Capitolo C14:
grammatiche e linguaggi acontestuali

Contenuti delle sezioni

25 pagine

C14:**0.01** *In questo capitolo introduciamo le grammatiche acontestuali, entità che si possono considerare meccanismi basati su regole di riscrittura di stringhe che consentono di individuare linguaggi formali i quali, a loro volta, pemettono di trattare importanti elementi di gran parte dei linguaggi artificiali di rilevante interesse applicativo.*

Le grammatiche acontestuali hanno il vantaggio di facilitare l'attribuzione di strutture significative alle stringhe che generano; esse inoltre forniscono le basi per le procedure l'interpretazione efficiente da parte del computer delle formule matematiche consentite da molti linguaggi per l'elaborazione dei dati.

In questo capitolo vengono presentate le proprietà di base delle grammatiche e dei linguaggi acontestuali ed in particolare i meccanisni che consentono di dare a queste grammatiche forme che risultano vantaggiose per individuare varie caratteristiche dei linguaggi generati e meccanismi di grande importanza nelle applicazioni riguardanti i linguaggi procedurali e la loro compilazione, tema accennato nelle ultime sezioni che verrà ripreso successivamente.

Presenteremo anche alcune grammatiche acontestuali piuttosto semplici al fine di dare una prima idea della possibilità di controllare espressioni algebriche ed espressioni similari, sia per quanto riguarda il loro significato operativo, sia per la possibilità di automatizzare la loro interpretazione.

A fianco delle grammatiche acontestuali introdurremo anche i riconoscitori a pila, le macchine sequenziali che si possono considerare riconoscitori di Rabin e Scott potenziati con l'aggiunta di un dispositivo, la pila, che consente loro di accettare o rifiutare le stringhe dei linguaggi acontestuali, sostanzialmente più elaborati dei razionali.

C14:a. produzioni, grammatiche e linguaggi acontestuali

C14:**a.01** Per definire una grammatica acontestuale servono due alfabeti disgiunti, il primo costituito dai simboli che formano le stringhe del linguaggio che la grammatica individua, il secondo formato da simboli che intervengono nei processi formali per la "generazione" delle stringhe del linguaggio.

Questi due insiemi di simboli vengono detti, risp., **alfabeto dei terminali** ed **alfabeto degli ausiliari** e tipicamente verranno denotati, risp., con T e X o con loro lievi varianti.

Nel seguito di questo capitolo utilizzeremo prevalentemente lettere maiuscole come A, B ed S per denotare simboli ausiliari, lettere minuscole della `presente fonte` per i segni terminali, lettere come u, v e w per le stringhe di terminali e lettere greche come α e β per le stringhe che possono contenere sia ausiliari che terminali.

C14:**a.02** Consideriamo quindi due alfabeti disgiunti T ed X e denotiamo con V la loro unione.

Diciamo **arborescenza di derivazione elementare** o **arborescenza.DE** relativa ad X e T una arborescenza-1, cioè una arborescenza distesa di altezza 1 con la radice etichettata da un carattere di X e con i nodi foglia etichettati da cartteri di V.

Si considerano arborescenze.DE anche quelle con una sola foglia etichettata da $\sqcup$.

Un tale digrafo arricchito si può considerare come la rappresentazione grafica di una cosiddetta **produzione acontestuale** sugli alfabeti X e T: con questo termine si intende una coppia $\langle X, \sigma\rangle \in \mathsf{X} \times \mathsf{V}^*$ che usualmente si presenta con la scrittura $X \to \sigma$.
Per esempio le produzioni

$$S \to \texttt{()} \quad , \quad A \to \texttt{(}A + A\texttt{)} \quad \text{e} \quad B \to \sqcup$$

sono rappresentate, risp., dalle arborescenze-1 etichettate:

//input pC14a02

Le arborescenze.DE si possono considerare come le componenti elementari delle cosiddette **arborescenze di derivazione** o **arborescenze.der** Queste sono arborescenze distese con i nodi etichettati che si possono ottenere con successive operazioni di innesto di arborescenze.DE, operazioni ternarie nelle quali si possono fondere un nodo foglia e un nodo radice solo se portano la stessa etichetta.

C14:**a.03** Si dice **grammatica acontestuale** o context free grammar o concisamente **grammatica-F** una quaterna:

$$\boldsymbol{G} \;=\; \big\langle S, \mathsf{X}, \mathsf{T}, \mathcal{P}\big\rangle \;,$$

dove X e T sono due alfabeti disgiunti, S è un elemento di X detto **simbolo di partenza** delle derivazioni (o anche assioma della grammatica), $\mathcal{P}$ è una collezione finita di produzioni acontestuali su X e su T.

Due semplici grammatiche-F sono le seguenti:

$$\boldsymbol{G}_a = \big\langle\ S\ ,\ \{S\}\ ,\ \{\texttt{(},\texttt{)}\}\ ,\ \{\ S \to \texttt{(}S\texttt{)}\ ,\ S \to \texttt{()}\}\ \big\rangle\ ,$$
$$\boldsymbol{G}_{Dyck} = \big\langle S, \{S\}, \{\texttt{(},\texttt{)}\}\ ,\ \{S \to \texttt{()}\ ,\ S \to \texttt{(}S\texttt{)}\ ,\ S \to S\,S\}\big\rangle\ .$$

L'insieme delle grammatiche-F lo denotiamo con **GrmF**

C14:**a.04** *In seguito vedremo grammatiche di tipi più generali delle acontestuali; in questo capitolo ci limitiamo alle grammatiche di questo tipo e per brevità le chiameremo semplicemente grammatiche e ometteremo l'aggettivo acontestuale anche parlando di produzioni e derivazioni.*

C14:**a.05** Ad una grammatica-F $\boldsymbol{G} = \langle S, \mathsf{X}, \mathsf{T}, \mathcal{P} \rangle$ risulta associato l'insieme delle arborescenze.DE costruibili con le sue produzioni.

Diciamo **prodotto di un'arborescenza di derivazione** della grammatica $\boldsymbol{G}$ la stringa di V^* che si ottiene leggendo da sinistra a destra i simboli che costituiscono le etichette delle sue foglie.
Il prodotto di un'arborescenza.DE si chiama anche schema di frase.

Diciamo **arborescenza generativa di una grammatica-F** ogni arborescenza di derivazione la quale abbia la radice etichettata dal simbolo di partenza e le foglie etichettate solo da simboli terminali o dalla stringa muta.

Diciamo inoltre **linguaggio generato dalla grammatica-F** $\boldsymbol{G}$ l'insieme delle stringhe che sono il prodotto delle arborescenze generative della $\boldsymbol{G}$ stessa: questo linguaggio si denoterà con $\boldsymbol{G}^{\mathcal{G}\mathrm{en}}$.

Si dice **linguaggio acontestuale** un linguaggio che può essere generato da una grammatica acontestuale.

L'insieme dei linguaggi acontestuali sarà denotato con **LngF** o con **F** ed estendendo il dominio della funzione $^{\mathcal{G}\mathrm{en}}$ agli insiemi di grammatiche possiamo enunciare

$$\mathbf{LngF} \;=\; \mathbf{GrmF}^{\mathcal{G}\mathrm{en}} \quad .$$

C14:**a.06** Esaminiamo la grammatica-F:

$$\boldsymbol{G}_a \;:=\; \langle\; S \;,\; \{S\} \;,\; \{\texttt{(},\texttt{)}\} \;,\; \{\; S \to \texttt{(}S\texttt{)} \;,\; S \to \texttt{()}\} \;\rangle \;.$$

Non è difficile rendersi conto che l'insieme delle arborescenze di generazione della grammatica $\boldsymbol{G}_a$ è costituito dalla seguente successione:

```
//input pC14a06
```

Il linguaggio generato da $\boldsymbol{G}_a$ è quindi formato da tutte le stringhe che presentano un certo numero positivo di parentesi aperte seguite da un ugual numero di parentesi chiuse:

$$\boldsymbol{G}_a{}^{\mathcal{G}\mathrm{en}} \;:=\; \{\; \texttt{()},\; \texttt{(())},\; \texttt{((()))},\ldots\} \;=\; \sum_{n \,\in\, \mathbb{P}} \texttt{(}^n\texttt{)}^n .$$

C14:**a.07** Un'altro esempio è dato da:

$$\boldsymbol{G}_b \;:=\; \langle\; S, \{S\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{S \to \mathtt{aa}S\mathtt{ccc}, S \to \mathtt{b}\} \;\rangle .$$

L'insieme delle arborescenze di generazione di $\boldsymbol{G}_b$ è fornito dalla seguente successione:

```
//input pC14a07
```

Il linguaggio generato da $\boldsymbol{G}_b$ è formato da tutte le stringhe che presentano un numero pari $2\,n$ di lettere `a` seguite da una `b` e da $3\,n$ lettere `c`:

$$\boldsymbol{G}_b{}^{\mathcal{G}\mathrm{en}} \;=\; \{\; \mathtt{b}, \mathtt{aabccc}, \mathtt{aaabcccccc}, \ldots \;=\; \sum_{n \,\in\, \mathbb{P}} \mathtt{a}^{2n}\,\mathtt{bc}^{3n} .$$

C14:**a.08** Le grammatiche acontestuali (insieme a vari altri sistemi di riscrittura) hanno numerose applicazioni che richiedono di entrare nei dettagli del loro modo di operare e delle loro implementazioni. Non meraviglia quindi che sia necessario esaminarle anche attraverso modalità specifiche per la definizione formale di loro raggruppamenti.

Qui prima di trattare altri esempi conviene introdurre una modalità di scrittura abbreviata per le grammatiche-F che chiameremo prima notazione delle GrmF.

Se in una grammatica si hanno più produzioni con lo stesso ausiliario X a primo membro, invece di scrivere

$$X \to \sigma_1,\ X \to \sigma_2,\ ...,\ X \to \sigma_r$$

useremo la scrittura più sintetica:

$$X \to \sigma_1 \,|\, \sigma_2 \,|\, ... \,|\, \sigma_r .$$

Il simbolo "$|$" che compare in questa notazione ha il significato di alternativa: esso potrebbe leggersi come "oppure" o come *vel*.

La prima notazione delle GrmF segue la convenzione di denotare con lettere maiuscole i caratteri ausiliari e con lettere minuscole i terminali. In tal modo si possono evitare le indicazioni esplicite di X e T.

Si conviene inoltre di presentare come primo gruppo di produzioni quello avente a primo membro il simbolo di partenza, in modo da evitare il primo membro della quaterna usata nella definizione generale.
La prima notazione delle GrmF si può quindi ridurre alla presentazione dei raggruppamenti delle produzioni.

L'ultima convenzione per la nostra notazione consiste nel delimitare i raggruppamenti delle produzioni tra le parentesi specifiche $\lceil$ e $\rfloor$.

La scrittura abbreviata per le grammatiche precedenti è quindi:

$$\boldsymbol{G}_a = \lceil S \to (S) \,|\, () \rfloor \quad , \qquad \boldsymbol{G}_b = \lceil S \to \mathtt{aa}S\mathtt{ccc} \,|\, \mathtt{b} \rfloor \ .$$

Da una tale notazione abbreviata si possono ricavare tutti gli elementi della scrittura completa anche se non si rispetta la regola delle lettere maiuscole per i caratteri variabili e delle lettere minuscole per carateeri terminali. Infatti i primi membri dei raggruppamenti di produzioni costituiscono l'insieme dei simboli ausiliari; il primo ausiliario incontrato è il simbolo di partenza; tutti gli altri simboli che compaiono nei secondi membri delle produzioni costituiscono i simboli terminali.

C14:**a.09** Si dice **grammatica-F lineare** una grammatica-F $\boldsymbol{G} = \langle S, \mathsf{X}, \mathsf{T}, \mathcal{P} \rangle$ le cui produzioni sono della forma $X \to w\,Y\,z$ o $X \to \sqcup$ con $X, Y \in \mathsf{X}$ e $w, z \in \mathsf{T}^*$, cioè con $\mathcal{P} \subset_\varphi \mathsf{X} \times (\mathsf{T}^* \times \mathsf{T}^*) \cup \{ \sqcup \}$

Le produzioni dei tipi precedenti presentano un secondo membro che se diverso da $\sqcup$, contiene un solo segno ausiliario e si dicono lineari.

Il termine "lineare" viene suggerito dall'algebra dei polinomi e dal parallelo tra variabili e costanti nei polinomi usuali e simboli ausiliari e simboli terminali nelle grammatiche-F.

Le due grammatiche precedenti $\boldsymbol{G}_a$ e $\boldsymbol{G}_b$ sono grammatice-F lineari, in quanto sono costituite da produzioni che a secondo membro presentano al più un ausiliario.

Le arborescenze di derivazione delle grammatiche-F-lineari hanno un caratteristico aspetto a "lisca di pesce".

Caso particolare sono le **grammatiche-F lineari a destra**, grammatiche le cui produzioni sono della forma $X \to w\,Y$, con $w \in \mathsf{T}^*$, oppure della forma $X \to \sqcup$. I linguaggi generati da queste grammatiche ci sono già noti.

C14:**a.10** **Prop.** Le grammatiche-F lineari a destra generano tutti e soli i linguaggi razionali.

Dim.: Si tratta di precisare come associare a una generica grammatica lineare a destra $\boldsymbol{G}$ un riconoscitore-RSND che accetta il linguaggio $\boldsymbol{G}^{\mathcal{G}\mathrm{en}}$ e di stabilire come far corrispondere a un generico riconoscitore-RSD $\boldsymbol{R}$ una grammatica lineare a destra che genera il linguaggio $\boldsymbol{R}^{\mathcal{G}\mathrm{en}}$.

Vediamo quindi come si può trasformare una grammatica lineare a destra $\langle S, \mathsf{X}, \mathsf{T}, \mathcal{P} \rangle$ in un riconoscitore.
Innanzi tutto si può trasformare la grammatica "snellendo" le sue produzioni, cioè facendo in modo che le sue produzioni, oltre che del tipo $X \to \sqcup$, possano essere solo del tipo $X \to \mathtt{a}Y$, cioè siano contenute in $\mathsf{X} \times \mathsf{TX}$.
In effetti ogni produzione della forma $X \to \mathtt{a}_{j_1}\mathtt{a}_{j_2}...\mathtt{a}_{j_r}Y$, introducendo $r-1$ nuovi ausiliari X_1, ..., X_{r-1}, si può rimpiazzare, con le $X \to \mathtt{a}_{j_1}X_1$, $X_1 \to \mathtt{a}_{j_2}X_2$, ..., $X_{r-1} \to \mathtt{a}_{j_r}Y$, senza cambiare il linguaggio generato: la modifica comporta che nelle arborescenze di derivazione e di generazione ogni occorrenza della arborescenza.DE sia rimpiazzata da una sottoarborescenza di altezza $r-1$ con una tipica "forma a pettine" formata da r arborescenze.DE con due figli, in modo che non cambino i prodotti delle arborescenze di generazione.

//input pC14a10

C14:**a.11** La nuova grammatica viene poi sostituita dal riconoscitore sull'alfabeto T che ha come stati gli ausiliari nonché un nuovo stato Z che è l'unico stato finale, come solo stato iniziale S e come transizioni le terne $\langle X, \mathtt{a}, Y \rangle$ corrispondenti alle produzioni $X \to \mathtt{a}Y$, le terne $\langle X, \mathtt{a}, Z \rangle$ corrispondenti alle produzioni $X \to \mathtt{a}$ e le terne $\langle X, \sqcup, Z \rangle$ corrispondenti alle produzioni $X \to \mu$.

Si vede allora che le stringhe accettate dal nuovo riconoscitore sono esattamente quelle generate dalla grammatica: le passeggiate utili sul riconoscitore corrispondono alle arborescenze di generazione della grammatica.
Quindi ogni linguaggio generato da una grammatica lineare a destra è razionale.

Mostriamo ora come ricavare da un riconoscitore di Rabin e Scott deterministici una grammatica lineare a destra equivalente.

Come ausiliari di questa grammatica assumiamo gli stati del riconoscitore. Dato che si assume che il riconoscitore ha un solo stato iniziale, ad esso si affida il ruolo di ausiliario di partenza.

Per l'insieme delle produzioni, in corrispondenza a ogni transizione $\langle X, \mathtt{a}, Y \rangle$ poniamo la $X \to \mathtt{a}Y$ e in corrispondenza a ogni stato finale Z la produzione $Z \to \sqcup$.
Con considerazioni analoghe alle precedenti si vede che le stringhe generate dalla nuova grammatica sono esattamente quelle accettate dal riconoscitore dato ∎

A questo punto possiamo affermare che le grammatiche acontestuali costituiscono uno strumento per trattare linguaggi di portata superiore dei riconoscitori di Rabin e Scott; infatti l'insieme dei linguaggi acontestuali, contenendo linguaggi dotati di infinite derivate da sinistra come $\sum_{n\in\mathbb{N}} (^n)^n$, include propriamente l'insieme dei linguaggi razionali.

C14:b. stringhe di parentesi e depositi a pila

C14:**b.01** Studiamo ora una grammatica con molte applicazioni chiamata **grammatica di Dyck**,

$$\boldsymbol{G}_{Dyck} = \left\lceil S \to \texttt{()} \,\middle|\, \texttt{(}S\texttt{)} \,\middle|\, SS \right\rceil .$$

Osserviamo che essa possiede una produzione non lineare con lo stesso ausiliario nel primo e nel secondo membro; questo comporta che il linguaggio che essa genera, $\mathsf{L}_c := \boldsymbol{G}_c^{\mathcal{G}\text{en}}$, è meno facile da descrivere di quelli visti in precedenza.
Osserviamo innanzi tutto che $\mathsf{L}_a = \sum_{n\in\mathbb{N}} \texttt{(}^n\texttt{)}^n \subseteq \mathsf{L}_c$, in quanto l'insieme delle produzioni di $\boldsymbol{G}_c$ include quello relativo a $\boldsymbol{G}_a$.

In generale una grammatica con un insieme di produzioni più esteso di una seconda è sicuramente capace di generare tutte le stringhe generate da quest'ultima.

In effetti in L_c si trovano molte stringhe estranee ad L_a: alcune come:

```
(()) ()   e   () ((())) (())
```

sono ottenute giustapponendo stringhe di L_a; altre come:

```
( (())() )   e   ( (())()((())) )
```

sono ottenute delimitando tra coppie di parentesi stringhe dei tipi precedenti.

Si constata che le prime sono ottenute dalle derivazioni che presentan come produzione iniziale la $S \to S\,S$, mentre le seconde provengono dalle derivazioni ch hanno la $S \to \texttt{(}\,S\,\texttt{)}$ come prima produzione.

Questo ampliamento si può portare avanti illimitatamente, cioè in L_c si possono trovare stringhe ottenute con un numero arbitrario di operazioni di giustapposizione e di delimitazione tra parentesi di stringhe di L_a come le seguenti:

```
(((()(()))(()()(()))))    e    (((()(()))(())(((())())())))
```

C14:**b.02** Si può comunque dire che le stringhe di L_c sono costituite da coppie di parentesi che si giustappongono e che le coppie si annidano le une nelle altre.

Si può stabilire se una stringa w, composta da parentesi tonde, appartiene a L_c con un procedimento di riduzione progressiva che consiste nel cancellare via via dalla stringa trattata coppie di parentesi `()` contigue: la stringa w è corretta sse questo processo può condurre all'eliminazione di tutte le parentesi.

Un algoritmo ancora più semplice ed efficiente, per stabilire sse una stringa $w \in \{\texttt{(},\texttt{)}\}^*$ appartiene ad L_c procede a scorrere la stringa da sinistra verso destra tenendo conto ad ogni passo del numero delle parentesi aperte incontrate diminuito del numero delle parentesi chiuse; in ogni momento dello scorrimento si chiede che il numero delle parentesi aperte sia maggiore o uguale del numero delle parentesi chiuse e che alla fine della stringa i due numeri coincidano.

Una esecuzione di questo algoritmo nel caso di alcune stringhe di L_c incontrate in precedenza è illustrata dai seguenti schemi:

```
( ( ) ) ( )          ( ( ( ) ) ( ) ( ( ( ) ) ) )
0 1 2 1 0 1 0        0 1 2 3 2 1 2 1 2 3 4 3 2 1 0
```

Due processi con rifiuto di stringhe di parentesi tonde sono invece:

```
( ) ( ( ( ) ) ( ( ) )          ( ( ( ) ( ) ) ) ) ( ( )
0 1 0 1 2 3 2 1 2 3 2 1         0 1 2 3 2 3 2 1 0 -1
```

Le stringhe di L_c si possono considerare ottenute da espressioni generiche contenenti solo parentesi tonde cancellando tutti gli operandi e gli operatori. Una stringa di Dyck fornisce dunque una caratteristica essenziale di varie espressioni: per ciascuna di esse dice in qual modo si articola in sottoespressioni.

C14:**b.03** Nella scrittura di espressioni estese spesso si trova comodo servirsi di coppie di parentesi di diversa forma e di diversa estensione per evidenziare meglio i loro accoppiamenti.
Questa constatazione suggerisce generalizzazioni della grammatica $\boldsymbol{G}_c$ in grado di tenere sotto controllo stringhe di sole parentesi ricavate da queste espressioni. Nel caso di tre forme di parentesi, le tonde, le quadre e le graffe, si ha la seguente grammatica:

$$\boldsymbol{G}_d = \left[\; S \to \texttt{()} \,\middle|\, \texttt{[]} \,\middle|\, \texttt{\{\}} \,\middle|\, \texttt{(}S\texttt{)} \,\middle|\, \texttt{[}S\texttt{]} \,\middle|\, \texttt{\{}S\texttt{\}} \,\middle|\, SS \; \right] .$$

Esempi di stringhe di $\mathsf{L}_d = \boldsymbol{G}_d^{\mathcal{G}\mathrm{en}}$ sono:

`{[()]}` `[()()()]` `{[()]()[(())()]}[()] ;`

le corrispondenti arborescenze generative sono:

?DSGN.31:.5 //input pC14b03

//input pC14b03B

C14:**b.04** Cerchiamo ora un procedimento per l'accettazione di queste stringhe. Quello visto per L_c potrebbe indurre a servirsi di tre contatori in grado di garantire che a ogni punto non si siano incontrate più parentesi chiuse delle aperte corrispondenti e che alla fine ogni parentesi aperta sia seguita da una coniugata chiusa.
Questo procedimento però non permette di scartare stringhe come la `[()]` nelle quali tra due parentesi accoppiate si trovano parentesi non accoppiate.

Queste situazioni inaccettabili vengono rivelate da un meccanismo elementare chiamato **deposito a pila** o pushdown store o stack che risulta di estesa portata algoritmica e merita di essere approfondito.

Un deposito a pila è in grado di contenere una sequenza di simboli che possiamo descrivere come disposti uno sopra l'altro e che consente di leggere, eliminare e aggiungere solo i simboli nelle posizioni più in alto.

Una tale descrizione si può chiamare basso–alto: alternativamente talora può essere più comoda una descrizione sinistra–destra secondo la quale i simboli sono affiancati e possono essere consultati, scritti o cancellati solo all'estremità destra.

//input pC14b04

Un deposito a pila si può assimilare a un binario morto nel quale si fanno stazionare vagoni o locomotive che momentaneamente non servono: gli ultimi che vi sono stati parcheggiati sono i primi che possono essere recuperati.

Un deposito di questo tipo viene detto anche **deposito LIFO**, ove LIFO è acronimo dell'espressione inglese *Last In First Out* che caratterizza le manovre alle quali possono essere sottoposti gli oggetti in esso stazionati: in ogni momento è l'ultimo inserito nel deposito quello che potrà essere estratto per primo.

Un'altra descrizione intuitiva del deposito a pila lo assimila a un contenitore cilindrico di monete dotato alla base di un pistone che viene spinto in alto da una molla. Quando viene inserita una nuova moneta

la molla viene compressa e il pistone si abbassa consentendo solo alla moneta più in alto di emergere in modo da poter essere ripresa. Quando si estrae la moneta più in alto la molla spinge in alto le monete rimaste consentendo la ripresa della moneta che precedentemente si trovava nella seconda posizione dall'alto.

C14:**b.05** Un deposito a pila si può implementare in ogni linguaggio di programmazione procedurale in modo estremamente semplice mediante un array monodimensionale che in ogni istante è riempito solo nelle prime posizioni e mediante una variabile intera che individua la posizione attuale della sua cosiddetta **testa**, cioè la sua posizione estrema (la più in alto nella raffigurazione basso–alto, la più a destra nella raffigurazione sinistra–destra).

Alla base della pila deve essere posto un simbolo particolare, diverso da tutti quelli che possono essere successivamente inseriti, in modo da segnalare nel modo più semplice la situazione nella quale la pila non contiene alcun oggetto depositabile.

Le due operazioni di base per una pila sono l'inserimento di un nuovo simbolo, chiamata operazione **push**, e l'estrazione di un oggetto, operazione **pop**.

Si tratta di operazioni assai semplici e il pregio dei depositi a pila consiste nel fatto che opportunamente utilizzati permettono di effettuare manovre molto incisive.

C14:**b.06** Vediamo ora come riconoscere se una stringa w di segni del genere parentesi, anche di forme diverse, appartiene ad esempio al linguaggio L_d attraverso un solo scorrimento da sinistra a destra e servendosi di un deposito a pila.

Inizialmente nella pila si trova solo il simbolo di base. Ad ogni passo se nella w viene letta una parentesi aperta si inserisce nella pila tale simbolo.
Se invece viene letta una parentesi chiusa occorre controllare la testa della pila: se si trova la corrispondente parentesi aperta, la si cancella abbassando la testa del deposito e si prosegue; in caso contrario si è individuata una parentesi chiusa non accoppiata con una aperta precedente e si rifiuta la stringa.

Se in corrispondenza di una parentesi chiusa in testa alla pila si ha il simbolo di base, si rifiuta la w per eccesso di parentesi chiuse. Se alla conclusione nella pila rimangono ancora parentesi aperte, si rifiuta la w per eccesso di parentesi aperte.
Si ha l'accettazione della w solo nel caso di conclusione con deposito vuoto, cioè con la pila contenente il solo simbolo di base.

C14:**b.07** Osserviamo che L_d, linguaggio infinito, possiede stringhe lunghe quanto si vuole; quindi il deposito a pila usato deve essere illimitatamente estendibile, deve essere in grado di registrare una quantità illimitata di informazioni.
Questa illimitatezza (potenziale) rende il deposito a pila un meccanismo di memoria sostanzialmente superiore agli stati, in numero finito, consentiti ai riconoscitori di Rabin e Scott.

Si osserva anche che non è possibile riuscire ad accettare un linguaggio a parentesi aumentando il numero degli stati di un riconoscitore-RSND. Anche limitandoci al linguaggio-F $\boldsymbol{G}_{Dyck}{}^{\mathcal{G}\mathrm{en}}$ la illimitatezza potenziale della lunghezza delle sue stringhe implica la illimitatezza del numero degli stati.

C14:c. portata delle grammatiche acontestuali

C14:**c.01** Le grammatiche, come tutti gli strumenti che permettono di individuare linguaggi, possono presentare numerose varianti equivalenti. In effetti la definizione di grammatica lascia molte libertà ed in una grammatica potrebbero comparire simboli e produzioni che non intervengono in alcuna derivazione di stringhe del linguaggio generabile, anche componenti palesemente inutili.

Procederemo ora a individuare alcune forme normali delle grammatiche che permettono di evitare la presenza di elementi superflui e di servirsi di produzioni particolari in modo da facilitare lo studio dei linguaggi acontestuali.

C14:**c.02** I simboli ausiliari delle grammatiche, come accade per gli stati dei riconoscitori e come ogni altro raggruppamento di componenti strumentale di ogni meccanismo formale, possono essere modificati con una qualsiasi biiezione.
Dunque se si deve operare con due meccanismi formali (generatori o accettatori) grammatiche è lecito supporre che gli insiemi dei rispettivicomponenti strumentali siano disgiunti. In particolare se si trattano due grammatiche a struttura di frase si può assumere che si servano di due insiemi disgiunti di segni ausiliari.

C14:**c.03** Preliminarmente introduciamo una configurazione grafica associata a ogni grammatica a struttura di frase $\boldsymbol{G} = \langle S, \mathsf{X}, \mathsf{T}, \mathcal{P} \rangle$ che costituisce un utile riferimento per molte delle considerazioni che seguono.

Si chiama **digrafo delle dipendenze** tra i simboli della $\boldsymbol{G}$ il digrafo che ha come nodi i simboli ausiliari e terminali della $\boldsymbol{G}$ ed ha come archi tutte le coppie $\langle X, \mathsf{a} \rangle$ ottenibili da produzioni della forma $u\,X\,v \;\to\; x\,\mathsf{a}\,y$ con $u, v, x, y \in (\mathsf{X} \cup \mathsf{T})^*$; inoltre in presenza di produzioni della forma $X \to \sqcup$ vanno aggiunti al digafo il nodo μ e gli archi $\langle X, \mu \rangle$.
Questo digrafo lo denotiamo con $\boldsymbol{G}^{ddG}$.

C14:**c.04** Un carattere ausiliario di una grammatica si dice **ausiliario fecondo** sse permette di generare stringhe di soli terminali.

I caratteri fecondi si possono distinguere dagli eventuali nonfecondi costruendo il loro insieme per fasi successive.

Nella prima fase si segnano come fecondi gli ausiliari, e i nodi del digrafo delle dipendenze, che sono primi membri di produzioni aventi secondi membri costituiti solo da terminali o dalla $\sqcup$. Evidentemente se non si trovano tali caratteri la grammatica non genera alcuna stringa.

In ciascuna delle fasi successive si segnano come fecondi gli ausiliari primi membri di produzioni aventi secondi membri costituiti solo da terminali e da ausiliari già trovati fecondi. Per questo risulta utile servirsi del digrafo delle dipendenze.
Il processo si conclude o con l'esclusione della presenza di ausiliari nonfecondi o con la prima fase che non individua nessun nuovo ausiliario fecondo.

Gli eventuali ausiliari rimasti non segnati non possono essere etichette di radice di alcuna arborescenza di derivazione con foglie costituite solo da terminali e quindi non possono essere etichetta di alcuna sottoarborescenza della arborescenza di generazione della grammatica.

L'eliminazione da una grammatica degli ausiliari nonfecondi e di tutte le produzioni che li coinvolgono non riduce il linguaggio generato: ci si può quindi limitare a considerare grammatiche prive di ausiliari non fecondi.

Si ha quindi il seguente criterio per la vacuità dei linguaggi acontestuali.

C14:**c.05** **Prop.** Il linguaggio generato da una grammatica $\boldsymbol{G}$ è vuoto sse il suo simbolo di partenza non è fecondo ∎

Può essere utile che il carattere di partenza non compaia nei secondi membri delle produzioni. Per una grammatica priva di questo requisito è sufficiente introdurre un nuovo carattere S', fargli assumere il ruolo di carattere di partenza ed introdurre la produzione $S' \to S$.

C14:**c.06** Tra le produzioni acontestuali quelle della forma $X \to \sqcup$ sono le sole ad avere il secondo membro di lunghezza inferiore al primo. Come vedremo può essere utile eliminare queste produzioni, eccettuata al più la $S \to \sqcup$, senza modificare il linguaggio generato.

Anche questa normalizzazione è attuabile con un algoritmo valido per tutte le grammatiche-F.
Il procedimento prevede successive fari in ciascuna delle quali si elimina una produzione $X \to \sqcup$ con $X \neq S$. Per questo occorre aggiungere alla grammatica produzioni corrispondenti a ogni produzione che presenti l'ausiliario X a secondo membro.

Una produzione $Y \to \alpha\, X\, \beta$ con X assente da α e β va affiancata dalla $Y \to \alpha\beta$; ad una produzione $Y \to \alpha X \beta X \gamma$ con X assente da α, β e γ si devono aggiungere $Y \to \alpha X \beta\gamma$, $Y \to \alpha\beta X\gamma$ e $Y \to \alpha\beta\gamma$.
In generale a una produzione che presenta k occorrenze di X nel secondo membro si devono affiancare le $2^k - 1$ produzioni in ciascuna delle quali venga cancellato uno dei sottoinsiemi propri di occorrenze della X.

Le arborescenze di generazione della nuova grammatica differiscono da quelle della precedente semplicemente per la cancellazione delle passeggiate aventi i nodi diversi dall'iniziale e dal finale etichettati dai precedenti ausiliari e aventi il nodo finale etichettato da $\sqcup$.
Quindi la nuova grammatica equivale a quella di partenza.

C14:**c.07** Dalla costruzione precedente deriva anche il seguente criterio per la presenza della $\sqcup$ nei linguaggi acontestuali.

Prop. Un linguaggio generato da una grammatica $\boldsymbol{G}$ contiene la stringa muta $\sqcup$ sse la riduzione della $\boldsymbol{G}$ nella forma precedente lascia presente la produzione $S \to \sqcup$ ∎

C14:**c.08** Un'altra normalizzazione delle grammatiche riguarda l'eliminazione delle produzioni delle forma $X \to Y$, con X ed Y ausiliari.

Per realizzare questa eliminazione occorre innanzi tutto precisare, ad esempio ricavandolo come sottodigrafo del digrafo delle dipendenze, il digrafo i cui nodi sono gli ausiliari e i cui archi sono le coppie $\langle X, Y\rangle$ corrispondenti alle produzioni $X \to Y$. Le passeggiate di questo digrafo consentono di individuare tutte le coppie di ausiliari X e Z per i quali si abbia una catena di produzioni $X \to X_1 \to ... \to X_{t-1} \to Z$.
Evidentemente non cambia il linguaggio generato dalla grammatica di partenza se a essa si aggiungono tutte le produzioni $X \to Z$ relative alle coppie suddette, scartando ovviamente le produzioni $X \to X$, evidentemente inutili.

A questo punto resta di procedere ad eliminare dalla grammatica le produzioni $(X \to Z) \in \mathsf{X} \times \mathsf{X}$, una per una.
Denotato con $Z \to \alpha_1 \big| ... \big| \alpha_r$ il gruppo delle produzioni aventi come primo membro Z e secondo membro diverso da un ausiliario, si può eliminare la $x \to Z$ aggiungendo il reggruppamento di produzioni $X \to \alpha_1 \big| ... \big| \alpha_r$.

Infatti la conseguenza di questi rimpiazzamenti di produzioni sulle arborescenze di generazione consiste semplicemente nella sostituzione di ogni sottoarborescenza corrispondente a una catena di produzioni del tipo $X \to X_1 \to ... \to X_{t-1} \to \alpha$ con l'arborescenza.DE relativa alla nuova produzione $X \to \alpha$.

C14:**c.09** Diciamo che una grammatica $\boldsymbol{G} = \langle S, \mathsf{X}, \mathsf{T}, \mathcal{P} \rangle$ è una **grammatica in forma normale di base** sse le sue produzioni possono assumere solo le seguenti forme:

$$(1) \qquad S \to \sqcup \quad , \quad X \to \mathtt{a} \quad , \quad X \to \alpha \quad , \quad \text{con } X \in \mathsf{X} \ , \ \mathtt{a} \in \mathsf{T} \ , \ \alpha \in \mathsf{V}^{\geq 2} \ .$$

Le considerazioni precedenti si possono riassumere nell'enunciato che segue.

(2) Prop.: Una grammatica-F si può sempre trasformare in una equivalente-L nella forma normale di base ∎

C14:**c.10** Ora possiamo chiarire la cosiddetta **decidibilità del problema dell'appartenenza per le grammatiche acontestuali**.

Prop. Esiste un procedimento che, per una qualunque grammatica-F $\boldsymbol{G} = \langle S, \mathsf{X}, \mathsf{T}, \mathcal{P} \rangle$ e una qualsiasi stringa $w \in \mathsf{T}^*$ permette di decidere se $w \in \boldsymbol{G}^{\mathcal{G}\text{en}}$ o meno.

Dim.: Per quanto visto sulla presenza della $\sqcup$ [c05], possiamo limitarci a esaminare la generazione delle $w \in \mathsf{T}^+$.
Si può inoltre porre la grammatica nella forma normale di base. Con una tale grammatica, le arborescenze di generazione che possono produrre una stringa di data lunghezza costituiscono un insieme finito; quindi si può decidere se w appartenga o meno a $\boldsymbol{G}^{\mathcal{G}\text{en}}$ esaminando un insieme finito di arborescenze di generazione ∎

Osserviamo che le precedenti indicazioni operative sono estremamente semplici, ma trascurano del tutto di perseguire l'efficienza del procedimento.
Il suddetto problema di appartenenza si pone nella pratica quotidiana ogni volta che si rende necessario sottoporre a traduzione un programma: in questi casi l'efficienza è importante e per poter procedere in tale maniera è stato necessario sviluppare una vera e propria tecnologia con una sua elaborata base teorica, la **teoria del parsing**, rispetto alla quale i presenti discorsi costituiscono dei preliminari.

C14:**c.11** Le produzioni della forma $X \to \alpha$ con $\alpha \in \mathsf{V}^{\geq 2}$ si possono rimpiazzare con produzioni $X \to \beta$ con $\beta \in \mathsf{X}^{\geq 2}$. Per questo basta rimpiazzare ogni terminale $\mathtt{a}$ che compare nelle stringhe α con un nuovo ausiliario A che compare come primo membro solo nella produzione $A \to \mathtt{a}$. Conseguenza di questa modifica delle produzioni per le arborescenze di generazione è la sostituzione di arborescenze.DE del tipo $X \to \alpha$ con arborescenze di derivazione di altezza 2 del tipo mostrato in figura:

//input pC14c11

C14:**c.12** Si può anche procedere a uno "snellimento" delle produzioni che consiste nel rimpiazzare una produzione $X \to Y_1 ... Y_r$ con le $r-1$ produzioni $X \to Y_1\, X_2$, $X_2 \to Y_2\, X_3$, ..., $X_{r-1} \to Y_{r-1}\, Y_r$.

Esso ha come conseguenza per le arborescenze di generazione la sostituzione di ogni arborescenza.DE con $r > 2$ figli ausiliari con una arborescenza-$r-1$ di derivazione formata da arborescenze.DE con 2 figli e avente "forma a pettine":

//input pC14c12

Una grammatica si dice **GRAMMATICA-f in forma normale di Chomsky** sse le sue produzioni possono avere solo le seguenti forme:

$$S \to \sqcup \qquad X \to \mathtt{a} \qquad X \to YZ \qquad \text{con} \;\; X, Y, Z \in \mathsf{X} \,, \;\; \mathtt{a} \in \mathsf{T} \,.$$

Risulta quindi dimostrato l'enunciato che segue.

Teorema (teorema di Chomsky) Ogni grammatica si può trasformare in una equivalente in forma normale di Chomsky ∎

C14:**c.13** Osserviamo che la trasformazione di una grammatica in una equivalente nella forma suddetta dopo le eventuali semplificazioni, consistenti nella eliminazione di ausiliari non fecondi, si riduce a una sequenza di sostituzioni di una singola produzione con nuove produzioni aventi forme di pochi tipi e nel complesso più numerose.

La grammatica-F normalizzata quindi in genere è più pesante di quella di partenza; inoltre le nuove produzioni possono essere meno significative delle originali. In effetti sapere che una grammatica può porsi nella precedente forma normale non risulta vantaggiosa per elaborazioni particolari, ma serve a trarre conclusioni generali sulla portata complessiva delle grammatiche-F e sulle caratteristiche generali dei linguaggi acontestuali.

Questo va visto come uno dei molti casi di conflitto tra esigenze di una teoria di ampia portata ed esigenze delle applicazioni specifiche.

C14:**c.14** Veniamo ora a un enunciato molto utile per conoscere i confini dell'insieme dei linguaggi acontestuali.

(1) Lemma: (teorema di Bar Hillel-Perles-Shamir, 1961) Per ogni grammatica-F $\boldsymbol{G} = \langle S, \mathsf{T}, \mathsf{X}, \mathcal{P} \rangle$ si trova un intero positivo k tale che ogni stringa del linguaggio $\mathsf{L} = \boldsymbol{G}^{\mathcal{G}\text{en}}$ di lunghezza superiore o uguale a k si può porre nella forma $u\, w\, x\, y$, dove $v\, x \neq \sqcup$ e per ogni $n \in \mathbb{N}$ si ha $u\, v^n\, w\, x^n\, y \in \mathsf{L}$.

Dim.: Per la dimostrazione è comodo supporre che la grammatica sia nella forma normale di Chomsky. In tal caso si vede facilmente che una stringa di L che sia prodotto di un'arborescenza di altezza h deve avere lunghezza non superiore a 2^{h-1}: infatti una arborescenza-h che si serve di produzioni della forma di Chomsky può avere nodi con al più due figli nei livelli 0, 1, ..., $h-1$, cioè può avere al più 2^{h-1} nodi ai livelli $h-1$ ed h.

Se H è il numero di ausiliari della $\boldsymbol{G}$ ogni stringa del linguaggio di lunghezza maggiore o uguale a 2^H è il prodotto di un'arborescenza di generazione Ψ_1 di altezza superiore o uguale ad $H+1$. In una tale arborescenza si deve avere una passeggiata massimale sul quale un certo ausiliario R è ripetuto.

Chiamiamo ν_1 e ν_2 i due nodi etichettati da tale ausiliario, con ν_1 predecessore di ν_2. Nella arborescenza Ψ_1 si possono distinguere 5 porzioni.

La terza ψ_w è la sottoarborescenza dei discendenti di ν_2. La seconda ψ_v e la quarta ψ_x fanno parte della sottoarborescenza dei discendenti di ν_1 dalla quale siano stati eliminati i discendenti di ν_2: ψ_v è costituita dai discendenti verso sinistra dei nodi sulla passeggiata da ν_1 a ν_2, ψ_x è costituita dai discendenti sulla destra.
Similmente ψ_u e ψ_y si individuano nella sottoarborescenza ottenuta eliminando dalla Ψ_1 i discendenti di ν_1 considerando, risp., i discendenti sulla sinistra e sulla destra dei nodi sulla passeggiata dalla radice a ν_1.

//input pC14c14

C14:**c.15** Osserviamo che alcune delle porzioni di arborescenza precedenti potrebbero essere vuote: ν_1 potrebbe coincidere con la radice e in tal caso sarebbero vuote ψ_u e ψ_y; se ν_1 fosse padre di ν_2 una delle due porzioni ψ_v e ψ_x sarebbe vuota.
Se ν_1 fosse raggiungibile dalla radice con la passeggiata più a sinistra [più a destra], mancherebbe ψ_u [ψ_y];
Se ν_2 fosse raggiungibile da ν_1 con la passeggiata più a sinistra [più a destra], mancherebbe ψ_v [ψ_x].

Non può comunque accadere che sia vuota ψ_w e che siano vuote contemporaneamente ψ_v e ψ_x.
Chiamando, risp., u, v, w, x ed y le stringhe che si leggono sui terminali delle cinque porzioni ψ_u, ψ_v, ψ_w, ψ_x e ψ_y, abbiamo la fattorizzazione preannunciata $u\,v\,w\,x\,y$ della stringa generata dalla Ψ_1.

Si tratta ora di ricavare altre arborescenze di generazione per la grammatica modificando la Ψ_1. Una Ψ_0 si ottiene dalla Ψ_1 eliminando i discendenti di ν_1 e innestando in questo nodo la ψ_w: essa produce la stringa $u\,w\,y = u\,v^0\,w\,x^0\,y$.

Una Ψ_2 si ottiene invece eliminando dalla Ψ_1 i discendenti di ν_2 e innestando in tale nodo la sotto-arborescenza che ha ν_1 come radice: questa arborescenza si può considerare formata da ψ_u, da due repliche di ψ_v, da ψ_w, da due repliche di ψ_x e da ψ_y: essa produce $u\,v^2\,w\,x^2\,y$.
Con analoghi processi di ricomposizione (detti "a pompaggio") si può ottenere per ogni n positivo un'arborescenze Ψ_n contenente n repliche di ψ_v ed n repliche di ψ_x ed in grado di produrre $u\,v^n\,w\,x^n\,y$

∎

La seguente figura presenta la costruzione dell'arborescenza Ψ_3 corrispondente alla Ψ_1 della figura precedente:

U/input pC14c15

C14:**c.16** Un ausiliario R di una grammatica $\boldsymbol{G}$ si dice **ausiliario ricorsivo** sse esiste un'arborescenza di derivazione della $\boldsymbol{G}$ avente R come etichetta della radice e di una foglia.
Evidentemente gli ausiliari ricorsivi di una grammatica sono tutti e soli gli ausiliari che si trovano sui cicli del suo digrafo delle dipendenze.
Chiaramente un linguaggio acontestuale è infinito sse viene generato da una grammatica nella forma di Chomsky nella quale si trovano ausiliari ricorsivi.

Risulta quindi decidibile anche il cosiddetto **problema della infinitezza dei linguaggi generati da grammatiche-F**, acontestuali; esiste cioè un procedimento che consente di stabilire per una qualsiasi grammatica se essa genera un linguagggio finito o infinito.

C14:**c.17** Il lemma precedente consente di individuare numerosi linguaggi non acontestuali, linguaggi cioè che non possono essere generati da alcuna grammatica acontestuale.

Prop. $\displaystyle\sum_{n=1}^{\infty} \mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$ non è un linguaggio acontestuale.

Dim.: Se esistesse una grammatica in grado di generarlo, per un n sufficientemente elevato si potrebbe scrivere $a^n\,b^n\,c^n = u\,v\,w\,x\,y$.
Questa scrittura però è in conflitto con la possibilità di avere nel linguaggio anche tutte le stringhe $u\,v^k\,w\,x^k\,y$. Infatti v ed x non possono contenere lettere diverse, perché in tal caso una stringa $u\,v^k\,x^k\,y$ con $k \geq 2$ presenterebbe un alternarsi di lettere superiore a quello, molto ridotto, delle stringhe $\mathtt{a}^n\,\mathtt{b}^n\,\mathtt{c}^n$.
Se fosse invece $v = \mathtt{a}^j$, dovrebbe essere $x = \mathtt{b}^j$ o $x = \mathtt{c}^j$ per mantenere equilibrio tra le $\mathtt{a}$ e un'altra lettera, ma anche in tal modo non si avrebbe equilibrio con la lettera rimanente ∎

C14:**c.18** **(1) Prop.**: La collezione dei linguaggi acontestuali è chiusa per unione.

Dim.: Consideriamo due grammatiche-F $\boldsymbol{G}_1 = \langle {}_1, \mathsf{T}, \mathsf{X}_1, \mathcal{P}_2 \rangle$ e $\boldsymbol{G}_2 = \langle S_2, \mathsf{T}, \mathsf{X}_2, \mathcal{P}_2 \rangle$; si tratta si individuare una grammatica-F $\boldsymbol{G}_3$ in grado di generare tutte le stringhe che possono generare le due suddette grammatiche.
Come si è osservato, possiamo supporre che le due grammatiche date si servano di due insiemi di ausiliari disgiunti: $\mathsf{X}_1 \cap \mathsf{X}_2 = \emptyset$.
Come simbolo di partenza della $\boldsymbol{G}_3$ assumiamo un nuovo carattere S_3; come insieme degli ausiliari le assegnamo $\mathsf{X}_1 \dot\cup \mathsf{X}_2$ e come insieme delle produzioni $\{S_3 \to S_1 \,|\, S_2\} \dot\cup \mathcal{P}_1 \dot\cup \mathcal{P}_2$.
È allora evidente che l'insieme delle arborescenze di derivazione della $\boldsymbol{G}_3$ è l'unione delle arborescenze della $\boldsymbol{G}_1$, ciascuna preceduta dalla produzione $S_3 \to S_1$ e delle arborescenze della $\boldsymbol{G}_2$, ciascuna preceduta dalla produzione $S_3 \to S_2$.
Questo comporta che sia $\boldsymbol{G}_3^{\mathcal{G}\mathrm{en}} = \boldsymbol{G}_1^{\mathcal{G}\mathrm{en}} \cup \boldsymbol{G}_2^{\mathcal{G}\mathrm{en}}$ ∎

(2) Prop.: La collezione dei linguaggi acontestuali non è chiusa per intersezione e non è chiusa per complementazione.

Dim.: Il precedente linguaggio non acontestuale è intersezione dei due linguaggi $\sum_{n,k\in\mathbb{N}} \mathtt{a}^k\,\mathtt{b}^k\,\mathtt{c}^n$ e $\sum_{n,k\in\mathbb{N}} \mathtt{a}^k\,\mathtt{b}^n\,\mathtt{c}^n$ che si sono già dimostrati acontestuali.
Inoltre se la collezione dei linguaggi acontestuali fosse chiusa per complementazione, sarebbe chiusa anche per intersezione, dato che

$$\mathsf{L}_1 \cap \mathsf{L}_2 = \mathsf{T}^* \setminus ((\mathsf{T}^* \setminus L_1) \cup (\mathsf{T}^* \setminus \mathsf{L}_2))$$ ∎

C14:**c.19** **Eserc.** Dimostrare che il riflesso di un linguaggio acontestuale L è acontestuale.

C14:d. generazione di espressioni algebriche

C14:**d.01** Le espressioni sono oggetti formali che si incontrano in tutte le esposizioni di argomnti scientifici e tecnologici.
Mentre è piuttosto facile dare di esse un'idea intuitiva, è invece necessario essere molto meticolosi quando si deve dare una loro definizione precisa, in particolare quando si vogliono rendere comprensibili le espressioni a dispositivi elettronici come le calcolatrici tascabili e i computers.

Iniziamo con espressioni algebriche nelle quali intervengono soltanto le operazioni di somma e di prodotto (che denoteremo con "+" e "∗" come si fa nella massima parte dei linguaggi procedurali) e soltanto operandi rappresentati da lettere come `a, b, ... , h`.

C14:**d.02** La più semplice grammatica che permette di generare espressioni di questo tipo è

$$\boldsymbol{G}_{espl} := \left[S \to (S+S) \,\middle|\, (S*S) \,\middle|\, \mathtt{a} \,\middle|\, \mathtt{b} \,\middle|\, ... \,\middle|\, \mathtt{h} \right] .$$

Le sue più semplici arborescenze generative sono

//input pC14d02

Risulta quindi che sono espressioni di $\mathsf{L}_{espl} := \boldsymbol{G}_{espl}{}^{\mathcal{G}\mathrm{en}}$ le stringhe formate dai singoli operandi e le quintuple formate da una coppia di parentesi che racchiude una terna formata da un primo operando, da un operatore binario e da un secondo operando.

Come si è osservato per le espressioni razionali, le parentesi iniziale e finale sono superflue e si dovrebbero accettare anche espressioni come `a+b` e `c*d`. In effetti le parentesi che racchiudono ogni espressione non ridotta a un solo operando sono dovute al fatto che si fa riferimento ad una grammatica molto semplice. Le espressioni relative a queste grammatiche si dicono **espressioni completamente parentesizzate**.

Per generare espressioni più comprensibili e più usuali come $\mathtt{a}+\mathtt{b}, 2\mathtt{a} \leq \mathtt{a}*(\mathtt{b}+\mathtt{c})$, invece delle equivalenti completamente parentesizzate $(\mathtt{a}+\mathtt{b}), (\mathtt{a}+\mathtt{a})$ e $(\mathtt{a}*(\mathtt{b}+\mathtt{c}))$, si deve ricorrere a una grammatica più articolata.

C14:**d.03** Esaminiamo alcune arborescenze generative un po' più complesse:

//input pC14d03

Osserviamo che tutte le sottoarborescenze sono anche arborescenze generative; questo è dovuto al fatto che la grammatica $\boldsymbol{G}_{espl}$ possiede un solo ausiliario. In generale comunque tutte le sottoarborescenze di un'arborescenza di generazione aventi la radice etichettata dal simbolo di partenza sono esse stesse arborescenze di generazione. Nelle espressioni si individuano quindi **sottoespressioni**, cioè sottostringhe facenti parte del linguaggio L_{espl}.

Si nota anche che ogni espressione si può trasformare in una nuova trasformando una sua qualsiasi sottoespressione in un'altra. Per esempio accanto alle espressioni $E =$ `((a+b)*(c+d))` e alla `((f+g)*h)`, possiamo considerare come espressioni accettabili la più complessa `((a+((f+g)*h))*(c+d))` e la più semplice `((a+b)*c)`.
Si riscontra quindi una completa intercambiabilità delle sottoespressioni; in particolare si possono scambiare tra di loro tutti gli operandi semplici.

C14:**d.04** Su arborescenze generative come le precedenti si basa l'attribuzione di un significato operativo a ogni espressione. Per esempio la `((a+b)*(c+d))` individua un possibile processo di calcolo il quale richiede di:

- prendere in considerazione i quattro valori degli operandi `a, b, c` e `d`;
- sommare i primi due e assegnare il risultato ad un registro temporaneo che si può pensare associato al nodo etichettato da S "padre" della sottoespressione `(a+b)`;
- sommare i valori di `c` e `d` e attribuire la somma a un secondo registro associato al nodo ascendente della `(c+d)`;
- moltiplicare i valori contenuti nei due registri temporanei ed assegnare il prodotto a un registro associato alla radice dell'arborescenza; in questo registro è disponibile il valore dell'intera espressione.

C14:**d.05** Una interpretazione di questo tipo si può adattare a ogni arborescenza generativa della $\boldsymbol{G}_{espl}$ e può essere ripresa, con le opportune modifiche, per espressioni generate da molte varianti di questa grammatica.
Per esempio la grammatica:

$$\boldsymbol{G}_f = \big[\ S \to \texttt{(}S+S\texttt{)} \,\big|\, \texttt{(}S-S\texttt{)} \,\big|\, \texttt{(}S*S\texttt{)} \,\big|\, \texttt{(}S/S\texttt{)} \,\big|\, \texttt{a} \,\big|\, \texttt{b} \,\big|\, \texttt{c} \,\big|\, \texttt{d} \,\big|\, \texttt{e} \,\big|\, \texttt{f} \,\big|\, \texttt{g}\ \big]\ .$$

genera le espressioni completamente parentesizzate nei quattro operatori aritmetici e aventi come operandi `a, b, ..., g`. La seguente figura mostra un esempio di arborescenza generativa di $\boldsymbol{G}_f$ e di espressione facente parte di $\mathsf{L}_f := \boldsymbol{G}_f{}^{\mathcal{G}\text{en}}$:

//input pC14d05

C14:**d.06** Prevedibili estensioni delle grammatiche precedenti possono tener conto di altri operatori numerici binari.
Per esempio denotando l'operatore resto con "%" e l'operatore esponenziazione "↑" si può considerare la grammatica:

$$\boldsymbol{G}_g = \big[S \to \texttt{(}S+S\texttt{)} \,\big|\, \texttt{(}S-S\texttt{)} \,\big|\, \texttt{(}S*S\texttt{)} \,\big|\, \texttt{(}S/S\texttt{)} \,\big|\, \texttt{(}S\%S\texttt{)} \,\big|\, \texttt{(}S\uparrow S\texttt{)} \,\big|\, \texttt{a} \,\big|\, \texttt{b} \,\big|\, \texttt{c} \,\big|\, \texttt{d} \,\big|\, \texttt{e} \,\big|\, \texttt{f} \,\big|\, \texttt{g} \,\big|\, \texttt{h}\big].$$

Un'arborescenza generativa di $\boldsymbol{G}_g$ è:

//input pC14d06

C14:**d.07** Finora si sono considerati solo operatori binari infissi, cioè collocati tra i loro due operandi. Non è difficile trattare anche **operatori unari prefissi**, cioè scritti a sinistra dell'operando, come il meno unario, che qui conviene denotare con $-$, e **operatori unari suffissi** collocati a destra dell'operando, come il passaggio al valore reciproco "$^{-1}$" o il passaggio alla matrice trasposta come "$^{\mathsf{T}}$".
Una grammatica per espressioni che possono contenere i due precedenti operatori unari è la seguente:

$$\boldsymbol{G}_h = \big[\ S \to \texttt{(}S+S\texttt{)} \,\big|\, \texttt{(}S-S\texttt{)} \,\big|\, \texttt{(}S*S\texttt{)} \,\big|\, \texttt{(}S/S\texttt{)} \,\big|\, \texttt{(}-S\texttt{)} \,\big|\, \texttt{(}S^{-1}\texttt{)} \,\big|\, \texttt{a} \,\big|\, \texttt{b} \,\big|\, \ldots\ \big]\ .$$

Una semplice arborescenza generativa di tale grammatica è:

//input pC14d07

C14:**d.08** Un altro arricchimento delle espressioni riguarda la presenza di richiami di funzioni con un certo numero di argomenti. Nel caso di uno o due argomenti si tratta di sottoespressioni equivalenti, risp., agli operatori unari e binari. Per potere trattare le funzioni basta servirsi di produzioni come le seguenti:

$$S \to \texttt{sin[}S\texttt{]} \quad , \quad S \to \texttt{max[S,S]} \quad , \quad S \to \texttt{min[}S,S,S\texttt{]}$$

.

C14:**d.09** Altre espressioni utili riguardano entità eterogenee, entità di tipi diversi che possono essere elaborate e costruite da operatori e funzioni.

Nei linguaggi procedurali rivestono ruoli importanti espressioni che forniscono valori booleani, cioè valori appartenenti al dominio $\{\mathsf{true}, \mathsf{false}\}$, mediante composizioni di operandi e sottoespressioni numeriche con operatori relazionali ($<$, $\geq$, ...) e mediante composizioni di espressioni e operandi booleani con operatori booleani.

Una grammatica in grado di generare queste espressioni è la seguente:

$$\begin{aligned}\boldsymbol{G}_i = \Big[\ & B \to \texttt{(}B \wedge B\texttt{)} \,\big|\, \texttt{(}B \vee B\texttt{)} \,\big|\, \texttt{(}\neg B\texttt{)} \,\big|\, \texttt{(}A < A\texttt{)} \,\big|\, \texttt{(}A > A\texttt{)} \,\big|\, \texttt{(}A \leq A\texttt{)} \,\big|\, \texttt{(}A \geq A\texttt{)} \,\big|\\ & \texttt{(}A = A\texttt{)} \,\big|\, \texttt{(}A \neq A\texttt{)} \,\big|\, \texttt{l} \,\big|\, \texttt{m} \,\big|\, \texttt{n} \,\big|\, \ldots\ ,\\ & A \to \texttt{(}A + A\texttt{)} \,\big|\, \texttt{(}A - A\texttt{)} \,\big|\, \texttt{(}A * A\texttt{)} \,\big|\, \texttt{(}A/A\texttt{)} \,\big|\, \texttt{a} \,\big|\, \texttt{b} \,\big|\, \texttt{c} \,\big|\, \ldots\ \Big]\ .\end{aligned}$$

Una semplice arborescenza generativa di tale grammatica è:

//input pC14d09

Si osserva che la struttura delle arborescenze di generazione di queste grammatiche differisce da quella delle arborescenze viste in precedenza solo in quanto si hanno operandi espliciti di due tipi, l'aritmetico (`a,b,c,...`) e il booleano (`l,m,n,...`) e si hanno due ausiliari diversi (A, B) corrispondenti ai due possibili tipi (l'aritmetico e il booleano).

C14:**d.10** Vediamo ora una semplice grammatica in grado di generare espressioni numeriche le quali evitano parentesi ridondanti in quanto non necessarie per l'interpretazione corretta del loro significato operativo.

$$\boldsymbol{G}_x \ := \ \Big[\ E \to E + T \,\big|\, T\ ,\ T \to T * F \,\big|\, F\ ,\ F \to \texttt{(}E\texttt{)} \,\big|\, I\ ,\ I \to \texttt{a} \,\big|\, \texttt{b} \,\big|\, \texttt{c} \,\big|\, \texttt{d}\ \Big].$$

I molti simboli ausiliari in una grammatica come questa spesso vengono vantaggiosamente presentati servendosi di termini significativi racchiusi tra parentesi angolate invece di lettere con funzioni di ausiiari.

Per la precedente $\boldsymbol{G}_x$ si potrebbe scrivere $\langle espressione \rangle$ in luogo di E, $\langle termine \rangle$ in luogo di T, $\langle fattore \rangle$ invece di F ed $\langle identificatore \rangle$ invece di I.

La grammmatica $\boldsymbol{G}_x$, come la $\boldsymbol{G}_{espl}$, è in grado di generare solo espressioni riguardanti operatori numerici da comporre con operazioni di somma e prodotto; essa però è sensibilmente più complessa della $\boldsymbol{G}_{espl}$: in effetti solo con un maggior numero di simboli ausiliari e di produzioni si possono evitare le parentesi ridondanti, cioè si possono avere le cosiddette **espressioni concisamente parentesizzate**

Vediamo alcune arborescenze generative di questa grammatica:

//input pC14d10

Si tratta di arborescenze più complesse di quelle della grammatica $\boldsymbol{G}_{espl}$ che portano a espressioni completamente parentesizzate equivalenti. Esse però si possano semplificare nelle stesse arborescenze

sintattiche ottenibili dalle grammatiche per le espressioni completamente parentesizzate e quindi interpretabili in modo molto simile.

C14:**d.11** Concludiamo presentando una grammatica in grado di generare espressioni concisamente parentesizzate nelle quali compaiono anche le operazioni di sottrazione, divisione e cambiamento di segni:

$$\boldsymbol{G}_y \ := \ \left[\ E \to -T \big| E + T \big| E - T \big| T \ , \ T \to T * F \big| T / F \big| F \ , \ F \to (E) \big| I \ , \ I \to \mathtt{a} \big| \mathtt{b} \big| \mathtt{c} \big| \mathtt{d} \right].$$

C14:e. arborescenze sintattiche delle espressioni

C14:**e.01** L'arborescenza generativa di un'espressione relativa a una delle grammatiche viste in precedenza è collegata direttamente ad una nozione generale e di grande importanza come quella di grammatica.
Questa arborescenza però presenta informazioni ridondanti e per taluni scopi è opportuno trasformarla in una struttura ad albero più essenziale, la cosiddetta **arborescenza sintattica dell'espressione**.

Vediamo dunque alcune semplificazioni che si possono apportare a una arborescenza generativa senza farle perdere informazioni effettive.
Innanzi tutto si possono eliminare tutti i nodi etichettati da parentesi e gli archi che portano a essi. Nella struttura ad albero così ottenuta i nodi padre sono etichettati da simboli ausiliari come S estranei all'espressione: queste etichette si possono eliminare rimpiazzandole con i simboli degli operatori "fatti risalire" dal nodo foglia e facendo sparire anche questi nodi e gli archi relativi.

C14:**e.02** Con i suddetti cambiamenti le arborescenze generative presentate nelle figure precedenti si trasformano, risp., nelle seguenti ben più semplici arborescenze distese con nodi etichettati solo da operatori e operandi:

//input pC14e02

//input pC14e02B

//input pC14e02C

Questi grafi forniscono nel modo più evidente il significato operativo delle espressioni associate. Essi mostrano che il processo di calcolo individuato da una espressione prevede successive composizioni di valori forniti dagli operandi espliciti; queste danno nuovi valori che si possono pensare depositati temporaneamente in registri associati ai nodi ascendenti che, nell'arborescenza sintattica, sono etichettati dall'operatore che riguarda la composizione.
Le successive composizioni portano alla fine a un solo valore che risulta associato alla radice dell'arborescenza e che costituisce il valore attuale dell'espressione.

C14:**e.03** La trasformazione di un'arborescenza generativa nel corrispondente arborescenza sintattica e la costruzione che costituisce la valutazione dell'espressione non sono legate ai particolari significati degli operandi e degli operatori.
Quello che conta è la presentazione di un complesso di composizioni di oggetti sui quali si conoscono i modi di operare (gli operandi sui quali si sanno eseguire le operazioni che li coinvolgono) e che tra queste composizioni vi sono priorità di esecuzione derivanti esclusivamente da un ordinamento tra le etichette dei nodi dell'arborescenza sintattica. Questo ordinamento viene retto dalle cosiddette "relazioni di discendenza".

La nozione di arborescenza sintattica e il significato che una tale configurazione attribuisce alla espressione corrispondente sono validi per tutte le grammatiche che, come quelle viste in precedenza, si basano su arborescenze-1 esprimenti operazioni. Come abbiamo visto, le più tipiche e comuni di queste operazioni sono le binarie e a esse corrispondono arborescenze-1 con una radice, contraddistinta dall'operatore, dalla quale escono due archi che entrano nei due nodi etichettati dagli operandi.

Nel caso degli operatori unari si hanno arborescenze-1 formate da un solo arco che esce da un nodo etichettato dall'operatore unario ed entra in un nodo etichettato dall'operando sul quale l'operatore deve agire.

C14:**e.04** Si potrebbero avere anche operatori ternari, quaternari o di arietà maggiore: in questi casi si avrebbero, risp., arborescenze-1 con tre, quattro o più figli: si pensi per esempio alle operazioni di ricerca del massimo e del minimo tra più numeri: in questo caso il numero degli operandi può addirittura variare con le diverse occorrenze. Due esempi di arborescenze sintattiche di espressioni riguardanti tali operatori sono:

//input pC14e04

C14:**e.05** Come vedremo in seguito, si possono avere anche composizioni nelle quali gli operandi e il risultato sono di tipi diversi: per questo si devono usare arborescenze-1 i cui nodi sono destinati a valori di tipi diversi, tipi che possono dedursi dagli operatori che forniscono le etichette dei nodi e dagli operandi che caratterizzano i figli di tali nodi.

L'unica restrizione per i processi di calcolo forniti da queste strutture ad albero riguarda il fatto che si deve trattare di composizioni univoche, cioè costruzioni che portano a un unico valore risultato sia nelle operazioni parziali che nel processo complessivo.

Nella pratica dei calcoli possono essere molto utili procedimenti che portano a più valori. Questi procedimenti si collocano al di là della portata delle espressioni che abbiamo analizzate e che abbiamo prospettate: per essi si devono utilizzare algoritmi che abbiano una portata più generale.

Conviene segnalare fin da ora che le arborescenze sintattiche verranno riprese introducendo formalmente i depositi a pila in :g ed esaminando le macchine in grado di gestire i complessi di operazioni che le arborescenze esprimono.

C14:f. grammatiche per semplici frasi italiane

C14:**f.01** Consideriamo la seguente grammatica:

$$\boldsymbol{G}_m \ := \ \left[\ F \to N \,|\, V \,|\, O \ \right] ,$$
$$N \to \texttt{Anna} \,|\, \texttt{Carlo} \,|\, \texttt{Giuseppe} \,|\, \texttt{Marco} \,|\, \texttt{Maria} \,|\, \texttt{Tina}$$
$$V \to \texttt{mangia} \,|\, \texttt{compera} \,|\, \texttt{desidera} \,|\, \texttt{richiede} \,|\, \texttt{rifiuta}$$
$$O \to \texttt{una mela} \,|\, \texttt{una pesca} \,|\, \texttt{un gelato} \,|\, \texttt{uno strudel} \ \left] \right. \quad .$$

Un'arborescenza generativa di tale grammatica è:

//input pC14f01

Tutte le arborescenze generative hanno lo stesso aspetto in quanto differiscono solo per le stringhe di terminali che discendono dai tre nodi figli della radice. $\boldsymbol{G}_m$ è quindi in grado di generare $6 \cdot 5 \cdot 4 = 120$ frasi costituite da un soggetto seguito da un verbo transitivo e da un complemento oggetto.

In effetti le grammatiche acontestuali sono state introdotte dal linguista Noam Chomsky intorno al 1955 come uno dei modelli che consentivano di inquadrare semplici frammenti di una lingua naturale.

L'esempio precedente, molto schematico, mostra come, con una grammatica piuttosto ridotta, si possano controllare e strutturare insiemi di frasi piuttosto estesi. Infatti è facile ampliare i gruppi di produzioni precedenti perché comprendano un elevato numero di soggetti, di verbi e di complementi oggetto ottenendo la possibilità di inquadrare un numero maggiore di frasi.

Proseguendo in questa direzione, tuttavia, si rischia di consentire frasi prive di senso comune come `Luisa mastica un chinotto` o anche più surreali.

Evidentemente per avere schemi linguistici utilizzabili si devono proporre grammatiche più elaborate delle precedenti.
Queste estensioni rivestono un certo interesse in quanto consentono di controllare insiemi di frasi più articolate, ma incontrano ancora dei limiti. Per riuscire a controllare la ricchezza di espressioni che caratterizzano ogni linguaggio naturale si rendono necessari modelli ben più complessi delle grammatiche acontestuali.
In particolare si rende necessario controllare caratteristiche delle componenti delle frasi che dipendono dal contesto nel quale esse si vengono a trovare, mentre le grammatiche acontestuali consentono di scambiare senza restrizioni le stringhe che discendono da un nodo ausiliario.

C14:**f.02** Entro limiti piuttosto stretti comunque le grammatiche da noi studiate consentono di avere delle descrizioni precise e semplici di frasi di linguaggi naturali. Nelle grammatiche utilizzate per questi scopi può essere opportuno sostituire i simboli ausiliari con le più significative scritture della forma esemplificata in `d10`; ora le scritture tra parentesi angolate devono riallacciarsi alle diciture utilizzate dai linguisti.
In tal modo i simboli ausiliari vengono sostituiti dalle cosiddette **categorie sintattiche** presentate con scritture chiaramente leggibili delimitate da parentesi angolate.

La grammatica precedente potrebbe riscriversi nella seguente forma:

⟨frase ⟩ ::= ⟨soggetto ⟩ ⟨verbo ⟩⟨complemento oggetto ⟩

⟨soggetto ⟩ ::= `Anna` |`Carlo` |`Giuseppe` |`Marco` |`Maria` |`Tina`

⟨verbo ⟩ ::= `mangia` |`compera` |`desidera` |`richiede` |`rifiuta`

⟨complemento oggetto ⟩ ::= `una mela` |`una pesca` |`un gelato` |`uno strudel`

Queste scritture hanno un significato piuttosto evidente. La prima formula dice che le frasi che consideriamo sono costituite da una prima parte con il ruolo di soggetto, da una seconda parte appartenente alla categoria sintattica ⟨verbo ⟩ e da un complemento oggetto. Le successive formule allineano le possibili istanze delle categorie a primo membro, ovvero gli elementi degli insiemi denotati con le scritture a primo membro.

C14:g. depositi a pila e riconoscitori a pila

C14:**g.01** Procediamo ora a definire formalmente i **riconoscitori a pila**, le macchine sequenziali che hanno lo scopo di accettare tutti e soli i linguaggi acontestuali e che, come si è visto discorsivamente in b04, costituiscono un notevole arricchimento dei riconoscitori-RS.

Come questi i riconoscitori a pila sono caratterizzati da un insieme finito di stati interni e sono in grado di leggere stringhe scorrendole in una direzione e subendo di conseguenza una evoluzione caratterizzata da transizioni da stato a stato.
In più essi dispongono di un dispositivo di memoria, la pila, con prestazioni essenzialmente semplici ma con capacità di ricordo illimitate, almeno ion linea di principio.

L'esame delle prestazioni degli accettatori a pila e dei loro arricchimenti costituiti dai trasduttori a pila, è decisamente più impegnativo di quello dei riconoscitori-RS e dei riconoscitori-trasduttori-R.
Inizialmente tratteremo in modo intuitivo i riconoscitori e i trasduttori a pila, illustrando esempi significativi di tali macchine e cercando di dare un'idea intuitiva delle loro proprietà principali.

C14:**g.02** Riprendiamo da a07 la grammatica $\boldsymbol{G}_b := \left\lceil s \to (S) \middle| () \right\rfloor$ e il linguaggio

$$\mathsf{L}_b \;=\; {\boldsymbol{G}_b}^{\mathcal{G}\mathrm{en}} \;=\; \{n \in \mathbb{N} :|\; \mathtt{a}^{\,n}\mathtt{bc}^{3\,n}\} \quad .$$

Esso viene accettato dal riconoscitore descritto dal seguente pluridigrafo arricchito:

//input pC14g02

In esso voi sono archi etichettati, oltre che dal carattere che induce la transizione, dal comando riguardante una azione da compiere sul deposito a pila.
Il segno ↓ richiede l'inserimento (push) del carattere che segue; il segno ↑ richiede l'estrazione (pop) del carattere che segue che deve trovarsi in cima al deposito.

Si osservi come i due circuiti di lunghezza 2 e 3 di questo riconoscitore consentano di considerare accorpate le coppie aa e le terne ccc che si devono verificare coniugate.

Consideriamo il linguaggio delle stringhe palindrome $\{w \in \{\mathtt{a},\mathtt{b}\}^* :|\; w\,\mathtt{c}\,w^{\leftarrow}\}$.
Esso viene accettato dal riconoscitore a pila schematizzato da:

//input pC14g02B

Si osservi che in questa macchina, contrariamente alle precedenti, il deposito a pila può contenere simboli diversi ed esso viene utilizzato per ricordare non solo il numero ma anche la individualità dei simboli incontrati nella prima parte della stringa esaminata. Per i riconoscitori precedenti il deposito a pila fungeva semplicemente da contatore e la cosa ha consentito di semplificare la loro implementazione.

C14:h. esecutore di espressioni di Lukasiewicz postfisse

C14:**h.01** Come si sono arricchiti i riconoscitori di Rabin e Scott ottenendo gli accettatori trasduttori a stati finiti, così si possono arricchire con dispositivi di emissione i riconoscitori a pila giungendo agli **accettatori-trasduttori a pila**.

Noi non tratteremo in dettaglio queste macchine, ma ci limiteremo a considerarne esempi interessanti in grado di operare su particolari rappresentazioni di espressioni matematiche per riconoscerne la correttezza e per automatizzare la loro valutazione.
I meccanismi che esporremo, in effetti, costituiscono forme semplificate di meccanismi che svolgono un ruolo centrale nei compilatori di gran parte dei linguaggi di programmazione.

C14:**h.02** Torniamo a considerare le arborescenze sintattiche relative alla generazione di espressioni matematiche.
Da un'arborescenza sintattica si possono ricavare con un procedimento generale, cioè non condizioato dal significato particolare di operatori ed operandi, scritture chiamate **notazioni di Lukasievicz** o notazioni polacche che equivalgono alle espressioni dotate di parentesi e riguardanti operatori binari infissi.
Come vedremo si tratta di scritture a prima vista poco chiare, di lettura assai meno agevole delle espressioni usuali. Le notazioni polacche si rivelano però più essenziali in quanto non contengono segni che non influiscono direttamente sul risultato come le parentesi o le virgole che separano gli argomrnti di funzioni e costituiscono la base di algoritmi molto semplici finalizzati al calcolo automatico delle espressioni.

C14:**h.03** Consideriamo dunque un'arborescenza sintattica e ricordiamo i due metodo che consentono di visitare in modo sistematico tutti i suoi nodi, l'**procedimento Top Down Left Right** e il **procedimento Top Down Right Left**.

Il primo metodo, applicato alle arborescenze sintattiche viste in precedenza fornisce le seguenti stringhe:

`++abc`	`**ab+cd`	`/-+ab−c`$^{-1}$`d`
`/−−abc+*de+fg`	`+/+abc↑*−de%fgh`	`∨∧<+abc≠d*ef¬l.`

Queste sono dette **notazioni polacche prefisse** per le relative espressioni.

Il secondo metodo dà invece:

`+c+ba`	`**dc+ba`	`/−`$^{-1}$`dc-+ba`
`/++gf*ed−c−ba`	`+↑h*%gf−ed/c+ba`	`∨¬l∧ ≠*fed<c+ba.`

Più di queste stringhe interessano direttamente le loro riflesse:

`ab+c+`	`ab+cd+*`	`ab+-cd`$^{-1}$ `−/`
`ab−c−de*fg++/`	`ab+c/de−fg%*h↑+`	`ab+c<def*≠ ∧l¬∨.`

Queste si dicono **notazioni polacche suffisse** (o postfisse) per le relative espressioni.

C14:**h.04** Queste notazioni sono equivalenti alle arborescenze sintattiche: infatti, oltre ad essere in grado di ricavarle algoritmicamente dalle arborescenze sintattiche, conosciamo un procedimento generale che consente di compiere la trasformazione inversa. Vediamo come si ricava da una notazione prefissa l'arborescenza sintattica corrispondente.

- Si scorre la notazione prefissa da sinistra a destra e in corrispondenza di ogni operando `x` incontrato si traccia un nodo etichettato da `x` e munito di un arco entrante;

- questo arco lo si fa uscire dall'ultimo nodo operatore tracciato che non sia saturo, cioè dal quale non escono tutti gli archi previsti dalla sua arietà; per individuare il nodo padre attuale si può utilizzare un deposito a pila verticale in cima della quale si inseriscono i nodi operatore e si cancellano quelli che si sono rivelati saturi.

Le notazioni polacche sono dunque equivalenti alle notazioni parentesizzate e infisse, sono meno leggibili ma hanno il vantaggio di essere più essenziali.

C14:**h.05** Il rilevante pregio di queste notazioni stà nella possibilità di far agire su di essi meccanismi esecutori, che, servendosi solo di un deposito a pila, effettuano la valutazione dell'espressione. Vediamo la cosa per le notazioni postfisse.
Innanzi tutto osserviamo che in ciascuna posizione della pila dell'esecutore si può trovare o un valore che interviene nel calcolo (tipicamente un valore numerico) o l'indicazione di un operatore.
L'esecutore si serve di una testina di lettura che scorre la notazione suffissa da sinistra a destra e opera come segue:

- se legge un operando pone il suo valore in cima alla pila;
- se legge un operatore estrae dalla cima della pila gli operandi che esso richiede (uno, due o più); esegue quindi l'operazione richiesta su questi operandi e pone il risultato in cima alla pila al posto degli operandi estratti.

Alla fine della lettura della notazione postfissa nella pila si deve trovare soltanto un valore, quello che costituisce il risultato della valutazione.

C14:**h.06** Consideriamo due esempi riguardanti espressioni aritmetiche nelle quali compaiono come operandi costanti numeriche esplicite.
Si abbia l'espressione `((2+5)*3)` e l'equivalente notazione polacca postfissa `25+3*`. I passi nel quale l'esecutore sviluppa i calcoli sono:

//input pC14h06

Considerando poi le espressioni equivalenti `(-(6+1))/(4-(2`$^{-1}$ `))` e `61+- 42`$^{-1}$`-/`
si ha la seguente evoluzione:

//input pC14h06B

C14:**h.07** Si osserva che l'algoritmo presentato consente anche di verificare la correttezza di una notazione sufissa: il procedimento giunge a una conclusione corretta e utile se e solo se la notazione è corretta, cioè sse ogni operatore può disporre di un numero adeguato di operandi e ogni operando viene elaborato da un operatore.
Due casi di evoluzioni che si concludono con la segnalazione di errore sono i seguenti:

//input pC14h07

//input pC14h07B

`Testi dell'`*esposizione* `in http://www.mi.imati.cnr.it/alberto/ e in http://arm.mi.imati.cnr.it/Matexp/`