei seguiti della candidata di riduzione è $\{\mathbf{or}, \swarrow\}$; la condizione $SLR(1)$ è pure verificata in quanto la candidata di spostamento è $D \to C^{\bullet}\ \mathbf{and}\ D$, con $\mathbf{and} \notin \{\mathbf{or}, \swarrow\}$.

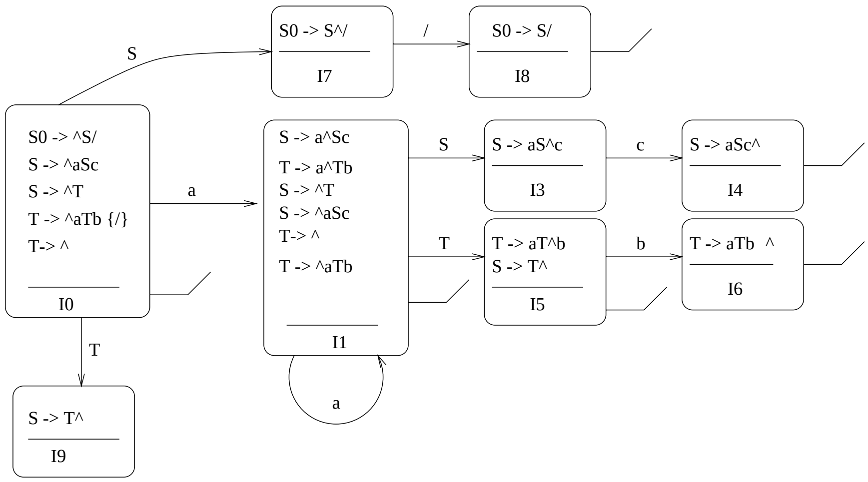

Figure 28: L'automa riconoscitore dei prefissi ascendenti LR(0) per l'esercizio 5.4.

5.4 Esercizio

Si consideri il linguaggio $L = \{a^n b^p c^q \mid n \geq 0 \wedge n = p + q\}$ dell'Esercizio 4.1 e si verifichi se è LR(0) o LR(1).

Soluzione 5.4

La grammatica G data nella Soluzione dell'Esercizio 4.1:

$$G = \begin{cases} S \to aSc \mid T \\ T \to aTb \mid \epsilon \end{cases}$$

non è LR(0), come si può facilmente verificare dall'automa riconoscitore mostrato in Figura 28, e non è nemmeno SLR. La grammatica è tuttavia LR(1), come si vede dal riconoscitore dei prefissi ascendenti LR(1) di Figura 29.

Dalla figura si può facilmente notare che G è anche LALR(1): basta accorpare gli stati con le stesse candidate, cioè I_1 con I_2, I_3 con I_7, I_5 con I_9, I_8 con I_4 e I_6 con I_{10}: gli insiemi di prospezione così ricavati sono adeguati per LALR(1). Gli insiemi possono essere ricavati anche in base alla definizione di LALR(1), riportata nell'esercizio 5.2 a partire dall'automa riconoscitore LR(0).

In I_0 l'insieme di prospezione LALR(1) per $T \to^\bullet$ è $\{\swarrow\}$. Infatti, l'unica derivazione destra del tipo $S_0 \Rightarrow^* \gamma T z$ tale che $\delta(I_0, \gamma) = I_0$, è la derivazione: $S_0 \Rightarrow S \swarrow \Rightarrow T \swarrow$ (perchè il prefisso γ deve essere uguale a ϵ se si vuole che $\delta(I_0, \gamma) = I_0$).

In I_1 l'insieme di prospezione LALR(1) per $T \to^\bullet$ è $\{b, c\}$ perchè le derivazioni destre che portano in I_1 devono generare una frase del tipo $a^+ T z$, e sono quindi di uno dei due tipi seguenti:

$$S_0 \Rightarrow^+ a^p S c^p \swarrow \Rightarrow a^p T c^p \swarrow$$

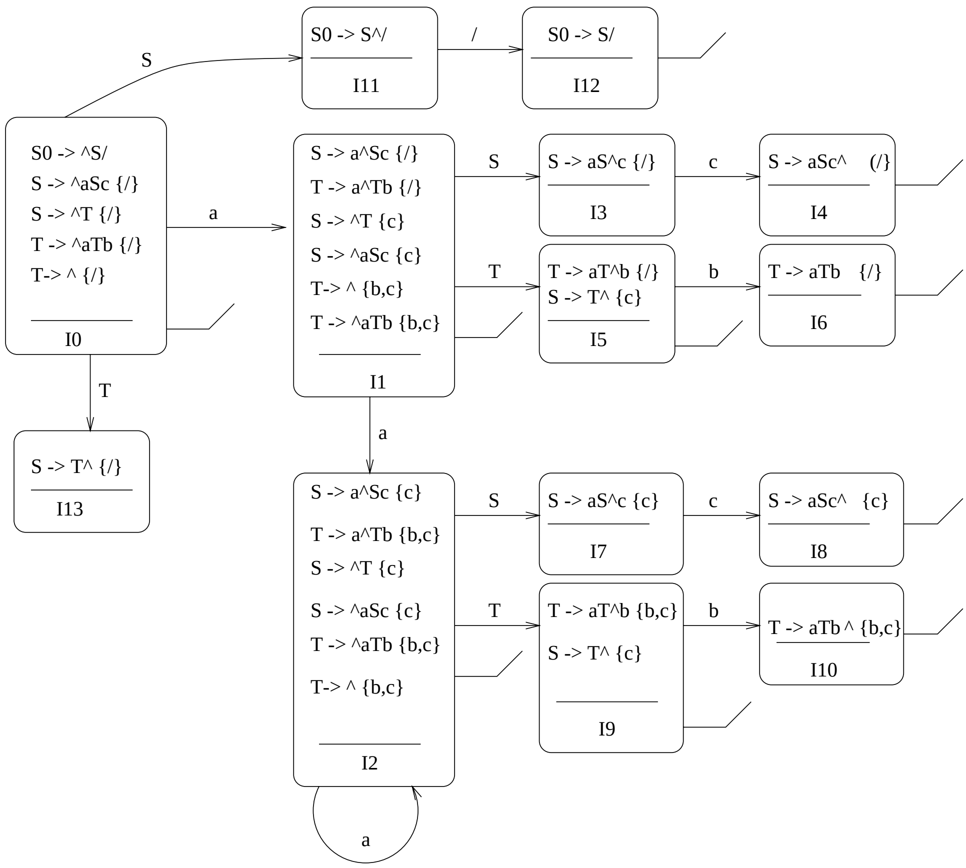

Figure 29: L'automa riconoscitore dei prefissi ascendenti LR(1) per l'esercizio 5.4.

$$S_0 \Rightarrow^+ a^p T c^p \swarrow \Rightarrow^+ a^p a^q T b^q c^q \swarrow$$

Infine, nell'ultimo stato inadeguato, I_5, la candidata di riduzione $S \to T^\bullet$ ha come insieme di prospezione LALR(1) $\{c\}$, poichè le derivazioni destre che da I_0 portano in I_5 devono generare una frase del tipo $a+Sz$, e sono quindi tutte del tipo:

$$S_0 \Rightarrow^+ a^p S c^p \swarrow$$

Si osservi che sia la grammatica G che il linguaggio L non sono LL(k) per nessun k, ma che tuttavia la grammatica è LR(1). Infatti la classe dei linguaggi LR(1) è più ampia di quella che contiene i linguaggi LL(k).

Si può addirittura costruire una versione, più complicata, di grammatica per L che è anche LR(0). Questo non s̀orprendente visto che la classe dei linguaggi LR(1) coincide con la classe dei linguaggi deterministici, mentre LR(0) coincide con la classe dei linguaggi deterministici per i quali nessuna frase è prefisso proprio di un'altra frase: usando una marca di fine stringa che non compare nell'alfabeto terminale, le due classi vengono di fatto a coincidere. Nonostante questo, una grammatica LR(1) può non essere LR(0), come in questo caso. La trasformazione in LR(0) è però sempre possibile, nell'ipotesi di considerare una marca di fine stringa ed essendo disposti ad accettare una grammatica più complicata e meno intuitiva.

Una grammatica LR(0) per L è la seguente G':

$$G' = \begin{cases} S_0 \to S \swarrow | \swarrow | \; T \swarrow \\ S \to aSc \mid ac \mid aTc \\ T \to aTb \mid ab \end{cases}$$

La verifica che la grammatica G' è LR(0) è lasciata al lettore. Rispetto a G, G' è LR(0) in quanto non contiene le due produzioni $S \to T$ e $T \to \epsilon$, la cui presenza porta a costruire per G alcuni stati inadeguati.

5.5 Esercizi proposti

5.5.1 Esercizio

Sia data la sintassi:

$$G = \begin{cases} S \to aAc \\ A \to bAb \mid b \end{cases}$$

Si verifichi se G è LR(1) e, se necessario, si costruisca una sintassi equivalente LR(0) o LR(1). Si progetti un analizzatore ascendente per G.

5.5.2 Esercizio

Per ciascuna delle seguenti grammatiche si verifichi se la grammatica è LR(0), SLR(1), LALR(1), LR(1):

$$G_1 = \begin{cases} S \to aAaa \mid bAba \\ A \to b \mid \epsilon \end{cases}$$

$$G_2 = \begin{cases} S \to X \swarrow \\ X \to bA \mid aB \\ A \to a \mid aX \mid bAA \\ B \to b \mid bX \mid aBB \end{cases}$$

Si mostri inoltre una traccia del calcolo svolto da un analizzatore ascendente, su una stringa a scelta.

6 Traduzioni sintattiche

6.1 Esercizio

Si costruisca uno schema di traduzione semplice per calcolare la traduzione:

$$\tau((a)^n c(b)^n) = d^{\lfloor n/2 \rfloor} e^n, \text{ con } n \geq 0$$

Descrivere un trasduttore a pila deterministico che esegue la traduzione, usando la tecnica LL(1) e la tecnica LR(0), e un trasduttore ricorsivo discendente.

Soluzione 6.1

La sintassi sorgente deve generare il linguaggio $L = \{(a)^n c(b)^n \mid n \geq 0\}$. La grammatica più ovvia per questo è la seguente:

$$G_0 = \{ \; S \to aSb \mid c$$

Lo schema di traduzione potrebbe allora scrivere una d per ogni coppia di a, e una e per ogni b. Non si può però ricavare una sintassi pozzo a partire da G_0, in quanto occorre distinguere i due casi: quando si genera una a e non si scrivono d, e quando si genera una coppia di a e si scrive una d in uscita.

Questo può essere ottenuto semplicemente con il seguente schema di traduzione:

$$G_t = \{ \; S \to aa\{d\}Sbb\{ee\} \mid acb\{e\} \mid c$$

Ad esempio, $\tau(aaacbbb) = deee$ è così ottenuta:
Sorgente: $S \Rightarrow aaSbb \Rightarrow aaacbbb$.
Pozzo: $S \Rightarrow dSee \Rightarrow deee$.
La grammatica non è LL(1). La si può rendere LL(1) ad esempio in questo modo:

$$G'_t = \begin{cases} S \to aTb\{e\} \mid c \\ T \to a\{d\}Sb\{e\} \mid c \end{cases}$$

Ad esempio, $\tau(aacbb) = dee$ è così ottenuta:
Sorgente: $S \Rightarrow aTb \Rightarrow aaTbb \Rightarrow aacbb$.
Pozzo: $S \Rightarrow Te \Rightarrow dTee \Rightarrow dee$.

Lo schema di traduzione è chiaramente deterministico, in quanto la grammatica sorgente è addirittura LL(1). Infatti:

$$\mathcal{G}(S \to aTb) = \{a\}$$

$$\mathcal{G}(S \to c) = \{c\}$$

$$\mathcal{G}(T \to aSb) = \{a\}$$

$$\mathcal{G}(T \to c) = \{c\}$$

L'automa a pila che implementa la traduzione si ottiene facilmente dall'automa riconoscitore LL(1) aggiungendo opportune azioni di emissione. Il risultato è riportato nella tabella seguente:

Cima Pila	Carattere di ingresso		
	a	b	c
S	push($\{e\}bTa$)		push(c)
T	push($a\{e\}Sb\{d\}$)		push(c)
a	avanza		
b		avanza	
c			avanza
$\{e\}$	emetti e	emetti e	emetti e
$\{d\}$	emetti d	emetti d	emetti d

Analogamente per il trasduttore ricorsivo discendente:

```
procedure S;
if cc ∈ {a} then cc:= PROSSIMO;
                 T;
                 if cc ≠ "b" then ERRORE else cc := PROSSIMO endif;
                 EMETTI("e");
          elsif cc ∈ {c} then cc := PROSSIMO
          else ERRORE
endif
end S;
```

```
procedure T;
if cc ∈ {a} then cc:= PROSSIMO;
                 EMETTI("d");
                 S;
                 if cc ≠ "b" then ERRORE else cc := PROSSIMO endif;
                 EMETTI("e");
            elsif cc ∈ {c} then cc := PROSSIMO
            else ERRORE
endif
end T;
```

Per potere costruire un traduttore con tecnica LR(0), occorre portare la grammatica pozzo in forma postfissa. In questo caso si può introdurre un nonterminale D da usare al posto di $\{d\}$ nello schema di traduzione. Per non rendere la grammatica inadeguata per l'analisi LR(0), evitiamo l'introduzione di ϵ-produzioni (come succederebbe per la grammatica sorgente se si aggiungesse la produzione $D \to \{d\}$). Conviene allora usare D per generare una a nella grammatica sorgente e una d in quella pozzo. Lo schema di traduzione risultante è il seguente:

$$G'_t = \begin{cases} S \to aTb\{e\} \mid c \\ T \to DSb\{e\} \mid c \\ D \to a\{d\} \end{cases}$$

Lo schema LR(0) (che è lo stesso dell'esercizio 5.1) è riportato in figura 30.

Il trasduttore a spostamento e riduzione basato sul riconoscitore LR(0) si ottiene facilmente dall'automa a spostamento e riduzione dell'esercizio 5.1 aggiungendo opportune azioni di emissione da eseguire durante le riduzioni. Si veda la Tabella 6.

6.2 Esercizio

Si costruisca uno schema di traduzione semplice per calcolare la traduzione: $\tau(x) = x^R x$, con $x \in \{a,b\}^*$. Tracciare poi lo schema di un trasduttore a pila che esegua la traduzione. È possibile costruire un trasduttore deterministico?

Soluzione 6.2

Notiamo innanzitutto che $\tau(xx^R) = x^R$ si può definire banalmente:

$$G_{t_0} = \left\{ S \to aSa\{a\} \mid bSb\{b\} \mid \epsilon \right.$$

Inoltre, $\tau(xx^R) = x$ è ottenibile con il seguente schema:

$$G_{t_1} = \left\{ S \to \{a\}aSa \mid \{b\}bSb \mid \epsilon \right.$$

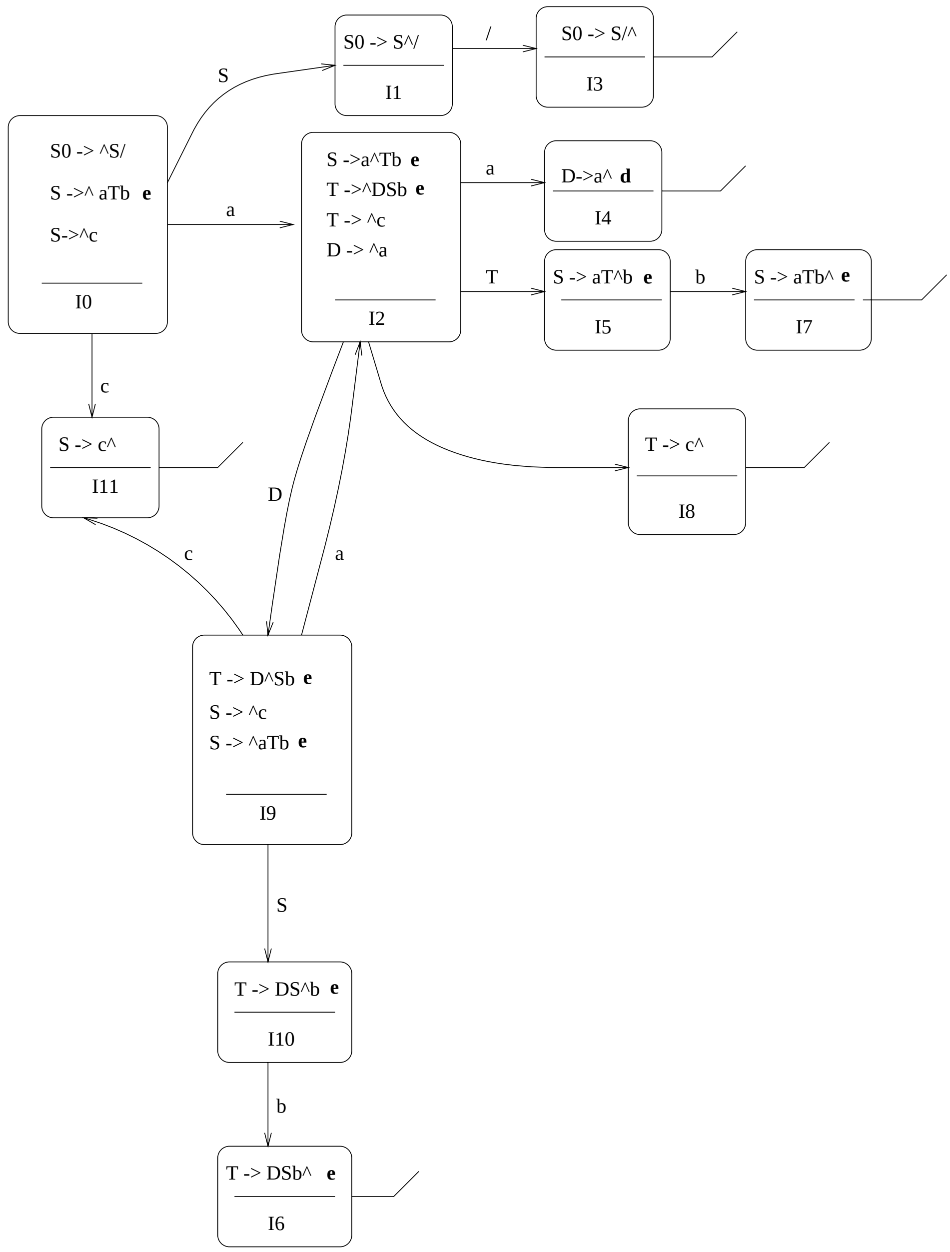

Figure 30: L'automa riconoscitore dei prefissi ascendenti LR(0) per l'esercizio 6.1. I caratteri della grammatica pozzo, da emettere al momento di una riduzione, sono segnati in grassetto.

Cima Pila	Carattere di ingresso				
	a	b	c	$\swarrow$	ϵ
I_0	Sposta I_2		Sposta I_{11}		
I_1				Sposta I_3	
I_2	Sposta I_4		Sposta I_8		
I_3					accetta
I_4					riduci $D \to a$ emetti d
I_5		Sposta I_7			
I_6					riduci $T \to DSb$ emetti e
I_7					riduci $S \to aTb$ emetti e
I_8					riduci $T \to c$
I_9	Sposta I_2	Sposta I_{11}			
I_{10}		Sposta $I6$			
I_{11}					riduci $S \to c$

Table 6: Il trasduttore a spostamento e riduzione dell'esercizio 6.1.

Per ottenere la traduzione desiderata, basta a questo punto combinare queste traduzioni in modo opportuno.

Lo schema di traduzione risultante è il seguente:

$$G_t = \left\{ \ S \to \{a\}Sa\{a\} \mid \{b\}Sb\{b\} \mid \epsilon \right.$$

Un esempio di derivazione è:

Sorgente: $S \Rightarrow Sa \Rightarrow Sba \Rightarrow Sbba \Rightarrow Sbbba \Rightarrow bbba$.

Pozzo: $S \Rightarrow aSa \Rightarrow abSba \Rightarrow abbSbba \Rightarrow abbbSbbba \Rightarrow abbbbbba$.

Un trasduttore a pila per la τ si può costruire sfruttando il nondeterminismo. Prima di cominciare a leggere, l'automa "indovina", con una serie di ϵ-mosse il riflesso x^R della stringa da riconoscere, lo emette e lo impila. Questo è possibile in quanto si può costruire un automa che ad ogni passo sceglie nondeterministicamente se emettere e impilare il carattere a, oppure il carattere b oppure cominciare a leggere l'ingresso. A un certo punto, l'automa comincia a leggere la stringa in ingresso, emettendola in uscita e confrontandola con la stringa salvata sulla pila. Se le due stringhe sono una il riflesso dell'altra, allora l'automa accetta, e quindi la stringa $x^R x$ è il risultato della traduzione dell'ingresso x. Se invece l'automa non accetta, i caratteri emessi non fanno parte della traduzione, in quanto, per stabilire il risultato della traduzione, la definizione di trasduttore non considera le computazioni fallite.

Stato / cima pila	Ingresso		
	ϵ	a	b
q_0, Z_0	(q_0, A, a) (q_0, B, b) $(q_0, \epsilon, \epsilon)$		
q_0, A	(q_0, AA, a) (q_0, AB, b) (q_1, A, ϵ)		
q_0, B	(q_0, AA, a) (q_0, AB, b) (q_1, B, ϵ)		
q_1, A		(q_1, ϵ, a)	
q_1, B			(q_1, ϵ, b)

Table 7: La funzione di transizione di un trasduttore a pila che calcola $\tau(x) = x^R x$, per l'esercizio 6.2.

La definizione dell'automa trasduttore può essere così precisata. Sia Z_0 il simbolo iniziale della pila; l'alfabeto di ingresso è $\{a, b\}$, quello della pila $\{Z_0, A, B\}$. L'insieme di stato è $\{q_0, q_1\}$. L'automa accetta quando la pila è vuota e il nastro di ingresso è vuoto. Il risultato della traduzione della stringa di ingresso è definito come l'insieme delle stringhe emesse durante computazioni che portano ad accettazione (in questo caso, per ogni ingresso ci sarà una e una sola computazione che porta ad accettazione, e pertanto il risultato della trasduzione sarà una stringa e non un insieme di stringhe).

La funzione di transizione dell'automa è descritta nella Tabella 7, in cui sulle righe sono riportate solo le coppie (stato, simbolo in cima alla pila) per le quali la funzione è definita per almeno un valore dell'ingresso. Poiché l'automa è non deterministico, in una stessa configurazione sono possibili più mosse, riportate nella stessa cella della tabella. Ogni mossa è rappresentata da una tripla (stato, stringa da impilare, stringa di uscita). L'automa esegue una *pop* ad ogni passo, cambia di stato, esegue una *push* della stringa da impilare ed emette la stringa di uscita.

È facile comprendere che nessun automa a pila deterministico è in grado di eseguire questa traduzione.

6.3 Esercizio

Si costruisca uno schema di traduzione finito per calcolare la traduzione: $\tau(x)$ è la funzione identità fino alla prima occorrenza, da sinistra, della stringa aba in x, restituisce c per questa prima occorrenza, è ancora la funzione identità dopo l'occorrenza.

Ad esempio, $\tau(aab) = aab$, $\tau(aba) = c$, $\tau(aabbabababa) = aabbcbaba$.

Stato	ingresso		
	a	b	ϵ
q_0	(q_1, ϵ)	(q_0, b)	
q_1	(q_1, a)	(q_2, ϵ)	(q_4, a)
q_2	(q_0, c)	(q_0, abb)	(q_4, ab)
q_3	(q_3, a)	(q_3, b)	
q_4			

Table 8: Il trasduttore finito per l'esercizio 6.3. Il primo elemento di ogni coppia è lo stato prossimo, il secondo la stringa emessa in uscita. Gli stati q_0, q_3 e q_4 sono finali.

Si costruisca il trasduttore finito che calcola la traduzione.

Soluzione 6.3

Uno schema di traduzione è il seguente:

$$G_t = \begin{cases} S \rightarrow aT \mid b\{b\}S \mid \epsilon \\ T \rightarrow bU \mid a\{a\}T \mid \{a\} \\ U \rightarrow a\{c\}V \mid b\{abb\}S \mid \{ab\} \\ V \rightarrow a\{a\}V \mid b\{b\}V \mid \epsilon \end{cases}$$

Il corrispondente trasduttore finito è dato in Tabella 8.

6.4 Esercizio

Si progetti uno schema sintattico semplice di traduzione per calcolare la traduzione:

$$\tau(a^{n_1}?a^{n_2}?a^{n_3}\ldots?a^{n_k}) = b^{n_1}c^{n_1}1b^{n_2}c^{n_2}2b^{n_3}c^{n_3}\ldots b^{n_k}c^{n_k}$$

con $k \geq 1, n_i \geq 1$. La traduzione sostituisce i separatori "?" di posto dispari con 1, quelli di posto pari con 2.

La traduzione può essere calcolata da un trasduttore deterministico?

Soluzione 6.4

Una soluzione è data dal seguente schema:

$$G_t = \begin{cases} S \rightarrow T \mid T?\{1\}T \mid T?\{1\}T?\{2\}S \\ T \rightarrow a\{b\}T\{c\} \mid \epsilon \\ \end{cases}$$

È evidente che si può costruire un trasduttore deterministico. Per convincersene, si può anche modificare la grammatica in modo da renderla LL(1) (come si può facilmente verificare):

$$G_{t1} = \begin{cases} S \to TU \\ U \to ?\{1\}TW \mid \epsilon \\ W \to \epsilon \ mid?\{2\}TS \\ T \to a\{b\}T\{c\} \mid \epsilon \end{cases}$$

6.5 Esercizi proposti

6.5.1 Esercizio

Si costruisca un trasduttore finito che calcola la seguente traduzione:
$\tau(a^{3n}) = b^{2n}, n \geq 0$

6.5.2 Esercizio

Si costruisca uno schema di traduzione sintattico che calcola la seguente traduzione, per $n \geq 0$:

$$\tau(a^n c^* b^n) = \begin{matrix} p^{n/2} & n \text{ pari} \\ d^{n+1} & n \text{ dispari} \end{matrix}$$

Si verifichi se la traduzione può essere calcolata in modo deterministico e si costruisca il relativo automa trasduttore.

7 Grammatiche ad attributi

7.1 Esercizio

Costruire una grammatica ad attributi che generi il linguaggio non libero

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

Soluzione 7.1

Possiamo procedere in due modi:

1. Costruire una grammatica per $\mathbf{a}^+\mathbf{b}^+\mathbf{c}^+$, i cui attributi verificano se vi sia lo stesso numero di occorrenze di a, di b e di c.

2. Costruire una grammatica per $\mathbf{a^n b^n c}^+$ e verificare poi che il numero di occorrenze di c sia proprio n.

Seguendo il primo metodo definiamo la seguente grammatica, scritta in una forma comoda per semplificare gli attributi:

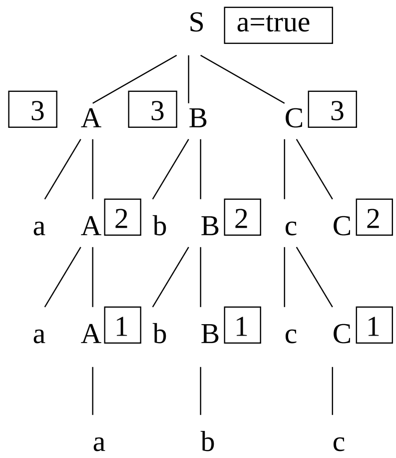

Figure 31: Albero decorato per la stringa $aaabbbccc$ per la grammatica ad attributi dell'esercizio 7.1.

$$G_1 = \begin{cases} 1) S \to ABC \\ 2) A \to aA \\ 3) A \to a \\ 4) B \to bB \\ 5) B \to b \\ 6) C \to cC \\ 7) C \to c \end{cases}$$

Gli attributi sono i seguenti: $aofS$ restituisce $true$ se la stringa generata è una frase di L, mentre $numofA$, $numofB$ e $numofC$ rappresentano rispettivamente il numero di a, b e c generati da A, B, C.

$$\begin{cases} 1) aofS := (numofA = numofB \wedge numofB = numofC) \\ 2) numofA_0 := 1 + numofA_1 \\ 3) numofA := 1 \\ 4) numofB_0 := 1 + numofB_1 \\ 5) numofB := 1 \\ 6) numofC_0 := 1 + numofC_1 \\ 7) numofC := 1 \end{cases}$$

Un esempio di albero di derivazione per la frase $aaabbbccc$, decorato con i corrispondenti attributi, è mostrato in figura 31.

Il secondo metodo è mostrato dalla seguente grammatica:

$$G_1 = \begin{cases} 1)S \rightarrow XC \\ 2)X \rightarrow aXb \\ 3)X \rightarrow ab \\ 4)C \rightarrow cC \\ 5)C \rightarrow c \end{cases}$$

Gli attributi sono i seguenti: $aofS$ restituisce true se la stringa generata è una frase di L, mentre $numofX$ restituisce il numero di a (uguale a quello delle b) e $numofC$ rappresenta il numero di c generati da C.

$$\begin{cases} 1)aofS := (numofX = numofC) \\ 2)numofX_0 := 1 + numofX_1 \\ 3)numofX := 1 \\ 4)numofC_0 := 1 + numofC_1 \\ 5)numofC := 1 \end{cases}$$

7.2 Esercizio

Costruire una grammatica ad attributi che generi il linguaggio non libero

$$L = \{ww \mid q \in (0 \cup 1)^*\}$$

Si possono usare solo attributi di tipo intero o booleano.

Soluzione 7.2

Una soluzione può essere facilmente costruita se si considera una stringa di 0 e 1 come una rappresentazione binaria di un opportuno numero intero. Si può allora definire una grammatica che considera frasi divise in due metà di pari lunghezza, i cui attributi calcolano il valore intero corrispondente alle due metà. Poichè le due stringhe sono di pari lunghezza, sono uguali se, e solo se, rappresentano lo stesso numero intero. In questo caso, infatti, non vi è alcuna ambiguità nella rappresentazione dovuta alla possibilità che vi siano zeri iniziali (0010 ha lo stesso valore di 010 e di 10).

Per calcolare il valore numerico delle due metà basta osservare che aggiungere un bit v a sinistra, cioè nella posizione più significativa, di una stringa di n bit che rappresenta il numero b, definisce un numero binario di valore $b + v * 2^{n-1}$. Aggiungere un bit v a destra, cioè nella posizione meno significativa, di una stringa di n bit che rappresenta il numero b, definisce un numero binario di valore $2 * b + v$.

Si può allora usare una produzione che attacchi un bit a sinistra e uno a destra, con opportuni attributi che tengano conto della profondità dell'albero.

La grammatica costruisce simmetricamente stringhe del tipo $(0 \cup 1)^n (0 \cup 1)^n$. Gli attributi sono *des* e *sin* per i valori della parte destra e sinistra della frase, *val* per il valore di un singolo bit, *d* per la profondità dell'albero.

$1) S \to T$	$aofS := sinofT = desofT$
$2) T \to XTX$	$sinofT_0 := valofX * 2^{dofT_1} + sinofT_1$ $desofT_0 := valofX + 2 * desofT_1$ $dofT_0 := dofT_1 + 1$
$3) T \to \epsilon$	$desofT := 0$ $sinofT := 0$ $dofT := 0$
$4) X \to 0$	$valofX := 0$
$5) X \to 1$	$valofX := 1$

Un albero decorato per la stringa 0101 e un albero per 0111 sono mostrati in figura 32.

Si noti che il metodo è facilmente generalizzabile a qualunque alfabeto finito: dato un qualunque alfabeto finito di cardinalità k a,b, ... z basta assegnare un valore tra 0 e k ad ogni lettera. Le formule per aggiungere cifre a sinistra e a destra sono uguali, salvo che la cifra 2 è sostituita da k.

7.3 Esercizio

Si definisca una grammatica ad attributi per il linguaggio delle dichiarazioni e di chiamate di procedura. Una dichiarazione ha la forma: **proc** Id $(P_1, P_2, \ldots, P_n)$; *lista-chiamate-di-procedura* **end** dove Id è un identificatore che rappresenta il nome della procedura e i P_i sono gli identificatori dei parametri formali, che si suppongono tutti dello stesso tipo (intero). Vi è almeno un parametro formale per procedura.

Una chiamata ha la seguente forma: **call** Id $(A_1, A_2, \ldots A_n)$, dove ogni parametro attuale A_i deve essere un intero positivo.

Una frase consiste di una o più dichiarazioni e chiamate, terminate da un carattere punto. Ad esempio, una frase è:

proc P1(A, B)
call P1(2,3)
proc P2(X,Y)
call P2(4,5)
call P1(1,2)
call P2(3,6)
end.

La grammatica deve svolgere i seguenti controlli semantici:

- non vi sono due dichiarazioni di procedura con lo stesso nome;
- le chiamate di procedura sono relative solo alle procedure che sono state dichiarate in un punto precedente del programma;

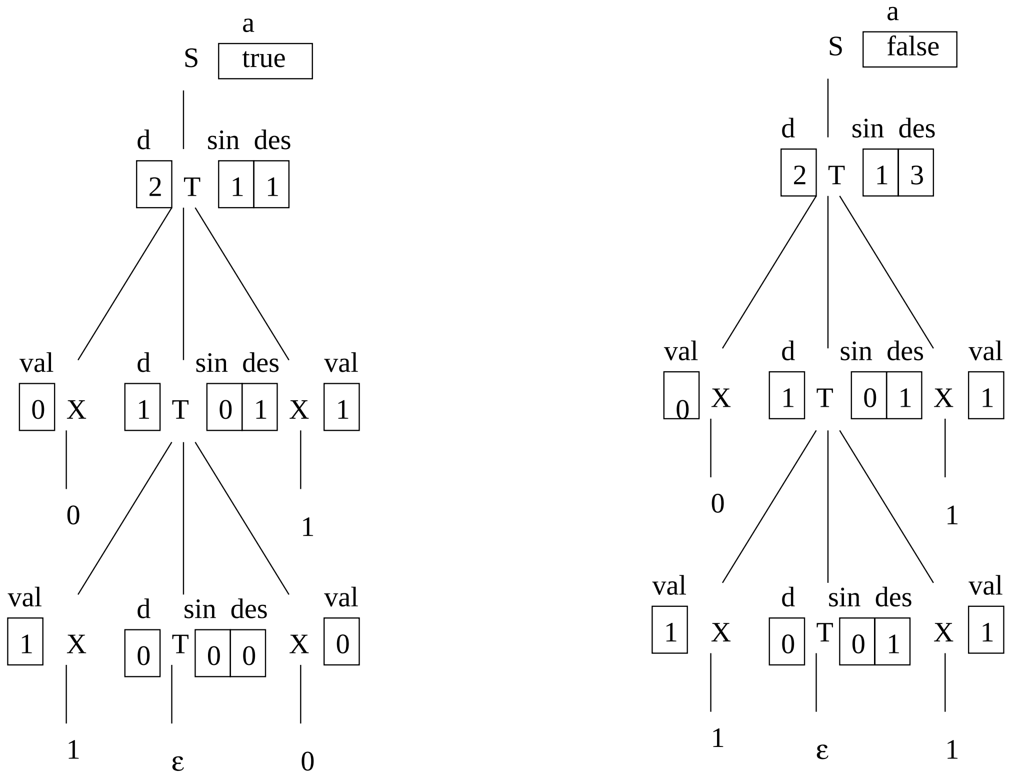

Figure 32: Alberi decorati per la stringhe 0101 e 0111 per la grammatica ad attributi dell'esercizio 7.2.

$1) program \rightarrow prg$	$aofprogram := aofprg$ $envofprg := \emptyset$
$2) prg_0 \rightarrow prcd\ prg_1$	$aofprg_0 := aofprcd \wedge aofprg_1$ $envofprg_1 := envofprg_0 \bigcup updofprcd$ $envofprcd := envofprg_0$
$3) prg \rightarrow .$	$aofprg := true$
$4) prcd \rightarrow \mathbf{proc} pId(fPar)cll$	$aofprcd := aofcll \wedge valofpId \notin nomi(envofprcd)$ $updofprcd := \{(valofpId, numoffPar)\}$ $envofcll := envofprcd \bigcup \{(valofpId, numoffPar)\}$
$5) fPar \rightarrow id$	$numoffPar := 1$
$6) fPar_0 \rightarrow id, fPar_1$	$numoffPar_0 := 1 + numoffPar_1$
$7) cll_0 \rightarrow \mathbf{call} pId(actPar)cll_1$	$aofcll_0 := (pId, numofactPar) \in envofcll_0 \wedge aofcll_1$ $envofcll_1 := envofcll_0$
$8) cll \rightarrow \mathbf{end}$	$aofcll := true$
$9) actPar \rightarrow int$	$numofactPar := 1$
$10) actPar_0 \rightarrow int, actPar_1$	$numofactPar_0 := numofactPar_1 + 1$

Table 9: La grammatica ad attributi per l'esercizio 7.3

- i parametri attuali di una chiamata sono in numero pari a quelli formali della dichiarazione corrispondente;

Si descriva, almeno in parte, un analizzatore sintattico-semantico integrato per la grammatica.

Soluzione 7.3

La grammatica, illustrata in Tabella 9, è costituita dai non-terminali *program*, *prg*, *prcd*, *cll*, *fPar*, *actPar*, *pId*, *int*. Gli attributi da considerare sono: *aofprogram*, che restituisce *true* o *false* a seconda che i controlli semantici diano esito positivo o negativo; *envofprogram*, *envofprog*, ecc., che sono attributi *ereditati* costituiti da un insieme di coppie $< nome, numero >$ che rappresentano l'environment di una dichiarazione, vale a dire i nomi e il numero dei parametri delle procedure visibili in un certo punto del programma. Vi sono poi altri attributi, tutti sintetizzati, come *updofprcd* che contiene la coppia nome-numero corrispondente alla dichiarazione corrente; *numofactPar* e *numoffPar* che rappresentano il numero di parametri di una dichiarazione e di una chiamata rispettivamente. I nonterminali *pId*, *id* e *int* non sono espansi ulteriormente in quanto si suppone che il loro (unico) attributo sintetizzato *val* sia ritornato da un opportuno analizzatore lessicale.

La regola (1) serve solo per inizializzare l'ambiente di *prg*. Da *prg* si deriva un programma sintatticamente corretto, costituito da una o più dichiarazioni di procedura (*prcd*), tramite le regole (2) e (3). Ogni procedura è costruita secondo la regola (4). L'attributo sintetizzato *envofprg* permette di verificare se l'identificatore della procedura è già stato utilizzato. Si suppone che vi sia un analizzatore lessicale che restituisce un identificatore per pId e un intero per int (in caso contrario, l'albero sintattico sarà scorretto).

La grammatica è in forma normale di Bochman e sono verificate le condizioni di tipo L, in quanto gli attributi ereditati di ogni regola dipendono al più da attributi di nonterminali fratelli che si trovano più a sinistra e dagli attributi ereditati della parte sinistra della regola. La grammatica è LL(2), a causa delle produzioni 5, 6, 9 e 10. Le altre produzioni verificano le condizioni LL(1). Costruiamo un valutatore di attributi a discesa ricorsiva solo per il nonterminale *prcd*, per il quale è sufficiente considerare un valutatore LL(1).

La procedura riceve come parametri di ingresso il valore degli attributi ereditati di *prcd* e restituisce in uscita il valore di quelli sintetizzati. La procedura costruisce l'albero sintattico e intanto valuta gli attributi.

procedure prcd (in $envofprcd$**; out** $aofprcd$**,** $updofprcd$**);**
in ingresso: attributi ereditati di prcd *
in uscita: attributi sintetizzati di prcd *
if $cc \in \{proc\}$ **then**
  cc:= **PROSSIMO;**
  *** non ci sono attributi ereditati di pId ***
  *** calcolo attributi sintetizzati di pId: ***
  pId(valofpId);
  if $cc \neq$ "(" **then ERRORE**
  else cc:=**PROSSIMO;**
    *** non ci sono attributi ereditati di fPar ***
    *** calcolo attributi sintetizzati di fPar: ***
    call fPar($numoffPar$**)**
    if $cc \neq$ ")" **then ERRORE**
    else cc:= **PROSSIMO;**
      *** calcolo attributi ereditati di cll: ***
      $envofcll$ **:=** $envofprcd \bigcup \{(valofpId, numofpId)\}$
      *** calcolo attributi sintetizzati di cll: ***
      call cll($envofcll$**,** $aofcll$**);**
      *** calcolo attributi sintetizzati di prcd: ***
      $updofprcd$ **:=** $\{(valofpId, numofpId)\}$
      $aofprcd$ **:=** $aofcll \wedge valofpId \in nomi(envofprcd)$
    endif
  endif
else ERRORE
endif
end prcd;

7.4 Esercizio

Descrivere un insieme di attributi per la grammatica dell'esercizio 5.1 in modo da calcolare, come attributo $dofS$, la profondità di un albero di derivazione della grammatica. Mostrare una traccia di funzionamento di un automa a pila a spostamento e riduzione per l'analisi sintattico-semantica integrata sulla frase acb.

Soluzione 7.4

La grammatica ad attributi è:

$$G = \begin{cases} 1) S \to aTb \\ 2) S \to c \\ 3) T \to DSb \\ 4) T \to c \\ 5) D \to a \end{cases}$$

$$\begin{cases} 1) dofS := dofT + 1 \\ 2) dofS := 1 \\ 3) dofT := 1 + dofS \\ 4) dofT := 1 \end{cases}$$

La grammatica ad attributi è di tipo elementare, in quanto tutti gli attributi sono sintetizzati, ogni attributo $dofX$ dipende solo dagli attributi della parte destra o da attributi sintetizzati di X_0, e la grammatica è in forma normale di Bochman. Si può pertanto usare l'automa a spostamento e riduzione presentato per l'esercizio 5.1, che nelle mosse di riduzione per produzioni del tipo $X \to \alpha$ salva sulla pila i valori degli attributi del nonterminale X. Nei successivi passi di riduzione i valori degli attributi di X e degli altri nonterminali salvati sulla pila possono essere utilizzati per il calcolo degli attributi dei nonterminali di sinistra delle produzioni ridotte.

Una traccia di esecuzione sulla stringa acb è riportata in Tabella 10. Si osservi che il calcolo di $dofS$ quando si riduce $S \to aTb$ richiede la conoscenza degli attributi di T, che però sono sulla pila.

7.5 Esercizio

Si consideri il linguaggio delle parentesi dato dalla seguente grammatica:

$$G = \begin{cases} 1) S \to (S)S \\ 2) S \to T; S \\ 3) S \to \epsilon \\ 4) T \to aT \\ 5) T \to bT \\ 6) T \to \epsilon \end{cases}$$

Pila	Nastro di ingresso	azione
I_0 -	a c b $\swarrow$	sposta I_2
I_0 a I_2 - - -	c b $\swarrow$	sposta I_8
I_0 a I_2 c I_8 - - - - -	b $\swarrow$	riduci $T \to c$ $dofT := 1$
I_0 a I_2 T I_5 - - - d=1	b $\swarrow$	sposta I_7
I_0 a I_2 T I_5 b I_7 - - - d=1 - -	$\swarrow$	riduci $S \to aTb$ $dofS := dofT + 1$
I_0 S I_1 $\swarrow$ I_3 - d=2 - - - -	ϵ	accetta

Table 10: Traccia del funzionamento dell'automa di Tabella 1 sulla stringa acb.

Un esempio di frase è:

$$(abb; ab; (aab; aba;)abbb; ()a; aa;)$$

Si costruisca una grammatica ad attributi tale per cui il linguaggio generato coincide con le stringhe di $L(G)$ per le quali tutti gli identificatori (cioé le stringhe di a e di b) che si trovano allo stesso livello di annidamento siano distinti.

Ad esempio, la frase precedente è corretta, mentre la seguente non lo è:

$$(abb; ab; (aab; aba;)abbb; (aab;)a; aa;)$$

in quanto vi sono due sottostringhe aab allo stesso livello di annidamento (il secondo).

Soluzione 7.5

La soluzione fa uso di due attributi ereditati, env e lev. L'attributo env ha come valore un insieme di coppie $(identificatore, livello)$; lev è costituito da un numero naturale che rappresenta il livello di annidamento. Vi sono poi gli attributi sintetizzati a, che può assumere un valore booleano che denota la correttezza semantica di una frase, upd, che contiene una coppia $(identificatore, livello)$, e val che è la stringa associata a un identificatore.

La grammatica ad attributi è G per la parte sintattica, a cui corrispondono gli opportuni assegnamenti degli attributi.

$envofS$ è inizializzato con l'insieme vuoto, $levofS$ con il numero zero.

$$G_a = \begin{cases} 1) S_0 \to (S_1) S_2 & levof S_1 := levof S_0 + 1 \\ & levof S_2 := levof S_0 \\ & envof S_1 := envof S_0 \\ & envof S_2 := envof S_0 \cup updof S_1 \\ & updof S_0 := updof S_1 \cup updof S_2 \\ & aof S_0 := aof S_1 \wedge aof S_2 \\ 2) S_0 \to T; S_1 & levof S_1 := levof S_0 \\ & updofT := \{(valofT, levof S_0)\} \\ & envof S_1 := envof S_0 \cup updofT \\ & updof S_0 := \{(valofT, levof S_0)\} \cup updof S_1 \\ & aof S_0 := aof S_1 \wedge (valofT, levof S_0) \notin envof S_0 \\ 3) S \to \epsilon & updof S := \emptyset \\ & aof S := true \\ 4) T_0 \to aT_1 & valofT_0 := a \bullet valofT_1 \\ 5) T_0 \to bT_1 & \\ & valofT_0 := b \bullet valofT_1 \\ 6) T \to \epsilon & valofT := \epsilon \end{cases}$$

La grammatica verifica che gli identificatori allo stesso livello siano distinti, confrontando l'attributo *upd* con l'attributo *env*.

Si osservi che questo tipo di confronto non è lo stesso dei linguaggi di programmazione che ammettono procedure annidate, in quanto il confronto in questo caso considera solo il livello di annidamento, mentre nei linguaggi di programmazione anche l'annidamento stesso.

7.6 Esercizio

È data la grammatica ad attributi G_X di fig. 33.

Si decida se la grammatica risulta valutabile in una sola scansione, e in tal caso si costruisca la corrispondente procedura di valutazione. Si ipotizzi che gli attributi siano inizializzati correttamente.

Soluzione 7.6

Affinchè gli attributi di una grammatica possano essere valutati in una sola scansione, è necessario e sufficiente che siano soddisfatte le seguenti condizioni per ciascuna produzione $P : A_0 \longrightarrow A_1 \dots A_r$:

1. Il grafo delle dipendenze funzionali dip_P, $\forall P : A_0 \longrightarrow A_1 \dots A_r$ è tale per cui:
   (a) dip_P è aciclico;
   (b) Non esistono cammini da un attributo sintetizzato di A_i a un attributo ereditato dello stesso A_i;

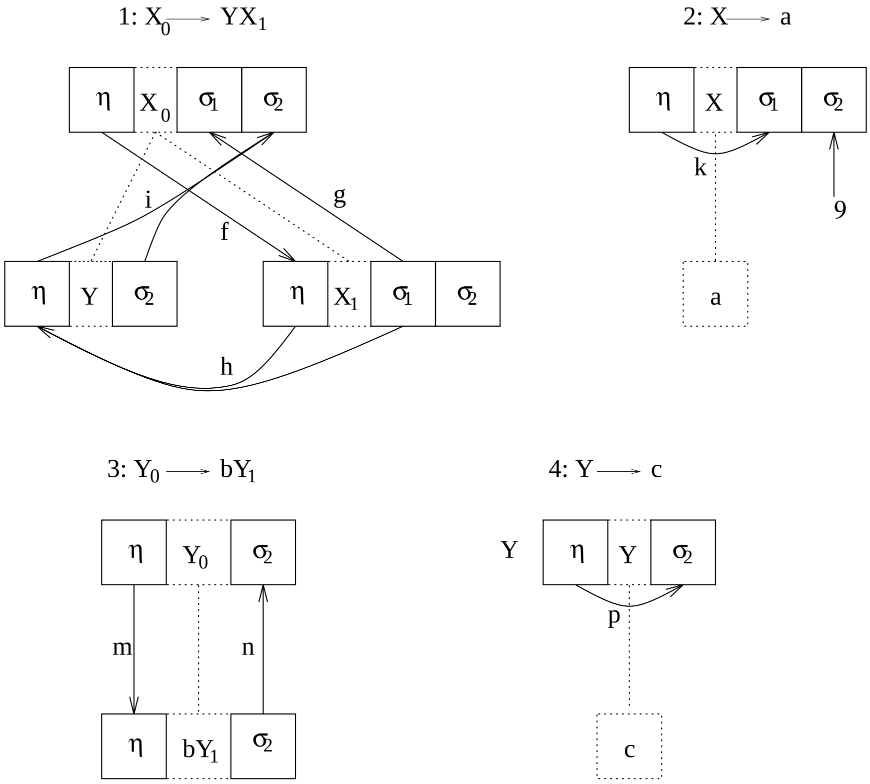

Figure 33: Grammatica ad attributi G_X

(c) Non esistono dipendenze di attributi ereditati di A_i da un attributo sintetizzato di A_0.

2. Il grafo fra_P dei fratelli è aciclico per ogni P.

La grammatica G_X soddisfa tutte queste condizioni. In particolare valutiamo il punto (2): l'attributo η di Y nella produzione (1:) dipende da attributi di X_1, per cui si costruisce un ordine topologico in cui $X \prec Y$. Non vi sono poi altre occorrenze di relazioni di precedenza.

L'ordine topologico per gli attributi ereditati è immediato, visto che vi è un solo attributo ereditato e bisogna rispettare la condizione appena vista. Per quanto riguarda gli attributi sintetizzati, si può scegliere un ordine topologico in cui $\sigma_1 \prec \sigma_2$, ma l'ordine è indifferente, visto che entrambe le possibilità vanno bene.

Realizziamo il codice della procedura di visita, ipotizzando che $\eta of X$ sia iniazializzato a 1:

program
  procedure R_X(**in** η *of* X_0, T; **out** σ_1 *of* X_0, σ_2 *of* X_0)
    case alternativa(T) of
      1: *calcola in ordine topologico gli attributi ereditati di* X_1*:*

```
                η of X₁ := f(η of X₀);
                calcola attributi sintetizzati di X₁:
                R_X(η of X₁, T₁, σ₁ of X₁, σ₂ of X₁);
                calcola in ordine topologico gli attributi ereditati di Y:
                η of Y := h(η of X₁, σ₁ of X₁);
                calcola attributi sintetizzati di Y:
                R_Y(η of Y, T₂; σ₂ of Y);
                calcola in ordine topologico gli attributi sintetizzati di X₀
                σ₁ of X₀ := g(σ₁ of X₁);
                σ₂ of X₀ := i(η of Y, σ₂ of Y);
            2:  σ₁ of X₀ := k(η of X₀);
                σ₂ of X₀ := 9;
      endcase
   end R_X;

   procedure R_Y(in η of Y₀, T; out σ₂ of Y₀)
      case alternativa(T) of
            3:  η of Y₁ := m(η of Y₀);
                R_Y(η of Y₁, T₁, σ₂ of Y₁);
                σ₂ of Y₀ := n(σ₂ of Y₁);
            4:  σ₂ of Y₀ := p(η of Y₀);
      endcase
   end R_Y;

   Leggi l'albero T₀;
   R_X(1,T₀; σ₁ of X, σ₂ of X);
   emetti il valore di σ₂ of X
end
```

7.7 Esercizio

È data la grammatica ad attributi G'_X di fig. 34.

Si decida se la grammatica risulta valutabile in una o più scansioni.

Soluzione 7.7

La grammatica non è ad una scansione perchè l'attributo η *of* X_1 dipende da σ *of* X_1. Le altre condizioni risultano però verificate:

- Il grafo delle dipendenze funzionali dip_P, $\forall P : A_0 \longrightarrow A_1 \dots A_r$ è aciclico.
- Non esistono dipendenze di attributi ereditati di A_i da un attributo sintetizzato di A_0.

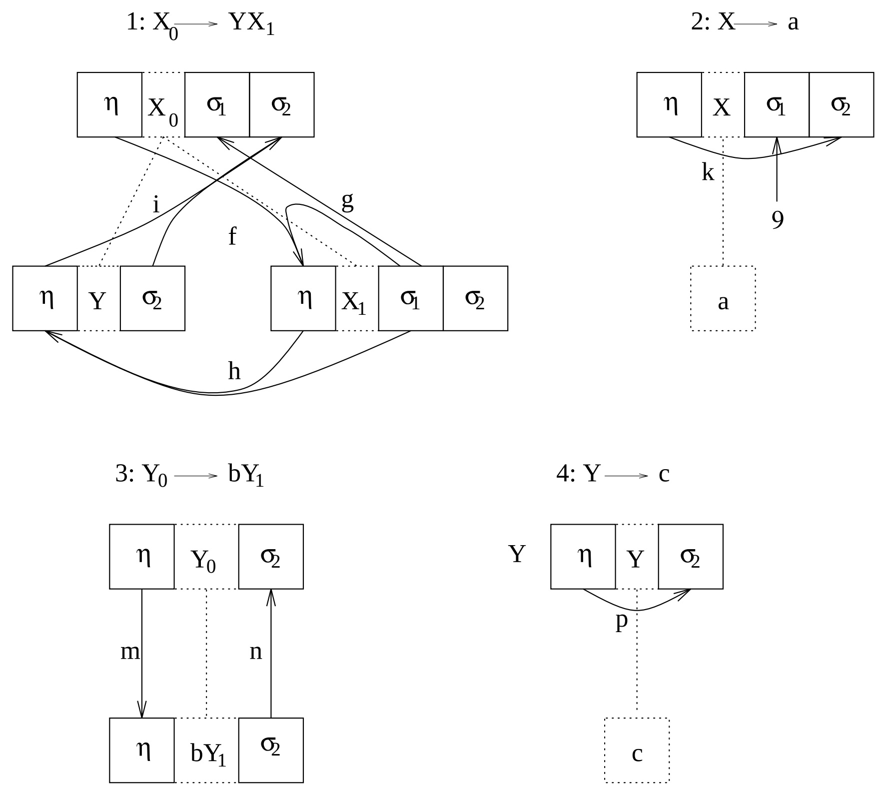

Figure 34: Grammatica ad attributi G'_X

- Il grafo fra_P dei fratelli è aciclico per ogni P.

Per valutare se la grammatica risulta valutabile in più scansioni bisogna costruire il grafo semplice delle dipendenze (Figura 35).

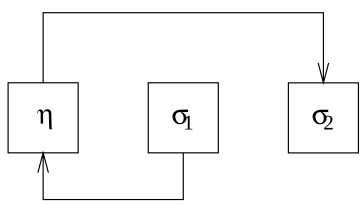

Figure 35: Grafo semplice delle dipendenze per la grammatica G'_X

Il grafo risulta aciclico, quindi i componenti connessi massimali sono costituiti dai semplici nodi; visto che il grafo è costituito da 3 nodi, la grammatica sarà valutabile al massimo con 3 scansioni. Non è difficile intuire che è possibile accorpare le scansioni per η e per σ_2, in quanto la dipendenza "proibita" era solo fra η e $sigma_1$.

Per la scansione che calcola σ_1, qualunque ordinamento topologico dei fratelli e degli attributi va bene. Per la scansione che calcola η e σ_2 è necessario scegliere, per la produzione $X \to YX$, l'ordine topologico derivato dal graf o semplice di dipendenza è: $X \prec Y$.

7.8 Esercizio

È data la grammatica ad attributi G''_X di fig. 36.

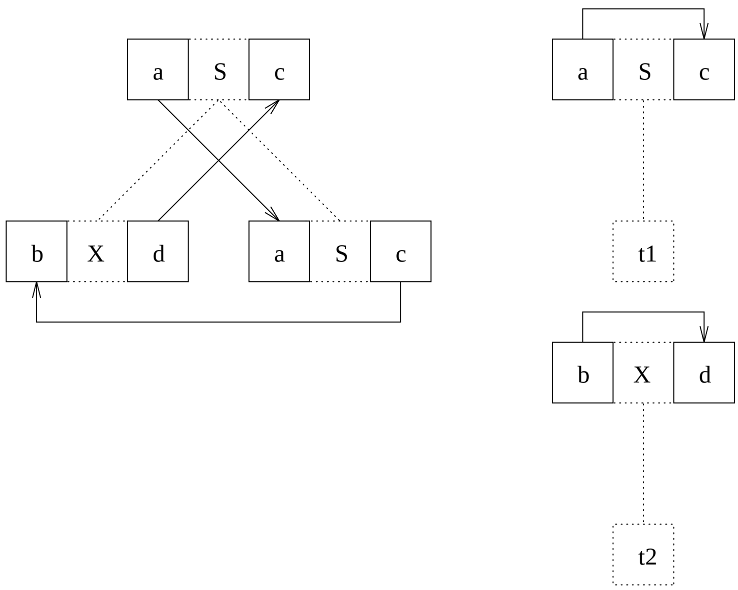

Figure 36: Grammatica ad attributi G''_X

Si decida se la grammatica risulta valutabile in una o più scansioni, e in tal caso si costruisca la corrispondente procedura di valutazione.

Soluzione 7.8

La grammatica è valutabile con una scansione. Infatti:

1. Il grafo delle dipendenze funzionali dip_P, $\forall P : A_0 \longrightarrow A_1 \dots A_r$ è tale per cui:
   (a) dip_P è aciclico;
   (b) Non esistono cammini da un attributo sintetizzato di A_i a un attributo ereditato dello stesso A_i;
   (c) Non esistono dipendenze di attributi ereditati di A_i da un attributo sintetizzato di A_0.
2. Il grafo fra_P dei fratelli è aciclico per ogni P.

L'ordine topologico dei fratelli da utilizzare nella scansione è $\{ S, X\}$, mentre per quanto riguarda gli attributi non c'è un problema di ordinamento in quanto esiste un unico attributo sintetizzato ed un unico attributo ereditato per ogni non-terminale.

La procedura di visita sarà:

```
procedure S(in a of S0, T; out c of S0)
    case alternativa(T) of
        S ⟶ XS:
                    a of S1 := f1(a of S0);
                    S(a of S1, T1, c of S1);
                    b of X := f2(c of S1);
                    X(b of X, T2; d of X);
                    c of S0 := f3(d of X);
        S ⟶ t1:
                  c of S0 := f4(a of S0);
        endcase
    end S;

    procedure X(in b of X, T; out d of X)
        d of X := f5(b of X);
    end Y;
```

7.9 Esercizio

Di ognuna delle seguenti grammatiche, di cui sono dati i grafi delle dipendenze funzionali, si dica:

- se la grammatica è ben formata, ovvero se tutti gli attributi risultano senza ambiguità sintetizzati o ereditati;
- se è nella forma normale di Bochmann;
- se è della classe L;
- se è valutabile con una scansione;
- se è valutabile con un numero fisso di scansioni.

Si supponga che gli attributi siano inizializzati in modo opportuno

7.9.1 Grammatica *(i)*

Soluzione 7.9.1

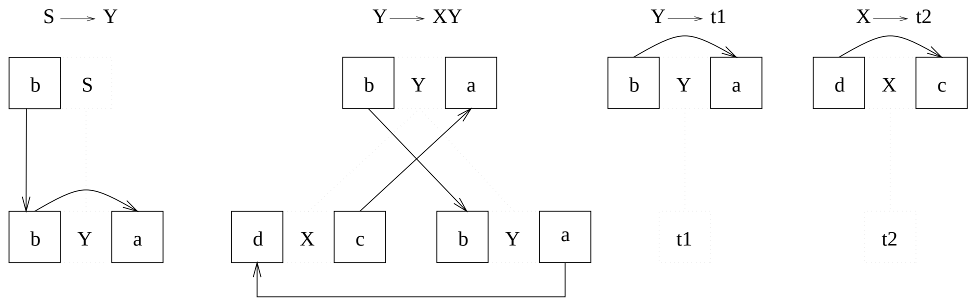

Figure 37: Grammatica (i)

- No, la grammatica non è ben formata perché l'attributo a risulta sia sintetizzato (per le dipendenze nelle produzioni $Y \longrightarrow XY$ e $Y \longrightarrow t_1$) che ereditato (per la dipendenza nella produzione $S \longrightarrow Y$). Per rendere corretta la grammatica, eliminiamo la dipendenza che compare nella produzione $S \longrightarrow Y$, rendendo l'attributo a sintetizzato. Si noti che se invece si invertisse il verso della freccia, si otterrebbe uniformità nella classificazione degli attributi, ma la grammatica risulterebbe ugualmente scorretta in quanto ciclica.
- La grammatica modificata è nella forma normale di Bochmann.
- Non è della classe L, in quanto in $Y \to XY$ c'è attributo $dofX$ che dipende da $aofX$
- La grammatica modificata è calcolabile con una scansione, scegliendo, per la produzione $Y \to XY$, l'ordine fra fratelli $Y \prec X$.

7.9.2 Grammatica *(ii)*

Soluzione 7.9.2

- La grammatica è ben formata.
- La grammatica è nella forma normale di Bochmann.
- La grammatica non appartiene alla classe L, perché esiste una dipendenza del primo figlio dal secondo nella produzione $S \to YX$.
- La grammatica non è valutabile con una scansione. Infatti, c'è un ciclo ($X \to Y \to X$) nel grafo dei fratelli della produzione $S \to YX$.
- Disegnando il grafo semplice delle dipendenze (rappresentato in Figura 39), si osservano due sottografi connessi massimali, evidenziati in Figura 40. Si può quindi determinare se gli attributi di ogni componente sono valutabili con una

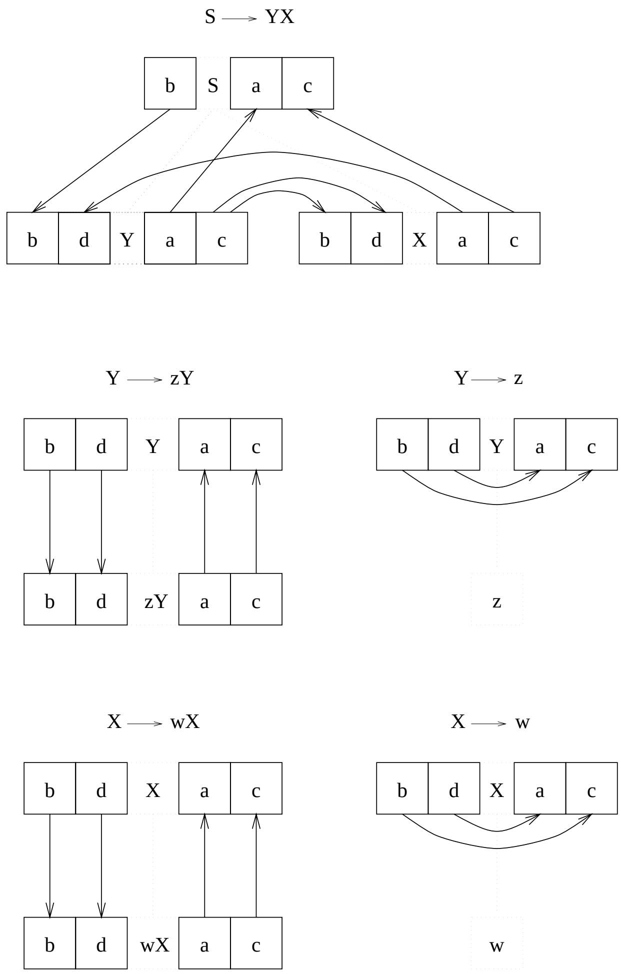

Figure 38: Grammatica *(ii)*

scansione. In entrambi i componenti, il grafo dei fratelli di $S \to YX$ risulta aciclico (con il singolo arco $Y \to X$ per il primo componente, e il singolo arco $X \to Y$ per il secondo). La coppia di attributi $\{b, c\}$ è quindi valutabile con una scansione che scende prima nel ramo Y e poi analizza il ramo X, mentre la coppia di attributi $\{a, d\}$ è valutabile da una scansione successiva che scende invece prima nel ramo X per poi analizzare il ramo Y. Le due scansioni in questo caso sono caratterizzate da ordini di visita differenti. Si osserva inoltre che le due scansioni non possono essere accorpate (altrimenti la grammatica risulterebbe ad una scansione, cosa che abbiamo mostrato non possibile).

7.9.3 Grammatica *(iii)*

Soluzione 7.9.3

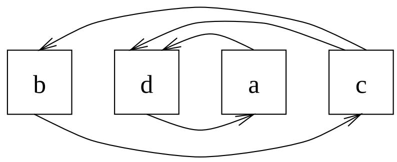

Figure 39: Grafo semplice delle dipendenze per la Grammatica (ii)

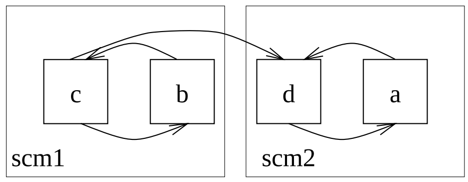

Figure 40: I sottografi connessi massimali del grafo semplice delle dipendenze per la Grammatica (ii)

- La grammatica è ben formata, a patto che gli attributi d *of* X in $S \to X$ e b *of* X in $S \to XT_2$ siano inizializzati.

- La grammatica è nella forma normale di Bochmann.

- La grammatica non è valutabile con una scansione e quindi non appartiene nemmeno alla classe L, in quanto vi sono, in $S \to X$ e $S \to Xt_2$, attributi ereditati che dipendono da attributi sintetizzati dello stesso nonterminale. Un valutatore semantico non sarebbe quindi in grado di calcolare tutti gli attributi ereditati di X prima di tutti gli attributi sintetizzati.

- Disegnando il grafo semplice delle dipendenze (rappresentato in Figura 42), si osserva un unico sottografo connesso massimale che racchiude tutti gli attributi. La grammatica non è quindi valutabile neanche con un numero superiore di scansioni. La grammatica presenta un ciclo nel grafo approssimato delle dipendenze, ma in effetti la grammatica è valutabile, in quanto i grafi delle dipendenze funzionali sono aciclici; ciò si può verificare costruendo i grafi delle dipendenze per ognuna delle quattro frasi $(t_3, t_4, t_3t_2, t_4t_2)$ generate dalla grammatica. Questo comportamento anomalo della grammatica è giustificato dal fatto che per ogni albero della grammatica c'è un proprio ordinamento topologico, non compatibile ocn quello di altri alberi. Ad esempio, per $S \to X \to t_3$ l'unico ordinamento è *abc*, mentre per $S \to Xt_2 \to t_4t_2$ l'ordinamento è *cda*. Si tratta pertanto di un caso degenere, abbastanza raro nella pratica, in cui il metodo di valutazione multiscansione fallisce, anche quando la grammatica non è ciclica.

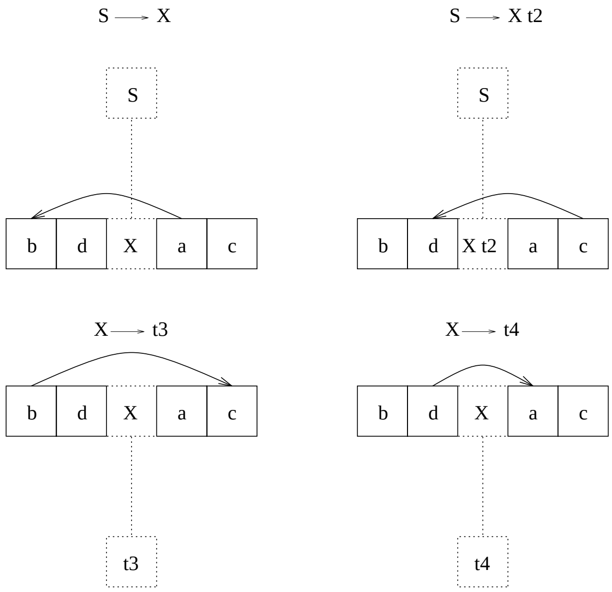

Figure 41: Grammatica *(iii)*

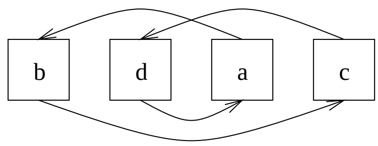

Figure 42: Grafo semplice delle dipendenze per la Grammatica (*iii*)

7.9.4 Grammatica *(iv)*

Soluzione 7.9.4

- La grammatica è ben formata.
- La grammatica non è nella forma normale di Bochmann, che richiede che tutti gli attributi interni (cioè che compaiono a sinistra di un assegnamento) di ogni produzione dipendano solo da attributi esterni (cioè che compaiono a destra di un assegnamento). Infatti nell'ambito della produzione $Y \longrightarrow XY$ compare la coppia di assegnamenti:

    1. $cofX := \eta(aofY_0)$
    2. $aofY_1 := \zeta(cofX, dofX)$

  L'attributo $cofX$ è interno (in quanto compare a sinistra di un assegnamento) e quindi $aofY_1$ è un attributo interno (si trova a sinistra di un assegnamento) che dipende a sua volta da un interno. Per rendere la grammatica in forma normale, basta sostituire i due assegnamenti precedenti con:

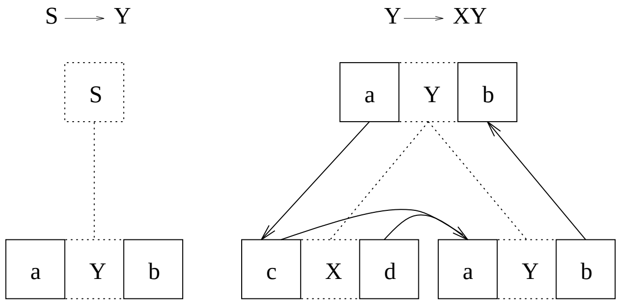

Figure 43: Grammatica *(iv)*

1. $cofX := \eta(aofY_0)$
2. $aofY_1 := \zeta(\eta(aofY_0), dofX)$

o

- La grammatica appartiene alla classe L.
- La grammatica è quindi anche valutabile con una scansione.

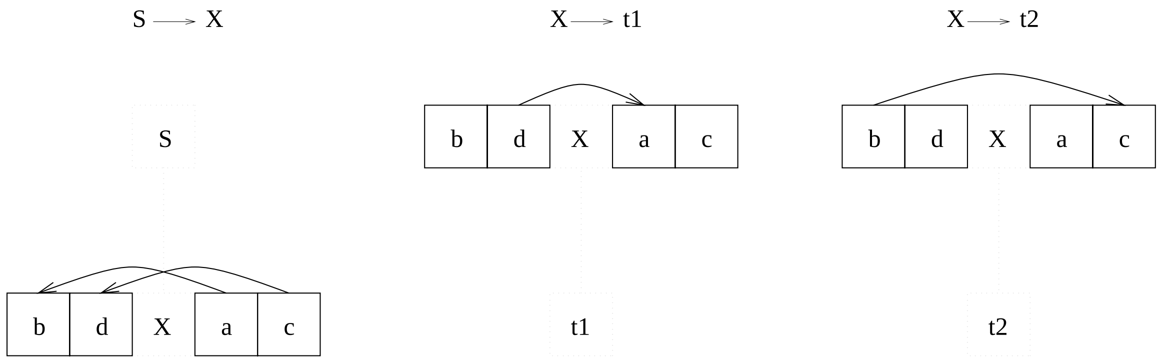

Figure 44: Grammatica (v)

7.9.5 Grammatica *(v)*

Soluzione 7.9.5

- La grammatica è ben formata.
- La grammatica è nella forma normale di Bochmann.

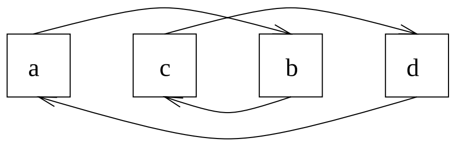

Figure 45: Grafo semplice delle dipendenze per la grammatica (v)

- Gli attributi di X nella produzione $S \longrightarrow X$ dipendono direttamente dagli attributi sintetizzati di X, e quindi la grammatica non rispetta le condizioni per essere valutabile in una scansione. Di conseguenza la grammatica non appartiene neanche alla classe L.
- La grammatica non è neanche valutabile in più scansioni in quanto non risulta possibile definire un ordinamento topologico tra gli attributi che valga per qualsiasi albero sintattico. Costruendo il grafo semplice delle dipendenze si ottiene infatti un grafo ciclico caratterizzato da un unico grafo connesso massimale (vedi Figura 45).

Si osserva che costruendo le dipendenze approssimate per la produzione $S \longrightarrow X$ si ottiene il ciclo: $a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$: la grammatica non è neanche assolutamente aciclica e anche il metodo di Katayama non può essere applicato. La grammatica risulta però aciclica, in quanto analizzando gli alberi sintattici corrispondenti alle due frasi del linguaggio non si osserva la presenza di cicli nelle dipendenze degli attributi. Si tratta ovviamente di un caso degenere.

7.10 Esercizio

È data la grammatica ad attributi G_A di fig. 46. Nei grafi i nomi delle funzioni semantiche sono scritti a fianco degli archi che collegano gli argomenti al risultato; ad es.:

- η of $Y := h(\sigma_1 of X_0, \eta of X_1)$;
- $\eta of X_0 := 1$

Si esaminino le dipendenze funzionali e si costruiscano le procedure ricorsive che valutano gli attributi, usando il metodo di Katayama.

Soluzione 7.10

La grammatica risulta assolutamente aciclica.

Utilizzando il metodo di Katayama costruiamo il seguente programma per il calcolo del valore degli attributi:

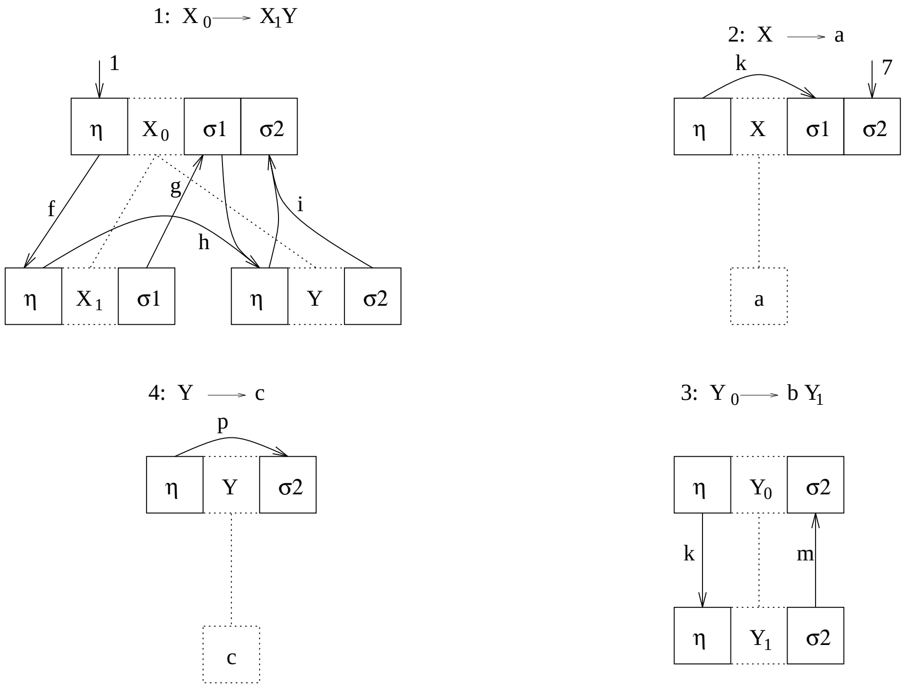

Figure 46: Grammatica ad attributi G_A

program

  procedure R_{X,σ_2}(**in** η *of* X_0, T; **out** σ_2 *of* X)

    case `alternativa(`T`) of`

      1: R_{X,σ_1}(η *of* X_0, T; σ_1 *of* X_0);

        η *of* X_1 := f(η *of* X_0);

        η *of* Y := h(η *of* X_1, σ_1 *of* X_0);

        R_{Y,σ_2}(η *of* Y, T_1; σ_2 *of* Y);

        σ_2 *of* X := i(η *of* Y, σ_2 *of* Y);

      2: σ_2 *of* X := 7;

    endcase

  end R_{X,σ_2};

  procedure R_{X,σ_1}(**in** η *of* X_0, T; **out** σ_1 *of* X_0)

    case `alternativa(`T`) of`

      1: η *of* X_1 := f(η *of* X_0);

        R_{X,σ_1}(η *of* X_1, T_1; σ_1 *of* X_1);

        σ_1 *of* X_0 := g(σ_1 *of* X_1);

      2: σ_1 *of* X_0 := k(η *of* X_0);

    endcase

  end R_{X,σ_1};

```
procedure R_{Y,σ2}(in η of Y0, T; out σ2 of Y0)
   case alternativa(T) of
        3:  η of Y1 := k(η of Y0);
            R_{Y,σ2}(η of Y1, T1; σ2 of Y1);
            σ2 of Y0 := m(σ2 of Y1);
        4:  σ2 of Y0 := p(η of Y0);
   endcase
end R_{Y,σ2};

Leggi l'albero T0;
R_{X,σ2}(1,T0; σ2 of X);
emetti il valore di σ2 of X
end
```

7.11 Esercizi proposti

7.11.1 Esercizio

Si deve calcolare per mezzo di una grammatica ad attributi il valore, scritto come una frazione, di una espressione di numeri frazionari. Ad esempio:

$1/3 + 5/2 + 2 + (1/2 + 3)/8 = 29/6$
$2/(1 + 2/5) + 1/8 = 223/112$
$(1/2 + 3)/0 = indefinito$

L'espressione può contenere i numeri interi, gli operatori + e / e le parentesi. Si progettino la sintassi del linguaggio, gli attributi semantici e le regole semantiche per il calcolo dell'attributo valore. Si disegni l'albero semantico decorato per il secondo esempio mostrato. Si indichi quale tecnica di valutazione degli attributi è possibile impiegare.

7.11.2 Esercizio

Un'agenzia marittima invia all'armatore dei telex codificati per annunciare la partenza dal porto (sigle NA, LI, GE, VE, TS, ...) di una o più navi. Ciascuna nave è individuata da un nome e dal prossimo porto di destinazione e contiene un insieme di container. Ogni container è individuato da un numero intero e da un porto di destinazione. All'arrivo, il telex deve essere controllato e decodificato. Ad esempio, il telex: NA (LAURA LI (1235 LI, 78990 GE)), (COSTA TS (67 TA, 3345 AN, 7899 VE)) deve essere tradotto nel messaggio:

"Da Napoli parte LAURA per Livorno con 2 container: 1235 per Livorno, 78990 per Genova; parte COSTA per Trieste con 3 container: 567 per Taranto, 3345 per Ancona, 7899 per Venezia".

Si definisca una sintassi per i telex e uno schema di traduzione ad attributi per controllare il telex (non possono esservi due navi con lo stesso nome, né due container con lo stesso numero, ecc.) e per stampare il messaggio in chiaro.

Si costruisca almeno in parte il programma che calcola la traduzione definita dallo schema.

7.11.3 Esercizio

Si progetti una grammatica (oppure uno schema di traduzione) ad attributi per calcolare la derivata $\partial/\partial x$ di una espressione. Una espressione è un polinomio i cui termini possono essere costanti intere (ad esempio 4 o 19), potenze della variabile x (ad esempio, x, $x^2, x^3 \ldots$) o prodotti di costanti e potenze. Ad esempio, $\partial/\partial x(x^2 + 5x^3) = 2x + 15x^2$.

Nel calcolo della derivata è possibile non semplificare le espressioni, lasciando anche termini o fattori inutili, come ad esempio $1x$.

Si modifichi la grammatica in modo che i termini possano contenere anche funzioni trigonometriche e loro potenze, come in $x^2 sin(x) + 3x^4 cos^3(x)$.

7.11.4 Esercizio

Si consideri la grammatica ad attributi G_a:

1: $S \longrightarrow X$	γ of $S := g_1(\gamma$ of $X)$
	α of $X := h_1(\beta$ of $X)$
2: $X_0 \longrightarrow X_1 Y$	β of $X_0 := f_2(\beta$ of X_1, β of $Y)$
	γ of $X_0 := g_2(\alpha$ of X_0, γ of X_1, γ of $Y)$
	α of $X_1 := h_2(\beta$ of $Y)$
	α of $Y := h_3(\alpha$ of $X_0)$
3: $X \longrightarrow Y$	β of $X := f_3(\beta$ of $Y)$
	γ of $X := g_3(\gamma$ of $Y)$
	α of $Y := h_3(\alpha$ of $X)$
4: $Y \longrightarrow a$	β of $Y := cost$
	γ of $Y := g_4(\alpha$ of Y, β of $Y)$

1. Si indichino gli attributi sintetizzati ed ereditati. Si verifichi se G_a è in forma normale di Bochmann. Si verifichi se G_a è assolutamente aciclica, e se è valutabile in una o più passate.

2. Si costruiscano le procedure semantiche del valutatore degli attributi.

7.11.5 Esercizio

È data la seguente grammatica ad attributi

1: $Z \longrightarrow X$	c of $X := 0$
	a of $X := 0$
	f of $Z := b$ of $X + d$ of X
2: $X_0 \longrightarrow Y X_1$	c of $X_1 := c$ of X_0
	d of $X_0 := d$ of X_1
	a of $X_1 := a$ of $X_0 + d$ of $Y + d$ of X_1
	c of $Y := a$ of X_0
	e of $Y := d$ of Y
	b of $X_0 := b$ of $Y + b$ of X_1
3: $Y \longrightarrow X$	a of $X := e$ of $Y + c$ of Y
	c of $X := c$ of Y
	b of $Y := b$ of X
4: $X \longrightarrow ab$	b of $X := a$ of X
	d of $X := c$ of X

1. Si determini di quale tipo sono le dipendenze funzionali tra gli attributi (L, S, a più passate, assolutamente aciclica, aciclica).

2. Si costruisca un valutatore degli attributi con una tecnica appropriata, supponendo noto l'albero sintattico.