Written by Marco & Alioscia

Linguaggi formali e grammatiche

Dispense ad integrazione del Corso di Informatica Teorica

Indice

Capitolo
1 Introduzione alla dispensa

Introduzione

I fondamenti (teorici) dell'informatica sono lo studio dei modelli per il calcolo. Tale disciplina, la cui nascita si può far risalire al 1930 (prima dell'esistenza del moderno calcolatore) negli studi dei logici Church, Gödel, Kleene, Post e Turing è alla base degli sviluppi pratici e teorici del calcolo mediante calcolatore. Ad esempio è possibile garantire in modo formale l'esistenza di una macchina universale (calcolatore con programma software) o caratterizzare rigorosamente la nozione di programma come sequenza/insieme di istruzioni. Si dimostra l'esistenza di funzioni non calcolabili da nessun calcolatore e, di conseguenza, di problemi non risolubili in modo completo e automatico mediante calcolatore. Lo studio formale dei linguaggi generabili a partire da un dato alfabeto è invece alla base dello sviluppo delle tecniche di traduzione dei moderni linguaggi di programmazione. Tali metodologie e risultati devono costituire, per un Informatico, ma più in generale per chi utilizza professionalmente un calcolatore, una base di conoscenze indispensabile come le nozioni fondamentali di algebra e di analisi lo sono per il matematico.
Scopo del testo è dunque quello di fornire gli strumenti formali e le nozioni fondamentali per studiare problemi trattabili e non mediante calcolatore.
Gli unici prerequisiti sono una conoscenza di base di teoria elementare degli insiemi e di logica.

Capitolo
2 Notazione e concetti di base

Introduzione

In questo breve capitolo viene stabilita la notazione impiegata nel testo relativamente ai concetti base di matematica e logica utilizzati.

1. Insiemi

Con lettere latine maiuscole denoteremo in genere insiemi di oggetti.
Alcuni insiemi hanno un nome particolare che li identifica univocamente: ad esempio N rappresenta l'insieme dei numeri naturali ($N = \{0, 1, 2, 3, \ldots\}$).
$x \in A$ significa che x è un *elemento* dell'insieme A. Mediante la rappresentazione *intensionale* di insiemi $\{x \mid E(x)\}$ si identifica l'insieme costituito dagli x che soddisfano (rendono vera) una data espressione E, la quale dipende da x ed ha valori booleani. Se gli elementi di un insieme hanno un indice e sono tutti rappresentabili come una sequenza (anche infinita) di elementi x_i al variare di $i \in I$, allora l'insieme costituito da questi oggetti è rappresentato con $\{x_i\}_{i \in I}$. $A \subseteq B$ indica che A è un sottoinsieme di B, ovvero che ogni elemento di A è anche elemento di B. Con la notazione $A \subset B$ si denoterà l'inclusione *stretta* ovvero vale che $A \subseteq B$ e $A \neq B$.
Con $\wp(A)$ si denota *l'insieme delle parti* dell'insieme A, ovvero l'insieme costituito da tutti i suoi sottoinsiemi: $\wp(A) = \{X \mid X \subseteq A\}$. Unione ed intersezione di insiemi sono rispettivamente denotati con i simboli $\bigcup$ e $\bigcap$.
Il simbolo \ rappresenta la *differenza insiemistica*: $A \setminus B = \{x \mid x \in A \wedge x \notin B\}$.
$\overline{A}$ denota il *complemento* di A, ovvero $x \in \overline{A}$ se e solo se $x \notin A$. Se assumiamo (come faremo spesso nel seguito) che gli insiemi sono insiemi di numeri naturali, allora considerando N come insieme *universo*, si ha che $\overline{A} = N \setminus A$.
Dati due o più insiemi, è possibile costruire coppie, triple, etc. di oggetti. In generale una *n-upla* di oggetti $x_1, \ldots, x_n$ di un insieme A è rappresentata con $\langle x_1, \ldots, x_n \rangle$. L'insieme delle *n-uple* di elementi di A è rappresentato con A^n e corrisponde al *prodotto cartesiano* di A *n*-volte.
Insiemi diversi possono essere messi in prodotto cartesiano:

$$A_1 \times A_2 \times \ldots \times A_n = \{\langle x_1, \ldots, x_n \rangle \mid x_1 \in A_1, \ldots, x_n \in A_n\}$$

indica l'insieme delle n-uple di oggetti appartenenti rispettivamente agli insiemi $A_1, \ldots, A_n$.

2. Relazioni e funzioni

Una relazione (binaria) è un sottoinsieme del prodotto cartesiano di due insiemi; dati A e B, $\Re \subseteq A \times B$ è una relazione su A e B. Ad esempio, la relazione di ordinamento sui numeri

naturali $"\leq" \subseteq N \times N$ è definita nel modo seguente: $\leq = \{\langle x, y\rangle \mid x \in N, y \in N, x \leq y\}$. Il fatto che $\langle x, y\rangle \in \leq$ viene solitamente indicato con $x \leq y$.

Dato un insieme *S*, una relazione $R \subseteq S \times S$ è una relazione di *equivalenza* se gode delle proprietà:

riflessiva: per ogni $a \in S$ si ha che *a R a*,

simmetrica: per ogni $a, b \in S$ si ha che se *a R b* allora *b R a*, e

transitiva: per ogni $a, b, c \in S$ si ha che se *a R b* e *b R c* allora *a R c*.

E possibile definire relazioni *n*-arie, ovvero $\Re \subseteq A_1 \times \ldots \times A_n$. Sia $k < n$. Una relazione $\Re$ *n*-aria è una *funzione* di *k* argomenti se, dati $x_1 \in A_1, \ldots, x_k \in A_k$, esiste uno ed un solo $z_{k+1} \in A_{k+1}, \ldots, z_n \in A_n$ tale che $\langle x_1, \ldots, x_k, z_{k+1}, \ldots, z_n\rangle \in \Re$. In questo caso, la funzione è definita in $A_1 \times \ldots \times A_k$ ed ha valori in $A_{k+1} \times \ldots \times A_n$, ovvero è del tipo:

$$\Re : A_1 \times \ldots \times A_k \rightarrow A_{k+1} \times \ldots \times A_n$$

Nel caso $n = 2$ e $k = 1$, otteniamo la definizione usuale di funzione di un argomento: $\Re : A_1 \rightarrow A_2$ come relazione binaria $\Re \subseteq A_1 \times A_2$.

Denoteremo come $\vec{x}$ una generica *n*-upla (vettore) di oggetti. La lunghezza di tale *n*-upla si evincerà dal contesto. Una funzione è *iniettiva* se $f(\vec{x}_1) = f(\vec{x}_2)$ implica che $\vec{x}_1 = \vec{x}_2$ per ogni x_1, x_2; è *suriettiva* se per ogni elemento $\vec{y}$ del *codominio* (nel caso sopra $A_{k+1} \times \ldots \times A_n$) esiste $\vec{x}$ nel *dominio* (nel caso sopra $A_1 \times \ldots \times A_k$) tale che $f(\vec{x}) = \vec{y}$.

Una funzione può non essere definita su alcuni argomenti, ovvero se la funzione di *k* argomenti è data dalla relazione $\Re \subseteq A_1 \times \ldots \times A_n$, $k < n$, allora possono esistere degli $\in A_i$ argomenti di $\Re$ (ovvero $i \leq k$) tali che nessuna *n*-upla $\langle \ldots, x, \ldots\rangle$ avente *x* nella i-esima posizione appartiene a $\Re$. In questo caso diremo che la funzione $\Re$ è *parziale*.

Al contrario, una funzione sempre definita su ogni argomento è detta *totale*.

NOTA 2.1. Normalmente si usa parlare di funzioni $f: A \rightarrow B$ e $f(x)$ è definita per ogni elemento *x* dell'insieme *A* (dominio). Ad esempio, la funzione $f(x) = 1000$ *div x*, ove *div* ritorna il quoziente della divisione intera, è definita da $A = N \setminus \{0\}$ a $B = N$. In questo testo, per mantenere un'analogia con le funzioni calcolabili mediante un calcolatore (che potrebbe non fornire risultato per certi valori di input), si preferisce parlare di $f: N \rightarrow N$ e parlare di funzioni parziali.

Nel seguito faremo largo uso di funzioni parziali e totali. Al fine di evitare confusione, saremo soliti rappresentare le funzioni secondo la notazione usuale:

$$\Re : A_1 \times \ldots \times A_k \rightarrow A_{k+1} \times \ldots \times A_n$$

identificando in questo modo gli insiemi su cui vengono considerati gli argomenti e gli insiemi su cui la funzione restituisce risultati. In particolare, rappresenteremo con lettere minuscole latine le funzioni totali; ad esempio:

$$f : A_1 \times \ldots \times A_k \rightarrow A_{k+1} \times \ldots \times A_n$$

rappresenta una funzione totale di k argomenti, mentre rappresenteremo con lettere minuscole greche le funzioni parziali:

$$\varphi : A_1 \times \ldots \times A_k \rightarrow A_{k+1} \times \ldots \times A_n$$

rappresenta una funzione parziale di *k* argomenti. Poiché tra l'insieme delle funzioni parziali vi sono anche le funzioni totali, nel caso non vi siano ipotesi sulla totalità della funzione considerata, rappresenteremo con lettere greche minuscole una generica funzione.

Per indicare che una funzione φ è definita in *x*, scriveremo $\varphi(x)\downarrow$. Analogamente, per indicare che una funzione φ non è definita in *x*, scriveremo $\varphi(x)\uparrow$, o equivalentemente $\varphi(x)=\uparrow$. Il simbolo $\uparrow$ rappresenta la non definizione della funzione φ. Supponiamo quindi di ammettere come possibile valore della funzione anche il valore indefinito $\uparrow$. Questo ci permette di definire funzioni parziali del seguente tipo:

$$\varphi : A_1 \times \ldots \times A_k \rightarrow \left(A_{k+1} \cup \{\uparrow\}\right) \times \ldots \times \left(A_n \cup \{\uparrow\}\right)$$

Data dunque una generica funzione (parziale o totale) φ chiameremo *dominio* della funzione φ l'insieme su cui φ è definita, ovvero l'insieme:

$$dom(\varphi) = \{x \mid \varphi(x)\downarrow\}$$

Analogamente chiameremo *immagine* o *range* di φ l'insieme dei risultati definiti della funzione, ovvero:

$$range(\varphi) = \{x \mid \exists z.\varphi(z) = x \neq \uparrow\}$$

Un valore della funzione φ è un elemento di $range(\varphi)$.

In genere la definizione di una funzione $f(X, Y) = X + Y$ crea un'associazione (relazione) tra il nome *f* della funzione ed il corpo della funzione stessa $X + Y$. Eseguire *f* non significa altro che eseguire il suo corpo. In alcuni contesti è utile potersi riferire ad una funzione senza essere costretti ad associare ad essa un nome; il concetto di funzione ha infatti un suo significato a prescindere dal nome che gli viene assegnato con la definizione. A questo scopo viene introdotta la cosidetta λ-notazione. Per esempio la funzione che riceve due argomenti e ne restituisce la somma può essere definita con: $\lambda X, Y\ .\ X + Y$ dove λ è un simbolo speciale, generalmente seguito da una lista di variabili separate da una virgola (i *parametri* della funzione). Il corpo della funzione segue il punto: si tratta di un'espressione in cui possono apparire le variabili introdotte dopo il simbolo λ. Queste variabili risultano *legate* all'interno dell'espressione (bound variables), mentre si definiscono *libere* quelle variabili che appaiono nel corpo e non appaiono dopo il simbolo lambda.

L'applicazione di una funzione definita con la λ-notazione ha la seguente forma: alla definizione della funzione racchiusa fra parentesi seguono tante espressioni quanti sono i parametri. Ad esempio: $(\lambda X, Y\ .\ X + Y)7\ 4$ associa 7 a *X* e 4 a *Y*.

3. Cenni di Logica Matematica

Un *linguaggio del prim'ordine* **L** è costituito da

- *un'insieme di simboli relazionali o predicativi,*
- *un'insieme di simboli funzionali,*
- *un'insieme di simboli di costante e*
- *da un'insieme di simboli logici.*

I simboli logici presenti in ogni linguaggio sono:

- le parentesi ")", "(" e la virgola ","
- un insieme numerabile di variabili $v_0, v_1, \ldots$
- un insieme di connettivi: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- i quantificatori $\forall$ (per ogni) ed $\exists$ (esiste).

Si ricorda che l'implicazione e la doppia implicazione sono superflue in quanto $\varphi \to \psi$ è equivalente a $(\neg\varphi) \vee \psi$ mentre $\varphi \leftrightarrow \psi$ è equivalente a $(\varphi \to \psi) \wedge (\psi \to \varphi)$. Anche uno tra $\vee$ e $\wedge$ è superfluo (si vedano le leggi di De Morgan più avanti), mentre uno solo dei due quantificatori è sufficiente. Tuttavia tali operatori, di uso comune, sono usualmente accettati per fornire maggiore chiarezza alle formule.
Un *termine* di un linguaggio del prim'ordine è definito ricorsivamente nel seguente modo:

1. una variabile *v* è un termine;
2. un simbolo costante *c* è un termine;
3. se $t_1, \ldots, t_m$ sono termini e *f* è un simbolo funzionale *m*-ario, allora $f(t_1, \ldots, t_m)$ è un termine.

Se *p* è un simbolo relazionale *n*-ario di **L** e se $t_1, \ldots, t_n$ sono termini allora $p(t_1, \ldots, t_n)$ è una *formula atomica*. Una *formula* di un linguaggio del prim'ordine è definita ricorsivamente nel seguente modo:

1. una formula atomica è una formula;
2. se φ è una formula allora $(\neg\varphi)$ è una formula;
3. se φ e ψ sono formule, anche $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$ e $(\varphi \to \psi)$ sono formule;
4. se φ è una formula e *v* è una variabile allora $\forall v(\varphi)$ e $\exists v(\varphi)$ sono formule.

Per semplificare la notazione, si assumono le comuni convenzioni per eliminare, qualora ciò non generasse ambiguità, alcune coppie di parentesi dalle formule.
Il significato (la semantica) di una formula del prim'ordine è stabilito qualora venga fornita una *interpretazione*, associata ad un insieme non vuoto detto *dominio* (ad esempio, l'insieme dei numeri naturali *N*)

- per i simboli di costante (ad esempio, la costante *c* è il numero 5, *d* il numero 10),
- per i simboli di funzione (ad esempio, il simbolo funzionale binario *f* sta a significare il '+' tra numeri interi) e
- venga assegnata un'interpretazione ai predicati sul dominio, ovvero, in parole povere, un valore di verità (*vero* oppure *falso*) agli atomi costruiti con i simboli predicativi e gli oggetti del dominio (ad esempio, $f(5; 5) = 10$ è vero).

Fissata una interpretazione per un linguaggio del prim'ordine, usando la semantica degli operatori, è possibile estendere l'interpretazione ad ogni formula chiusa (ovvero, ogni variabile che occorre in essa è presente come quantificata) del linguaggio. Ad esempio, nell'interpretazione accennata sopra, la formula $\exists v(v = f(c,c))$ è vera in quanto per $v = 10$ è soddisfatta, mentre la formula $\forall v(v = f(c,c))$ è falsa (per esempio, si prenda $v = 11$).
Per quanto concerne l'utilizzo della logica del prim'ordine nel prosieguo, sarà fondamentale saper negare una formula logica, ovvero data una formula φ, scrivere in modo equivalente ma più leggibile la formula $\neg\varphi$.
Relativamente alle formule che non hanno quantificatori come simboli più "esterni", le regole da utilizzare sono le seguenti:

Doppia negazione: $\neg(\neg\varphi) = \varphi$
De Morgan 1: $\neg(\varphi \wedge \psi) = (\neg\varphi) \vee (\neg\psi)$
De Morgan 2: $\neg(\varphi \vee \psi) = (\neg\varphi) \wedge (\neg\psi)$

Si osservi in particolare che: $\neg(\varphi \to \psi)$ è equivalente a $\varphi \wedge \neg\psi$. Per quanto riguarda la semplificazione della **negazione di quantificatori** valgono:

- $\neg\exists\varphi = \forall\neg\varphi$
- $\neg\forall\varphi = \exists\neg\varphi$

Pertanto, ad esempio,

$$\neg\forall n\exists y(r(y) \wedge y > n \wedge \exists u\exists v(y = u + v \wedge \forall i(r(u + v^i))))$$

ove *r* è un simbolo unario di predicato, risulta essere:

$$\exists n \forall y (\neg r(y) \vee y \leq n \vee \forall u \forall v (y \neq u + v \vee \exists i (\neg r(u + v^i))))$$

Esplicitando le implicazioni, quest'ultima diventa:

$$\exists n \forall y ((r(y) \wedge y > n) \rightarrow \forall u \forall v (y = u + v \rightarrow \exists i (\neg r(u + v^i))))$$

Capitolo

3 Automi a stati finiti

Introduzione

In questo capitolo sarà presentato il concetto di automa a stati finiti. Si partirà dal formalismo più semplice (automi a stati finiti deterministici) per poi presentare diverse tipologie di automa, che si dimostreranno essere tutte equivalenti dal punto di vista delle potenzialità del loro utilizzo, ovvero del riconoscimento di un dato linguaggio.

1. Alfabeti e linguaggi

Un *simbolo* è un'entità primitiva astratta che non sarà definita formalmente (come punto, linea, ecc.). Lettere e caratteri numerici sono esempi di simboli.
Una *stringa* (o *parola*) è una sequenza finita di simboli giustapposti (uno dietro l'altro). Ad esempio, se *a*, *b*, *c* sono simboli, *abcba* è una stringa.
La *lunghezza* di una stringa *w*, denotata come |w|, è il numero di *occorrenze* di simboli che compongono una stringa. Ad esempio, $|abcba| = 5$. La stringa vuota, denotata con ε è la stringa costituita da zero simboli: $|\varepsilon| = 0$.
Sia $w = a_1 \ldots a_n$ una stringa. Ogni stringa della forma:

- $a_1 \ldots a_j$, con $j \in \{1,\ldots, n\}$ è detta un *prefisso* di *w*;
- $a_i \ldots a_n$, con $i \in \{1,\ldots, n\}$ è detta un *suffisso* di *w*;
- $a_i \ldots a_j$, con $i,j \in \{1,\ldots, n\}$, $i \leq j$, è detta una s*ottostringa* di *w*;
- ε è sia prefisso che suffisso che sottostringa di *w*.

Si osservi che se $n = 0$, allora $w = \varepsilon$. Dunque ε è sia prefisso che suffisso di ε.

Ad esempio, i prefissi di *abc* sono ε, *a*, *ab*, e *abc*. I suffissi sono ε, *c*, *bc*, e *abc*. Le sottostringhe sono:
ε
a *ab* *abc*
b *bc*
c

Un prefisso, un suffisso o una sottostringa di una stringa, quando non sono la stringa stessa, sono detti *propri*.

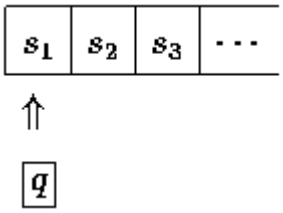
La concatenazione di due stringhe *v* e *w* è la stringa *vw* che si ottiene facendo seguire alla prima la seconda. La concatenazione è una operazione (associativa) che ammette come identità la stringa
vuota ε.
Un *alfabeto* Σ è un insieme finito di simboli. Un *linguaggio formale* (in breve linguaggio) è un insieme di stringhe di simboli da un alfabeto Σ. L'insieme vuoto Ø e l'insieme {ε} sono due linguaggi formali di qualunque alfabeto. Con Σ^* verrà denotato il linguaggio costituito da tutte le stringhe su un fissato alfabeto Σ. Dunque

$$\Sigma^* = \{a_1 \ldots a_n : n \geq 0, a_i \in \Sigma\}$$

Ad esempio, se $\Sigma = \{0\}$, allora $\Sigma^* = \{\varepsilon, 0, 00, 000, \ldots\}$; se $\Sigma = \{0, 1\}$, allora $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$.

2. Automi

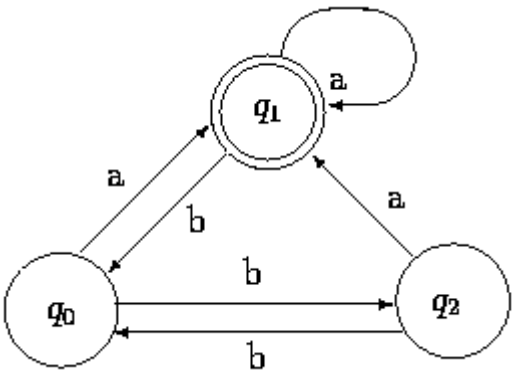
Un automa a stati finiti è un modello matematico di un sistema avente input, ed eventualmente output, a valori discreti. Il sistema può essere in uno stato tra un insieme finito di stati possibili. L'essere in uno stato gli permette di tener traccia della storia precedente. Un buon esempio di automa a stati finiti è costituito dall'ascensore: l'input è la sequenza di tasti premuti, mentre lo stato è il piano in cui si trova.
Un automa a stati finiti si può rappresentare mediante una testina che legge, spostandosi sempre nella stessa direzione, un nastro di lunghezza illimitata contenente dei simboli. La testina si può trovare in un certo stato; a seconda dello stato q e del simbolo s_i letto, la testina si porta in un altro stato (o rimane nello stesso) e si sposta a destra per apprestarsi a leggere il simbolo successivo:



Quando la lettura dei simboli termina, a seconda dello stato raggiunto dalla testina, l'automa fornisce un risultato di accettazione o di refutazione della stringa (parola) letta.
Il comportamento dell'automa si definisce in maniera univoca mediante una tabella, detta *matrice di transizione*, come ad esempio:

	a	b
q_0	q_1	q_2
q_1	q_1	q_0
q_2	q_1	q_0

Un automa siffatto si rappresenta bene anche con un grafo della forma seguente:



Nel prossimo paragrafo si fornirà una definizione formale del concetto ora esposto, necessaria per una trattazione precisa dell'argomento.

NOTA 3.3. Il fatto che, a differenza delle MdT, la testina non possa produrre degli output (eventualmente sul nastro) non è essenziale (si vedano, ad esempio, le macchine di Moore e di Mealy. Intuitivamente, anche se scrivesse qualcosa sul nastro, non lo potrebbe più riutilizzare. Neppure il permettere la bidirezionalità aumenterebbe di fatto la potenzialità del dispositivo. Intuitivamente, la testina andrebbe su e giù ma sempre sugli stessi dati e con un controllo finito. E' la somma delle due caratteristiche che permette di passare da questo formalismo al più potente formalismo di calcolo costituito dalla MdT.

3. Automi deterministici

Un automa a stati finiti deterministico (DFA) è una quintupla $\langle Q, \Sigma, \delta, q_0, F \rangle$ dove:

- Q è un insieme finito di stati ;
- Σ è un alfabeto (alfabeto di input);
- $\delta : Q \times \Sigma \to Q$ è la *funzione di transizione*;
- q_0 è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli *stati finali*.

NOTAZIONE 3.4. Useremo p, q, r con o senza pedici per denotare stati, P, Q, R, S per insiemi di stati, a, b con o senza pedici per denotare simboli di Σ, x, y, z, u, v, w sempre con o senza pedici per denotare stringhe.

Dalla funzione δ si ottiene in modo univoco la funzione $\hat{\delta} : Q \times \Sigma^* \to Q$ nel modo seguente:

$$\begin{cases} \hat{\delta}(q, \varepsilon) = q \\ \hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a) \end{cases}$$

Una stringa x è detta essere accettata da un DFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ se $\hat{\delta}(q_0, x) \in F$. Il *linguaggio accettato da M* , denotato come $L(M)$ è l'insieme delle stringhe accettate, ovvero:

$L(M) = \{ x \in \Sigma^* : \hat{\delta}(q_0, x) \in F \}$

Un linguaggio L è detto *regolare* se è accettato da qualche DFA, ovvero se esiste M tale che $L = L(M)$.

LEMMA 3.8. Siano vx e wx due stringhe e $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un DFA. Allora, $\hat{\delta}(q_0, v) = \hat{\delta}(q_0, w)$ implica $\hat{\delta}(q_0, vx) = \hat{\delta}(q_0, wx)$.

LEMMA 3.9. Sia $L \subseteq \Sigma^*$ un *linguaggio accettato* da un DFA M. Allora esiste $M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$ tale che $\Sigma^* = \Sigma$ e $L(M') = L(M)$.

4. Automi non-deterministici

Un automa a stati finiti non-deterministico (NFA) è una quintupla $\langle Q, \Sigma, \delta, q_0, F \rangle$ dove Q, Σ, q_0 e $F \subseteq Q$ mantengono il significato visto per gli automi deterministici, mentre la *funzione di transizione* δ è ora definita

$$\delta : Q \times \Sigma \to \wp(Q).$$

Si osservi che ora è ammesso: $\delta(q, a) = \emptyset$ per qualche $q \in Q$ ed $a \in \Sigma$.
Anche per gli NFA dalla funzione δ si ottiene in modo univoco la funzione $\hat{\delta} : Q \times \Sigma^* \to \wp(Q)$ nel modo seguente:

$$\begin{cases} \hat{\delta}(q, \varepsilon) = \{q\} \\ \hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q,w)} \delta(p, a) \end{cases}$$

Una stringa x è detta *essere accettata* da un NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ se $\hat{\delta}(q_0, x) \cap F \neq \emptyset$. Il *linguaggio accettato da M* è l'insieme delle stringhe accettate, ovvero:

$$L(M) = \left\{ x \in \Sigma^* : \hat{\delta}(q_0, x) \cap F \neq \emptyset \right\}$$

Mentre la rappresentazione mediante tabella degli automi non deterministici è profondamente diversa da quella per i deterministici (in ogni casella si deve inserire ora un insieme di stati), la rappresentazione a grafo rimane pressoché immutata. L'unica differenza è che da un nodo possono uscire più archi (o nessuno) etichettati dallo stesso simbolo.
Dal punto di vista invece del modello con testina e nastro, il non determinismo va immaginato come la possibilità *contemporanea* di procedere la computazione in ognuno degli stati raggiunti. Si apre dunque un gran numero di computazioni virtuali parallele.

5. Equivalenza tra DFA e NFA

In questo paragrafo si mostrerà che i linguaggi accettati dai DFA e dagli NFA coincidono. Poiché un DFA si può vedere come un NFA in cui $\delta(q, a)$ restituisce sempre insiemi costituiti da un solo stato (detti anche *singoletti*), si ha che ogni linguaggio regolare è un linguaggio accettato da un qualche NFA. Per l'implicazione inversa ci serve il seguente teorema:

TEOREMA 3.10 (Rabin-Scott, 1959). Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un NFA. Allora esiste un DFA M' tale che $L(M) = L(M')$.

PROOF. Definisco $M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$ come segue:

- $\Sigma' = \Sigma$;
- $Q' = \wp(Q)$ (sarebbe più preciso definire $Q' = \{q_1, \ldots, q_{2|Q|}\}$ e poi stabilire una corrispondenza biunivoca fra tali stati e gli elementi di $\wp(Q)$. Tuttavia, con tale abuso sintattico la dimostrazione diventa molto più snella);
- $q'_0 = \{q_0\}$;
- $F' = \{P \subseteq Q : P \cap F \neq \emptyset\}$;

- $\delta'(P,a) = \bigcup_{p \in P} \delta(p,a)$ per $P \in \wp(Q)$.

Mostriamo per induzione sulla lunghezza della stringa di input *x* che

$$\hat{\delta}(q_0, x) = \hat{\delta}'(q'_0, x)$$

Base: Per $|x| = 0$ il risultato è banale, poiché $q'_0 = \{q_0\}$ e $x = \varepsilon$.

Passo: Supponiamo che l'ipotesi induttiva valga per tutte le stringhe *x* tali che $|x| \leq m$. Sia *xa* una stringa di lunghezza $m + 1$. Allora:

$$\begin{aligned}
\hat{\delta}'(q'_0, xa) &= \delta'(\hat{\delta}'(q'_0, x), a) && \text{Def. di } \hat{\delta}' \\
&= \delta'(\hat{\delta}'(q_0, x), a) && \text{Ip. ind.} \\
&= \bigcup_{p \in \hat{\delta}(q_0, x)} \delta(p, a) && \text{Def. di } \delta' \\
&= \hat{\delta}(q_0, xa) && \text{Def. di } \hat{\delta}
\end{aligned}$$

Il teorema segue dal fatto che $x \in L(M)$ sse $\hat{\delta}(q_0, x) \cap F \neq \emptyset$ sse $\hat{\delta}'(q'_0, x) \cap F \neq \emptyset$ sse $\hat{\delta}'(q'_0, x) \in F'$ sse $x \in L(M')$. ف

6. Automi con ε-transizioni

In questo paragrafo sarà presentato un terzo tipo di automa che estende il modello non-deterministico ma che, come sarà mostrato nel Teorema 3.12, ne è equivalente dal punto di vista dei linguaggi accettati.

Un NFA con *ε-transizioni* è una quintupla $\langle Q, \Sigma, \delta, q_0, F \rangle$ dove Q, Σ, q_0 e $F \subseteq Q$ sono come per gli automi non deterministici, mentre la *funzione di transizione* δ è ora definita

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \wp(Q).$$

L'idea è che da uno stato è permesso passare ad un altro stato anche senza "leggere" caratteri di input.

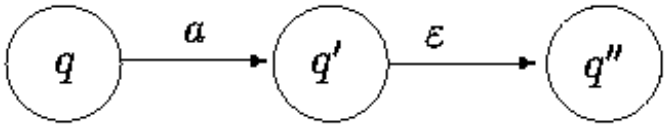
La costruzione della funzione $\hat{\delta} : Q \times \Sigma^* \to \wp(Q)$ nel caso dei *ε-NFA* risulta leggermente più complessa che nei casi precedenti. Per far ciò si introduce la funzione *ε-closure* che, applicataad uno stato, restituisce l'insieme degli stati raggiungibili da esso (compreso sé stesso) mediante ε-transizioni. La costruzione di tale funzione è equivalente a quella che permette di conoscere i nodi raggiungibili da un nodo in un grafo e può facilmente essere calcolata a partire dalla funzione δ (un arco $p \to q$ si ha quando $q \in \delta(p, \varepsilon)$). Il concetto di ε-closure si estende in modo intuitivo ad insiemi di stati:

$$\varepsilon - closure(p) = \bigcup_{p \in P} \varepsilon - closure(p)$$

$\hat{\delta}$ si può ora definire nel modo seguente:

$$\begin{cases} \hat{\delta}(q, \varepsilon) = \varepsilon - closure(q) \\ \hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q,w)} \varepsilon - closure(\delta(p, a)) \end{cases}$$

Si noti che in questo caso $\hat{\delta}(q,a)$ può essere diverso da $\delta(q,a)$. Ad esempio, nell'automa:



Si ha che $\delta(q,a)=\{q'\}$, mentre
$\hat{\delta}(q,a)=\bigcup_{p\in\hat{\delta}(q,\varepsilon)}\varepsilon-closure(\delta(p,a))=\{q',q''\}$.
Definiamo dunque il *linguaggio accettato* dall'automa
L(*M*) = $\{x\in\Sigma^*:\hat{\delta}(q_0,x)\cap F\neq\emptyset\}$.

7. Equivalenza di ε-NFA e NFA

Ogni NFA è, per definizione, un caso particolare di un ε-NFA. Con il seguente teorema mostreremo che la classe dei linguaggi riconosciuti dagli automi ε-NFA non estende propriamente quella dei linguaggi riconosciuti da automi NFA:

TEOREMA 3.12. Sia *M* = $\langle Q,\Sigma,\delta,q_0,F\rangle$ un *ε-NFA*. Allora esiste un *NFA M'* tale che *L*(*M*) = *L*(*M'*).

PROOF. Definisco *M'* = $\langle Q',\Sigma',\delta',q'_0,F'\rangle$ come segue:

- $Q'=Q$,
- $\Sigma'=\Sigma$,
- $q'_0=q_0$
- $F'=\begin{cases}F\cup\{q_0\} & \text{se } \varepsilon\text{-}closure(q_0)\cap F\neq\emptyset\\ F & \text{altrimenti}\end{cases}$
- $\delta'(q,a)=\hat{\delta}(q,a)$.

Dobbiamo mostrare che

$$\hat{\delta}(q_0,x)\cap F\neq\emptyset \quad \text{sse} \quad \hat{\delta}'(q'_0,x)\cap F'\neq\emptyset \qquad (1)$$

Se *x* = ε, si ha che: $\hat{\delta}(q_0,\varepsilon)=\varepsilon-closure(q_0)$ per definizione.

D'altro canto: $\hat{\delta}'(q_0,\varepsilon)=\{q_0\}$. Per definizione di *F'*, si ha che $q_0\in F'$ sse $\varepsilon-closure(q_0)\neq 0$.

Mostreremo per induzione su |*x*| ≥ 1 che $\hat{\delta}'(q'_0,x)=\hat{\delta}(q_0,x)$.

Base: |*x*| = 1. Allora *x* = *a* per qualche simbolo $a\in\Sigma$. Ma allora
$\hat{\delta}'(q'_0,a)=\delta'(q'_0,a)=\hat{\delta}(q_0,a)$ per definizione.

Passo: Assumiamo che la tesi valga per tutte le stringhe *x* tali che 1 ≤ |*x*| ≤ *m*. Sia *xa* una stringa di lunghezza *m* + 1. Allora:

$\hat{\delta}'(q'_0,xa)$ $=\bigcup_{p\in\hat{\delta}'(q'_0,x)}\delta'(p,a)$ Def. di $\hat{\delta}'$

$= \bigcup_{p \in \hat{\delta}'(q'_0, x)} \hat{\delta}(p, a)$ Def. di δ'

$= \bigcup_{p \in \hat{\delta}(q_0, x)} \hat{\delta}(p, a)$ Ip. ind.

$= \bigcup_{p \in \hat{\delta}(q_0, x)} \varepsilon - closure(\delta(p, a))$ Def. di $\hat{\delta}$ e (*)

$= \hat{\delta}(q_0, xa)$ Def. di $\hat{\delta}$

(*) si osservi che per definizione di $\hat{\delta}$, la ε-closure(*p*) è contenuta in $\hat{\delta}(q_0, x)$.

Per concludere la dimostrazione, si deve mostrare la proprietà (1), ovvero che $\hat{\delta}'(q'_0, x)$ contiene uno stato di *F*' sse $\hat{\delta}(q_0, xa)$ contiene uno stato di *F*.

Poiché $F \subseteq F'$ e $\hat{\delta}'(q'_0, x) = \hat{\delta}(q_0, x)$ l'implicazione (←) deriva immediatamente.
Per l'altra implicazione, l'unico problema si avrebbe nel caso:

1. $q_0 \in \hat{\delta}'(q'_0, x)$,
2. $q'_0 \in F'$,
3. $q_0 \notin F$.

Poiché $\hat{\delta}'(q'_0, x) = \hat{\delta}(q_0, x)$, si ha che $q_0 \in \hat{\delta}(q_0, x)$. Ma, per definizione di $\hat{\delta}$, anche ogni elemento della sua ε-closure appartiene a $\hat{\delta}(q_0, x)$.

Tale ε-closure, poiché $q_0 \in F'$, interseca *F*. ڤ

COROLLARIO 3.13. Le classi di linguaggi riconosciute da DFA, NFA e ε-NFA coincidono (linguaggi regolari).

PROOF. Immediato dai Teoremi 3.10 e 3.12. ڤ

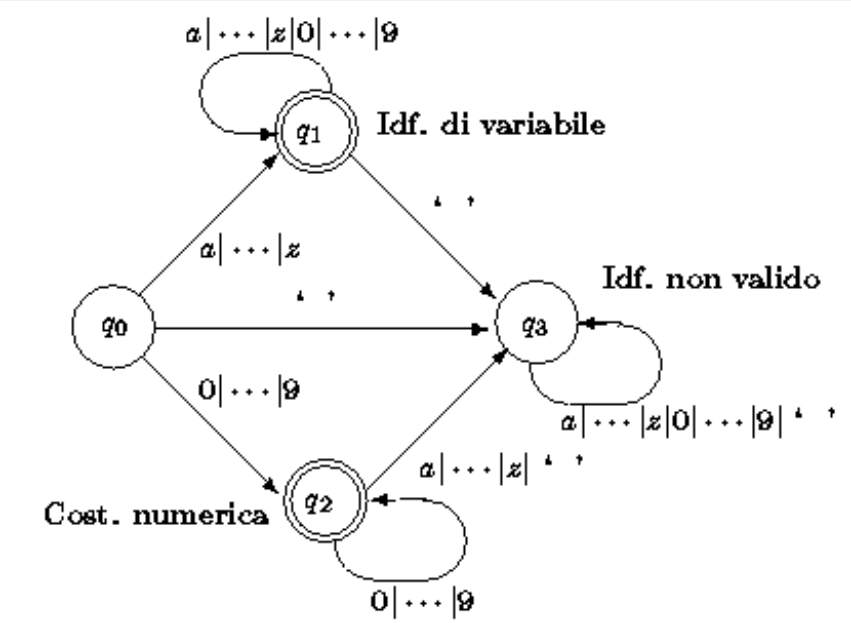
8. Automi con output

Gli automi visti fin qui, tutti equivalenti dal punto di vista della classe di linguaggi accettati (riconosciuti) non sono in grado di fornire alcun tipo di output se non il messaggio (booleano) del raggiungimento di uno stato finale o meno. Esistono in letteratura due tipi di automi provvisti di output: le macchine di *Moore* e quelle di *Mealy*. Tali *macchine* sono automi a stati finiti deterministici (e dunque ereditano la classe di linguaggi accettati) che possiedono un alfabeto di output:

- le macchine di Moore forniscono in output un simbolo in funzione di ogni stato raggiunto. Poiché per una stringa di *n* elementi si raggiungono al più *n* + 1 stati (compreso quello iniziale), si può ottenere un output di (al più) *n* + 1 caratteri.
- le macchine di Mealy forniscono in output un simbolo ogni volta che avviene una transizione $\delta(q, a)$. In questo caso al più *n* caratteri saranno forniti in output.

Gli output possono essere utili per differenziare stati finali. Così, ad esempio, una stringa accettata può anche avere una più precisa classificazione (si veda l'Esempio 3.14).
Le macchine di Moore e Mealy vengono impiegate per la sintesi di reti sequenziali e sono formalismi tra loro equivalenti.

ESEMPIO 3.14. Il seguente automa è una macchina di Moore che, oltre a riconoscere se una stringa sia o meno un elemento di una espressione numerica, segnala se si tratta di un numero intero o di un identificatore di variabile. L'output è rappresentato dalle scritte (Idf. di variabile, ecc.) posizionate accanto ai vari stati.



Si osservi come esso racchiuda in sè due DFA privi di output.

Capitolo

4 Espressioni regolari

Introduzione

In questo capitolo verrà illustrato un modo alternativo per descrivere i linguaggi accettati da automi finiti (linguaggi regolari).

1. Operazioni sui linguaggi

Sia Σ un alfabeto e L, L_1, L_2 insiemi di stringhe di Σ^*. La *concatenazione* di L_1 e L_2, denotata come L_1L_2 è l'insieme:

$$L_1L_2 = \{xy \in \Sigma^* : x \in L_1, y \in L_2\}.$$

Definiamo ora

$$\begin{cases} L^0 = \{\varepsilon\} \\ L^{i+1} = LL^i \end{cases}$$

Si osservi che $L^0 = \{\varepsilon\} \neq \emptyset$. La *chiusura* (*di Kleene*) di L, denotata come L^* è l'insieme:

$$L^* = \bigcup_{i \geq 0} L^i$$

mentre la *chiusura positiva di L*, denotata come L^+ è l'insieme: $L^+ = \bigcup_{i \geq 1} L^i$ (si verifica immediatamente che $L^+ = LL^*$, dunque tale operatore, seppur comodo, non è essenziale). Si osservi come la definizione sia consistente con la nozione di Σ^* fin qui adoperata.

2. Definizione formale

Sia Σ un alfabeto. Le *espressioni regolari su* Σ e gli *insiemi* che esse denotano sono definiti ricorsivamente nel modo seguente:

1. $\emptyset$ è una espressione regolare che denota l'insieme vuoto.
2. ε è una espressione regolare che denota l'insieme $\{ \varepsilon \}$.
3. Per ogni simbolo $a \in \Sigma$, a è una espressione regolare che denota l'insieme $\{a\}$.
4. Se r e s sono espressioni regolari denotanti rispettivamente gli insiemi R ed S, allora $(r + s)$, (rs), e (r^*) sono espressioni regolari che denotano gli insiemi $R \cup S$, RS, R^* rispettivamente.

Se r è una espressione regolare, indicheremo con $L(r)$ il linguaggio denotato da r. Tra espressioni regolari valgono delle uguaglianze che permettono la loro manipolazione algebrica. Varrà che $r = s$ se e solo se $L(r) = L(s)$.

NOTAZIONE 4.1. Qualora non si crei ambiguità, nello scrivere espressioni regolari saranno omesse le parentesi. Le precedenze degli operatori sono le stesse dell'algebra, intepretando * come esponente e la concatenazione come prodotto (si veda Esercizio 4.2). E' ammesso usare l'abbreviazione r^+ per l'espressione regolare rr^*.

3. Equivalenza tra DFA e ER

TEOREMA 4.3 (McNaughton,Yamada, 1960). Sia r una espressione regolare. Allora esiste un ε-*NFA M* tale che $L(M) = L(r)$.

PROOF. Costruiremo un ε-NFA siffatto, con un unico stato finale, per induzione sulla complessità strutturale dell'espressione regolare *r*.

Base: Ci sono tre casi base:

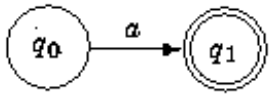
- l'automa:

  riconosce il linguaggio {ε};

- l'automa

  riconosce il linguaggio $\emptyset$;

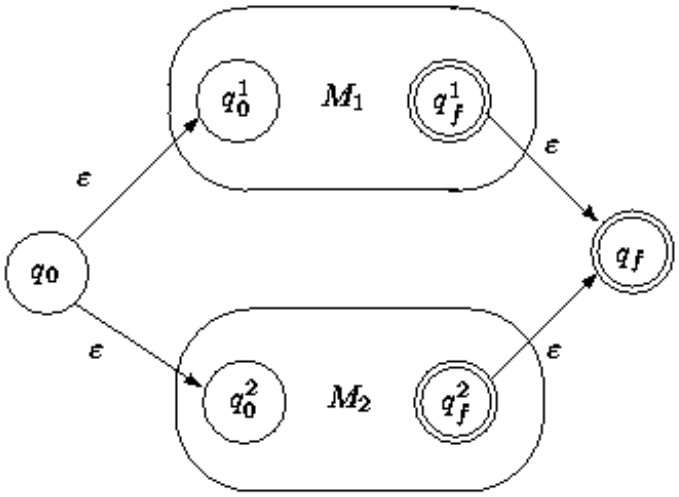
- l'automa



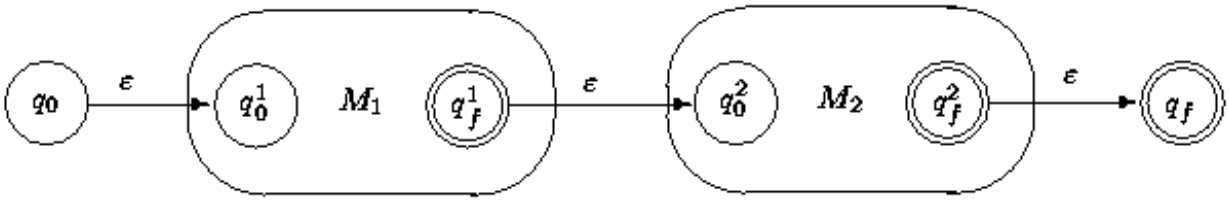
  riconosce il linguaggio $\{a\}$.

Passo: Anche qui abbiamo tre casi da analizzare:

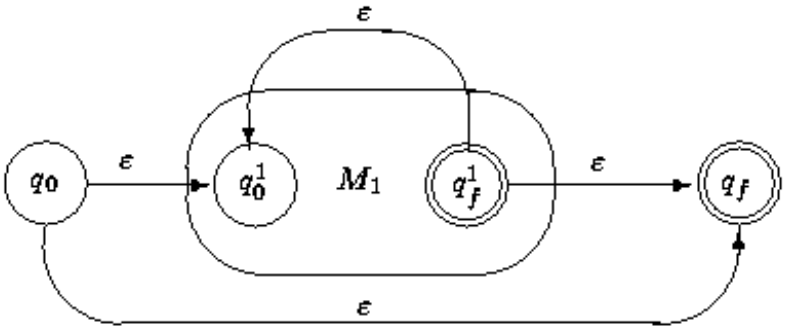
- $r = r_1 + r_2$. Per $i = 1, 2$, sia M_i , con stato iniziale q_0^i e stato finale q_f^i l'automa che riconosce $L(r_i)$. L'esistenza di tali automi è assicurata dall'ipotesi induttiva. Il seguente automa, con stato iniziale q_0 e stato finale q_f riconosce il linguaggio $L(r) = L(r_1) \bigcup L(r_2)$:



- $r = r_1 r_2$. Per i = 1, 2, sia M_i , con stato iniziale q_0^i e stato finale q_f^i l'automa che riconosce $L(r_i)$. L'esistenza di tali automi è assicurata dall'ipotesi induttiva. Il seguente automa, con stato iniziale q_0 e stato finale q_f riconosce il linguaggio $L(r_1)L(r_2)$:



- $r = r_1^*$. Sia M_1, con stato iniziale q_0^1 e stato finale q_f^1 l'automa che riconosce $L(r_1)$. L'esistenza di tali automi è assicurata dall'ipotesi induttiva. Il seguente automa, con stato iniziale q_0 e stato finale q_f riconosce il linguaggio $L(r) = (L(r_1))^*$:



Le dimostrazioni che tali automi riconoscono esattamente i linguaggi a loro assegnati è lasciata per esercizio. ڤ

Si vuole ora mostrare il contrario, ovvero che preso a caso un automa M (senza perdita di generalità è sufficiente considerare un DFA), il linguaggio accettato da M è denotato da una espressione regolare. Si darà una dimostrazione costruttiva di questo fatto.

TEOREMA 4.4. Sia M un *DFA*. Allora esiste una espressione regolare r tale che $L(M) = L(r)$.

PROOF. (Traccia) Si costruirà, a partire da M = $\langle Q, \Sigma, \delta, q_0, F \rangle$ un sistema di equazioni in cui ad ogni stato q_i viene associata una variabile Q_i. Si fornirà una metodologia per calcolare

una soluzione a tale sistema. La soluzione calcolata assegnerà una espressione regolare r_i ad ogni variabile Q_i, che ne denota il linguaggio. Se F = $\{q_{j_1},\ldots,q_{j_p}\}$, avremo che:

$$r = r_{j_1} + \ldots + r_{j_p}$$

e $L(r)$ è il linguaggio accettato dall'automa.
Per ogni $q_i \in Q$, siano:

- $q_{i_1},\ldots,q_{i_h}$ gli stati diversi da q_i tali che esiste $a_{i_j} \in \Sigma$ t.c. $\delta(q_{i_j}, a_{i_j}) = q_i$;
- $b_{i_1},\ldots,b_{i_k}$ i simboli tali che $\delta(q_i, b_{i_j}) = q_i$.

Assegnamo a q_i l'equazione:

$$Q_i = (Q_{i_1} a_{i_1} + \ldots + Q_{i_h} a_{i_h})(b_{i_1} + \ldots + b_{i_k})^*$$

Nel caso che q_i fosse q_0 bisogna aggiungere $+\varepsilon$ alla parte destra.
Un tale sistema si può risolvere mediante manipolazioni algebriche sulle espressioni regolari presenti ed usando il metodo di *sostituzione* (standard) qualora (come accade all'inizio) Q_i non compaia nella parte destra dell'equazione che lo definisce.
Se non fosse così, poichè le variabili Q_j compaiono sempre all'inizio delle espressioni presenti nelle parti destre delle equazioni, ci si riesce a ricondurre, mediante manipolazioni algebriche, ad una equazione della forma:

$$Q_i = Q_i s_1 + s_2$$

per opportuni s_1 ed s_2 in cui Q_i non compare. La soluzione di questa equazione è $Q_i = s_2 s_1^*$; infatti:

$$(s_2 s_1^*)s_1 + s_2 = s_2 s_1^+ + s_2 = s_2(s_1^+ + \varepsilon) = s_2 s_1^*$$

Si applichi dunque la sostituzione $Q_i = s_2 s_1^*$ al resto del sistema. ف

Capitolo

5 Proprietà dei linguaggi regolari

Introduzione

Dai Capitoli 2 e 4 si è imparato che se un linguaggio è riconosciuto da un automa a stati finiti (equivalentemente, DFA, NFA, ε-NFA) allora è un linguaggio regolare. Argomenti di questo capitolo saranno i principali risultati e metodi che saranno utili per stabilire se un dato linguaggio sia o meno regolare.

1. Il "Pumping Lemma"

Il primo metodo che proponiamo e che viene spesso usato per mostrare che un linguaggio non è regolare deriva dal seguente risultato noto come *Pumping Lemma*:

LEMMA 5.1 (Bar-Hillel, Perles, Shamir, 1961). Sia L un linguaggio regolare. Allora esiste una costante $n \in N$ tale che per ogni $z \in L$ z 2 L, se $|z| \geq n$, esistono tre stringhe *u*, *v*, *w* tali che:
1. $z = uvw$,
2. $|uv| \leq n$,
3. $|v| > 0$, e
4. per ogni i ≥ 0 vale che $uv^i w \in L$.

PROOF. Sia $M = \langle \{q_0,\ldots,q_{n-1}\}, \Sigma, \delta, q_0, F \rangle$ DFA tale che $L = L(M)$ (esiste poiché *L* è regolare). Sia $z = a_1 \ldots a_m$, $m \geq n$, $z \in L$ (si noti l'arbitrarietà della scelta di *z*; se una tale *z* non esistesse, il lemma varrebbe banalmente). Per $i = 1,\ldots, n$ ($n \geq m$) si consideri l'evoluzione degli stati $\hat{\delta}(q_0, a_1 \ldots a_i)$. In tal modo, considerando anche lo stato iniziale q_0, si raggiungono in tutto $n + 1$ stati. Poiché gli stati dell'automa sono *n* per ipotesi, esiste (almeno) uno stato $\overline{q}$ raggiunto (almeno) due volte.

Dunque avremo una situazione del tipo

$$\hat{\delta}(q_0, a_1 \ldots a_{i_1}) = \overline{q}$$
$$= \hat{\delta}(q_0, a_1 \ldots a_{i_1} \ldots a_{i_2}) \quad i_2 > i_1$$

A questo punto, si prenda $u = a_1 \ldots a_{i_1}$, $v = a_{i_1+1} \ldots a_{i_2}$, $w = a_{i_2+1} \ldots a_m$. E' immediato, usando il Lemma 3.8, mostrare che $\hat{\delta}(q_0, z) = \hat{\delta}(q_0, uv^i w)$ per ogni $i \geq 0$. Per costruzione si ha inoltre che: $|uv| \leq n$. ف

COROLLARIO 5.2. *n* del Lemma 5.1 può essere preso come il numero di stati del più piccolo automa riconoscente *L*.

Proof. Immediato dalla dim. del Lemma. ف

Si osservi come il lemma asserisca la veridicità di una formula logica quantificata non banale, ovvero, per ogni linguaggio L, se L è regolare, allora vale:

$$(1.1)\quad \exists n \in \mathbb{N}\ \forall z \left(\begin{array}{l} (z \in L \wedge |z| \geq n) \rightarrow \exists u, v, w \\ \left(\begin{array}{l} z = uvw \wedge |uv| \leq n \wedge |v| > 0 \wedge \\ \forall i \, (i \in \mathbb{N} \rightarrow uv^i w \in L) \end{array} \right) \end{array} \right)$$

Il Pumping lemma viene largamente usato per mostrare che un dato linguaggio non è regolare (qualora non lo sia!). Per mostrare ciò, si assume che L non sia regolare e si deve mostrare che vale la negazione della formula (1.1), ovvero (si veda il Capitolo 3 per le tecniche di complementazione di una formula logica):

$$(1.2)\ \forall n \in \mathbb{N}\ \exists z \left(\begin{array}{l} z \in L \wedge |z| \geq n \wedge \\ \forall u, v, w \, (z = uvw \wedge |uv| \leq n \wedge |v| > 0 \\ \quad \rightarrow \exists i \, (i \in \mathbb{N} \wedge uv^i w \notin L) \end{array} \right)$$

Nel prossimo esempio si illustrerà un corretto uso di questa tecnica, che dato l'annidamento di quantificazioni, può indurre ad errore.

ESEMPIO 5.3. Il linguaggio $L = \{0^i 1^i : i \geq 0\}$ non è regolare.

Supponiamo, per assurdo, sia regolare e cerchiamo di verificare la formula (1.2). Si prenda un numero naturale n arbitrario e si scelga (questo è il punto in cui bisogna avere la "fortuna" di scegliere una stringa "giusta" tra tutte quelle di lunghezza maggiore o uguale a n) la stringa $z = 0^n 1^n$. Per tale stringa vale che $z \in L$ e $|z| \geq$ n.

Rimane da dimostrare che comunque si suddivida la stringa z in sottostringhe uvw (in modo tale cioè che $z = uvw$) tali da soddisfare i vincoli $|uv| \leq n$ e $|v| > 0$ esiste almeno un $i > 0$ tale per cui $uv^i w \notin L$.

I modi per partizionare z sono molti (per esercizio, si provi a pensare quanti) però, il fatto che $|uv| \leq n$ permette di ricondursi al seguente unico "schema di suddivisione":

$u = 0^a$,

$v = 0^b$,

$w = 0^c 1^n$,

con $a + b + c = n$, $b > 0$. "Pompando" v si ottengono stringhe al di fuori del linguaggio, da cui l'assurdo. Ad esempio, con $i = 0$ vale che $uv^0w = 0^a 0^c 1^n \notin$ L, poichè $a + c < n$.

2. Proprietà di chiusura

La definizione di linguaggio è una definizione di tipo insiemistico ($L \subseteq \Sigma^*$). Pertanto è lecito parlare di unione, intersezione, complementazione (etc.) di linguaggi. In questo paragrafo mostreremo che la proprietà di essere un linguaggio regolare si conserva sotto queste usuali operazioni.

TEOREMA 5.5. I linguaggi regolari sono chiusi rispetto alle operazioni di unione, concatenazione e chiusura di Kleene.

PROOF. Immediato dalla definizione di espressione regolare e dai Teoremi 4.3 e 4.4.

ف

TEOREMA 5.6. I linguaggi regolari sono chiusi rispetto alla operazione di complementazione. Ovvero, se $L \subseteq \Sigma^*$ è regolare, anche $\overline{L} = \Sigma^* \setminus L$ è regolare.

PROOF. Sia $M = \langle Q, \Sigma', \delta, q_0, F \rangle$ il DFA che riconosce L. Per il Lemma 3.9, possiamo assumere $\Sigma' = \Sigma$. Allora, banalmente, $M' = \langle Q, \Sigma, \delta, q_0, Q \setminus F \rangle$ riconosce $\overline{L}$.

ف

COROLLARIO 5.7. I linguaggi regolari sono chiusi rispetto all'intersezione.

PROOF. Immediato dal fatto che $L_1 \cap L_2 = \overline{(\overline{L}_1 \cup \overline{L}_2)}$. ف

3. Risultati di decidibilità

In questo paragrafo saranno mostrati alcuni risultati relativi al problema di effettuare delle decisioni riguardanti linguaggi regolari.

TEOREMA 5.10 (Vuoto-infinito). L'insieme delle stringhe accettate da un DFA M con n stati è:
1. non vuoto se e solo se accetta una stringa di lunghezza inferiore a n;
2. infinito se e solo se l'automa accetta una stringa di lunghezza l, $n \le l < 2n$

PROOF.
1. (←) Se accetta una stringa (di lunghezza di inferiore a n) allora è banalmente non vuoto.
(→) Supponiamo sia non vuoto. Allora, per definizione M accetta almeno una stringa z, $|z| = m$. Se $m < n$, la tesi è provata. Altrimenti come conseguenza del pumping lemma, $z = uvw$ con $|v| \ge 1$, e $uv^0w = uw$ è accettata da M . Se $|uw| < n$ la tesi è provata, altrimenti si proceda iterativamente ripartendo con $z = uw$ (che ha lunghezza strettamente minore a m). Dopo al più $m - n$ iterazioni si otterrà una stringa di lunghezza inferiore a n;

2. (←) Supponiamo M accetti una stringa z di lunghezza l, $n \le l < 2n$. Allora, per il pumping Lemma, $z = uvw$, $|v| \ge 1$ e $\{uv^iw : i \in \mathrm{N}\} \subseteq L(M)$. Ma $\{uv^iw : i \in \mathrm{N}\}$ è un insieme infinito.
(→) Sia $L(M)$ infinito. Allora esiste $z \in L(M)$ tale che $|z| = m \ge 2n$. Per il pumping Lemma $z = uvw$ con $|uv| \le n$ e $|v| \ge 1$ (e dunque $|uw| \ge n$). Per il pumping Lemma, $z' = uw \in L(M)$.
Se $|z'| < 2n$, allora la tesi è dimostrata. Altrimenti, si reiteri il procedimento partendo dalla stringa (più corta) $z' = uw$. In un numero finito di passi si trova la stringa cercata.

ف

COROLLARIO 5.11. Sia M DFA. Allora i problemi '$L(M) = \emptyset$' e '$L(M)$ è infinito' sono entrambi decidibili (cioè esiste una procedura effettiva per determinare la loro veridicità o falsità).

TEOREMA 5.12 (Equivalenza). Dati due DFA M_1 e M_2 , il problema di stabilire se $L(M_1) = L(M_2)$ è decidibile.

PROOF. $L(M_1) = L(M_2)$ è insiemisticamente equivalente a

$$(L(M_1) \cap \overline{L(M_2)}) \cup (\overline{L(M_1)} \cap L(M_2)) = \emptyset$$

Dai Teoremi 5.5 e 5.6 sappiamo esistere un automa M_3 tale che

$$L(M_3) = (L(M_1) \cap \overline{L(M_2)}) \cup (\overline{L(M_1)} \cap L(M_2))$$

Il risultato segue dalla decidibilità del problema del vuoto (Corollario 5.11). ف

Capitolo

6 Grammatiche libere dal contesto

Introduzione

Una grammatica è, intuitivamente, un insieme di regole che permettono di generare un linguaggio. Un ruolo fondamentale tra le grammatiche è costituito dalle grammatiche libere dal contesto mediante le quali vengono solitamente descritti i linguaggi di programmazione.

1. Definizione formale

Una grammatica libera dal contesto *(CF, context free)* è una quadrupla $G = <V, T, P, S>$ ove:

- V è un insieme finito di variabili (dette anche simboli non terminali),
- T è un insieme finito di simboli terminali ($V \cap T = \varnothing$),
- P è un insieme finito di produzioni; ogni produzione è della forma $A \rightarrow \alpha$ ove: $A \in V$ è una variabile, e $\alpha \in (V \cap T)$.
- $S \in V$ è una variabile speciale, detta simbolo iniziale.

ESEMPIO 6.1: Sia G la grammatica seguente:

$$G = \{ \{ E \}, \{ or, and, not, (,), 0, 1 \}, P, E \}$$

ove P è costituito da:

$E \rightarrow 0$

$E \rightarrow 1$

$E \rightarrow (E \text{ or } E)$

$E \rightarrow (E \text{ and } E)$

$E \rightarrow (\text{not } E)$

Come vedremo, G genererà le possibili espressioni booleane.

NOTAZIONE 6.2*:* Per le grammatiche saranno utilizzate le seguenti notazioni: lettere maiuscole *A, B, C, D, E, S* denotano variabili, *S* (a meno che non sia esplicitamente detto il contrario) il simbolo iniziale. *A, b, c, d, e, 0, 1* denotano simboli terminali; *X, Y, Z* denotano simboli che possono essere sia terminali che variabili; *u, v, w, x, y, z* denotano stringhe di terminali, $\alpha, \beta, \gamma, \delta$ stringhe generiche di simboli (sia terminali che non). Se $A \to \alpha_1 , \ldots , A \to \alpha_n \in P$ esprimeremo questo fatto scrivendo: $A \to \alpha_1 \mid \ldots \mid \alpha_n$. Con *A-produzione* denoteremo una generica produzione con la variabile *A* a sinistra. La grammatica dell'esempio 6.1 può essere dunque schematicamente rappresentata da:

$$E \to 0 \mid 1 \mid (E \text{ or } E) \mid (E \text{ and } E) \mid (\text{not } E).$$

2. Linguaggio generato

Le grammatiche servono a generare un linguaggio. Daremo ora le definizioni necessarie a definire il linguaggio generato da una grammatica $G = <V, T, P, S>$. Definiremo le relazioni $\overset{G}{\Longrightarrow}, \overset{G}{\Longrightarrow}_i, \overset{G}{\Longrightarrow}_* \subseteq (V \cup T)^* \times (V \cup T)^*$ nel seguente modo:

- se $A \to \beta \in P$ e $\alpha, \gamma \in (V \cup T)^*$, *allora* $A\gamma \overset{G}{\Longrightarrow} \alpha\beta\gamma$. Diremo in questo caso che da $\alpha A \gamma$ deriva immediatamente $\alpha\beta\gamma$

- se $\alpha_1, \ldots, \alpha_i \in (V \cup T)^*$, $i \geq 1$, e

$$\bigwedge_{j=1}^{i-1} \alpha_j \overset{G}{\Longrightarrow} \alpha_{j+1}$$

allora $\alpha_1 \overset{G}{\Longrightarrow}_i \alpha_m$. In questo caso diremo che da α_1 *deriva* α_m in *i* passi;

- se esiste *i* tale per cui $\alpha_1 \overset{G}{\Longrightarrow}_i \alpha_m$, allora $\alpha_1 \overset{G}{\Longrightarrow}_* \alpha_m$. Diremo in tal caso che da α_1 *deriva* α_m.

Si noti che vale $\alpha \overset{G}{\Longrightarrow}_* \alpha$ e $\alpha \overset{G}{\Longrightarrow}_0 \alpha$ per ogni $\alpha \in (V \cup T)^*$.

Il *linguaggio generato da G* è:

$$L(G) = \{ w \in T^* : S \overset{G}{\Longrightarrow}_* w \}$$

L è un linguaggio libero dal contesto (CF) se esiste una grammatica CF G tale che L = L(G). Due grammatiche G_1 e G_2 sono equivalenti se $L(G_1) = L(G_2)$.

ESEMPIO 6.2: Σ^* è un linguaggio CF. Infatti, sia $\Sigma = \{s1, \ldots, sn\}$, allora Σ^* è generato da:

$$S \to \varepsilon \mid s1S \mid \ldots \mid s1S$$

ESEMPIO 6.3: La grammatica schematicamente definita da: $S \rightarrow ASB \mid \varepsilon$, $A \rightarrow 0$, $B \rightarrow 1$ genera il linguaggio

$$\{ 0^n 1^n : n \geq 0 \}$$

Per quanto dimostrato nell'esempio 5.3 usando il *pumping lemma* (che vedremo più avanti), il linguaggio $\{ 0^n 1^n : n \geq 0 \}$ non è un linguaggio regolare. Pertanto l'insieme dei linguaggi regolari e quello dei linguaggi CF non sono lo stesso insieme. Vedremo, comunque, che il primo insieme è incluso nel secondo.

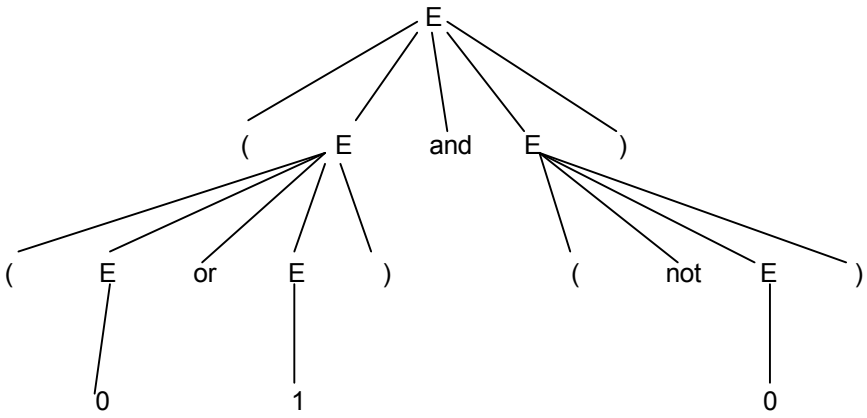
3. Alberi di derivazione

Le derivazioni generate da una grammatica vengono ben visualizzate mediante l'ausilio di strutture dati ad albero.
Sia $G = <V, T, P, S>$. Si una grammatica CF. Un albero è un *albero di derivazione (parse tree) per* G se:

1. ogni vertice ha una etichetta, presa tra $V \cup T \cup \{ \varepsilon \}$ l'etichetta della radice appartiene a V ;
2. se un vertice interno (ovvero, non una foglia) ha etichetta A, allora $A \in V$;
3. se un vertice n è etichettato con A e $n_1, ..., n_k$ sono (ordinatamente, da sinistra a destra) i vertici gli di A etichettati con $X_1, ..., X_k$, allora
$$A \rightarrow X_1, ..., X_k \in P$$
4. se un vertice n ha etichetta ε, allora n è una foglia ed è l'unico figlio di suo padre.

Si noti che la definizione di albero non impone che tutte le foglie siano etichettate con simboli terminali. L'albero avente un solo nodo etichettato da S è un caso particolare di albero di derivazione. Gli alberi verranno utilizzati per dare una controparte grafica delle derivazioni. Se vi sono foglie etichettate con simboli non terminali, allora l'albero rappresenta una derivazione parziale. Diremo che un albero descrive una stringa $\alpha \in (V \cup T)^*$ se α è proprio la stringa che possiamo leggere dalle etichette delle foglie da sinistra a destra.

ESEMPIO 6.4: La grammatica dell'esempio 6.1 genera, in particolare, la stringa w = ((0 or 1) and (not 0)). La derivazione per w è rappresentata dall'albero seguente (che descrive w):



Dato un albero ed un suo nodo *n*, il *sottoalbero* identificato da quel nodo è costituito da quel nodo più tutti i suoi discendenti (gli, nipoti, ecc.), nonchè le varie etichette associate ad essi.

TEOREMA 6.1. Sia G = <V, T, P, S> una grammatica CF. Allora $S \overset{G}{\Longrightarrow}_* \alpha$ *se e solo se esiste un albero di derivazione con radice etichettata S per G che descrive* α.

PROOF. Si proverà un enunciato più forte, ovvero: dato un qualunque simbolo non terminale A $\in$ V, $A \overset{G}{\Longrightarrow}_* \alpha$ se e solo se esiste un albero di derivazione per *G* che descrive la cui radice è etichettata A.

(→) Proviamo l'enunciato per induzione sul numero i di passi di derivazione per $A \overset{G}{\Longrightarrow}_* \alpha$

Base: $i = 0$ $A \overset{G}{\Longrightarrow}_0 \alpha$ implica, per definizione, che α = A. Ma l'albero con unico nodo (radice) etichettato con A è un albero di derivazione (parziale) che descrive A stesso.

Passo: Supponiamo $A \overset{G}{\Longrightarrow}_i \beta_1 B \beta_3 \overset{G}{\Longrightarrow}_1 \beta_1 \beta_2 \beta_3$ (con $\beta_1\beta_2\beta_3 = \alpha$). Per ipotesi induttiva esiste un albero di derivazione T con radice etichettata A che descrive $\beta_1 B \beta_3$. Ma per definizione, $\beta_1 B \beta_3 \overset{G}{\Longrightarrow}_1 \beta_1\beta_2\beta_3$ se e solo se esiste la produzione $B \rightarrow \beta_2$ in P . Ma allora, applicando la regola (4) di definizione di albero, dall'albero T si ottiene l'albero che soddisfa i requisiti.

(←) Proviamo l'enunciato per induzione sul numero di nodi interni dell'albero di derivazione etichettato in *A* che descrive α .

4. Ambiguità delle derivazioni

Se ad ogni passo di una derivazione la produzione è applicata al simbolo non terminale più a sinistra, allora la derivazione è detta sinistra (leftmost). Similmente, se viene applicata sempre a quello a destra, allora è detta destra (rightmost). Se $w \in L(G)$, dal Teorema 6.7 sappiamo che per essa esiste almeno un albero di derivazione con radice etichettata S. Si può dimostrare che, fissato un albero di derivazione con radice etichettata S, esattamente una derivazione sinistra (suggerimento: si visiti l'albero in preordine) ed una derivazione destra (simmetricamente) possono essere desunte.

Un albero di derivazione rappresenta dunque un certo insieme di possibili derivazioni distinte di una stessa stringa. Ci possono tuttavia essere più alberi di derivazione per la stessa parola:

ESEMPIO 6.5. Si consideri la grammatica

$E \rightarrow E + E \mid E * E \mid 0 \mid 1 \mid 2$

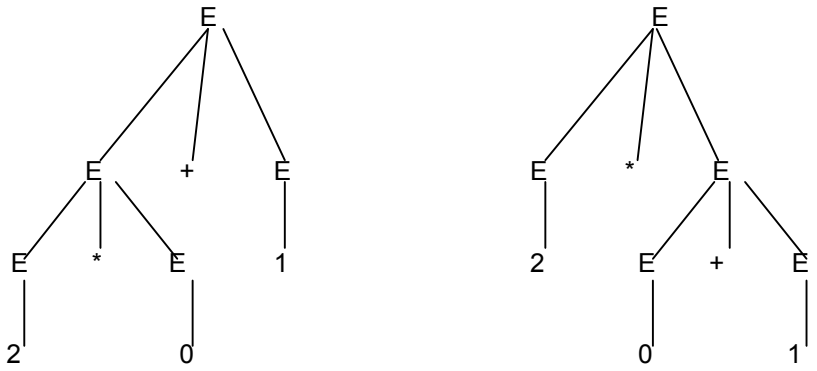
Una derivazione sinistra è:

$$E \overset{G}{\Longrightarrow} E + E \overset{G}{\Longrightarrow} E * E + E \overset{G}{\Longrightarrow}_3 2 * 0 + 1$$

Un'altra derivazione sinistra per la stessa stringa è:

$$E \overset{G}{\Longrightarrow} E*E \overset{G}{\Longrightarrow} 2*E+E \overset{G}{\Longrightarrow}_2 2*0+1$$

Tuttavia gli alberi associati alle due derivazioni (che potrebbero essere visti come alberi per la computazione dell'espressione associata alla stringa generata) sono diversi:



Il primo è intuitivamente associato a (2 * 0) + 1 = 1 il secondo, invece, 2 * (0 + 1) = 2.

Una grammatica CF tale per cui esiste una parola con più di un albero di derivazione con radice etichettata S è detta *ambigua*. Un linguaggio CF per cui ogni grammatica che lo genera è ambigua è detto essere *inerentemente ambiguo*.

Un esempio di tali linguaggi è:

$$L = \{a^n b^n c^m d^m : n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n : n \geq 1, m \geq 1\}$$

5. Semplificazione

Vi sono molti modi per restringere la forma delle produzioni permesse alle grammatiche CF senza alterare la classe dei linguaggi riconosciuti dall'insieme delle possibili grammatiche. In questo paragrafo si presenteranno due di queste trasformazioni, che conducono alla forma normale di Chomsky e a quella di Greibach.
Mostreremo che ogni linguaggio CF può essere generato da una grammatica G dalle seguenti due proprietà:

1. ogni variabile e ogni simbolo terminale di G compaiono in almeno una derivazione di una qualche parola di L;
2. non ci sono produzioni della forma $A \to B$ con A e B variabili.

Se, inoltre, $\varepsilon \notin L$, non avremo bisogno di produzioni $A \to \varepsilon$; sempre sotto tale ipotesi potremo richiedere che ogni produzione sia della forma $A \to BC$ o $A \to a$ (Chomsky) oppure, alternativamente, che ogni produzione sia della forma $A \to a\alpha$ ove è una stringa di variabili (Greibach).

6. Forma normale di Chomsky

TEOREMA 6.2 (Chomsky, 1959). Ogni linguaggio CF senza ε è generato da una grammatica in cui tutte le produzioni sono della forma $A \to BC$ e $A \to a$.

PROOF. Partendo da una grammatica $G = <V, T, P, S>$ senza ε produzioni e senza produzioni unitarie (ossia produzioni nella forma $A \to B$, con A e B variabili).

Sia $A \to X_1, \ldots, X_m \in P$, m $\geq$ 1 (non vi sono ε-produzioni):

- se m = 1, allora, poichè non vi sono produzioni unitarie, $X_1 \in$ T: è già nella forma cercata;
- se m = 2 e $X_1, X_2 \in$ V, siamo nella forma cercata;
- se m $\geq$ 2 e qualcuno degli $X_i \in$ T, allora per ogni $X_i \in$ T introduco in V una nuova variabile, diciamo B_i, aggiungo la produzione $B_i \to X_i$ e rimpiazzo la produzione $A \to X_1, \ldots, X_m$ con $A \to Y1, \ldots, Ym$, ove Yi = Xi se Xi $\in$ V , Yi = B_i altrimenti;
- se m > 2 e tutti gli X_i sono variabili, allora rimpiazzo la produzione $A \to X_1, \ldots, X_m$ con le produzioni $A \to BX_3, \ldots X_m$, $B \to X_1X_2$, ove B è una nuova variabile da aggiungere a V .

E' immediato verificare che il procedimento termina e che l'insieme finale di produzioni (P') è nella forma normale di Chomsky.

Per accertare l'equivalenza delle due grammatiche è sufficiente verificare che la grammatica $G' = <V', T, P', S>$ (ove V' si ottiene da V con l'aggiunta delle nuove variabili) ottenuta è equivalente a quella di partenza. Ovvero, per A $\in$ V :

$$A \overset{G}{\Longrightarrow} * w \text{ sse } A \overset{G'}{\Longrightarrow} * w$$

7. Forma normale di Greibach

Per raggiungere questa forma normale abbiamo bisogno di due lemmi preliminari.

LEMMA 6.1 (Unfolding). Sia $G = <V, T, P, S>$ una grammatica CF, sia $A \to \alpha_1 B \alpha_2$ una produzione di P e $B \to \beta_1 | \ldots | \beta_2$ l'insieme delle B-produzioni. Sia $G' = <V, T, P', S>$ ottenuta da G eliminando tutte le produzioni $A \to \alpha_1 B \alpha_2$ da P e aggiungendo le produzioni $A \to \alpha_1 B \alpha_2 | \ldots | \alpha_1 B \alpha_2$. Allora $L(G) = L(G')$.

LEMMA 6.2 (Eliminazione ricorsione sinistra). Sia $G = <V, T, P, S>$ una grammatica CF e sia $A \to A\alpha_1 | \ldots | A\alpha_m$ l'insieme delle A-produzioni di G per cui A è anche il simbolo più a sinistra della stringa di destra delle produzioni. Siano $A \to \beta_1 | \ldots | \beta_n$ le rimanenti A-produzioni. Sia $G' = <V \cup \{ B \}, T, P', S>$ la grammatica CF ottenuta aggiungendo una nuova variabile B a V e rimpiazzando tutte le A-produzioni con:

1. $A \to \beta_i | \beta_i$ per i $\in \{ 1, \ldots, n \}$;
2. $B \to \alpha_i | \alpha_i$ per i $\in \{ 1, \ldots, m \}$;

Allora $L(G') = L(G)$.

PROOF. Prima o poi, la sequenza di derivazioni:

$$A \overset{G}{\Longrightarrow} A\alpha_{i1} \overset{G}{\Longrightarrow} A\alpha_{i2}\alpha_{i1} \overset{G}{\Longrightarrow} \ldots$$

deve terminare con un'applicazione del tipo $A \overset{G}{\Longrightarrow} \beta_j$, altrimenti non si giungerà mai ad una stringa di terminali. Pertanto avremo una situazione del tipo:

$$A \overset{G}{\Longrightarrow}{}^{*} \beta_j \alpha_{in} \ \ldots \ \alpha_{i2} \alpha_{i1}$$

La stessa situazione può essere ottenuta mediante:

$$A \overset{G}{\Longrightarrow} \beta_j B \overset{G}{\Longrightarrow} \beta_j \alpha_{in} \overset{G}{\Longrightarrow}{}^{*} \beta_j \alpha_{in} \ldots \alpha_{i2} \alpha_{i1}$$

TEOREMA 6.3 (Greibach, 1965). Ogni linguaggio CF senza ε è generato da una grammatica in cui tutte le produzioni sono della forma $A \rightarrow a\alpha$, ove α è una stringa (eventualmente vuota) di simboli non terminali.

PROOF. Sia G una grammatica in forma normale di Chomsky. Sia $V = \{A_1, \ldots, A_m\}$. Si costruirà in tre passi una grammatica G' in forma normale di Greibach equivalente a G.

1. Si applichi il seguente algoritmo:

for k := 1 to m do

begin

for j := 1 to k - 1 do

elimina le produzioni $A_k \rightarrow A_i\alpha$ mediante unfolding

elimina le prod. $A_k \rightarrow A_k\alpha$ mediante elim. ricors. sinistra

end;

Nuove variabili $B_1, \ldots, B_h$ sono state introdotte dalla fase di eliminazione della ricorsione sinistra. L'equivalenza tra G e la grammatica ottenuta deriva dai lemmi appena dimostrati.

A questo punto tutte le produzioni sono della forma:

$$\begin{array}{lll} A_i & \rightarrow & A_j\alpha \qquad j > i \\ A_i & \rightarrow & \alpha\beta \qquad a \in T \\ B_i & \rightarrow & \gamma \end{array}$$

Inoltre, poichè siamo partiti da una grammatica in forma normale di Chomsky, si osserva che ha un prefisso della forma $A_{i1} \ \ldots \ A_{ih}$ seguito da un simbolo non terminale, seguito ancora da una sequenza eventualmente vuota di variabili. Per lo stesso motivo, β è una stringa di variabili. Per quanto riguarda le stringhe γ, esse iniziano con un terminale oppure con una coppia di $A_i A_j$, mai con un B_i .

2. Pertanto, ogni produzione $A_m \rightarrow \alpha$ ha la parte destra iniziante con un simbolo terminale. Posso effettuare l'unfolding delle A i -produzioni rimpiazzando A_m con ciascuna delle sue possibili parti destre. Posso iterare questo procedimento all'indietro, ottenendo produzioni della forma:

$$A_i \ \rightarrow \ \alpha\beta \qquad a \in T$$

per ogni i = 1, …, m, con stringa di variabili.

3. Si tratta ora di semplificare le produzioni $B_i \rightarrow \alpha..$. Applicando la sostituzione sul primo (eventuale) simbolo A_i della stringa, si giunge al risultato.

Un importante caso particolare di grammatiche in forma normale di Greibach sono le grammatiche CF in cui le produzioni hanno tutte la forma $A \rightarrow a$ oppure $A \rightarrow aB$. Tali grammatiche, che vedremo nei prossimi capitoli, sono grammatiche lineari destre ed hanno l'importante caratteristica di generare esattamente i linguaggi regolari.

Capitolo

7 Proprietà dei linguaggi liberi dal contesto

Introduzione

Per i linguaggi CF valgono delle proprietà simili a quelle studiate per i linguaggi regolari nel Capitolo 5.

1. Il pumping lemma per i linguaggi CF

LEMMA 7.1. (Bar-Hillel, Perles, Shamir, 1961). Sia L un linguaggio CF. Allora c'è una costante $n \in N$ (dipendente dal linguaggio L) tale che per ogni $z \in L$, $|z| \geq n$, esistono stringhe uvwxy tali che

1. *$z = uvwxy$,*
2. *$|vx| \geq 1$,*
3. *$|vwx| \leq n$, e*
4. *per ogni $i \geq 0$ vale che $uv^i wx^i y \in L$.*

PROOF. Sia $G = <V, T, P, S>$ la grammatica che genera $L \setminus \{ \varepsilon \}$.
Senza perdita di generalità, possiamo assumere che G sia in forma normale di Chomsky (se $\varepsilon \in$ L, potremmo costruire una nuova grammatica G' che estende G con un nuovo simbolo iniziale S' e le produzioni S' $\rightarrow \varepsilon$ e S' $\rightarrow$ S).

ESEMPIO 7.1. Il linguaggio $\{a^i b^i c^i : i \geq 1\}$ non è CF. Supponiamo per assurdo che lo sia. Dobbiamo mostrare la veridicità del complemento della formula dimostrata nel lemma, ovvero che per ogni $n \geq 1$ sappiamo trovare una stringa $z \in L$, $|z| \geq n$ che soddisfa una certa condizione. Scegliamo $z = a^n b^n c^n$. Dobbiamo mostrare che per ogni quintupla di stringhe u, v, w, x, y tali che $a^n b^n c^n = uvwxy$, $|vx| \geq 1$ e $|vwx| \leq n$ esiste un $i \geq 0$ tale per cui $uv^i wx^i y \notin L$.

Vi sono molti modi di partizionare z in uvwxy che soddisfano i requisiti. Tuttavia questi si possono raggruppare nelle seguenti casistiche:

$uvwx = a^h$ con $h \leq n$

$u = a^h$, $vwx = a^k b^m$ con $k > 0$, $k + m \leq n$

$u = a^n b^h$, $vwx = b^m$ con $m \leq n$

$u = a^n b^h$, $vwx = b^k c^m$ con $k + m \leq n$

$u = a^n b^n c^h$, $vwx = c^m$ con $m \leq n$

In ciascuna delle casistiche, tuttavia, si ha che $uv^0\, wx^0\, y = uwy \notin L$, in quanto vengono alterati al massimo due tra i numeri di ripetizioni dei tre tipi di carattere a, b e c.

2. Proprietà di chiusura

TEOREMA 7.2: I linguaggi CF sono chiusi rispetto all'unione, alla concatenazione, ed alla chiusura di Kleene.

PROOF. La dimostrazione è costruttiva. Siano $G_1 = <V_1, T_1, P_1, S_1>$ e $G_2 = <V_2, T_2, P_2, S_2>$ le grammatiche generanti i linguaggi L_1 e L_2 . Per semplicità si assuma che V_1 e V_2 siano disgiunti (altrimenti si proceda a ridenominare le variabili). Sia S una nuova variabile.

Allora

- $G_\cup = \langle V_1 \cup V_2 \cup \{ S \}, T_1 \cup T_2, P_1 \cup P_2 \cup P, S\rangle$, *ove P è:* $S \to S_1 \mid S_2$
  è la grammatica che genera $L_1 \cup L_2$.

- $G_o = \langle V_1 \cup V_2 \cup \{ S \}, T_1 \cup T_2, P_1 \cup P_2 \cup P, S\rangle$, *ove P è:* $S \to S_1S_2$
  è la grammatica che genera L_1L_2.

- $G_* = \langle V_1 \cup \{ S \}, T_1, P_1 \cup P, S\rangle$, *ove P è:* $S \to \varepsilon \mid S_1S$
  è la grammatica che genera L_1*.

ڤ

TEOREMA 7.3. I linguaggi CF non sono chiusi rispetto all'intersezione.

PROOF. Consideriamo $L_1 = \{a^i b^i c^j : i \geq 1; j \geq 1\}$ generato da:

$$S \to RC; R \to ab \mid aRb,\ C \to c \mid cC$$

e

$L_2 = \{a^i b^j c^j : i \geq 1; j \geq 1\}$ *generato da:* $S \to AR,\ R \to bc \mid bRc,\ C \to c \mid cC$

La loro intersezione è $\{a^i b^i c^i : i \geq 1\}$, mostrato essere non CF nell'esempio 7.2. ڤ

COROLLARIO 7.4. I linguaggi CF non sono chiusi rispetto alla complementazione.

Proof. Se lo fossero, allora lo sarebbero anche rispetto all'intersezione, in quanto

$A \cap B = \overline{\overline{A} \cup \overline{B}}$, contraddicendo il Teorema 7.3. ڤ

TEOREMA 7.5. Se L è un linguaggio CF e L' è un linguaggio regolare, allora $L \cap L'$ è un linguaggio CF.

3. Algoritmi di decisione

TEOREMA 7.6 (Vuoto, Finito, Infinto). Data una grammatica CF G = <V, T, P, S> problemi:

1. *L(G) = ∅*
2. *L(G) è finito*
3. *L(G) è infinito*

sono decidibili.

PROOF. Sia ora $L(G) \neq \varnothing$ e $G' = <V', T, P', S>$ una grammatica equivalente a G in forma normale di Chomsky e priva di simboli inutili.

Creiamo un grafo ζ avente:

- un nodo per ogni variabile $A \in V'$;
- un arco $\langle A, B \rangle$ ed un arco $\langle A, C \rangle$ per ogni produzione $A \rightarrow BC \in P'$.

Allora L(G) (= L(G')) è finito se e solo se ζ non ha cicli.
Se ζ non ha cicli, la finitezza è immediata: solo un numero finito di alberi di derivazione con radice etichettata da S possono essere costruiti.
Viceversa, supponiamo ζ abbia (almeno) un ciclo passante per un nodo associato ad una variabile A. Allora riusciamo a costruire un albero con un cammino dalla radice verso le foglie (qui è importante l'ipotesi che la grammatica non abbia simboli inutili) che attraversa due nodi etichettati con A. Ripetendo la dimostrazione del pumping lemma, si riescono a generare infiniti alberi di derivazione diversi. ڤ

THEOREMA 7.7 (Appartenenza). Data una grammatica CF $G = <V, T, P, S>$ e una stringa z, il problema $z \in L(G)$ è decidibile.

PROOF. Per il caso $z = \varepsilon$, si ha che $\varepsilon \in$ L sse esiste $S \rightarrow A_1 \ldots A_n$ ($n \geq 0$) in P tale per cui $A_i \in N$ per i = 1, …, n.

Sia $z \neq \varepsilon$ e $G' = <V', T, P', S>$ la grammatica equivalente a G in forma normale di Greibach. Allora, se z ha una derivazione, ne ha una di esattamente |z| passi. Generiamo esaustivamente tutte le derivazioni di |z| passi e verifichiamo se esiste una di queste che deriva z. ڤ

L'algoritmo impiegato nella dimostrazione del Teorema 7.7 ha complessità esponeziale. Fortunatamente per chi necessita di riconoscere o refutare stringhe (ad esempio, per capire se un programma Pascal o C contiene degli errori sintattici) può avvalersi di metodi più efficienti. In particolare, l'algoritmo di Cocke-Younger-Kasami (1965-67) oppure l'algoritmo di Valiant (1975).
Si osservi che nel caso di linguaggi regolari invece tale problema viene deciso in tempo O(n). Infatti, dato il DFA che accetta il linguaggio, la computazione avviene in tempo proporzionale alla lunghezza della stringa.
Per quanto riguarda il problema dell'equivalenza di due linguaggi liberi dal contesto, il problema è mostrato essere indecidibile.
Il problema corrispondente nel caso dei linguaggi regolari è invece decidibile.

Capitolo

8 Le grammatiche regolari

In questo breve capitolo si studieranno due famiglie di grammatiche CF che generano esattamente i linguaggi regolari.
Una grammatica CF si dice lineare destra se ogni produzione è della forma:

$$\begin{array}{llll} A & \rightarrow & wB & w \in T^*, \text{ oppure} \\ A & \rightarrow & w & w \in T^* \end{array}$$

più eventualmente $S \rightarrow \varepsilon$. Se invece tutte le produzioni sono della forma:

$$\begin{array}{llll} A & \rightarrow & Bw & w \in T^*, \text{ oppure} \\ A & \rightarrow & w & w \in T^* \end{array}$$

più eventualmente $S \rightarrow \varepsilon$, si dice lineare sinistra.

LEMMA 8.1. Data una grammatica lineare destra G esiste una grammatica G' equivalente (in forma normale di Greibach) tale che tutte le produzioni sono della forma:

$$\begin{array}{llll} A & \rightarrow & aB & a \in T^*, \text{ oppure} \\ A & \rightarrow & a & a \in T^* \end{array}$$

più eventualmente $S \rightarrow \varepsilon$.

PROOF. Ogni produzione della forma $A \rightarrow a_1a_2 \ldots a_nB$ viene sostituita dalle produzioni: $A \rightarrow a_1C_1$, $C_1 \rightarrow a_2C_2, \ldots, C_n \rightarrow a_2C_2, \ldots, C_{n-1} \rightarrow a_n$

THEOREMA 8.1. Se L è generato da una grammatica lineare destra, allora L è un linguaggio regolare.

PROOF. Grazie al Lemma 8.1 possiamo supporre che $G = \langle V, T, P, S \rangle$ tale che L(G) = L sia nella forma semplificata descritta nell'enunciato di tale lemma. Costruiremo un NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ che riconosce L.

- $Q = V \cup \{\perp\}$;
- $\Sigma = T$;
- $B \in \delta(A, a)$ sse $A \rightarrow aB \in P$, $\perp \in \delta(A, a)$ sse $A \rightarrow a \in P$;
- $q_0 = S$;
- $$F = \begin{cases} \{\perp, S\} & \text{se } S \rightarrow \varepsilon \in P \\ \{\perp\} & \text{altrimenti} \end{cases}$$

Si deve mostrare che $\hat{\delta}(S, w) \cap F \neq \varnothing$ se e solo se $S \overset{G}{\Longrightarrow}{}^* w$, per ogni stringa w.
Iniziamo mostrando, per induzione su $|w| \geq 0$, che per ogni $A \in V$ (dunque $A \neq \perp$)

$$A \in \hat{\delta}(S, w) \text{ sse } S \overset{G}{\Longrightarrow}{}^* wA$$

Base: Sia $w = \varepsilon$. Per definizione, $\hat{\delta}(S, w) = \{S\}$. Per la forma della grammatica $S \overset{G}{\Longrightarrow}{}^{*} \varepsilon A$ sse $S = A$.

Passo: Sia $w = va$.

$A \in \hat{\delta}(S, va)$	sse $A \in \bigcup_{B \in} \hat{\delta}_{(S, v)}\ \delta(B, a)$	def di $\hat{\delta}$
	sse $A \in \bigcup_{B : S \overset{G}{\Longrightarrow}{}^{*} vB}\ \delta(B, a)$	ipotesi induttiva
	sse $S \overset{G}{\Longrightarrow}{}^{*} vaA$	def di δ

Da questo risultato, per definizione di F , si ha che:

1. $S \overset{G}{\Longrightarrow}{}^{*} \varepsilon$ se e solo se $\hat{\delta}(S, \varepsilon) \cap F \neq \varnothing$
2. $S \overset{G}{\Longrightarrow}{}^{*} va$ se e solo se esiste A tale che $S \overset{G}{\Longrightarrow}{}^{*} vA$ *e* $A \to a \in P$.

   Ciò accade se e solo se $A \in \hat{\delta}(S, v)$ *e* $\perp \in \delta(A, a)$ ovvero $\perp \in \hat{\delta}(S, va)$ ف

TEOREMA 8.2. Se L è un linguaggio regolare, allora L è generato da una grammatica lineare destra.

PROOF. Sia $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ il DFA che riconosce L. Costruiamo una grammatica lineare destra $G = \langle V, T, P, S\rangle$ tale che L(G) = L nel modo seguente:

- $V = Q$;
- $T = \Sigma$;
- $F = \begin{cases} A \to aB \in P \text{ sse } \delta(A, a) = B \\ A \to \varepsilon \in P \text{ sse } A \in F \end{cases}$
- $S = q_0$;

Mostriamo che $\hat{\delta}(q_0, w) = p$ se e solo se $q_0 \overset{G}{\Longrightarrow}_{|w|} wp$ per induzione su |w|. Il risultato segue dalle ε-transizioni per gli stati di F . ف

Il Teorema 8.2 implica che ogni linguaggio regolare è anche CF.

Capitolo

9 La gerarchia di Chomsky

Introduzione

In questo capitolo riassumeremo brevemente dei risultati che permettono di classificare le grammatiche in base all'espressività dei linguaggi generabili. Questa classificazione divide le grammatiche in 4 tipi (partendo da 0). Le grammatiche di tipo 2 e 3, rispettivamente quelle libere dal contesto (CF) e regolari sono già state viste nei precedenti capitoli.

1. Grammatiche di tipo 0

Le grammatiche di tipo 0 (*a struttura di frase*) sono le grammatiche della forma $G = <V, T, P, S>$ in cui P contiene produzioni

$$\alpha \to \beta$$

ove $\alpha \in (V \cup T)^+$, $\beta \in (V \cup T)^*$.

TEOREMA 9.1. L = L(G) per G grammatica a struttura di frase se e solo se L è un insieme ricorsivamente enumerabile.

2. Grammatiche di tipo 1

Le grammatiche di tipo 1 (dipendenti dal contesto) sono le grammatiche della forma $G = <V, T, P, S>$ in cui P contiene produzioni

$$\alpha \to \beta$$

ove $\alpha \in (V \cup T)^+$, $\beta \in (V \cup T)^*$, e $|\alpha| \leq |\beta|$.

Per queste grammatiche esiste una forma normale per le produzioni, che devono essere del tipo:

$$\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$$

con $\beta \neq \varepsilon$.

TEOREMA 9.2. Ogni linguaggio L generato da una grammatica dipendente dal contesto è ricorsivo.
Tuttavia:

TEOREMA 9.3. Ci sono insiemi ricorsivi che non sono generati da nessuna grammatica dipendente dal contesto.

Nell'Esempio 7.1 abbiamo mostrato che fa $\{a^i b^i c^j : i \geq 1\}$ non è un linguaggio CF. Mostreremo ora che tale linguaggio è un linguaggio dipendente dal contesto.

ESEMPIO 9.1. La grammatica di tipo 1:

$S \rightarrow aSBC \mid aBC$

$CB \rightarrow BC$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

$aB \rightarrow ab$

genera esattamente il linguaggio $\{a^i b^i c^j : i \geq 1\}$.

3. Gerarchia

Ricordiamo:

- dagli esempi 5.3 e 6.4 sappiamo che il linguaggio fa $\{a^i b^i : i \geq 0\}$ è un linguaggio CF ma non regolare. Sappiamo inoltre che le grammatiche (CF) lineari destre e sinistre generano esattamente i linguaggi regolari.
- Dagli esempi 7.2 e 9.4 sappiamo che il linguaggio $\{a^i b^i c^j : i \geq 1\}$ è un linguaggio dipendente dal contesto ma non CF.
- Dai Teoremi 9.3 e 9.1 sappiamo che la classe di linguaggi generati da grammatiche dipendenti dal contesto è inclusa strettamente nella classe di linguaggi generati da grammatiche a struttura di frase.

Pertanto viene indotta una gerarchia di grammatiche, nota come *gerarchia di Chomsky*, fatta di inclusioni proprie, che si può trovare nella seguente figura.