di Andrea de Prisco

Grammatiche ed Automi

Dopo aver parlato un po' di calcolabilità, di macchine di Turing e di cardinalità transfinite, questo mese sarà la volta degli Automi a Stati Finiti Deterministici e non, altro capitolo abbastanza importante della computabilità o, meglio, della teoria dei linguaggi formali. «Linguaggi». Finalmente una parola di uso comune nella piccola informatica in mezzo ad altre parole meno diffuse. Anticipiamo subito che quanto state per leggere si usa regolarmente per inventare nuovi linguaggi di programmazione, progettare compilatori e interpreti. Siamo dunque molto meno sul teorico dei numeri scorsi.

Farina e sacchi

Anche se non strettamente necessario (quello che è successo il mese scorso in questa rubrica dovrebbe bastare) occorre ricordare ai lettori che l'intenzione di queste pagine non è certo quella di insegnare la teoria informatica in tutti i suoi aspetti e, perché no, le sue asperità, ma semplicemente mostrare qualche scorcio di «dietro le quinte» a tutti i lettori interessati a queste tematiche. Come più volte ripetuto, chi è maggiormente attratto da tali argomentazioni potrà documentarsi presso biblioteche e librerie scientifiche o, se in età scolare-quasi-universitaria, potrebbe pensare di iscriversi ad informatica presso una di quelle sedi universitarie che la mettono a disposizione.

Ricapitolando, si tratta solo di una piccola infarinatura al problema e la farina adoperata non è certamente appartenente al sacco del sottoscritto...

Andiamo a incominciare

Hardware in inglese sta per ferramenta, ferraglia (e perché no, per qualcosa di «duro» nel senso somaresco del termine). Eppure vedere all'opera un buon computer magari spettacolare come il Mac o l'Amiga fa ben altro effetto che uno sguardo alla cassetta dei ferri...

Se però prendiamo un computer, lo apriamo, togliamo da dentro tutto ciò che è software e proviamo a riaccenderlo l'effetto non sarebbe molto diverso: qualche chilo di parti metalliche, buttate lì, a far nulla.

Perché mi dice ciò? Semplice: un buon software è sempre meglio di un buon hardware (tanto per accendere ulteriori contrasti tra Amighi e Macchinisti). E per scrivere un buon software ci vuole un buon linguaggio di programmazione (altro che linguaggio macchina) ... e per inventare un buon linguaggio di programmazione c'è bisogno dei potenti mezzi messi a disposizione dalla ricerca informatica, coi quali ci si spinge sempre più verso l'automazione totale della realizzazione di compilatori, oggigiorno attuabile solo in parte.

Anche se non ne abbiamo mai parlato in queste pagine, i lettori più preparati (smanettoni compresi, che alla fin fine sanno più di tutti) sanno che un compilatore serve per *tradurre* un programma scritto in un linguaggio di programmazione ad alto livello (pascal, ada, pl/1, apl, modula 2 ecc.) in uno più basso come il linguaggio macchina o una forma intermedia interpretata da un apposito interprete.

Oltre a ciò, la compilazione di un programma avviene *sempre* in più fasi distinte. Di solito si individua una prima fase, ad opera dello scanner, che preso il testo sorgente individua le varie componenti (variabili, comandi, funzioni, ecc.) e le evidenzia. La seconda fase, di parsing, una volta ricevuto il programma sorgente con le componenti evidenziate (il programma non è più una manciata di caratteri ma una manciata di comandi, funzioni, variabili, espressioni, operatori) esegue l'analisi sintattica per trovare errori di tale tipo. Se questa fase termina con successo si procede (sempre in fasi distinte) alla generazione del codice

```
<NomeVariabile>::=<Lettera>|<Lettera><RestoNome>
<RestoNome>::=<Nil>|<Lettera><RestoNome>|<Cifra><RestoNome>
<Lettera>::= A|B|C|....|W|X|Y|Z
<Cifra>::=0|1|2|3|4|5|6|7|8|9
```

Figura 1 - Definizione del nome di una variabile.

eseguibile magari anche ottimizzato.

Le grammatiche e gli automi, argomento di questo mese, si usano proprio per questo: dare una rigorosa definizione sintattica del linguaggio di programmazione in modo da poter generare automaticamente scanner e parser (due automi) di cui sopra. In questo modo una parte del compilatore viene fatta automaticamente. Nel senso che, prese le specifiche sintattiche di un linguaggio, le passiamo ad un apposito *programmone*, il quale ci fornirà nel vero senso della parola il primo pezzo di compilatore. Il resto lo faremo a mano (o quasi).

Le Grammatiche

Cominciamo con un esempio. Supponiamo di avere un linguaggio di programmazione qualsiasi, in cui i nomi delle variabili possono essere formati da lettere e cifre purché il primo carattere non sia un numero. Ciò per distinguerle facilmente dai numeri veri e propri. Ad esempio nomi possibili sono A, AA, F104, Y10FIRE, mentre sono vietate variabili il cui nome sia 1022, 1234ABCD, 44GATTI ecc.

Per definire rigorosamente ciò si può usare la nota forma BNF, che come vedremo non è altro che una manciata di produzioni grammaticali. Questa particolarità delle variabili potremmo indicarla con le quattro produzioni di figura 1.

Si legge in questo modo: un nome di variabile è una lettera o una lettera seguita dalla parte terminale del nome. E con la prima produzione abbiamo finito. La parte terminale di un nome (produzione 2, <RestoNome>) può essere Nil, ovvero la stringa vuota (e usata per terminare la ricorsione che ora incontreremo) oppure una lettera seguita da una parte terminale di nome o una cifra seguita da una parte terminale di un nome. Per completezza le due produzioni che seguono indicano «cosa si intende» per lettera e per cifra.

Proviamo a generare un nome di variabile, ad esempio AZ15.

Un nome di variabile è una lettera o una lettera seguita da un <RestoNome>: ovviamente sceglieremo la seconda possibilità. Cominciamo dunque a sostituire la prima lettera con la A (rifacendoci alla terza produzione). A questo punto abbiamo A<RestoNome>. Deriviamo <RestoNome> sostituendo a questo, grazie alla seconda produzione <Lettera><RestoNome> ottenendo A<Lettera><RestoNome>. Come prima sostituiamo a <Lettera> una lettera, questa volta scegliendo la Z. Siamo arrivati a AZ<RestoNome>. Procedendo con la seconda produzione scegliamo <Cifra><RestoNome> per due volte di seguito e sostituendo alla prima <Cifra> un 1 e alla seconda un 5. A questo punto abbiamo AZ15<RestoNome> in cui sostituiremo a <RestoNome> la stringa vuota ottenendo AZ15.

Buona lavata di denti.

Dell'altro

Più informaticamente parlando, una grammatica è una quadrupla: un insieme di quattro oggetti. Abbiamo dei simboli, e delle produzioni. I simboli si suddividono a loro volta in terminali e non terminali e un particolare simbolo è detto iniziale perché da questo iniziamo a generare «frasi». Riassumendo, la quadrupla è composta da un insieme di simboli detto Alfabeto, da un sottoinsieme di questo detto «simboli terminali», da un elemento dell'Alfabeto detto «simbolo iniziale» e da un insieme di produzioni. Nell'esempio visto prima <NomeVariabile> è il simbolo iniziale; le lettere dell'alfabeto, <Nil> e le cifre sono simboli terminali, <Lettera>, <Cifra>, <RestoNome> sono i simboli non terminali. Qualora non fosse chiaro dal contesto, aggiungiamo che un simbolo si dice non terminale se da questo possiamo generare altri simboli mentre è terminale se non genera altro. E come si vede sempre da figura 1 lettere, cifre e <Nil> non producono altri simboli. Da notare che con la grammatica vista è possibile generare qualsiasi nome di variabile, anche lungo miliardi di lettere rispondente ai requisiti dati.

Dicevamo che nella definizione di una grammatica occorre fornire le produzioni che, come visto, servono per produrre «frasi». Immaginiamo ad esempio che la nostra grammatica sia formata dai seguenti simboli non terminali:

A,B

simboli terminali:

c,d

simbolo iniziale

S

Si noti come, per maggiore chiarezza, abbiamo indicato i simboli terminali in minuscolo e gli altri in maiuscolo.

L'insieme delle possibili produzioni potrebbe essere:

S → cAd | dBc | A | B
A → cd | cAd
B → dc | dBc

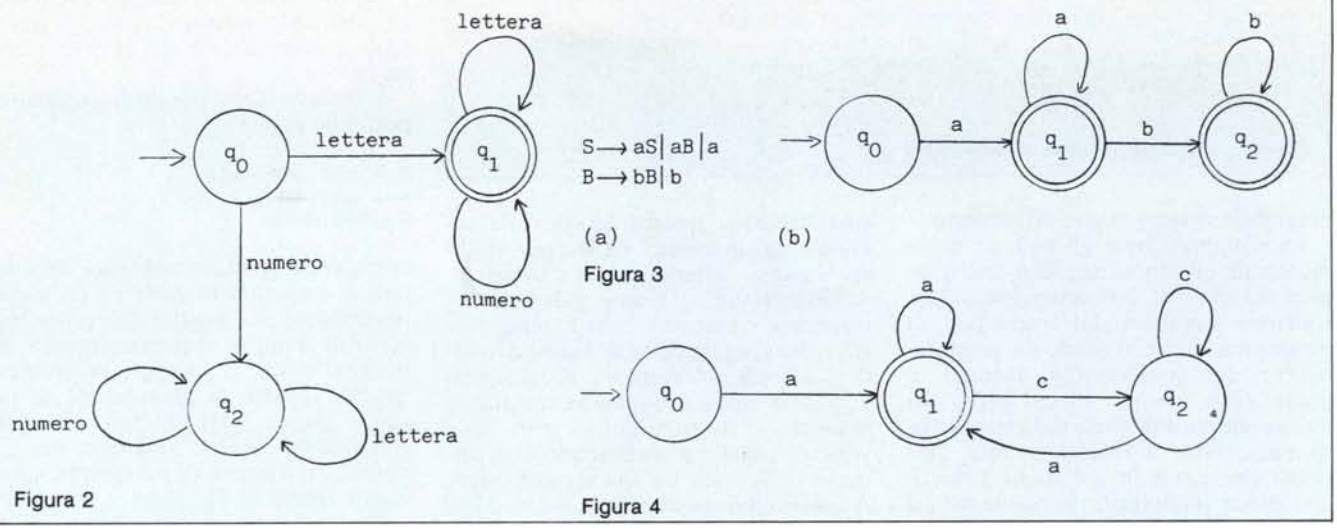
con questa scrittura abbiamo definito tutte le sequenze formate da un numero qualsiasi di *c* seguito da un pari numero di *d* più le sequenze formate da *d* seguite da *c*, sempre in numero uguale. Ovvero la grammatica di cui sopra genera frasi del tipo *cd, ccdd, ccccdddddd, dddccc, ddddcccc,* ecc. (si noti che il numero di *c* è sempre uguale al numero di *d*).

Per verificare quanto appena detto prendete carta e penna e provate a «produrre». Si parte dal simbolo iniziale *S* e da questo produciamo, ad esempio, la stringa *cAd*. *c* e *d* sono terminali quindi ce li porteremo dietro fino alla fine. Dalla seconda produzione scegliamo di sostituire la *A* con la stringa *cAd* ottenendo *ccAdd*. Potremmo continuare così all'infinito, ma decidiamo di terminare scegliendo a questo punto per la *A* la produzione *cd*. Otteniamo *cccddd*: essendo questa formata solo da terminali abbiamo finito ovvero abbiamo generato una frase.

Per ogni grammatica generativa esiste un automa in grado di riconoscere le frasi generate da questa. Ovvero un automa è un oggetto, espressamente costruito per una grammatica, il quale, preso una qualsiasi frase formata da simboli terminali restituisce «sì» o «no» a seconda se tale frase è stata generata o meno dalla grammatica da cui siamo partiti. Possiamo ad esempio immaginarcelo come una specie di Macchina di Turing, formato cioè da un nastro sul quale è stato incisa la stringa da analizzare, da una testina e da una parte controllo in grado di prendere decisioni in funzione del suo stato interno e del simbolo in lettura. Analizzata la stringa provvederà infine a scrivere il «sì» o il «no» sul nastro prima di fermarsi.

Nel nostro esempio, l'automa corrispondente alla grammatica, a frasi del tipo *cccddd, ddcc, cd* risponde «sì», a frasi non generabili come *dcdcd, ccc, d* risponderebbe «no».

Per mostrare dei semplici automi useremo le grammatiche regolari le quali, pur essendo grammatiche a tutti gli effetti, hanno alcune limitazioni circa le possibili produzioni. Analogamente le frasi generate saranno in un certo senso limitate non quantitativamente (una grammatica non banale genera sempre infinite frasi) ma come



complessità. Tanto per anticipare subito qualcosa, la grammatica appena mostrata non è regolare per il motivo che ora mostreremo e la complessità intrinseca delle frasi generate da questa risiede nel fatto che contengono un numero uguale di *c* e di *d*. La prima grammatica vista, quella dei nomi di variabili era regolare (!).

Grammatiche regolari

Dicesi grammatica regolare una qualunque grammatica le cui produzioni sono del tipo:

<NonTerminale> → <Terminale>
<NonTerminale> → <Terminale> <NonTerminale>

ovvero le rispettive parti destre delle produzioni (ciò che sta a destra della freccia) sono o un terminale o un terminale seguito da un (unico) non terminale. La grammatica vista prima non è regolare in quanto ha produzioni del tipo:

A → cd

(due terminali di fila) oppure:

A → cAd

(due terminali con in mezzo un bel non terminale).

Una grammatica regolare potrebbe essere la seguente:

S → a | aB
B → aB | cB | a

Le produzioni sono tutte del tipo mostrato e le frasi generate da queste sono sequenza qualunque di *a* e di *c* inizianti e terminanti per *a*. Es.: *acacaa, aaaa, accccca, accaacca,* ecc.

Automi a stati finiti deterministici

Per ogni grammatica regolare esiste un automa a stati finiti deterministico in grado di riconoscere le frasi generate da questa. Il funzionamento di un ASFD (acronimo dell'automa che stiamo trattando) può essere riassunto in un grafo come quelli mostrati in queste pagine. Ad esempio in figura 2 è mostrato il grafo dell'ASFD in grado di riconoscere se un nome di variabile è lecito oppure no.

In ogni grafo distinguiamo dei nodi contrassegnati da uno stato dell'automa (q0, q1, q2 ecc.) e degli archi tra i nodi contrassegnati da un termianle (o da una classe di terminali per semplificare il grafo). Lo stato iniziale dell'automa è contrassegnato da una freccia, quello finale (possono essere anche più di uno) da un doppio cerchio. Si parte dallo stato iniziale e leggendo mano mano i simboli si transla di stato a seconda dei simboli in lettura. Terminata la lettura, se l'automa si trova in uno stato finale la stringa è accettata (risposta «sì») se si ferma su uno stato non finale o abortisce prima del termine (da uno stato, con un dato simbolo in lettura non poteva andare avanti) la risposta è «no» e quindi la stringa non è riconosciuta.

In figura 2 il primo grafo. Proviamo a testare il nome di variabile AZ15. Partiamo dallo stato q0 e leggiamo la A. Trattasi di una lettera quindi trasliamo nello stato q1. Leggiamo una Z e, come da grafo, rimaniamo nello stato q1. Analogamente per l'1 e il 5. Abbiamo terminato la lettura, q1 è uno stato finale (è cerchiato due volte) quindi la stringa è riconosciuta. Proviamo a fare lo stesso con 44 GATTI che come detto non è valido. Partendo come prima da q0 leggiamo un numero, il primo 4 e trasliamo in q2. Come prima continueremo a ciclare in tale stato fino a lettura ultimata. q2 non è però uno stato finale quindi la stringa non è riconosciuta.

In figura 3A troviamo un altro esempio di grammatica regolare. Nella fattispecie tale grammatica genera stringhe formate da un certo numero di *a* seguito eventualmente da un certo numero di *b*. Si noti che nessun legame esiste tra la quantità di *a* e di *b* che formano la stringa. Abbiamo già detto che se così fosse la grammatica non sarebbe regolare.

In figura 3B è mostrato il grafo dell'automa che riconosce tali stringhe. Partendo dallo stato iniziale se leggiamo dal nastro una *a* trasliamo nello stato q1. Se leggiamo come primo carattere una *b* l'automa abortisce non essendo presente alcun arco da q0 etichettato in questo modo. Si noti che la stringa formata dalla sola *b* non è generata dalla grammatica di figura 3A. Restiamo nello stato q1 fintantoché continuiamo a leggere *a*. Se la stringa termina, ovvero era formata solo da tale carattere, essendo q1 uno stato finale la risposta è «sì». Se, di contro, incontriamo una *b* trasliamo nello stato q2 e lì restiamo fintantoché leggiamo delle *b*. Come prima, se la stringa così facendo termina, viene accettata, se incontriamo una *a* si abortisce come prima in quanto non sono ammesse stringhe fatte da un certo numero di *a* seguite da alcune *b* e poi di nuovo le *a*.

Infine, in figura 4 è mostrato il grafo dell'automa corrispondente alla grammatica regolare descritta nel paragrafo precedente. Lì l'unico stato finale è q1 al quale si arriva solo dopo aver letto una stringa iniziante e terminante per *a*.

Provare per credere.

Arrivederci.

LA PERFEZIONE DIVENTA MITO

QUAD-MITO - 5 1/4" 96 TPI DS/QD
Floppy disk a quadrupla densità, disegnato per aumentare la capacità di registrazione sino a 780 kb per dischetto.
Velocità di registrazione 5800 BPI

MEGA-MITO - 5 1/4" 96 TPI HIGH DENSITY
Floppy ad alta densità, disegnato per drive da 1.2 MEG (AT e compatibili).
Velocità di registrazione 9650 BPI

MICRO-MITO - 3 1/2" 135 TPI DS/DD
Costruito per l'era dei disk drive da 3 1/2".
Velocità di registrazione 8100 BPI

le misure della perfezione

Microforum
MANUFACTURING INC.
TORONTO - CANADA

944/A St. Claire Ave. West,
TORONTO, CANADA M6C 1C8 - Tel. (416) 656-6406
Tlx. 06-986766 Tor - Telefax (416) 222-5326