Capitolo 3

Espressioni e linguaggi regolari

Apriamo questo capitolo presentando le cosiddette espressioni regolari, un tipo di notazione per definire linguaggi già brevemente esemplificato nel Paragrafo 1.1.2. Le espressioni regolari si possono anche considerare come un linguaggio di programmazione in cui esprimere applicazioni rilevanti, relative per esempio all'ambito delle ricerche in testi o come componenti di un compilatore. Le espressioni regolari sono strettamente legate agli automi a stati finiti non deterministici e possono rappresentare una comoda alternativa alla notazione degli NFA per descrivere componenti software.

In questo capitolo, dopo averle definite, dimostriamo che le espressioni regolari sono in grado di definire tutti e soli i linguaggi regolari, discutiamo il modo in cui esse sono utilizzate in diversi sistemi software, quindi esaminiamo le leggi algebriche che si applicano loro. Nonostante differenze rilevanti, tali leggi somigliano in modo significativo a quelle dell'aritmetica.

3.1 Espressioni regolari

Da una descrizione dei linguaggi fondata su un tipo di macchina, gli automi a stati finiti deterministici e no, volgiamo ora l'attenzione a una descrizione algebrica: le "espressioni regolari". Scopriremo che le espressioni regolari definiscono esattamente gli stessi linguaggi descritti dalle varie forme di automa: i linguaggi regolari. Le espressioni regolari offrono comunque qualcosa in più rispetto agli automi: un modo dichiarativo di esprimere le stringhe da accettare. Perciò le espressioni regolari servono da linguaggio di input per molti sistemi che trattano stringhe. Vediamo alcuni esempi.

1. I comandi di ricerca, come `grep` nei sistemi UNIX, o i comandi equivalenti per la ricerca di stringhe in un browser o in programmi per la composizione di testi. In tutti questi casi si impiega una notazione simile alle espressioni regolari per

descrivere i *pattern* che l'utente vuole cercare in un file. Un sistema di ricerca converte l'espressione regolare in un DFA o in un NFA e simula l'automa sul file da esaminare.

2. I generatori di analizzatori lessicali, come Lex e Flex. Un analizzatore lessicale è quel componente di un compilatore che spezza un programma sorgente in unità logiche dette *token*, composte da uno o più caratteri e aventi un significato preciso. Esempi di *token* sono le "parole chiave" o *keyword* (ad es., `while`), gli identificatori (ad es., una lettera seguita da zero o più lettere o cifre) e certi simboli, come + o <=. Un generatore di analizzatori lessicali accetta descrizioni della forma dei *token*, in sostanza espressioni regolari, e produce un DFA che riconosce la sequenza dei *token* nell'input.

3.1.1 Gli operatori delle espressioni regolari

Le espressioni regolari denotano linguaggi. L'espressione regolare $\mathbf{01^* + 10^*}$ denota per esempio il linguaggio formato da tutte le stringhe composte da uno 0 seguito da qualsiasi numero di 1, oppure da un 1 seguito da qualsiasi numero di 0. Poiché non abbiamo ancora spiegato come interpretare le espressioni regolari, per il momento l'affermazione relativa al linguaggio di tale espressione deve essere accettata sulla parola. Tra breve definiremo tutti i simboli usati nell'espressione, in modo da chiarire perché l'interpretazione proposta è quella corretta. Prima di descrivere la notazione delle espressioni regolari, dobbiamo conoscere le tre operazioni sui linguaggi rappresentate dagli operatori delle espressioni regolari. Tali operazioni sono le seguenti.

1. L'*unione* di due linguaggi L ed M, indicata con $L \cup M$, è l'insieme delle stringhe che sono in L oppure in M, oppure in entrambi. Per esempio se $L = \{001, 10, 111\}$ e $M = \{\epsilon, 001\}$, allora $L \cup M = \{\epsilon, 10, 001, 111\}$.

2. La *concatenazione* dei linguaggi L ed M è l'insieme delle stringhe che si possono formare prendendo una qualunque stringa in L e concatenandola con qualsiasi stringa in M. Abbiamo definito la concatenazione di una coppia di stringhe nel Paragrafo 1.5.2; una stringa è seguita dall'altra per formare il risultato della concatenazione. Indichiamo la concatenazione di linguaggi con un punto oppure senza alcun operatore; l'operatore di concatenazione è spesso chiamato "punto" (*dot*). Per esempio, se $L = \{001, 10, 111\}$ e $M = \{\epsilon, 001\}$, allora $L \cdot M$, o semplicemente LM, è $\{001, 10, 111, 001001, 10001, 111001\}$. Le prime tre stringhe in LM sono le stringhe in L concatenate con ϵ. Dato che ϵ è l'identità per la concatenazione, le stringhe risultanti sono quelle in L. Le ultime tre stringhe in LM sono invece formate prendendo ogni stringa in L e concatenandola con la seconda stringa in M, che è 001. Per esempio 10 da L concatenato con 001 da M dà 10001 per LM.

3. La *chiusura* (o *star* o *chiusura di Kleene*[1]) di un linguaggio L viene indicata con L^* e rappresenta l'insieme delle stringhe che possono essere formate prendendo un numero qualsiasi di stringhe da L, eventualmente con ripetizioni (la stessa stringa può essere selezionata più di una volta), e concatenandole tutte. Per esempio, se $L = \{0, 1\}$, allora L^* consiste di tutte le stringhe di 0 e 1. Se $L = \{0, 11\}$, allora L^* consiste di quelle stringhe di 0 e 1 tali che gli 1 compaiono a coppie, per esempio 011, 11110 e ϵ, ma non 01011 o 101. In termini più formali, L^* è l'unione infinita $\cup_{i \geq 0} L^i$, dove $L^0 = \{\epsilon\}$, $L^1 = L$, e L^i, per $i > 1$ è $LL \cdots L$ (la concatenazione di i copie di L).

Esempio 3.1 Dato che l'idea di chiusura di un linguaggio non è immediatamente comprensibile, ne esaminiamo qualche esempio. In primo luogo, sia $L = \{0, 11\}$. $L^0 = \{\epsilon\}$, indipendentemente da L; la potenza di ordine 0 rappresenta la selezione di zero stringhe da L. Abbiamo poi $L^1 = L$, ovvero la scelta di una stringa di L. Dunque i primi due termini nell'espansione di L^* danno $\{\epsilon, 0, 11\}$.

Consideriamo poi L^2. Prendiamo due stringhe da L, con ripetizioni consentite, per cui ci sono quattro scelte, che danno $L^2 = \{00, 011, 110, 1111\}$. Analogamente L^3 è l'insieme delle stringhe che possono essere formate compiendo tre scelte delle due stringhe in L, cioè

$$\{000, 0011, 0110, 1100, 01111, 11011, 11110, 111111\}$$

Per calcolare L^* dobbiamo computare L^i per ogni i, e prendere l'unione di tutti questi linguaggi. L^i ha 2^i membri. Sebbene ogni L^i sia finito, l'unione degli infiniti termini L^i è generalmente un linguaggio infinito, come nel nostro esempio.

Sia ora L l'insieme di tutte le stringhe di 0. Si noti che L è infinito, a differenza dell'esempio precedente, in cui L era un linguaggio finito. Tuttavia non è difficile scoprire cos'è L^*. $L^0 = \{\epsilon\}$, come sempre, mentre $L^1 = L$. L^2 è l'insieme di stringhe formato prendendo una stringa di 0 e concatenandola con un'altra stringa di 0. Il risultato è ancora una stringa di 0. In effetti ogni stringa di 0 può essere scritta come la concatenazione di due stringhe di 0 (non si dimentichi che ϵ è una "stringa di 0" e può sempre essere una delle due stringhe che vengono concatenate). Dunque $L^2 = L$. Analogamente $L^3 = L$, e così di seguito. Perciò l'unione infinita $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots$ è L nel caso particolare che il linguaggio L sia l'insieme di tutte le stringhe di 0.

Come esempio finale, $\emptyset^* = \{\epsilon\}$. Notiamo che $\emptyset^0 = \{\epsilon\}$, mentre $\emptyset^i$ è vuoto per ogni $i \geq 1$, dato che nessuna stringa può essere selezionata dall'insieme vuoto. Effettivamente $\emptyset$ è uno degli unici due linguaggi la cui chiusura *non* è infinita. □

[1] La locuzione "chiusura di Kleene" fa riferimento a S. C. Kleene, che ideò la notazione delle espressioni regolari e quest'operatore.

Uso dell'operatore asterisco

Abbiamo incontrato per la prima volta l'operatore asterisco nel Paragrafo 1.5.2, dove l'abbiamo applicato a un alfabeto, per esempio Σ^*. Tale operatore forma tutte le stringhe i cui simboli sono scelti dall'alfabeto Σ. L'operatore di chiusura è essenzialmente lo stesso, sebbene vi sia una sottile differenza di tipo.

Supponiamo che L sia il linguaggio contenente stringhe di lunghezza 1, e che per ogni simbolo a in Σ esista una stringa a in L. Allora, sebbene L e Σ abbiano lo stesso "aspetto", sono di tipo diverso: L è un insieme di stringhe, mentre Σ è un insieme di simboli. D'altra parte L^* indica lo stesso linguaggio di Σ^*.

3.1.2 Costruzione di espressioni regolari

Qualsiasi algebra si forma a partire da alcune espressioni elementari, di solito costanti e variabili. Le algebre permettono poi di costruire ulteriori espressioni applicando un determinato insieme di operatori alle espressioni elementari e alle espressioni costruite in precedenza. Abitualmente è necessario anche un metodo per raggruppare gli operatori con i loro operandi, ad esempio attraverso le parentesi. Per esemplificare, l'algebra aritmetica comunemente nota parte da costanti come gli interi e i numeri reali, più le variabili, e costruisce espressioni più complesse con operatori aritmetici come $+$ e $\times$.

L'algebra delle espressioni regolari segue questo schema, usando costanti e variabili che indicano linguaggi e operatori per le tre operazioni del Paragrafo 3.1.1: unione, punto e star. Possiamo descrivere le espressioni regolari in maniera ricorsiva, come segue. In questa definizione, non solo descriviamo che cosa sono le espressioni regolari lecite, ma per ogni espressione regolare E descriviamo il linguaggio che rappresenta, indicandolo con $L(E)$.

BASE La base consiste di tre parti.

1. Le costanti ϵ e $\emptyset$ sono espressioni regolari, indicanti rispettivamente i linguaggi $\{\epsilon\}$ e $\emptyset$. In altre parole $L(\epsilon) = \{\epsilon\}$, e $L(\emptyset) = \emptyset$.

2. Se a è un simbolo qualsiasi, allora $\mathbf{a}$ è un'espressione regolare. Tale espressione denota il linguaggio $\{a\}$. Cioè $L(\mathbf{a}) = \{a\}$. Si noti che si usa il grassetto per indicare un'espressione corrispondente a un simbolo. La corrispondenza, per esempio che $\mathbf{a}$ si riferisce ad a, dovrebbe essere evidente.

3. Una variabile, generalmente una lettera maiuscola e in corsivo, come L, rappresentante un linguaggio arbitrario.

INDUZIONE Il passo induttivo consta di quattro parti, una per ognuno dei tre operatori e una per l'introduzione delle parentesi.

1. Se E ed F sono espressioni regolari, allora $E + F$ è un'espressione regolare che indica l'unione di $L(E)$ e $L(F)$. In altri termini $L(E + F) = L(E) \cup L(F)$.

2. Se E ed F sono espressioni regolari, allora EF è un'espressione regolare che indica la concatenazione di $L(E)$ e $L(F)$. Ossia $L(EF) = L(E)L(F)$. Si osservi che il punto può essere usato facoltativamente per indicare l'operatore di concatenazione, visto come operatore su linguaggi oppure come operatore in un'espressione regolare. Per esempio $\mathbf{0}\cdot\mathbf{1}$ è un'espressione regolare che ha lo stesso significato di $\mathbf{01}$ e rappresenta il linguaggio $\{01\}$. Tuttavia si eviterà di usare il punto come concatenazione nelle espressioni regolari.[2]

3. Se E è un'espressione regolare, allora E^* è un'espressione regolare che indica la chiusura di $L(E)$. Cioè $L(E^*) = \big(L(E)\big)^*$.

4. Se E è un'espressione regolare, allora anche (E), un E tra parentesi, è un'espressione regolare, e indica lo stesso linguaggio di E. In termini formali $L\big((E)\big) = L(E)$.

Esempio 3.2 Scriviamo un'espressione regolare per l'insieme delle stringhe che consistono di 0 e 1 alternati. Dapprima sviluppiamo un'espressione regolare per il linguaggio che consta di una singola stringa 01. Possiamo poi usare l'operatore asterisco per ottenere un'espressione per tutte le stringhe della forma $0101\cdots01$.

La regola di base per le espressioni regolari ci dice che $\mathbf{0}$ e $\mathbf{1}$ sono espressioni che indicano rispettivamente i linguaggi $\{0\}$ e $\{1\}$. Se concateniamo le due espressioni, otteniamo un'espressione regolare per il linguaggio $\{01\}$; questa espressione è $\mathbf{01}$. Come regola generale, se vogliamo un'espressione regolare per il linguaggio che consiste solamente della stringa w, usiamo w stesso come espressione regolare. Si noti che nell'espressione regolare i simboli di w saranno scritti di norma in grassetto, ma il cambiamento di carattere serve solo a distinguere le espressioni dalle stringhe e non dovrebbe essere considerato significativo.

Per ottenere tutte le stringhe di zero o più occorrenze di 01, usiamo ora l'espressione regolare $(\mathbf{01})^*$. Si osservi che mettiamo prima tra parentesi $\mathbf{01}$ per evitare di confonderci con l'espressione $\mathbf{01}^*$, il cui linguaggio è formato da tutte le stringhe consistenti di uno 0 e di un qualunque numero di 1. La ragione di questa interpretazione verrà chiarita nel Paragrafo 3.1.3, ma si può anticipare brevemente che lo star ha la precedenza sul punto e perciò l'argomento dello star viene selezionato prima di compiere concatenazioni.

[2] In realtà le espressioni regolari in UNIX impiegano il punto per uno scopo completamente diverso, ossia per rappresentare qualunque carattere ASCII.

Le espressioni e i loro linguaggi

Per essere precisi, un'espressione regolare E è appunto un'espressione, non un linguaggio. Quando vogliamo riferirci al linguaggio denotato da E, dovremmo usare $L(E)$. Tuttavia è consuetudine fare riferimento a E quando effettivamente si intende $L(E)$. Ricorreremo a tale convenzione finché sarà chiaro che si tratta di un linguaggio e non di un'espressione regolare.

Tuttavia $L\big((\mathbf{01})^*\big)$ non è esattamente il linguaggio che vogliamo, in quanto include solo le stringhe di 0 e 1 alternati che cominciano per 0 e finiscono per 1, mentre dobbiamo anche considerare la possibilità che ci sia un 1 all'inizio e uno 0 alla fine. Un modo può essere quello di costruire altre tre espressioni regolari che trattino le altre tre possibilità. In altre parole $(\mathbf{10})^*$ rappresenta le stringhe alternate che cominciano per 1 e finiscono per 0, $\mathbf{0}(\mathbf{10})^*$ può essere usato per stringhe che cominciano e finiscono per 0, e $\mathbf{1}(\mathbf{01})^*$ si impiega per stringhe che cominciano e finiscono per 1. L'espressione regolare completa è

$$(\mathbf{01})^* + (\mathbf{10})^* + \mathbf{0}(\mathbf{10})^* + \mathbf{1}(\mathbf{01})^*$$

Si faccia attenzione all'uso dell'operatore $+$ per effettuare l'unione dei quattro linguaggi, che nel complesso danno tutte le stringhe con alternanze di 0 e 1.

Esiste però un altro modo, che produce un'espressione regolare diversa e più concisa. Ricominciamo dall'espressione $(\mathbf{01})^*$. Se facciamo una concatenazione a sinistra con l'espressione $\epsilon + \mathbf{1}$, possiamo aggiungere un 1 facoltativo all'inizio. Analogamente aggiungiamo uno 0 facoltativo alla fine con l'espressione $\epsilon + \mathbf{0}$. Per esempio, ricorrendo alla definizione dell'operatore $+$:

$$L(\epsilon + \mathbf{1}) = L(\epsilon) \cup L(\mathbf{1}) = \{\epsilon\} \cup \{1\} = \{\epsilon, 1\}$$

Se concateniamo questo linguaggio con un qualsiasi altro linguaggio L, la scelta ϵ dà tutte le stringhe in L, mentre la scelta 1 dà $1w$ per ogni stringa w in L. Perciò un'altra espressione per l'insieme di stringhe che alternano 0 e 1 è:

$$(\epsilon + \mathbf{1})(\mathbf{01})^*(\epsilon + \mathbf{0})$$

Si noti che, per garantire che gli operatori siano raggruppati opportunamente, sono necessarie le parentesi prima e dopo ogni espressione aggiunta. □

3.1.3 Precedenza degli operatori delle espressioni regolari

Come altre algebre, gli operatori delle espressioni regolari hanno un ordine di precedenza, il che significa che gli operatori sono associati ai loro operandi in un ordine particolare. La

nozione di precedenza ci è familiare dalle espressioni aritmetiche ordinarie. Per esempio sappiamo che $xy + z$ raggruppa il prodotto xy prima della somma, dunque è equivalente all'espressione tra parentesi $(xy) + z$ e non all'espressione $x(y + z)$. In aritmetica raggruppiamo similmente da sinistra due operatori uguali, così $x - y - z$ è equivalente a $(x - y) - z$ e non a $x - (y - z)$. Per le espressioni regolari l'ordine di precedenza degli operatori è il seguente.

1. L'operatore star ha il livello più alto di priorità. Vale a dire, si applica solamente alla sequenza più breve di simboli alla sua sinistra che sia un'espressione regolare ben formata.

2. Il successivo, nell'ordine di precedenza, è la concatenazione o operatore "punto". Dopo aver raggruppato tutti gli star con i loro operandi, passiamo agli operatori di concatenazione con i loro operandi. In altre parole vengono raggruppate insieme tutte le espressioni che sono *giustapposte* (adiacenti, senza alcun operatore interposto). Dato che la concatenazione è un operatore associativo, non ha importanza in quale ordine raggruppiamo le concatenazioni consecutive, sebbene, dovendo scegliere, sia consigliabile farlo a partire da sinistra. Per esempio **012** viene raggruppato così: **(01)2**.

3. Per ultime vengono raggruppate le unioni (gli operatori $+$) con i loro operandi. Dato che anche l'unione è associativa, non è importante in quale ordine vengono raggruppate unioni consecutive, anche se si assumerà che il raggruppamento avvenga a partire da sinistra.

Ovviamente a volte non si desidera che in un'espressione regolare il raggruppamento segua l'ordine di precedenza degli operatori. In tal caso siamo liberi di usare le parentesi per raggruppare gli operandi come ci pare più opportuno. Oltre a ciò non è scorretto neppure mettere fra parentesi gli operatori che si vogliono raggruppare, anche se il raggruppamento desiderato è conforme alle regole di precedenza.

Esempio 3.3 L'espressione $\mathbf{01}^* + \mathbf{1}$ corrisponde a $\big(\mathbf{0}(\mathbf{1}^*)\big) + \mathbf{1}$. Per primo viene raggruppato l'operatore star. Poiché il simbolo $\mathbf{1}$ immediatamente alla sua sinistra è un'espressione regolare lecita, esso da solo è l'operando dello star. Raggruppiamo poi la concatenazione tra $\mathbf{0}$ e $(\mathbf{1}^*)$, che produce l'espressione $\big(\mathbf{0}(\mathbf{1}^*)\big)$. Infine l'operatore unione connette questa espressione e quella alla sua destra, che è $\mathbf{1}$.

Si noti che il linguaggio dell'espressione data, raggruppata secondo le regole di precedenza, è la stringa 1 più tutte le stringhe formate da uno 0 seguito da un qualunque numero di 1 (incluso nessuno). Se avessimo scelto di raggruppare il punto prima dello star, avremmo potuto usare le parentesi così: $(\mathbf{01})^* + \mathbf{1}$. Il linguaggio di quest'espressione è la stringa 1 e tutte le stringhe che ripetono 01, zero oppure più volte. Se volessimo raggruppare prima l'unione, potremmo metterla tra parentesi per produrre l'espressione

$\mathbf{0}(\mathbf{1}^* + \mathbf{1})$. Il linguaggio di quest'espressione è l'insieme di stringhe formate da uno 0 seguito da un numero arbitrario di 1. □

3.1.4 Esercizi

Esercizio 3.1.1 Scrivete le espressioni regolari per i seguenti linguaggi.

* a) L'insieme delle stringhe sull'alfabeto $\{a, b, c\}$ che contengano almeno una a e almeno una b.

b) L'insieme delle stringhe di 0 e 1 il cui decimo simbolo a partire da destra sia 1.

c) L'insieme delle stringhe di 0 e 1 con al massimo una coppia di 1 consecutivi.

! **Esercizio 3.1.2** Scrivete le espressioni regolari per i seguenti linguaggi.

* a) L'insieme di tutte le stringhe di 0 e 1 tali che ogni coppia di 0 adiacenti compaia prima di qualunque coppia di 1 adiacenti.

b) L'insieme delle stringhe di 0 e 1 il cui numero di 0 sia divisibile per cinque.

!! **Esercizio 3.1.3** Scrivete le espressioni regolari per i seguenti linguaggi.

a) L'insieme di tutte le stringhe di 0 e 1 che non contengano 101 come sottostringa.

b) L'insieme di tutte le stringhe con un numero uguale di 0 e 1 tali che in ogni prefisso la differenza tra il numero di 0 e il numero di 1 sia minore di 2.

c) L'insieme delle stringhe di 0 e 1 il cui numero di 0 sia divisibile per cinque e il numero di 1 sia pari.

! **Esercizio 3.1.4** Descrivete in italiano i linguaggi delle seguenti espressioni regolari:

* a) $(\mathbf{1} + \epsilon)(\mathbf{00}^*\mathbf{1})^*\mathbf{0}^*$

b) $(\mathbf{0}^*\mathbf{1}^*)^*\mathbf{000}(\mathbf{0} + \mathbf{1})^*$

c) $(\mathbf{0} + \mathbf{10})^*\mathbf{1}^*$.

*! **Esercizio 3.1.5** Nell'Esempio 3.1 abbiamo sottolineato che $\emptyset$ è uno dei linguaggi la cui chiusura è finita. Qual è l'altro?