Arturo Carpi

CORSO DI LAUREA IN INFORMATICA – A.A. 2022/23

Linguaggi Formali e Compilatori

Dispense del corso

Dipartimento di Matematica e Informatica
Università degli Studi di Perugia

Indice

versione: 25 maggio 2023

Parte I

Generalità

Capitolo 1

Le fasi della compilazione

Un programma, scritto in un linguaggio di alto livello come, ad esempio, C, Pascal, Lisp, Prolog eccetera, per poter essere eseguito deve essere *tradotto* in linguaggio macchina (il *codice oggetto*).

Tale operazione è eseguita da un apposito programma chiamato *compilatore* o *traduttore*.

Il processo di compilazione di un programma si può suddividere in varie fasi (vedi Fig. 1.1).

Innanzitutto possiamo dividere tale processo in una *fase analitica* in cui il programma sorgente viene esaminato per verificare se soddisfa le regole sintattiche e semantiche del linguaggio di programmazione e in una *fase sintetica* in cui viene generato il corrispondente codice oggetto.

La *gestione dell'input* ha il compito di leggere i caratteri che costituiscono il codice sorgente, interagendo col sistema operativo.

Nella fase di analisi possiamo distinguere *analisi lessicale*, *analisi sintattica* e *analisi semantica statica*.

L'analisi lessicale ha il compito di decomporre il codice sorgente, che si presenta come una sequenza di caratteri, in gruppi che logicamente rappresentano gli elementi atomici del linguaggio (*tokens*) come identificatori, costanti numeriche, operatori, parole chiave, eccetera.

Consideriamo, ad esempio, l'istruzione

```
WHILE A > 3 * B DO A := A - 1;
```

del linguaggio Pascal. L'analizzatore lessicale (o *scanner*) decodificherà questa istruzione nella sequenza di tokens

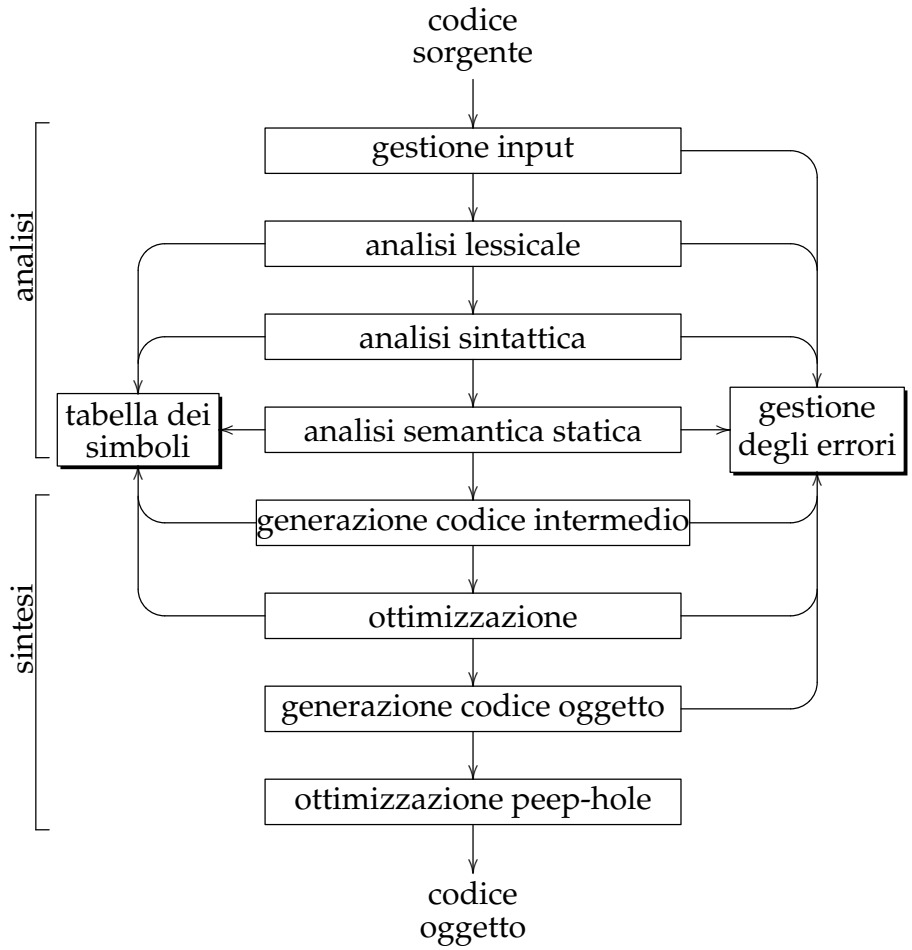


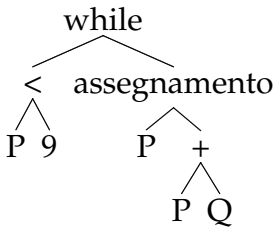
Figura 1.1: Fasi della Compilazione

`WHILE`	parola chiave	
`A`	identificatore	nome: A
`>`	operatore	confronto
`3`	costante	valore: 3
`*`	operatore	moltiplicazione
`B`	identificatore	nome: B
`DO`	parola chiave	
`A`	identificatore	nome: A
`:=`	operatore	assegnamento
`A`	identificatore	nome: A
`-`	operatore	sottrazione
`1`	constante	valore: 1
`;`	separatore	

L'analisi sintattica (o *parsing*) dovrà raggruppare i tokens prodotti dallo scanner in strutture sintattiche, scomponendo espressioni e costrutti nei loro componenti logici. L'analisi sintattica produce il cosiddetto *albero sintattico astratto*, una rappresentazione del programma che risulterà di grande utilità nella generazione del codice oggetto. Per esempio, il costrutto

```
WHILE (P < 9) P = P + Q;
```

produrrà l'albero sintattico astratto



L'analisi sintattica è spesso combinata con l'analisi semantica statica che ha il compito di verificare che le componenti della struttura sintattica soddisfino il tipo o lo scopo imposto dal contesto in cui sono analizzate. Per esempio, si consideri il costrutto

```
WHILE Espressione DO SequenzaComandi
```

del Pascal. Il linguaggio richiede che il valore di `Espressione` sia di tipo Boolean. Pertanto se `Espressione` corrispondesse a un'espressione o funzione di tipo differente, il compilatore dovrebbe rilevare l'errore.

Alla *fase analitica* segue la *fase sintetica*, in cui viene generato il codice oggetto. Il primo passo è in genere costituito dalla *generazione del codice intermedio*. Le

strutture dati prodotte nella fase analitica vengono utilizzate per generare uno schema di programma, o un codice ASSEMBLER, o anche solo un programma in un differente linguaggio di alto livello che sarà poi compilato separatamente. In ogni caso, il codice intermedio, a differenza del codice macchina, non specifica nel dettaglio gli indirizzi in memoria nè i registri da utilizzare per i calcoli e quindi resta, in larga parte, indipendente dalla macchina.

Per esempio, dal costrutto

```
WHILE (1<P && P < 9) P = P + Q;
```

si potrebbe ottenere un codice intermedio equivalente a

```
L0 if 1<P goto L1
   goto L3
L1 if P<9 goto L2
   goto L3
L2 P := P + Q
   goto L0
L3 ...
```

A questo punto interviene l'*ottimizzatore* che cerca di ridurre il tempo o lo spazio necessario all'esecuzione del codice intermedio. Per esempio il codice precedente potrebbe diventare

```
L0 if 1>=P goto L3
   if P>=9 goto L3
   P := P + Q
   goto L0
L3 ...
```

La parte più complessa è la *generazione del codice oggetto*. Qui si devono fissare le locazioni di memoria dei dati, generare il codice per accedere a tali dati, selezionare i registri per i calcoli intermedi e gli indirizzamenti, e così via. A differenza delle fasi precedenti, che sono in larga parte indipendenti dalla macchina su cui il programma sarà eseguito, in questa fase è necessario tener conto dell'architettura della piattaforma. È anche possibile che a questa fase segua un'ulteriore fase di ottimizzazione, detta ottimizzazione *peep-hole*, in cui si esaminano in dettaglio brevi sequenze di istruzioni consecutive, con l'obbiettivo di ridurre ulteriormente il numero delle operazioni da eseguire.

La realizzazione di un compilatore richiede l'uso di una struttura dati detta *tabella dei simboli* che tiene traccia dei nomi usati dal programma per individuare le variabili, le costanti, le funzioni insieme alle loro proprietà, come il tipo, la dimensione dello spazio occupato, il valore delle costanti.

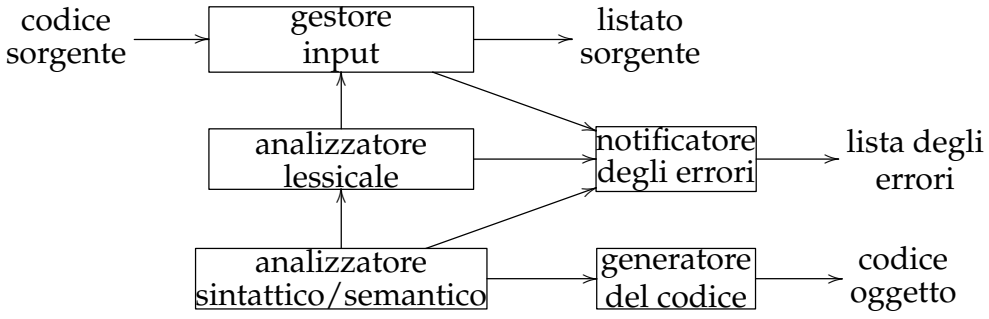


Figura 1.2: Organizzazione del compilatore

È anche necessario un gestore degli errori. Esso ha il compito di informare l'utente degli errori presenti nel programma e, nei limiti del possibile, di permettere il proseguimento della compilazione. Infatti è opportuno che la compilazione possa proseguire anche dopo l'individuazione di un errore, in modo da permettere all'utente di eseguire più correzioni in una sola volta.

In realtà, nei casi più frequenti, le varie fasi della compilazione non sono separate nel tempo (vedi Fig. 1.2). Un tipico compilatore ha come nucleo l'analizzatore sintattico (che realizza anche l'analisi semantica statica). L'analisi lessicale viene eseguita mediante la chiamata, da parte dell'analizzatore sintattico, di una funzione prossimoToken, implementata nell'analizzatore lessicale. A sua volta, questa funzione utilizza le funzioni lettura da archivio che realizzano la gestione dell'input. È ancora l'analizzatore sintattico ad attivare, mediante la chiamata di funzioni opportune, la generazione del codice.

Capitolo 2

Parole e linguaggi

In questa sezione introdurremo i principali oggetti che si utilizzano nella Teoria dei Linguaggi Formali.

Definizione 1 Chiameremo *alfabeto* un insieme finito non vuoto Σ di simboli. I suoi elementi sono detti *lettere*.

Per esempio, possiamo considerare gli alfabeti

$$\Sigma_0 = \{a, b\}\,, \quad \Sigma_1 = \{0, 1\}\,, \quad \Sigma_2 = \{a, b, c\}\,,$$

$$\Sigma_3 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}\,.$$

Definizione 2 Ogni sequenza finita di lettere di Σ si dice *parola* sull'alfabeto Σ. L'insieme delle parole sull'alfabeto Σ sarà denotato con Σ^*.

Per esempio, *a*, *abb*, *ababbabb* sono parole sull'alfabeto $\Sigma = \{a, b\}$. Come si è visto, una parola sull'alfabeto Σ è una sequenza

$$u = a_1 a_2 \cdots a_k$$

con $k \geq 0$, $a_1, a_2, \ldots, a_k \in \Sigma$. L'intero k si dice *lunghezza* della parola u. La parola di lunghezza 0 si denota con ε e viene detta *parola vuota*.

Definizione 3 Si considerino le parole $u = a_1 a_2 \cdots a_k$ e $v = b_1 b_2 \cdots b_h$ ($k, h \geq 0$, $a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_h \in \Sigma$). La *concatenazione* di u e v è la parola

$$uv = a_1 a_2 \cdots a_k b_1 b_2 \cdots b_h\,.$$

Per esempio la concatenazione delle parole $u = abb$ e $v = aaab$ è la parola $uv = abbaaab$.

La concatenazione è dunque un'operazione binaria su Σ^*. Si verifica facilmente che tale operazione gode della proprietà associativa, cioè risulta

$$(uv)w = u(vw)$$

per ogni $u, v, w \in \Sigma^*$. Inoltre la parola vuota funge da elemento neutro, cioè si ha $\varepsilon u = u\varepsilon = u$ per ogni $u \in \Sigma^*$. Pertanto, con l'operazione di concatenazione, l'insieme Σ^* assume la struttura di *monoide*.

Un'altra notevole proprietà della concatenazione è la *cancellatività* a destra e a sinistra: se si ha

$$u, v, w \in \Sigma\,, \quad uw = vw \quad (\text{risp.}, wu = wv)$$

allora si ha anche $u = v$.

Osserviamo infine che la concatenazione non gode della proprietà commutativa. Per esempio, se $u = abb$ e $v = aabb$, allora $uv = abbaabb$ mentre $vu = aabbabb$.

Definizione 4 Sia $n \geq 0$. La *potenza n-esima* di una parola u si definisce induttivamente come segue:

$$u^0 = \varepsilon\,, \qquad u^{n+1} = uu^n,\ n \geq 0\,.$$

Per esempio, se $u = abb$ allora $u^0 = \varepsilon$, $u^1 = u = abb$, $u^2 = abbabb$, $u^3 = abbabbabb$.

Definizione 5 Diremo che una parola v è un *fattore* di una parola w se risulta $w = xvy$ per opportune parole x, y. Nel caso in cui $x = \varepsilon$ (risp., $y = \varepsilon$) il fattore v si dice *prefisso* (risp., *suffisso*) di w. Diremo che v è un fattore *proprio* se $v \neq w$.

Per esempio, i fattori di abb sono ε, a, b, ab, bb e abb, i prefissi sono ε, a, ab e abb, e i suffissi sono ε, b, bb e abb.

Definizione 6 Ogni sottoinsieme di Σ^* si dice *linguaggio formale* (o, brevemente, *linguaggio*) sull'alfabeto Σ.

Diamo alcuni esempi di linguaggi formali sull'alfabeto $\Sigma = \{a, b\}$:

$$L_0 = \{a, b\}\,, \quad L_1 = \{a, ab, abb\}\,, \quad L_2 = \{ab^n a \mid n \geq 0\}\,, \quad L_3 = \emptyset\,, \quad L_4 = \Sigma^*\,.$$

Capitolo 3

Grammatiche

Come si è detto, un linguaggio formale è un qualsiasi insieme di parole su un alfabeto finito. Se il linguaggio considerato è costituito da un numero finito di parole distinte, è possibile descriverlo elencando le parole che lo compongono.

La descrizione di un linguaggio infinito può risultare molto più complessa. D'altra parte, come conseguenza del Teorema di Cantor, non può esistere una funzione iniettiva che associ a ogni linguaggio su un alfabeto con almeno due lettere una singola parola su un alfabeto finito. In altri termini, non è possibile associare a ciascun linguaggio una parola (la sua 'descrizione') che lo caratterizzi univocamente. Esistono pertanto linguaggi che non possono essere descritti.

Ciò premesso, è comunque possibile descrivere alcuni insiemi infiniti di parole. Per esempio, si può iniziare da un elenco finito di parole e aggiungere delle regole per costruire nuove parole da queste. Per esempio, la chiusura di Kleene di un linguaggio L è il linguaggio L^* è costituito dalla parola vuota, dalle parole di L e da tutte quelle che si ottengono concatenando due o più parole di L. Quindi, se L è un linguaggio finito, il linguaggio L^* è ben definito e, in generale, risulta infinito. Vediamo un esempio un pò più complesso.

Esempio 1 [1] Supponiamo di voler generare tutte le enumerazioni di nomi del tipo "Aldo, Bianca e Carlo" in cui tutti i nomi sono separati da virgole, tranne gli ultimi due, separati dalla congiunzione 'e'.

Una semplice procedura per generare queste liste è costituita dalles regole seguenti:

1. Aldo è un nome, Bianca è un nome, Carlo è un nome ;

2. Un nome è una lista di nomi ;

[1] Tratto da D. Grüne, C. J. Jacobs, Parsing Techniques, 1995

3. un nome seguito da una virgola e una lista di nomi è anch'essa una lista di nomi ;

4. Prima di terminare, se la lista finisce con ', nome' si deve sostituire ciò con 'e nome' .

Per essere più chiari nell'evidenziare il carattere 'operativo' della nostra costruzione potremmo riguardare i termini nome e listaDiNomi come segnaposto e evidenziare la possibilità di eseguire sostituzioni secondo regole predefinite. La nostra procedura diventa quindi:

1. nome può essere sostituito da Aldo
   nome può essere sostituito da Bianca
   nome può essere sostituito da Carlo

2. listaDiNomi può essere sostituito da nome
   listaDiNomi può essere sostituito da nome, listaDiNomi

3. , nome a fine lista si deve sostituire con e nome prima che nome sia sostituito a sua volta

4. Ci si arresta solo quando listaDiNomi e nome non compaiono più nella frase che stiamo costruendo;

5. Si inizia con listaDiNomi

Osserviamo che la regola 3 è diversa dalle altre dato che da un lato comporta una sostituzione obbligatoria, dall'altro fa riferimento a una precisa posizione nella nostra lista di nomi. Per uniformare le cose si può ricorrere a un 'indicatore di fine lista'. Si ottiene allora:

1. nome può essere sostituito da Aldo
   nome può essere sostituito da Bianca
   nome può essere sostituito da Carlo

2. listaDiNomi può essere sostituito da nome
   listaDiNomi può essere sostituito da nome, altriNomi fineLista

3. altriNomi può essere sostituito da nome
   altriNomi può essere sostituito da nome, altriNomi

4. , nome fineLista può essere sostituito da e nome

5. Ci si arresta solo quando listaDiNomi , altriNomi , fineLista e nome non compaiono più nella frase che stiamo costruendo

6. Si inizia con listaDiNomi

Si noti che sebbene per la regola 4 la sostituzione di ‘, nome fineLista’ con ‘e nome’ sia solo facoltativa, la regola 5 rende tale sostituzione, di fatto, obbligatoria.

Il procedimento illustrato nell’esempio precedente viene formalizzato dalla nozione di grammatica.

Definizione 7 Una *grammatica a struttura di frase* è una quadrupla

$$G = \langle V, \Sigma, P, S \rangle,$$

ove

- V è un alfabeto finito, detto *vocabolario totale*,
- $\Sigma \subseteq V$ è l’alfabeto dei *simboli terminali*,
- $S \in N = V - \Sigma$ è il *simbolo iniziale* o *assioma*,
- P è un insieme finito di espressioni della forma

  $$\alpha \to \beta$$

  con $\alpha \in V^* - \Sigma^*$ e $\beta \in V^*$, detto insieme delle *produzioni*.

Le lettere di $N = V - \Sigma$ si dicono *variabili*. Convenzionalmente, le variabili sono indicate con lettere latine maiuscole e i simboli terminali con lettere latine minuscole. Inoltre, le parole sull’alfabeto V sono indicate con lettere greche minuscole, mentre per le parole sull’alfabeto Σ dei simboli terminali si usano le lettere $u, v, w, \ldots$

Definizione 8 Siano $\alpha, \beta \in V^*$. Diremo che β è una *conseguenza diretta* di α in G se esistono parole $\gamma_1, \gamma_2 \in V^*$ e una produzione $\gamma \to \gamma'$ in P tali che

$$\alpha = \gamma_1 \gamma \gamma_2, \quad \beta = \gamma_1 \gamma' \gamma_2 \,.$$

In tal caso scriveremo $\alpha \Longrightarrow_G \beta$.

Diremo che β si *deriva* (o è una *conseguenza*) di α in G se risulta $\alpha = \beta$, oppure esistono $n > 0$, $\alpha_0, \alpha_1, \ldots, \alpha_n \in V^*$ tali che

$$\alpha = \alpha_0 \underset{G}{\Longrightarrow} \alpha_1 \underset{G}{\Longrightarrow} \cdots \underset{G}{\Longrightarrow} \alpha_n = \beta \,.$$

In tal caso scriveremo $\alpha \Longrightarrow_G^* \beta$.

Chiameremo *forme sentenziali* della grammatica G tutte le conseguenze del simbolo iniziale S. L’insieme delle forme sentenziali di G sarà denotato con $S(G)$.

Chiameremo *linguaggio generato* dalla grammatica G il linguaggio costituito dalle forme sentenziali di G che non contengono variabili. Il linguaggio generato dalla grammatica G sarà denotato con $L(G)$.

Si ha quindi

$$S(G) = \left\{\alpha \in V^* \;\middle|\; S \overset{*}{\underset{G}{\Longrightarrow}} \alpha\right\}, \quad L(G) = S(G) \cap \Sigma^* .$$

Nel seguito, nei simboli $\Longrightarrow_G$ e $\Longrightarrow_G^*$, ometteremo l'indice G quando ciò non crea confusione.

Esempio 2 Sia $G = \langle V, \Sigma, P, S\rangle$ la grammatica con il vocabolario totale $V = \{a, b, S\}$, l'alfabeto dei simboli terminali $\Sigma = \{a, b\}$ e con le produzioni

$$S \to ab\,, \qquad S \to aSb\,.$$

Si ha allora $S \Rightarrow aSb$, $aSb \Rightarrow aaSbb$, $aaSbb \Rightarrow aaaSbbb$, e quindi $S \Longrightarrow^* aaaSbbb$.

Si potrebbe verificare che l'insieme delle forme sentenziali di questa grammatica è

$$S(G) = \{a^n S b^n \mid n \geq 0\} \cup \{a^n b^n \mid n > 0\}\,.$$

Ne segue che il linguaggio generato da G è

$$L(G) = S(G) \cap \Sigma^* = \{a^n b^n \mid n > 0\}\,.$$

Esempio 3 Sia $G = \langle V, \Sigma, P, S\rangle$ la grammatica definita da

$$V = \{S, B, a, b, c\}\,, \quad \Sigma = \{a, b, c\}$$

e con le produzioni

$$S \to aSBc\,, \quad S \to abc\,, \quad cB \to Bc\,, \quad bB \to bb\,.$$

Vogliamo verificare che il linguaggio generato da G è

$$L = \{a^n b^n c^n \mid n > 0\}\,.$$

Iniziamo a verificare l'inclusione $L \subseteq L(G)$. Per fare ciò, dobbiamo far vedere che, per ogni $n > 0$, la parola $a^n b^n c^n$ è conseguenza dell'assioma S. A titolo esemplificativo, ci limiteremo a mostrare la cosa nel caso $n = 3$. Risulterà peraltro evidente che il procedimento si estende facilmente al caso generale. Per facilitare la lettura, in ciascuna forma sentenziale sottolineeremo il lato sinistro della produzione utilizzata per ottenere la forma sentenziale successiva. Abbiamo, innanzitutto

$$\underline{S} \Rightarrow a\underline{S}Bc \Rightarrow aa\underline{S}BcBc \Rightarrow aaabcBcBc\,.$$

Poi, tenendo conto della presenza della produzione $cB \to Bc$, abbiamo

$$aaab\underline{cB}cBc \Rightarrow aaabBc\underline{cB}c \Rightarrow aaabB\underline{cB}cc \Rightarrow aaabBBccc\,.$$

Infine, grazie alle produzioni $bB \to bb$, otteniamo

$$aaa\underline{bB}Bccc \Rightarrow aaab\underline{bB}ccc \Rightarrow aaabbbccc\,.$$

Dalle derivazioni precedenti segue che $S \Longrightarrow^* a^3b^3c^3$, cioè, $a^3b^3c^3 \in L(G)$.

Ora verifichiamo l'inclusione inversa. Osserviamo che tutte le forme sentenziali della grammatica G soddisfano le seguenti condizioni:

1. il numero delle a e uguale al numero delle c; tale numero è anche uguale al numero totale delle b e delle B;

2. se il simbolo iniziale S è presente, tutte le a si trovano alla sua sinistra e tutte le c alla sua destra;

3. in caso contrario, saranno presenti delle b (minuscole), tutte le a si trovano alla loro sinistra e tutte le c alla loro destra;

Invero, non è difficile verificare che se β è una conseguenza diretta di una parola α che soddisfa le Condizioni 1–3, allora anche β soddisfa tali condizioni. Poichè tali condizioni sono verificate dal simbolo iniziale, saranno quindi verificate necessariamente da ogni forma sentenziale. In particolare, ogni parola $w \in L(G)$ è una forma sentenziale priva di variabili e quindi soddisfa le Condizioni 1–3. Pertanto, sarà costituita da una sequenza di a, seguita da una sequenza di b e da una sequenza di c, tutte della stessa lunghezza.

Questo dimostra l'inclusione $L \subseteq L(G)$.

Concludiamo il paragrafo con l'importante nozione di equivalenza tra grammatiche.

Definizione 9 Due grammatiche si dicono *equivalenti* se generano lo stesso linguaggio.

Osserviamo che taluni autori usano il termine "debolmente equivalenti" in luogo di equivalenti.

Capitolo 4

La gerarchia di Chomsky

Con le grammatiche a struttura di frase abbiamo introdotto uno strumento che ci permette di dare una definizione 'operativa' di alcuni linguaggi formali. Diremo che un linguaggio è *ricorsivamente enumerabile* se esiste una grammatica a struttura di frase che lo genera. Come si è già osservato, non tutti i linguaggi possono avere una descrizione finita e quindi non tutti i linguaggi possono essere ricorsivamente enumerabili. D'altra parte, non conosciamo nessun meccanismo per la generazione di linguaggi che risulti più espressivo delle grammatiche. Di più, la tesi di Church, ampiamente condivisa tra gli studiosi di Informatica Teorica, è equivalente all'affermazione che un linguaggio generato da qualsivoglia procedura effettiva è necessariamente un linguaggio ricorsivamente enumerabile.

Le grammatiche risultano quindi uno strumento semplice ma completamente generale per definire linguaggi. Inoltre, il meccanismo della derivazione ci permette, in linea di principio, di produrre qualunque parola di un linguaggio, una volta che sia nota la grammatica che lo genera. Molto più complesso è però il problema di decidere, date una grammatica e una parola, se la parola appartiene o meno al linguaggio generato dalla grammatica stessa (problema di *appartenenza*). Un problema anche più complesso (*parsing*) consiste nel trovare una derivazione (se esistente) della parola nella grammatica data.

Per tale motivo, è spesso utile restringersi a considerare grammatiche in cui le produzioni abbiano una forma particolarmente semplice, e per le quali i problemi indicati qui sopra ammettano soluzioni efficienti. Per tale motivo, Noam Chomsky ha introdotto una gerarchia tra le grammatiche a struttura di frase e, di conseguenza, tra i linguaggi da esse generate.

4.1 Linguaggi di tipo 0 e tipo 1

Al livello 0 della gerarchia si trovano ovviamente le grammatiche a struttura di frase senza alcuna restrizione. I linguaggi generati da tali grammatiche sono i linguaggi *ricorsivamente enumerabili*, detti anche linguaggi di *tipo 0*.

Al livello 1 della gerarchia si trovano le grammatiche 'contestuali'.

Definizione 10 Una grammatica a struttura di frase $G = \langle V, \Sigma, P, S \rangle$ si dice *sensibile al contesto* (o *contestuale*) se tutte le produzioni hanno la forma[1]

$$\alpha_1 X \alpha_2 \to \alpha_1 \beta \alpha_2 \,, \qquad \text{con } X \in N, \quad \alpha_1, \alpha_2, \beta \in V^*, \quad \beta \neq \varepsilon \,.$$

Un linguaggio generato da una grammatica sensibile al contesto si dice linguaggio *sensibile al contesto* o di *tipo 1*.

Come si vede, le produzioni consistono nella sostituzione di una singola variabile X con una parola non vuota e tale sostituzione è possibile solo se la variabile compare nel 'contesto' prescritto ossia tra α_1 e α_2.

Gli acronimi CSG e CSL denotano rispettivamente le grammatiche e i linguaggi sensibili al contesto (*context-sensitive* in Inglese).

Una nozione più generale è quella di *grammatica monotòna*: una grammatica si dice *monotòna* se tutte le produzioni hanno la forma

$$\alpha \to \beta \quad \text{con } |\alpha| \leq |\beta| \,.$$

È evidente che tutte le grammatiche contestuali sono monotòne, mentre vi sono grammatiche monotòne che non sono contestuali. Però si può dimostrare che ogni grammatica monotòna è equivalente a una grammatica contestuale. Vale quindi il seguente

Teorema 1 *Un linguaggio è di tipo 1 se e solo se è generato da una grammatica monotòna.*

Esempio 4 Si consideri il linguaggio introdotto nell'Esempio 1. Se sostituiamo i nomi Aldo, Bianca, Carlo con le lettere a, b, c, la congiunzione 'e' col simbolo & e i termini listaDiNomi, altriNomi, nome, finelista rispettivamente con le variabili S, A N, F, otteniamo la grammatica G con le produzioni

$S \to N$	$A \to N$	$N \to a$
$S \to N, AF$	$A \to N, A$	$N \to b$
	$, NF \to \&N$	$N \to c$

[1] La definizione corretta di grammatica contestuale è leggermente più generale. Qui, per semplicità abbiamo tralasciato alcuni particolari. La stessa precisazione si applica alle grammatiche monotòne e alle grammatiche regolari, introdotte più avanti.

Il linguaggio generato da tale grammatica conterrà parole come, ad esempio

$$a, b, a, c\&b \quad \text{o} \quad b\&a \quad \text{o anche} \quad b\,.$$

Tale grammatica non è monotòna, in quanto la produzione $,NF \to \&N$ ha il lato sinistro più lungo del lato destro. Ma lo stesso linguaggio generato da G è generato anche dalla grammatica con le produzioni:

$S \to NVS$	$VN \to ,N$	$N \to a$	$U \to a$
$S \to U$	$VU \to \&U$	$N \to b$	$U \to b$
		$N \to c$	$U \to c$

In effetti, le produzioni della prima colonna esprimono il fatto che la nostra listaDiNomi può essere costituita o da un nome seguito dalla virgola (rappresentata dalla variabile V) e da una listaDiNomi, oppure dall'ultimoNome della lista (rappresentato dalla variabile U). Le produzioni della seconda colonna ci impongono di sostituire la variabile V con la congiunzione & se precede l'ultimoNome e con una ',' in tutti gli altri casi. La grammatica ottenuta risulta quindi contestuale.

Esempio 5 La grammatica G considerata nell'Esempio 3 è monotòna. Pertanto il linguaggio $L = \{a^n b^n c^n \mid n > 0\}$ è di tipo 1. Inoltre è possibile trovare una grammatica contestuale equivalente a G.

È opportuno precisare esplicitamente che i linguaggi di tipo 1 costituiscono una sottoclasse propria dei linguaggi di tipo 0. Esistono infatti dei linguaggi di tipo 0 che non sono di tipo 1. La costruzione di un tale linguaggio è però troppo complessa per essere affrontata in questa sede.

Una notevole proprietà dei linguaggi di tipo 1 è che esiste un algoritmo (per quanto, in generale, molto costoso) per decidere se una parola appartiene o meno a tale linguaggio. Esistono invece linguaggi di tipo 0 per cui un tale algoritmo non esiste.

4.2 Linguaggi di tipo 2

Al livello 2 della gerarchia si trovano le grammatiche in cui le produzioni consistono ancora nella sostituzione di una singola variabile X con una parola ma non si tiene più conto del contesto.

Definizione 11 Una grammatica a struttura di frase $G = \langle V, \Sigma, P, S\rangle$ si dice *non contestuale* (o *libera dal contesto*) se tutte le produzioni hanno la forma

$$X \to \beta\,, \quad \text{con } X \in N,\ \beta \in V^*\,.$$

Un linguaggio generato da una grammatica non contestuale si dice linguaggio *non contestuale* o di *tipo 2*.

Gli acronimi CFG e CFL denotano rispettivamente le grammatiche e i linguaggi non contestuali (*context-free* in Inglese).

Esempio 6 Si consideri ancora il linguaggio dell'Esempio 4. Tale linguaggio è generato anche dalla grammatica con le produzioni:

$$\begin{array}{lll} S \to N & M \to N\&N & N \to a \\ S \to M & M \to N, M & N \to b \\ & & N \to c \end{array}$$

che risulta non contestuale. In effetti, le produzioni della prima colonna esprimono il fatto che la nostra listaDiNomi è costituita o da un singolo nome o da moltiNomi (almeno due) e le produzioni della seconda colonna esprimono il fatto che la lista di moltiNomi è costituita o da un nome seguito da & e un'altro nome oppure da un nome seguito dalla virgola e da moltiNomi.

La grammatica considerata nell'Esempio 2 è non contestuale. Pertanto il linguaggio $L = \{a^n b^n \mid n > 0\}$ è di tipo 2.

Ovviamente, ogni linguaggio di tipo 2 è anche un linguaggio di tipo 1. Viceversa, vi sono linguaggi di tipo 1 che non sono di tipo 2. Un esempio è fornito dal linguaggio $L = \{a^n b^n c^n \mid n > 0\}$.

4.3 Linguaggi di tipo 3

Nelle grammatiche al livello 3 della gerarchia di Chomsky, una singola variabile X può essere sostituita solo da un terminale seguito eventualmente da una variabile.

Definizione 12 Una grammatica a struttura di frase $G = \langle V, \Sigma, P, S \rangle$ è *di tipo 3* (o *lineare destra*) se tutte le produzioni hanno le forme

$$X \to aY, \quad X \to a, \qquad X, Y \in N,\ a \in \Sigma\,.$$

Esempio 7 Si consideri ancora il linguaggio considerato nell'Esempio 4. Tale linguaggio è generato anche dalla grammatica con le produzioni:

$$\begin{array}{lllll} S \to a & S \to aR & R \to \&N & M \to aR & N \to a \\ S \to b & S \to bR & R \to , M & M \to bR & N \to b \\ S \to c & S \to cR & & M \to cR & N \to c \end{array}$$

che risulta di tipo 3. In effetti, le produzioni delle prime due colonne esprimono il fatto che la nostra listaDiNomi è costituita o da un nome (a, b o c) eventualmente seguito dal restoDellaLista, rappresentato dalla variabile R. A sua volta, come esplicitato nella terza colonna, il restoDellaLista può essere la congiunzione & seguita da un nome oppure la virgola seguita da moltiNomi. Infine le ultime due colonne esprimono il fatto che moltiNomi sono dati da un nome (a, b o c) seguito dal restoDellaLista mentre un nome può essere a, b o c.

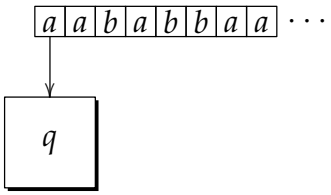
Ovviamente, ogni linguaggio di tipo 3 è anche un linguaggio di tipo 2. Viceversa, vi sono linguaggi di tipo 2 che non sono di tipo 3. Un esempio è fornito dal linguaggio $\{a^n b^n \mid n \geq 0\}$.

Parte II

Linguaggi regolari

Capitolo 5

Automi a stati finiti



Supponiamo di avere a disposizione un dispositivo dotato di un'unità di controllo che può assumere un numero finito di configurazioni interne (stati) e di un nastro di input, diviso in celle, ciascuna delle quali contiene una lettera. Il nostro dispositivo legge il nastro da sinistra a destra.

Dopo aver letto il contenuto di una cella, il nostro dispositivo assumerà un nuovo stato, dipendente esclusivamente dal simbolo letto e dallo stato all'istante precedente. Terminata la lettura del nastro il dispositivo, in base allo stato raggiunto, potrà accettare la parola scritta sul nastro o rifiutarla. Un tale dispositivo prende il nome di automa a stati finiti.

Definizione 13 Un *automa a stati finiti deterministico* è una quintupla

$$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle ,$$

dove

- Q è un insieme finito, detto insieme degli *stati*,
- Σ è un alfabeto, detto *alfabeto di input*,
- $\delta : Q \times \Sigma \to Q$ è la *funzione di transizione*,
- $q_0 \in Q$ è lo *stato iniziale*,

- $F \subseteq Q$ è l'insieme degli *stati finali*.

Per poter valutare quali parole sono accettate dall'automa, abbiamo bisogno di una funzione che descriva il comportamento dell'automa quando viene letto un input di lunghezza anche maggiore di 1. Pertanto, la funzione δ si estende induttivamente a una funzione $\widehat{\delta} : Q \times \Sigma^* \to Q$ ponendo

$$\begin{aligned} \widehat{\delta}(q, \varepsilon) &= q\,, && \text{per ogni } q \in Q\,, \\ \widehat{\delta}(q, va) &= \delta(\widehat{\delta}(q, v), a)\,, && \text{per ogni } q \in Q\,,\ v \in \Sigma^*\,,\ a \in \Sigma\,. \end{aligned}$$

Non è difficile verificare che per ogni $q \in Q$, $u, v \in \Sigma^*$ vale la seguente identità:

$$\widehat{\delta}(q, uv) = \widehat{\delta}(\widehat{\delta}(q, u), v)\,.$$

Definizione 14 Sia $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ un automa a stati finiti deterministico. Una parola $w \in \Sigma^*$ è *accettata* da $\mathcal{A}$ se $\widehat{\delta}(q_0, w) \in F$. L'insieme delle parole accettate da $\mathcal{A}$, cioè il linguaggio

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \widehat{\delta}(q_0, w) \in F\}$$

si dice *linguaggio riconosciuto* (o *accettato*) da $\mathcal{A}$.

Esempio 8 Si consideri l'automa $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ con $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $F = \{q_0, q_1\}$ e con δ definita dalla tabella seguente:

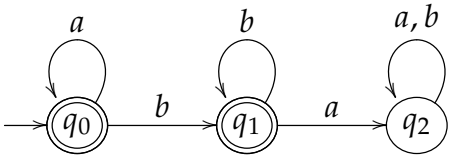
	a	b
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_2

Per calcolare $\delta(q_0, aaabb)$, osserviamo che

$$\begin{aligned} \widehat{\delta}(q_0, \varepsilon) &= q_0, \\ \widehat{\delta}(q_0, a) &= \delta(\widehat{\delta}(q_0, \varepsilon), a) = \delta(q_0, a) = q_0, \\ \delta(q_0, aa) &= \delta(\widehat{\delta}(q_0, a), a) = \delta(q_0, a) = q_0, \\ \delta(q_0, aaa) &= \delta(\widehat{\delta}(q_0, aa), a) = \delta(q_0, a) = q_0, \\ \delta(q_0, aaab) &= \delta(\widehat{\delta}(q_0, aaa), b) = \delta(q_0, b) = q_1, \\ \delta(q_0, aaabb) &= \delta(\widehat{\delta}(q_0, aaab), b) = \delta(q_1, b) = q_1. \end{aligned}$$

Poiché $q_1 \in F$, la parola *aaabb* è accettata da $\mathcal{A}$.

Può essere utile rappresentare un automa con un grafo diretto etichettato in cui i vertici sono gli stati dell'automa e gli archi sono le triple (q, a, q') con $q \in Q$, $a \in \Sigma$ e $q' = \delta(q, a)$). L'automa dell'Esempio 8 è rappresentato dal grafo seguente:



Come è d'uso, lo stato iniziale è denotato da una freccia entrante priva di etichetta e gli stati finali sono identificati dal bordo doppio.

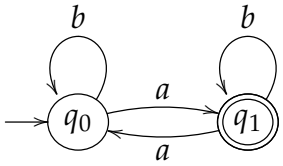
Osserviamo che i grafi associati agli automi a stati finiti deterministici sono caratterizzati dalla proprietà di avere esattamente una freccia uscente da ogni nodo con una data etichetta.

In un grafo siffatto, c'è un cammino da un nodo p a un nodo q con etichetta w se e soltanto se $q = \widehat{\delta}(p, w)$. Se ne conclude che le parole accettate dall'automa sono le etichette dei cammini che vanno dallo stato iniziale a uno degli stati finali.

Nell'esempio precedente, le etichette dei cammini da q_0 a q_0 sono le parole a^n con $n \geq 0$ mentre le etichette dei cammini da q_0 a q_1 sono le parole $a^n b^m$ con $n \geq 0$ e $m > 0$. Ne concludiamo che il linguaggio accettato da tale automa è

$$L = \{a^n b^m \mid n, m \geq 0\}.$$

Esempio 9 Si consideri l'automa a due stati con il seguente grafo:



Poichè la lettera a porta da uno stato all'altro, mentre la lettera b non fa cambiare stato, le etichette dei cammini dallo stato iniziale allo stato finale sono esattamente le parole con un numero dispari di occorrenze della lettera a. Pertanto questo automa riconosce il linguaggio

$$L = \{w \in \Sigma^* \mid |w|_a \text{ dispari}\},$$

ove $|w|_a$ denota il numero di occorrenze della lettera a in w.

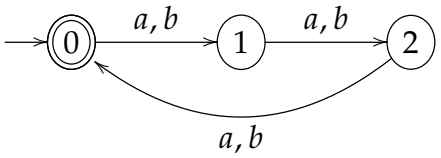
Esempio 10 Vogliamo realizzare un automa che riconosca il linguaggio

$$L = \{w \in \Sigma^* \mid |w| \text{ è multiplo di } 3\}$$

sull'alfabeto $\Sigma = \{a, b\}$. Costruiremo un automa con 3 stati $0, 1, 2$ tale che

$$\widehat{\delta}(0, w) = |w| \bmod 3\,, \quad w \in \Sigma^*\,.$$

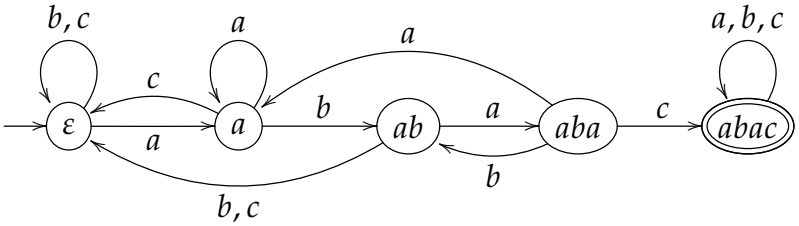
In effetti, è sufficiente porre $\delta(q, a) = \delta(q, b) = (q + 1) \bmod 3$. Si ottiene così l'automa



Esempio 11 Sia $\Sigma = \{a, b, c\}$. Vogliamo realizzare un automa che riconosca il linguaggio

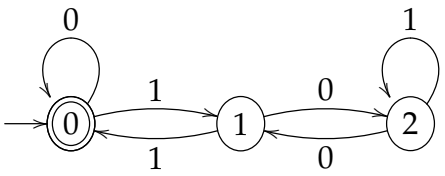
$$L = \{w \in \Sigma^* \mid \textit{abac} \text{ è fattore di } w\}\,.$$

Qui gli stati rappresentano i prefissi della parola *abac* e l'automa è costruito in modo che se $w \notin L$, allora $\widehat{\delta}(q_0, w)$ è il più lungo prefisso di *abac* che è anche suffisso di w.



Osserviamo che automi di questo tipo sono abitualmente usati in alcuni algoritmi per la ricerca nel testo (*pattern matching*).

Esempio 12 Si consideri l'alfabeto $\Sigma = \{0, 1\}$ e il linguaggio $L \subseteq \Sigma^*$ costituito dalle espansioni binarie dei multipli di 3. Come è noto, se $w \in \Sigma^*$ è l'espansione binaria dell'intero n allora $w0$ e $w1$ sono rispettivamente le espansioni binarie di $2n$ e $2n + 1$. Una relazione analoga vale per i rispettivi residui (mod 3). Un automa che riconosce L è quello rappresentato dal grafo seguente, in cui gli stati 'registrano' il residuo (mod 3) della parola letta.



Esempio 13 Si consideri il linguaggio $L = \{a^n b^n \mid n > 0\}$ sull'alfabeto $\Sigma = \{a, b\}$. Vogliamo mostrare che non esiste un automa a stati finiti che riconosce L.

Supponiamo, per assurdo, che esista un automa $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ che riconosce L. Consideriamo la lista infinita di stati $\widehat{\delta}(q_0, a^n)$, $n \geq 0$. Data la finitezza di Q, in tale lista ci sarà una ripetizione. Avremo cioè

$$\widehat{\delta}(q_0, a^n) = \widehat{\delta}(q_0, a^m)\,, \quad n \neq m\,,$$

per opportuni $n, m \geq 0$. Ne segue

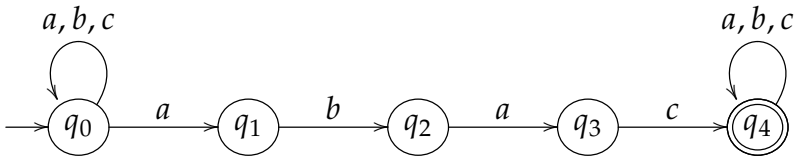
$$\widehat{\delta}(q_0, a^n b^n) = \widehat{\delta}(q_0, a^m b^n) = q\,.$$

Ora se $q \in F$, allora $\mathcal{A}$ accetta sia $a^n b^n$ che $a^m b^n$, se invece $q \notin F$, allora $\mathcal{A}$ non accetta né $a^n b^n$ né $a^m b^n$. In entrambi i casi si ha una contraddizione, poiché $a^n b^n \in L$ mentre $a^m b^n \notin L$. Se ne conclude che L non è riconosciuto da nessun automa a stati finiti.

Capitolo 6

Automi non deterministici

Come si è visto, a ogni automa a stati finiti deterministico $\mathcal{A}$ corrisponde un grafo e le parole accettate da $\mathcal{A}$ sono le etichette dei cammini dallo stato iniziale a uno stato finale. Tale grafo è caratterizzato dalla proprietà che per ogni stato q e ogni lettera a esiste esattamente un arco uscente da q con etichetta a. Questa proprietà può rendere arduo progettare un automa che accetti un dato insieme di parole. Si consideri ad esempio il linguaggio L definito nell'Esempio 11. Le parole di L sono esattamente le etichette dei cammini dallo stato iniziale allo stato finale del seguente grafo:



Tale grafo, molto più semplice di quello dell'Esempio 11, non corrisponde però a un automa a stati finiti deterministico, visto che dallo stato iniziale escono due frecce etichettate a. Può pertanto essere utile considerare un differente modello di automa in cui, ad ogni passo, siano possibili più transizioni in stati differenti. Pertanto a ogni input corrisponderanno varie computazioni e l'input sarà accettato se almeno una di tali computazioni termina in uno stato finale.

Definizione 15 Un *automa a stati finiti non deterministico* è una quintupla

$$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle ,$$

dove Q, Σ, q_0, F sono come nella Definizione 13 e

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

è la *funzione di transizione*.

La definizione delle parole accettate da un automa non deterministico richiede qualche attenzione.

Definizione 16 Sia $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ un automa a stati finiti non deterministico. Una parola $w = a_1 a_2 \cdots a_n$, $a_1, a_2, \ldots, a_n \in \Sigma$, $n \geq 0$, è *accettata* da $\mathcal{A}$ se esistono stati $q_1, q_2, \ldots, q_n \in Q$ tali che

$$q_i \in \delta(q_{i-1}, a_i)\,,\ 1 \leq i \leq n\,, \quad q_n \in F\,. \tag{6.1}$$

L'insieme delle parole accettate da $\mathcal{A}$ si dice *linguaggio accettato* (o *riconosciuto*) da $\mathcal{A}$ e si denota con $L(\mathcal{A})$.

Come nel caso degli automi deterministici, si può rappresentare un automa non deterministico $\mathcal{A}$ con un grafo diretto etichettato. In questo caso i vertici sono gli stati dell'automa e gli archi sono le triple (q, a, q'), con $q \in Q$, $a \in \Sigma$, $q' \in \delta(q, a)$. A differenza del caso degli automi deterministici, però, da uno stesso vertice può uscire più di una freccia con una data etichetta. È anche possibile che da un nodo non esca nemmeno una freccia etichettata con una data lettera a. Non è difficile poi convincersi che una parola w è accettata da $\mathcal{A}$ se e solo se w è l'etichetta di un cammino nel grafo con partenza dallo stato iniziale e termine in uno stato finale. Questo non esclude però che vi possano essere altri cammini con etichetta w che partono dallo stato iniziale e terminano in uno stato non finale.

Esempio 14 Si consideri il grafo dato all'inizio del paragrafo. La funzione di transizione del corrispondente automa finito non deterministico è data dalla seguente tabella:

	a	b	c	F
q_0	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0\}$	
q_1	$\emptyset$	$\{q_2\}$	$\emptyset$	
q_2	$\{q_3\}$	$\emptyset$	$\emptyset$	
q_3	$\emptyset$	$\emptyset$	$\{q_4\}$	
q_4	$\{q_4\}$	$\{q_4\}$	$\{q_4\}$	$\times$

È chiaro che ogni automa a stati finiti deterministico può essere identificato con l'automa non deterministico che ha lo stesso grafo e che, quindi, riconosce lo stesso linguaggio. Più sorprendente è il fatto che a ogni automa non deterministico corrisponde un automa deterministico che riconosce lo stesso linguaggio. Dimostreremo infatti il seguente

Teorema 2 *Sia $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ un automa a stati finiti non deterministico. Esiste effettivamente un automa a stati finiti deterministico $\mathcal{A}'$ tale che*

$$L(\mathcal{A}) = L(\mathcal{A}')\,.$$

L'idea centrale della dimostrazione è quella di realizzare un automa deterministico i cui stati 'registrino' l'insieme degli stati che si raggiungono con le computazioni di $\mathcal{A}$ sul medesimo input. Tale costruzione è possibile in quanto per un input va, $v \in \Sigma^*$, $a \in \Sigma$, tale insieme di stati dipende unicamente dall'insieme di stati corrispondente alla parola v e dalla lettera a.

Più precisamente, procediamo nel modo seguente: dato un automa non deterministico $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$, definiamo l'automa deterministico $\mathcal{A}' = \langle Q', \Sigma, \delta', s_0, F'\rangle$, come segue:

- l'insieme degli stati è l'insieme $Q' = \wp(Q)$ costituito dai sottoinsiemi di Q,
- lo stato iniziale è $s_0 = \{q_0\}$,
- gli stati finali sono tutti i sottoinsiemi di Q che intersecano F, cioè

$$F' = \{s \in \wp(Q) \mid s \cap F \neq \emptyset\} \tag{6.2}$$

- la funzione di transizione $\delta' \colon \wp(Q) \times \Sigma \to \wp(Q)$ è definita da

$$\delta'(s, a) = \bigcup_{q \in s} \delta(q, a)\,, \quad s \in \wp(Q),\ a \in \Sigma. \tag{6.3}$$

Verificheremo che $L(\mathcal{A}) = L(\mathcal{A}')$. Iniziamo a verificare una proprietà della funzione di transizione δ'.

Lemma 1 *Per ogni $w \in \Sigma^*$, $\delta'(s_0, w)$ è uguale all'insieme degli stati che, nel grafo dell'automa $\mathcal{A}$, si raggiungono da q_0 con un cammino etichettato w.*

DIMOSTRAZIONE: Procediamo per induzione su $|w|$.

Base: Nel caso in cui $|w| = 0$, si ha $w = \varepsilon$ e quindi, tenendo conto della definizione di $\widehat{\delta'}$ e di s_0, si ha $\widehat{\delta'}(s_0, w) = \widehat{\delta'}(s_0, \varepsilon) = s_0 = \{q_0\}$. E q_0 è anche l'unico stato che si raggiunge da q_0 con un cammino di lunghezza 0. Pertanto l'asserto è banalmente verificato.

Passo induttivo: Supponiamo ora $|w| > 0$. Allora possiamo scrivere $w = va$ con $v \in \Sigma^*$ e $a \in \Sigma$. Poniamo $s = \widehat{\delta'}(s_0, v)$. Per l'ipotesi induttiva, s è proprio l'insieme degli stati che, nel grafo dell'automa $\mathcal{A}$, si raggiungono da q_0 con un cammino etichettato v.

Dalla definizione di $\widehat{\delta'}$ e di δ' si ha

$$\widehat{\delta'}(s_0, w) = \delta'(v, a) = \bigcup_{p \in s} \delta(p, a)\,. \tag{6.4}$$

L'ultimo termine della precedente equazione rappresenta l'insieme degli stati che, nel grafo di $\mathcal{A}$, sono raggiunti da una freccia con etichetta a uscente da uno stato $p \in s$. Tenendo presente che s è l'insieme degli stati che si raggiungono da q_0 con un cammino etichettato v, si ottiene l'asserto. □

Ora siamo pronti per dimostrare il Teorema 2.

DIMOSTRAZIONE DEL TEOREMA 2: Chiaramente basta provare che gli automi $\mathcal{A}$ e $\mathcal{A}'$ accettano lo stesso linguaggio.

Data una parola w, poniamo $s = \delta(s_0, w)$. Come si è già osservato, la parola w è accettata dall'automa $\mathcal{A}$ se e solo se nel grafo dell'automa c'è un cammino con origine nello stato iniziale q_0, etichetta w e termine in uno stato $q \in F$. Poichè, per il lemma precedente, gli stati che vengono raggiunti dai cammini con origine q_0 ed etichetta w sono esattamente gli elementi di s, si conclude che w è accettata da $\mathcal{A}$ se e solo se

$$s \cap F \neq \emptyset\,.$$

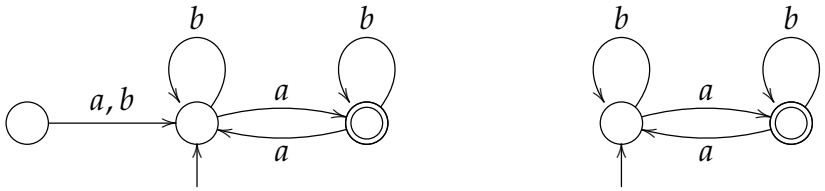
D'altra parte, w è accettata dall'automa deterministico $\mathcal{A}'$ se e solo se $s = \delta(s_0, w) \in F'$. Ma, per la definizione stessa di F', anche questa condizione è soddisfatta se e solo se $s \cap F \neq \emptyset$. Se ne conclude che gli automi $\mathcal{A}$ e $\mathcal{A}'$ accettano le stesse parole. □

Gli stati dell'automa deterministico $\mathcal{A}'$ sono le parti dell'insieme degli stati dell'automa non deterministico $\mathcal{A}$. Ciò significa che se il numero degli stati di $\mathcal{A}$ è n allora $\mathcal{A}'$ avrà 2^n stati. Per fortuna, in molti casi non tutti questi stati sono realmente necessari, in quanto alcuni di essi non sono mai raggiunti nel corso di una computazione.

Definizione 17 Sia $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ un automa a stati finiti deterministico. Uno stato $q \in Q$ si dice *accessibile* se $q = \delta(q_0, w)$ per qualche $w \in \Sigma^*$. L'automa si dice *ridotto* se tutti i suoi stati sono accessibili.

Chiaramente, se eliminiamo tutti gli stati non accessibili di un dato automa e restringiamo di conseguenza la funzione di transizione, otterremo un automa ridotto equivalente a quello dato.

Per esempio, nella figura seguente sono rappresentati un automa e il corrispondente automa ridotto:



Nel nostro caso è conveniente costruire l'automa ridotto corrispondente ad $\mathcal{A}'$ anzicché $\mathcal{A}'$ stesso.

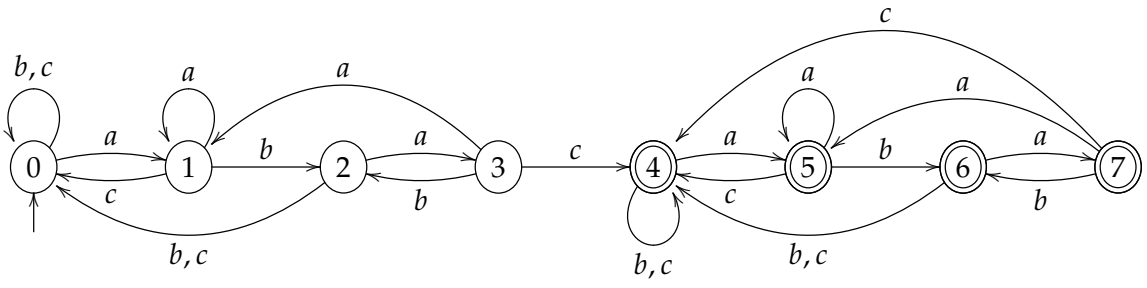
Nell'Algoritmo 1 utilizziamo una lista di stati che inizialmente conterrà solo lo stato iniziale $s_0 = \{q_0\}$. Per ogni stato r della lista e ogni lettera $a \in \Sigma$ calcoliamo $s = \delta'(r, a)$ mediante l'Equazione (6.3). Se otteniamo un nuovo stato lo aggiungiamo alla lista. Contemporaneamente, facendo uso dell'Equazione (6.2), costruiamo l'insieme F' degli stati finali.

Algoritmo 1: Determinizzazione

Ingresso: Un automa non deterministico $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$
Uscita: Un automa deterministico $\mathcal{A}' = \langle Q', \Sigma, \delta', s_0, F' \rangle$ equivalente

$s_0 \leftarrow \{q_0\}$;
$n \leftarrow 1$; `// numero degli stati`
$j \leftarrow 0$;
$F' \leftarrow \emptyset$;
mentre $j<n$ **fai**
  per ciascun $a \in \Sigma$ **fai**
    $s \leftarrow \bigcup_{q \in s} \delta(q, a)$;
    se $s = s_i$ *per qualche* $i < n$ **allora**
      $\delta'(j, a) \leftarrow i$
    altrimenti
      $s_n \leftarrow s$;
      $\delta'(j, a) \leftarrow n$;
      se $s \cap F \neq \emptyset$ **allora** $F' \leftarrow F' \cup \{n\}$;
      $n \leftarrow n + 1$
  $j \leftarrow j + 1$

Esempio 15 Applicando il precedente algoritmo all'automa dell'Esempio 14, si ottiene la Tabella 6.1 che corrisponde all'automa col grafo che segue:



i	s_i	$\delta'(i,a)$	$\delta'(i,b)$	$\delta'(i,c)$	$i \in F$
0	q_0	1	0	0	
1	q_0, q_1	1	2	0	
2	q_0, q_2	3	0	0	
3	q_0, q_1, q_3	1	2	4	
4	q_0, q_4	5	4	4	$\times$
5	q_0, q_1, q_4	5	6	4	$\times$
6	q_0, q_2, q_4	7	4	4	$\times$
7	q_0, q_1, q_3, q_4	5	6	4	$\times$

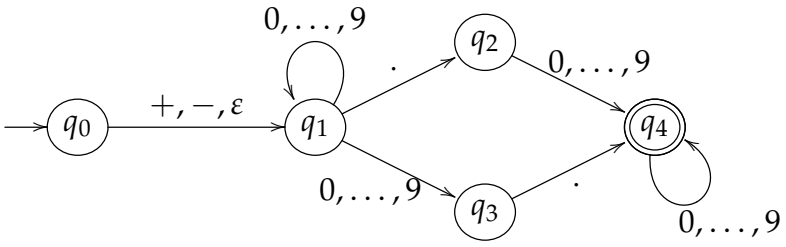
Tabella 6.1: Determinizzazione dell'automa dell'Esempio 14

6.1 ε-transizioni

Esempio 16 Sia L il linguaggio costituito dalle rappresentazioni decimali dei numeri razionali relativi. Un elemento di L si ottiene concatenando:

1. Il simbolo $+$ o $-$ o la parola vuota,
2. una sequenza finita di cifre,
3. il punto decimale
4. un'altra sequenza finita di cifre.

Una delle sequenze 2. o 4. può essere vuota ma non entrambe. Gli elementi di L sono quindi le etichette dei cammini nel grafo seguente con inizio nel vertice q_0 e termine nel vertice q_4.



Naturalmente, questo non è il grafo di un automa non deterministico, per la presenza dell'etichetta ε su uno degli archi. L'esempio precedente mostra che per disporre di maggiore espressività nel disegno degli automi può risultare utile prevedere delle ε-transizioni. Considereremo cioè la possibilità che il nostro automa possa saltare da uno stato a un altro senza leggere il nastro di input.

Definizione 18 Un *automa a stati finiti non deterministico con ε-transizioni* è una quintupla

$$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle ,$$

dove Q, Σ, q_0, F sono come nella Definizione 13 e

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

è la *funzione di transizione*.

Passiamo ora a definire quali sono le parole accettate da un automa siffatto.

Definizione 19 Sia $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ un automa a stati finiti non deterministico con ε-transizioni. Una parola $w \in \Sigma^*$ è *accettata* da $\mathcal{A}$ se esistono $n \geq 0$, $a_1, a_2, \ldots, a_n \in \Sigma \cup \{\varepsilon\}$ e $q_1, q_2, \ldots, q_n \in Q$ tali che

$$w = a_1 a_2 \cdots a_n , \quad q_i \in \delta(q_{i-1}, a_i) , \ 1 \leq i \leq n , \quad q_n \in F . \tag{6.5}$$

L'insieme delle parole accettate da $\mathcal{A}$ si dice *linguaggio accettato* (o *riconosciuto*) da $\mathcal{A}$ e si denota con $L(\mathcal{A})$.

Anche in presenza di ε-transizioni, si può rappresentare un automa non deterministico $\mathcal{A}$ con un grafo diretto etichettato. In questo caso i vertici sono gli stati dell'automa e gli archi sono le triple (q, a, s), con $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $s \in \delta(q, a)$. Non è difficile convincersi che, anche in questo caso, una parola w è accettata da $\mathcal{A}$ se e solo se esiste un cammino nel grafo corrispondente con origine nello stato iniziale, termine in uno stato finale ed etichetta w.

Esempio 17 Il grafo dell'Esempio 16 corrisponde all'automa finito non deterministico con ε-transizioni la cui funzione di transizione è data dalla seguente tabella:

	ε	$+$	$-$	$.$	$0, \ldots, 9$	F
q_0	$\{q_1, q_4\}$	$\{q_1, q_4\}$	$\{q_1, q_4\}$	$\emptyset$	$\emptyset$	
q_1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_2\}$	
q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$	$\{q_2\}$	
q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$	$\times$
q_4	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_5\}$	$\emptyset$	
q_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_5, q_6\}$	
q_6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\times$

Il Teorema 2 si estende al caso degli automi con ε-transizioni:

Teorema 3 *Sia $\mathcal{A}$ un automa a stati finiti non deterministico con ε-transizioni. Esiste effettivamente un automa a stati finiti deterministico $\mathcal{A}'$ tale che*

$$L(\mathcal{A}) = L(\mathcal{A}') .$$

Per poter costruire l'automa deterministico equivalente a un automa non deterministico con ε-transizioni, abbiamo bisogno di introdurre la nozione di ε-chiusura di uno stato. Informalmente, la ε-chiusura di uno stato q è l'insieme degli stati che da q si possono raggiungere utilizzando solo ε-transizioni. Più precisamente:

Definizione 20 Sia $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ un automa a stati finiti non deterministico con ε-transizioni. La *ε-chiusura* di uno stato $q \in Q$ è il più piccolo sottoinsieme $C_\varepsilon(q)$ di Q tale che

- $q \in C_\varepsilon(q)$,
- per ogni $p \in C_\varepsilon(q)$, $\delta(p, \varepsilon) \subseteq C_\varepsilon(q)$.

La *ε-chiusura* di un insieme di stati $S \subseteq Q$ è l'unione delle ε-chiusure dei singoli stati di S: $C_\varepsilon(S) = \bigcup_{q \in S} C_\varepsilon(q)$.

Per il calcolo effettivo della ε-chiusura di uno stato, si può osservare che $C_\varepsilon(q)$ è l'insieme degli stati accessibili da q nel grafo delle ε-transizioni dell'automa

$$G_\varepsilon = (Q, \{(r, s) \mid r \in Q,\ s \in \delta(p, \varepsilon)\}\,.$$

Una semplice procedura per il calcolo di $C_\varepsilon(q)$ è l'Algoritmo 2.

Algoritmo 2: Chiusura

Ingresso: Un automa non deterministico con ε-transizioni $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ e uno stato $q \in Q$
Uscita: La ε-chiusura di q

$R \leftarrow (q)$;
$p \leftarrow$ primo elemento di R;
ripeti
    appendi a R tutti gli elementi di $\delta(p, \varepsilon) - R$;
    $p \leftarrow$ successivo(p)
finché $p =$ NIL; `// fine lista`
ritorna R

La dimostrzione del Teorema 3 è simile a quella del Teorema 2. Più precisamente, dato un automa non deterministico con ε-transizioni $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$,

definiamo l'automa deterministico $\mathcal{A}' = \langle Q', \Sigma, \delta', s_0, F' \rangle$, con

$$
\begin{aligned}
Q' &= \wp(Q)\,, \\
\delta'(s,a) &= C_\varepsilon\left(\bigcup_{p\in s} \delta(p,a)\right), \qquad s \in \wp(Q),\ a \in \Sigma \\
s_0 &= C_\varepsilon(q_0)\,, \\
F' &= \{s \in \wp(Q) \mid s \cap F \neq \emptyset\}\,.
\end{aligned}
$$

Analogamente a quanto fatto nel caso degli automi a stati finiti senza ε-transizioni, si verifica che $L(\mathcal{A}') = L(\mathcal{A})$.

La costruzione dell'automa deterministico equivalente a un dato automa non deterministico con ε-transizioni è realizzata dall'Algoritmo 3, semplice generalizzazione dell'Algoritmo 1.

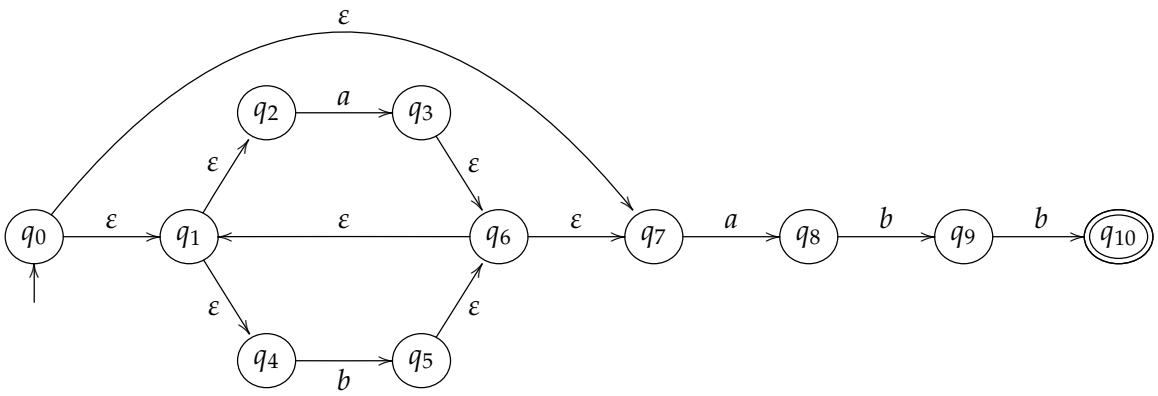
Algoritmo 3: Determinizzazione

Ingresso: Un automa non deterministico con ε-transizioni $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$

Uscita: Un automa deterministico $\mathcal{A}' = \langle Q', \Sigma, \delta', s_0, F' \rangle$ equivalente

```
s_0 ← C_ε(q_0);
n ← 1 ;                                   // numero degli stati
j ← 0;
F' ← ∅;
mentre j<n fai
    per ciascun a ∈ Σ fai
        s ← ⋃_{q∈s} δ(q, a);
        s ← C_ε(s);
        se s = s_i per qualche i < n allora
            δ'(j, a) ← i
        altrimenti
            s_n ← s;
            δ'(j, a) ← n;
            se s ∩ F ≠ ∅ allora F' ← F' ∪ {n};
            n ← n + 1
    j ← j + 1
```

Esempio 18 Si consideri il seguente automa $\mathcal{A}$



Usando l'Algoritmo 2 otteniamo la ε-chiusura degli stati di $\mathcal{A}$:

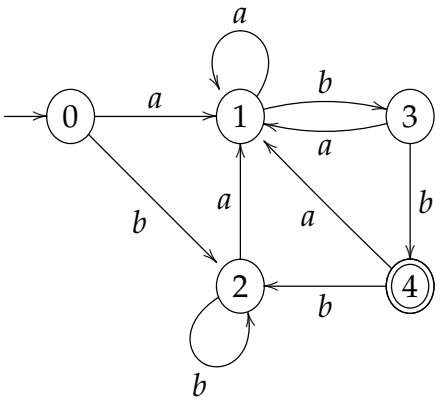
i	$C_\varepsilon(q_i)$
0	$\{q_0, q_1, q_2, q_4, q_7\}$
1	$\{q_1, q_2, q_4\}$
2	$\{q_2\}$
3	$\{q_1, q_2, q_3, q_4, q_6, q_7\}$
4	$\{q_4\}$
5	$\{q_1, q_2, q_4, q_5, q_6, q_7\}$

i	$C_\varepsilon(q_i)$
6	$\{q_1, q_2, q_4, q_6, q_7\}$
7	$\{q_7\}$
8	$\{q_8\}$
9	$\{q_9\}$
10	$\{q_{10}\}$

Con l'Algoritmo 3 si ottiene la tabella seguente

j	s_j	$\delta(j,a)$	$\delta(j,b)$	$j \in F$
0	$\{q_0, q_1, q_2, q_4, q_7\}$	1	2	
1	$\{q_1, q_2, q_3, q_4, q_6, q_7, q_8\}$	1	3	
2	$\{q_1, q_2, q_4, q_5, q_6, q_7\}$	1	2	
3	$\{q_1, q_2, q_4, q_5, q_6, q_7, q_9\}$	1	4	
4	$\{q_1, q_2, q_4, q_5, q_6, q_7, q_{10}\}$	1	2	$\times$

che corrisponde all'automa:



Capitolo 7

Teorema di Kleene

Come si è detto più volte, i linguaggi sono insiemi di parole. Pertanto possiamo operare su di essi con le operazioni Booleane di unione, intersezione e complemento. Introdurremo ora due nuove operazioni sui linguaggi.

Definizione 21 Siano L_1 e L_2 due linguaggi sull'alfabeto Σ. La *concatenazione* di L_1 e L_2 è il linguaggio

$$L_1 L_2 = \{uv \mid u \in L_1,\ v \in L_2\}\,.$$

Per esempio, la concatenazione dei linguaggi $L_1 = \{ab^n a \mid n \geq 0\}$ e $L_2 = \{b^m \mid m \geq 0\}$ è il linguaggio

$$L_1 L_2 = \{ab^n ab^m \mid n, m \geq 0\}\,.$$

Abbiamo così introdotto un'operazione binaria sull'insieme $\wp(\Sigma^*)$ dei linguaggi sull'alfabeto Σ. Tale operazione gode della proprietà associativa, e il linguaggio $\{\varepsilon\}$ funge da elemento neutro.

La *potenza n-esima* di un linguaggio L si definisce induttivamente come segue:

$$L^0 = \{\varepsilon\}\,, \qquad L^{n+1} = LL^n\,,\ n \geq 0\,.$$

Definizione 22 La *chiusura di Kleene* di un linguaggio L è il linguaggio

$$L^* = \bigcup_{n \geq 0} L^n = \{u_1 u_2 \cdots u_n \mid n \geq 0,\ u_1, u_2, \ldots, u_n \in L\}\,.$$

Per esempio, se $L = \{a,\ ab,\ abb\}$, allora avremo $L^0 = \{\varepsilon\}$, $L^1 = L$,

$$L^2 = \{aa, aab, aabb, aba, abab, ababb, abba, abbab, abbabb\}\,,$$
$$L^* = \{\varepsilon\} \cup \{ab^{i_1} ab^{i_2} a \cdots ab^{i_n} \mid n > 0, i_1, i_2, \ldots, i_n = 0, 1, 2\}\,.$$

Ora introdurremo le espressioni regolari. Si tratta di particolari parole, che contengono, oltre alle lettere di un alfabeto Σ, i simboli $\emptyset$, $+$, $*$, $($, $)$ e che permettono di denotare dei linguaggi sull'alfabeto Σ, ottenuti usando ripetutamente le operazioni di unione, concatenazione e chiusura di Kleene, a partire da linguaggi finiti.

Definizione 23 Dato un alfabeto Σ, consideriamo l'alfabeto $\widehat{\Sigma}$ ottenuto aggiungendo a Σ i nuovi simboli $\emptyset$, $+$, $*$, $($, $)$. Le *espressioni regolari* sull'alfabeto Σ sono le parole di $\widehat{\Sigma}^*$ definite dalle seguenti regole:

(i) Ogni lettera $a \in \Sigma$ è un'espressione regolare; $\emptyset$ è un'espressione regolare,

(ii) Se E e F sono espressioni regolari, allora $(E + F)$, (EF) e E^* sono espressioni regolari,

(iii) Solo le parole che si ottengono applicando un numero finito di volte le due regole precedenti sono espressioni regolari.

Possiamo associare a ogni espressione regolare un linguaggio sull'alfabeto Σ che chiameremo il linguaggio *denotato* dall'espressione regolare, definito ricorsivamente nel modo seguente.

Definizione 24 Per ogni $a \in \Sigma$, l'espressione regolare a *denota* il linguaggio $\{a\}$; l'espressione regolare $\emptyset$ *denota* il linguaggio vuoto.

Date due espressioni regolari E, F e detti rispettivamente L_E e L_F i linguaggi denotati da E e F, le espressioni regolari $(E + F)$, (EF) e E^* *denotano* rispettivamente i linguaggi $L_E \cup L_F$, $L_E L_F$ e L_E^*.

Diremo che un linguaggio è *regolare* se esiste un'espressione regolare che lo denota.

Spesso, per semplicità, nelle espressioni regolari si omettono le parentesi, quando non risultano necessarie per determinare il linguaggio denotato dall'espressione. A tal fine, si conviene che le operazioni vadano svolte col seguente ordine di priorità: chiusura di Kleene, concatenazione, somma. Ad esempio, l'espressione $bab + ab^*$ è equivalente all'espressione regolare $(((ba)b) + (ab^*))$.

Si osservi che l'espressione $\emptyset^*$ denota il linguaggio $\{\varepsilon\}$. Spesso si preferisce rimpiazzare tale espressione con il simbolo ε.

Esempio 19 Sia $\Sigma = \{a, b\}$. L'espressione regolare $(a + b)^*$ denota il linguaggio Σ^*, ossia l'insieme di tutte le parole sull'alfabeto Σ. L'espressione regolare $(a + ab)^*$ denota il linguaggio $\{a, ab\}^*$, cioè l'insieme delle parole sull'alfabeto Σ che iniziano con a e non contengono il fattore bb.

L'espressione regolare $(a+b)^*abb$ denota l'insieme delle parole sull'alfabeto Σ che hanno il suffisso abb.

L'espressione regolare $((a+b)(a+b)(a+b))^*$, o, più brevemente, $((a+b)^3)^*$, denota l'insieme delle parole sull'alfabeto Σ la cui lunghezza è un multiplo di 3.

Infine, l'espressione regolare $(b^*ab^*ab^*)^*b^*ab^*$ denota l'insieme delle parole sull'alfabeto Σ con un numero dispari di occorrenze della lettera a.

È opportuno osservare che ogni linguaggio finito è regolare. Invero, con la nostra convenzione sulle parentesi, se $w = a_1a_2\cdots a_n$ con $a_1, a_2, \ldots, a_n \in \Sigma$, allora la stessa w è un'espressione regolare che denota il linguaggio $\{w\}$. D'altra parte, l'espressione regolare $\emptyset^*$ denota la chiusura di Kleene del linguaggio vuoto che, per convenzione, è il linguaggio $\{\varepsilon\}$. Più in generale, un linguaggio finito $L = \{w_1, w_2, \ldots, w_n\}$ è denotato dall'espressione regolare $w_1 + w_2 + \cdots + w_n$.

Come vedremo, i linguaggi regolari sono caratterizzati dal fondamentale

Teorema 4 (Kleene) *Un linguaggio è regolare se e soltanto se è riconosciuto da un automa a stati finiti.*

Si presentano qui due problemi: il primo, detto *problema di sintesi*, consiste nel costruire effettivamente un automa a stati finiti che riconosca il linguaggio denotato da una data espressione regolare, il secondo detto *problema di analisi*, consiste nel trovare un'espressione regolare che denoti il linguaggio riconosciuto da un dato automa a stati finiti.

7.1 Sintesi

Osserviamo innanzitutto che i linguaggi denotati dalle espressioni regolari $\emptyset$ e a, $a \in \Sigma$, sono riconosciuti rispettivamente dagli automi

Osserviamo che tali automi sono provvisti di un unico stato finale. Nel seguito, per praticità, considereremo esclusivamente automi con un unico stato finale. Come vedremo, questa scelta non è restrittiva.

Per maneggiare espressioni regolari più complesse, ci sarà utile il lemma seguente.

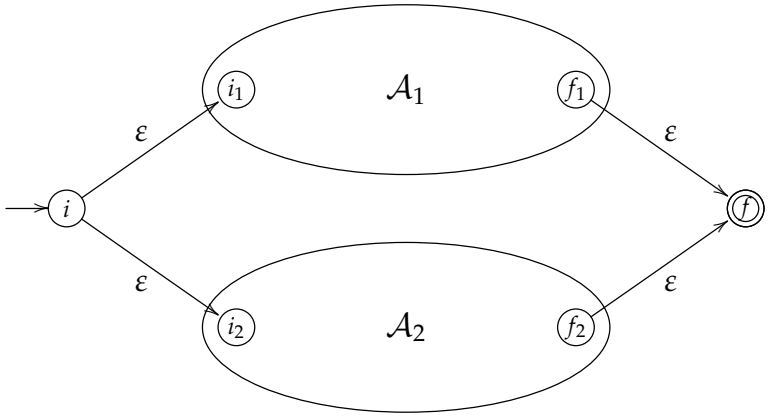
Lemma 2 *Siano $\mathcal{A}_1$ e $\mathcal{A}_2$ due automi non deterministici con ε-transizioni provvisti di un unico stato finale e siano rispettivamente L_1 e L_2 i linguaggi accettati da tali automi.*

È effettivamente possibile costruire degli automi non deterministici con ε-transizioni provvisti di un unico stato finale che riconoscono rispettivamente i linguaggi $L_1 \cup L_2$, $L_1 L_2$, L_1^*.

DIMOSTRAZIONE: Si ponga $\mathcal{A}_1 = \langle Q_1, \Sigma, \delta_1, i_1, \{f_1\}\rangle$ e $\mathcal{A}_2 = \langle Q_2, \Sigma, \delta_2, i_2, \{f_2\}\rangle$.

Possiamo supporre, senza perdita di generalità, che gli insiemi Q_1 e Q_2 siano disgiunti. Costruiamo un nuovo automa $\mathcal{A} = \langle Q, \Sigma, \delta, i, \{f\}\rangle$ nel modo seguente:

- $Q = Q_1 \cup Q_2 \cup \{i, f\}$, ove i, f sono due nuovi stati,
- i e f sono, rispettivamente lo stato iniziale e l'unico stato finale dell'automa $\mathcal{A}$.
- gli archi del grafo di $\mathcal{A}$ sono gli archi dei grafi di $\mathcal{A}_1$ e $\mathcal{A}_2$ e, inoltre, i quattro archi (i, ε, i_1), (i, ε, i_2), (f_1, ε, f), (f_2, ε, f).



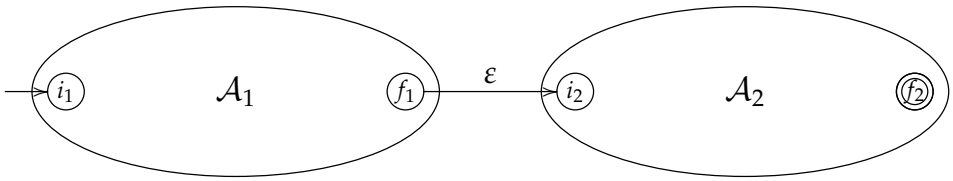
Risulta evidente che i cammini nel grafo di $\mathcal{A}$ dallo stato iniziale i allo stato finale f sono esattamente i cammini costituiti

- da una ε-transizione da i a i_1 seguita da un cammino di $\mathcal{A}_1$ da i_1 a f_1 e da una ε-transizione finale da f_1 a f
- oppure da una ε-transizione da i a i_2 seguita da un cammino di $\mathcal{A}_2$ da i_2 a f_2 e da una ε-transizione finale da f_2 a f.

Quindi le parole accettate da $\mathcal{A}$ sono esattamente quelle accettate da $\mathcal{A}_1$ o da $\mathcal{A}_2$. Se ne conclude che $\mathcal{A}$ riconosce il linguaggio $L_1 \cup L_2$.

Per riconoscere la concatenazione $L_1 L_2$ definiamo un nuovo automa $\mathcal{A}' = \langle Q, \Sigma, \delta', i_1, \{f_2\}\rangle$ nel modo seguente:

- $Q = Q_1 \cup Q_2$,
- lo stato iniziale di $\mathcal{A}_1$ e lo stato finale di $\mathcal{A}_2$ sono, rispettivamente lo stato iniziale e l'unico stato finale dell'automa $\mathcal{A}'$.
- gli archi del grafo di $\mathcal{A}$ sono gli archi dei grafi di $\mathcal{A}_1$ e $\mathcal{A}_2$ con l'aggiunta dell'arco (f_1, ε, i_2).



In queste condizioni, i cammini nel grafo di $\mathcal{A}'$ con origine in i_1 e termine in f_2 sono quelli che si ottengono concatenando un cammino di $\mathcal{A}_1$ da i_1 a f_1, la ε-transizione da f_1 a i_2 e un cammino di $\mathcal{A}_2$ da i_2 a f_2. Quindi le parole accettate da $\mathcal{A}'$ sono quelle che si ottengono concatenando una parola accettata da $\mathcal{A}_1$ con una parola accettata da $\mathcal{A}_2$. Si ha cioè $L(\mathcal{A}') = L_1 L_2$.

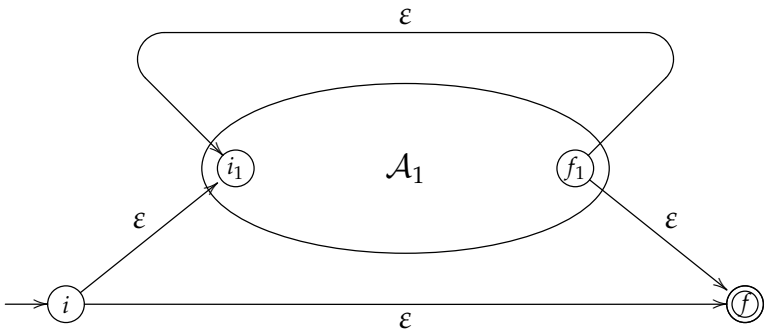
Occupiamoci infine del linguaggio L_1^*. Iniziamo a considerare l'automa $\bar{\mathcal{A}}$ ottenuto connettendo lo stato finale di $\mathcal{A}_1$ al suo stato iniziale con una ε-transizione.

I cammini di $\bar{\mathcal{A}}$ con origine in i_1 e termine in f_1 sono quelli che si ottengono concatenando un numero arbitrario di volte un cammino di $\mathcal{A}_1$ da i_1 a f_1 con una ε-transizione da f_1 a i_1 e teminando con un'ultimo cammino di $\mathcal{A}_1$ da i_1 a f_1. Ne segue che le parole accettate da $\bar{\mathcal{A}}$ sono esattamente quelle che si ottengono concatenando un numero arbitrario, ma positivo, di parole accettate da $\mathcal{A}_1$. Si ha quindi $L_1^* = L(\bar{\mathcal{A}}) \cup \{\varepsilon\}$.

Per completare la nostra costruzione ci basta quindi modificare $\bar{\mathcal{A}}$ in modo da accettare anche la parola vuota. Per fare questo ci basta aggiungere un nuovo stato iniziale i, un nuovo stato finale f, e connettere tramite ε-transizioni i a i_1, f_1 a f e i a f. In questo modo otteniamo l'automa $\mathcal{A}'' = \langle Q, \Sigma, \delta'', i, \{f\}\rangle$ definito nel modo seguente:

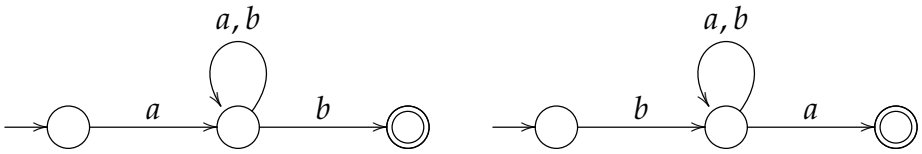
- $Q = Q_1 \cup \{i, f\}$, ove i, f sono due nuovi stati,
- i e f sono, rispettivamente lo stato iniziale e l'unico stato finale dell'automa $\mathcal{A}''$.
- gli archi del grafo di $\mathcal{A}$ sono gli archi dei grafi di $\mathcal{A}_1$ e, inoltre, i quattro archi (f_1, ε, i_1), (i, ε, i_1), (f_1, ε, f), (i, ε, f).

Tale automa riconosce il linguaggio L_1^*.

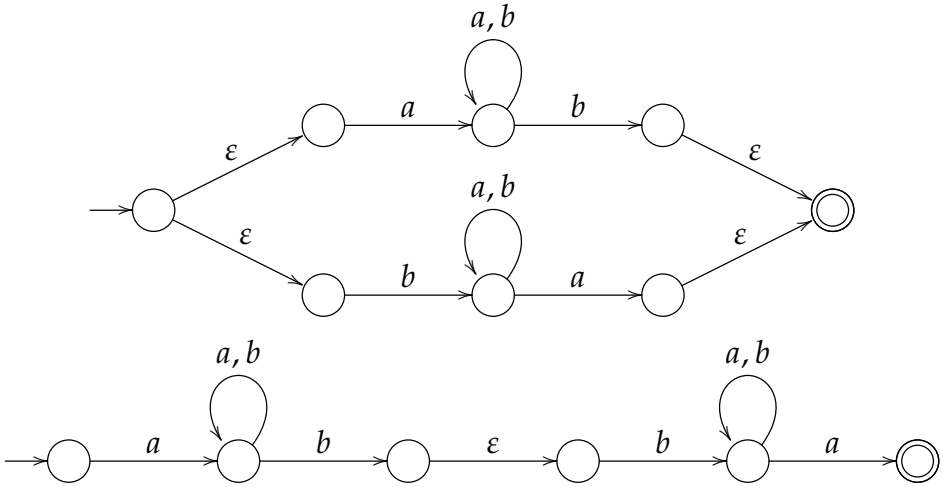


□

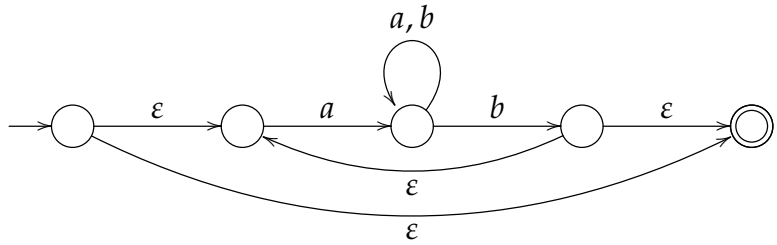
Esempio 20 Si considerino i seguenti due automi sull'alfabeto $\Sigma = \{a, b\}$:



Essi riconoscono rispettivamente i linguaggi $L_1 = \{aub \mid u \in \Sigma^*\}$ e $L_2 = \{bua \mid u \in \Sigma^*\}$. I linguaggi $L_1 \cup L_2$ e $L_1 L_2$ sono riconosciuti rispettivamente dagli automi:



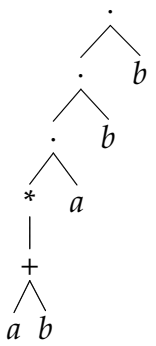
Il linguaggio L_1^* è riconosciutio dall'automa:



Corollario 1 *Data un'espressione regolare E, si può effettivamente costruire un automa a stati finiti che riconosce il linguaggio denotato da E. Si può inoltre supporre che tale automa abbia un unico stato finale.*

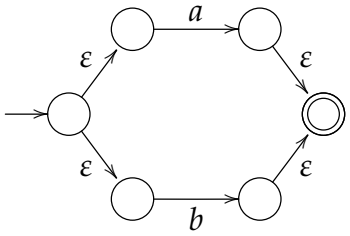
DIMOSTRAZIONE: Se $E = a$ con $a \in \Sigma$ o $E = \emptyset$, si è già visto che il linguaggio denotato da E è riconosciuto da un automa non deterministico con ε-transizioni con un unico stato finale. In ogni altro caso, si avrà $E = (F + G)$, oppure $E = (FG)$, oppure $E = F^*$, ove F e G sono espressioni regolari di lunghezza minore di E. Procedendo ricorsivamente, supponiamo di avere costruito due automi non deterministici con ε-transizioni con un unico stato finale che riconoscono rispettivamente il linguaggio L_F denotato da F e il linguaggio L_G denotato da G. Il linguaggio L_E denotato da E sarà allora uno dei linguaggi $L_F \cup L_G$, $L_F L_G$, o L_F^*. Il Lemma 2 ci assicura che è possibile costruire un automa provvisto di un unico stato finale che accetta L_E a partire dagli automi che accettano L_F e L_G. □

Esempio 21 Si consideri l'espressione regolare $(a + b)^*abb$. Tenuto conto dell'ordine di precedenza degli operatori, tale espressione regolare sarebbe meglio espressa come $((((a + b)^*a)b)b)$. A questa espressione possiamo associare il seguente albero che descrive come è stata ottenuta applicando le regole (i) e (ii) della Definizione 23.

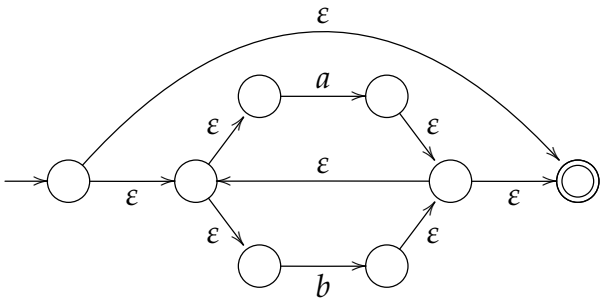


Gli automi associati alle espressioni regolari a e b sono rispettivamente

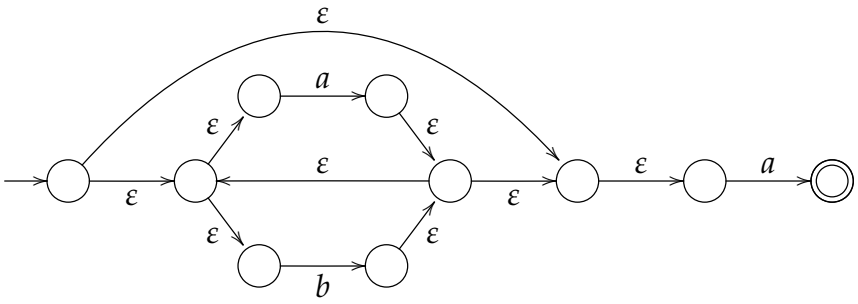
L'automa associato all'espressione regolare $(a + b)$ sarà quindi



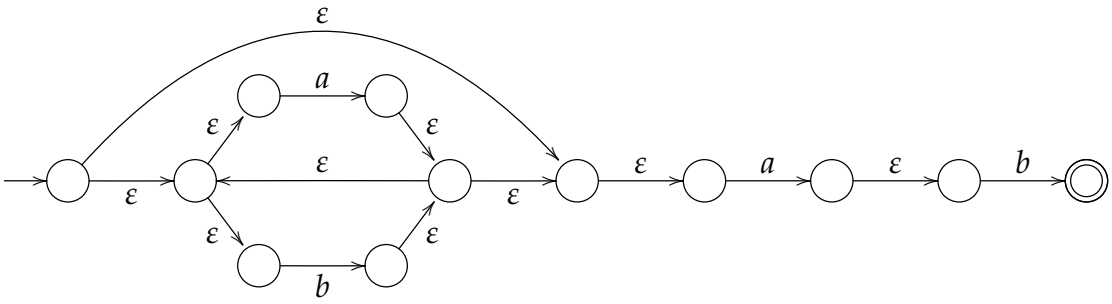
L'automa associato all'espressione regolare $(a+b)^*$ sarà poi



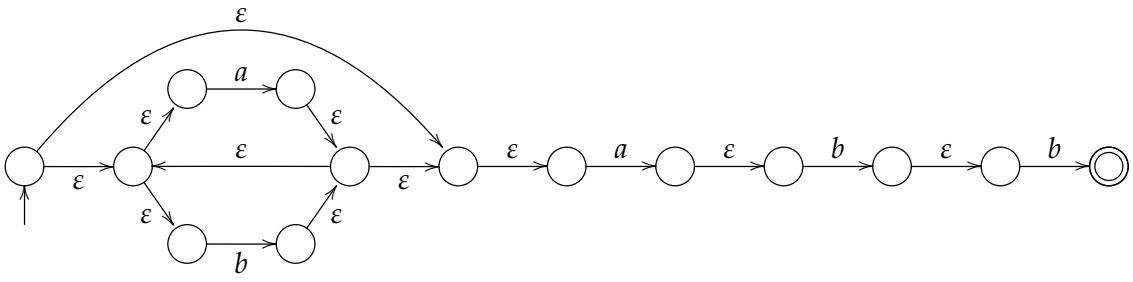
Ora costruiamo l'automa associato all'espressione regolare $((a+b)^*a)$:



poi l'automa associato all'espressione regolare $(((a+b)^*a)b)$:



e, infine, l'automa associato all'espressione regolare $((((a+b)^*a)b)b)$:



7.2 Analisi

Passiamo ora al problema di analisi che, come detto, consiste nel trovare un'espressione regolare che denota il linguaggio riconosciuto da un dato auoma a stati finiti. Sia dunque $\mathcal{A}$ un automa a stati finiti non deterministico con ε-transizioni. Possiamo supporre, senza perdita di generalità, che l'insieme degli stati sia

$$Q = \{q_1, q_2, \ldots, q_n\}, \quad n \geq 1,$$

e che lo stato iniziale sia q_1.

Si considerino i cammini nel grafo dell'automa $\mathcal{A}$ con origine e termine in due stati fissati e che attraversano esclusivamente stati di indice $\leq k$. Le etichette di tali cammini costituiscono un linguaggio per il quale cercheremo di trovare un'espressione regolare.

Osserviamo che nel caso $k = n$, non vi è nessuna restrizione sugli stati intermedi del cammino. Se riusciamo nel nostro intento, avremo quindi ottenuto anche delle espressioni regolari per le etichette di tutti i cammini con origine e termine fissati. Non sarà difficile, a questo punto, trovare un'espressione regolare per il linguaggio riconosciuto dall'automa.

Lemma 3 *Per $i, j = 1, 2, \ldots, n$, $k = 0, 1, \ldots, n$ si può determinare effettivamente un'espressione regolare E_{ijk} che denota l'insieme delle etichette dei cammini nel grafo dell'automa $\mathcal{A}$ con origine in q_i, termine in q_j e che attraversano esclusivamente stati di indice $\leq k$.*

DIMOSTRAZIONE: Procediamo per induzione su k.

Base: $k = 0$. Poiché non ci sono stati con indice ≤ 0, un cammino con origine in q_i, termine in q_j e che attraversa esclusivamente stati di indice ≤ 0 deve essere necessariamente un arco diretto da q_i a q_j. Un'espressione regolare E_{ij0} per tale linguaggio sarà quindi definita come segue:

se non ci sono archi da q_i a q_j, allora $E_{ij0} = \emptyset$; in caso contrario, dette $a_1, \ldots, a_m$ le etichette degli archi da q_i a q_j nel grafo dell'automa $\mathcal{A}$, si ha $E_{ij0} = a_1 + \cdots + a_m$,

Passo induttivo. Sia $k > 0$ e supponiamo di avere già calcolato le espressioni regolari E_{ijk-1} per $i, j = 1, 2, \ldots, n$.

I cammini nel grafo di $\mathcal{A}$ con origine in q_i, termine in q_j e che attraversano esclusivamente stati di indice $\leq k$ si possono ripartire in due categorie:

- i cammini con origine in q_i, termine in q_j e che attraversano esclusivamente stati di indice $\leq k - 1$
- i cammini costituiti dalla concatenazione di (vedi Fig. 7.1)

Figura 7.1: Calcolo di E_{ijk}

- un segmento iniziale da q_i a q_k che attraversa esclusivamente stati di indice $\leq k-1$,
- zero o più cammini con origine e termine in q_k e che attraversano esclusivamente stati di indice $\leq k-1$,
- un segmento finale da q_k a q_j che attraversa esclusivamente stati di indice $\leq k-1$.

Per l'induzione, possiamo supporre che i linguaggi costituiti dalle etichette dei cammini dei quattro tipi considerati siano denotati, rispettivamente, dalle espressioni regolari

$$E_{ijk-1}, \quad E_{ikk-1}, \quad E_{kkk-1}, \quad E_{kjk-1}.$$

Ne segue che l'espressione regolare

$$E_{ijk} = E_{ijk-1} + (E_{ikk-1})(E_{kkk-1})^*(E_{kjk-1})\,. \tag{7.1}$$

denota proprio il linguaggio costituito dalle etichette dei cammini da q_i a q_j che attraversano esclusivamente stati di indice $\leq k$. □

Il lemma precedente ci permette di risolvere il nostro problema di analisi.

Corollario 2 *Dato un automa (non deterministico con ε-transizioni) $\mathcal{A}$ si può effettivamente calcolare un'espressione regolare E che denota il linguaggio $L(\mathcal{A})$.*

DIMOSTRAZIONE: Sia n il numero degli stati dell'automa $\mathcal{A}$. Allora, l'espressione regolare E_{ijn} introdotta nel lemma precedente denota il linguaggio costituito dalle etichette dei cammini da q_i a q_j senza alcuna restrizione sugli stati attraversati.

Possiamo supporre, senza perdità di generalità, che lo stato iniziale sia q_1. Consideriamo ora la somma delle espressioni E_{1jn} con $q_j \in F$

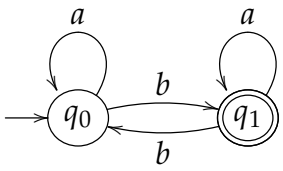
$$\sum_{q_j \in F} E_{1jn}\,. \tag{7.2}$$

Questa denoterà il linguaggio costituito dalle etichette dei cammini con origine nello stato iniziale q_1 e termine in uno stato finale. Pertanto, almeno nel caso

in cui $q_1 \notin F$, l'espressione regolare così ottenuta denota il linguaggio riconosciuto dall'automa $\mathcal{A}$. Chiaramente tale espressione può essere effettivamente calcolata a partire dalle E_{ij0} usando ricorsivamente l'Equazione (7.1).

Nel caso in cui lo stato iniziale q_1 sia anche uno stato finale, all'espressione regolare (7.2) andrà sommata l'espressione $\emptyset^*$. Invero, in tal caso l'automa accetta anche la parola vuota, che essendo etichetta di un 'cammino di lunghezza 0' non è catturata dalle formule (7.1). □

Esempio 22 Si consideri l'automa



Si ha

$$E_{110} = E_{220} = a + \emptyset^*\,, \quad E_{120} = E_{210} = b\,.$$

Tenendo conto delle (7.1) si ottiene

$$\begin{aligned} E_{121} &= E_{120} + (E_{110})(E_{110})^*(E_{120}) = b + (a+\emptyset^*)(a+\emptyset^*)^*b\,,\\ E_{221} &= E_{220} + (E_{210})(E_{110})^*(E_{120}) = a + \emptyset^* + b(a+\emptyset^*)^*b\,, \end{aligned}$$

e quindi tenendo conto della (7.2) si ottiene per il linguaggio riconosciuto da $\mathcal{A}$ l'espressione regolare seguente

$$\begin{aligned} E = E_{122} &= E_{121} + (E_{121})\big(E_{221}\big)^*(E_{221})\\ &= b + (a+\emptyset^*)(a+\emptyset^*)^*b + (b + (a+\emptyset^*)(a+\emptyset^*)^*b)\big(a+\\ &\qquad \emptyset^* + b(a+\emptyset^*)^*b\big)^*(a+\emptyset^* + b(a+\emptyset^*)^*b)\,. \end{aligned}$$

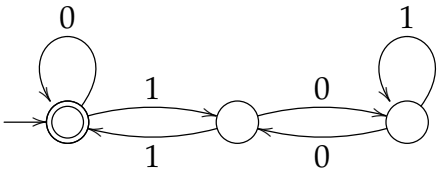
In realtà, si potrebbe verificare che anche l'espressione $(a+ba^*b)^*ba^*$, molto più breve, denota lo stesso linguaggio.

Se ci si riduce, come è sempre possibile, al caso in cui l'automa $\mathcal{A}$ abbia un unico stato finale q_n, che questo sia privo di frecce uscenti, che lo stato iniziale sia q_{n-1} e che non vi siano frecce entranti nello stato iniziale, allora risulta $L(\mathcal{A}) = E_{n-1\,n\,n-2}$. Tale espressione può essere calcolata con la seguente procedura:

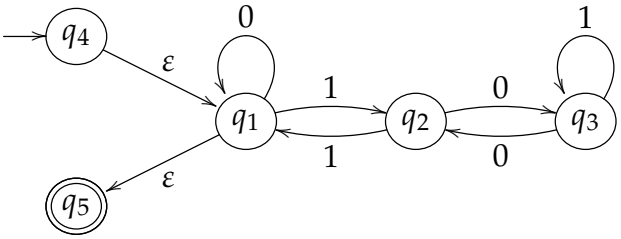
In realtà, l'aggiornamento (**1**) del valore di E_{ij} va eseguito solo se né E_{ik} né E_{kj} denotano il linguaggio vuoto. Per una maggior efficienza può quindi essere utile tenere traccia degli E_{ij} non vuoti su un grafo diretto (*metodo di eliminazione degli stati*), come nell'esempio seguente.

per $i \leftarrow 1, \ldots, n-1$ **fai**
  per $j \leftarrow 1, \ldots, n$ **fai**
    $E_{ij} \leftarrow E_{ij0}$
per $k \leftarrow 1, \ldots, n-2$ **fai**
  per $i \leftarrow k+1, \ldots, n-1$ **fai**
    per $j \leftarrow k+1, \ldots, n$ **fai**
(1) $E_{ij} \leftarrow E_{ij} + E_{ik}E_{kk}^{*}E_{kj}$
ritorna $E_{n-1,n}$

Esempio 23 Vogliamo determinare un'espressione regolare per il linguaggio accettato dall'automa seguente:



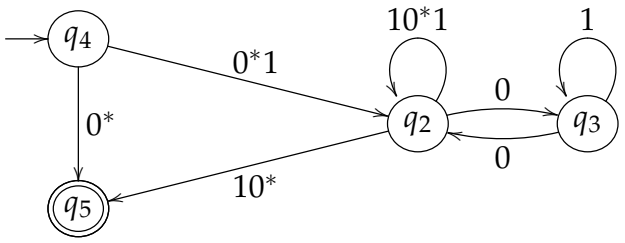
Come primo passo, ci ricondurremo al caso in cui vi sia un unico stato finale q_n, che questo sia privo di frecce uscenti, che lo stato iniziale sia q_{n-1} e che non vi siano frecce entranti nello stato iniziale. Per ottenere questo risultato è sufficiente aggiungere due nuovi stati, che fungeranno rispettivamente da stato iniziale e da stato finale e due ε-transizioni, una dal nuovo al vecchio stato iniziale e l'altra dal vecchio al nuovo stato finale. Inoltre numereremo gli stati in modo che i due indici massimi siano assegnati rispettivamente allo stato iniziale e allo stato finale.



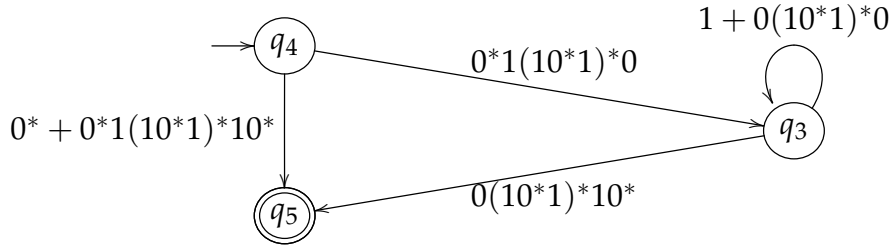
Possiamo riguardare le etichette degli archi come gli E_{ij0} introdotti nella dimostrazione del Lemma 3. L'eventuale assenza di una freccia fra due stati i e j comporta $E_{ij0} = \emptyset$.

A questo punto procediamo come segue: per ogni freccia $q_i \xrightarrow{E_{i1}} q_1 1$ che entra nello stato q_1 e ogni freccia $q_1 \xrightarrow{E_{1j}} q_j$ che esce da tale stato ($i, j \neq 1$) rimpiazziamo l'etichetta della freccia $q_i \xrightarrow{E_{ij}} q_j$ con $E_{ij} + E_{i1}E_{11}^{*}E_{1j}$. Se tale freccia è assente, la creiamo (con etichetta $E_{i1}E_{11}^{*}E_{1j}$). Dopo di ciò rimuoviamo lo stato q_1 e tutte le

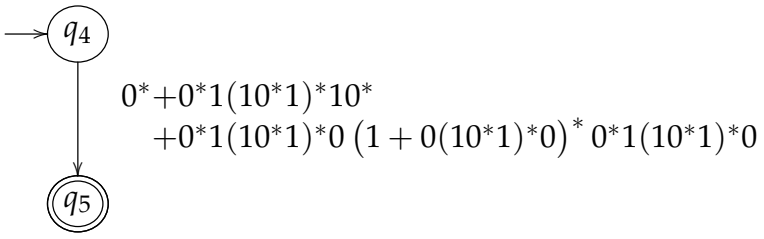
frecce che entrano o escono da esso. Otteniamo così il grafo seguente:



Ripetiamo la medesima operazione per lo stato q_2. Otteniamo allora il grafo



Infine ripetendo l'operazione per lo stato q_3, otteniamo



che ci dà l'espressione regolare cercata.

7.3 Conseguenze del teorema di Kleene

Il teorema di Kleene ci permette di dimostrare che i linguaggi regolari costituiscono un'algebra Booleana:

Proposizione 1 *L'unione e l'intersezione di due linguaggi regolari sono linguaggi regolari. Il complemento di un linguaggio regolare è un linguaggio regolare.*

DIMOSTRAZIONE: Siano L_1 e L_2 due linguaggi regolari. Dette rispettivamente E_1 e E_2 due espressioni regolari che li denotano, l'espressione regolare $(E_1 + E_2)$ denota il linguaggio $L_1 \cup L_2$, che pertanto è regolare.

Ora mostriamo che il complemento di un linguaggio regolare L è anch'esso regolare. Invero, per il teorema di Kleene, L è riconosciuto da un automa a stati

finiti deterministico $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$. Si verifica facilmente che l'automa $\mathcal{A}' = \langle Q, \Sigma, \delta, q_0, Q - F \rangle$ riconosce il linguaggio complementare $\Sigma^* - L$. Di nuovo per il teorema di Kleene, questo comporta che $\Sigma^* - L$ è un linguaggio regolare.

Infine consideriamo l'intersezione di due linguaggi regolari L_1 e L_2. Poiché, come è ben noto, l'intersezione si può esprimere mediante l'unione e il complemento

$$L_1 \cap L_2 = \Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2)),$$

dalle osservazioni precedenti segue che $L_1 \cap L_2$ è regolare. □

Come applicazione della proprietà precedente, otteniamo una procedura per verificare l'equivalenza di due espressioni regolari.

Proposizione 2 *Si può decidere effettivamente se due espressioni regolari denotano lo stesso linguaggio.*

DIMOSTRAZIONE: Iniziamo a descrivere una procedura effettiva per decidere se un'espressione regolare E denota il linguaggio vuoto. È sufficiente costruire l'automa a stati finiti che accetta il linguaggio denotato da E e verificare se nel grafo di tale automa c'è almeno un cammino dallo stato iniziale a uno stato finale.

Ora diamo una procedura effettiva per decidere se il linguaggio denotato da una data espressione regolare E è incluso nel linguaggio denotato da un'altra espressione regolare F. Siano rispettivamente con L_E e L_F i linguaggi denotati da E e F, osserviamo che si ha $L_E \subseteq L_F$ se e solo se L_E non interseca il complemento di L_F, cioè se e solo se il linguaggio $L_E \cap (\Sigma^* - L_F)$ è vuoto. Per quanto visto, tale linguaggio è regolare e si può effettivamente costruire un automa a stati finiti che lo riconosce. A questo punto, con la procedura vista in precedenza possiamo decidere se è vuoto o meno. Nel primo caso, L_E è incluso in L_F, nel secondo caso non è incluso.

Finalmente vediamo come decidere se due espressioni regolari E e F denotano lo stesso linguaggio: è sufficiente verificare se il linguaggio denotato da E è incluso in quello denotato da F e viceversa. □

Capitolo 8

Automa minimo

In questo capitolo, ci poniamo l'obbiettivo di costruire, per un dato linguaggio regolare L, l'automa deterministico col minimo numero di stati che riconosce L.

Iniziamo richiamando la nozione di congruenza destra.

Definizione 25 Sia Σ un alfabeto. Una relazione di equivalenza $\sim$ su Σ^* si dice una *congruenza destra* (risp., *sinistra*) se per ogni $u, v \in \Sigma^*$ tali che $u \sim v$ e per ogni lettera $a \in \Sigma$, si ha $ua \sim va$ (risp., $au \sim av$).

Una relazione che sia contemporaneamente una congruenza destra e una congruenza sinistra si dice *congruenza*.

Introduciamo ora un esempio notevole di congruenza destra.

Definizione 26 Sia L un linguaggio sull'alfabeto Σ. L'*equivalenza di Nerode* associata al linguaggio L è la relazione $\mathcal{N}_L$ definita in Σ^* da

$$u \,\mathcal{N}_L\, v \quad \text{se per ogni } y \in \Sigma^*,\ uy \in L \Longleftrightarrow vy \in L\,.$$

In altri termini, due parole $u, v \in \Sigma^*$ sono nella relazione $\mathcal{N}_L$ se hanno gli stessi 'completamenti' a destra in L.

È evidente che la relazione $\mathcal{N}_L$ è un'equivalenza. Mostriamo ora che si tratta in realtà di una congruenza destra.

Lemma 4 *Sia $L \subseteq \Sigma^*$ un linguaggio.*

1. *La relazione $\mathcal{N}_L$ è una congruenza destra,*
2. *L è unione di classi di equivalenza di $\mathcal{N}_L$,*

DIMOSTRAZIONE: 1. È evidente che $\mathcal{N}_L$ è un'equivalenza. Per mostrare che è una congruenza destra, dobbiamo verificare che se $u, v \in \Sigma^*$ sono tali che $u \,\mathcal{N}_L$

v, allora per ogni lettera $a \in \Sigma$ si ha $ua \mathcal{N}_L va$. Invero sia $u \mathcal{N}_L v$, $a \in \Sigma$ e $z \in \Sigma^*$. Dalla definizione di $\mathcal{N}_L$ (prendendo $y = az$) segue che $u\,az \in L \iff v\,az \in L$. Quindi si ha anche $ua \mathcal{N}_L va$.

2. Se $u \mathcal{N}_L v$, allora dalla definizione di $\mathcal{N}_L$ (prendendo $y = \varepsilon$), si ottiene $u \in L \iff v \in L$. Quindi due parole equivalenti o stanno entrambe in L o stanno entrambe fuori da L. Se ne conclude che ogni classe di equivalenza è interamente inclusa o in L o nel suo complemento. Pertanto L è unione di classi di equivalenza di $\mathcal{N}_L$. □

Ora enunciamo il fondamentale Teorema di Myhill-Nerode che dà una caratterizzazione algebrica dei linguaggi regolari. Ricordiamo che il numero di classi (eventualmente infinito) di un'equivalenza è detto *indice*.

Teorema 5 (Myhill-Nerode, 1958) *Sia L un linguaggio sull'alfabeto Σ. Le seguenti proposizioni sono equivalenti:*

(i) *L è regolare,*

(ii) *L è unione di classi di una congruenza destra su Σ^* di indice finito,*

(iii) *$\mathcal{N}_L$ ha indice finito.*

DIMOSTRAZIONE: Dimostreremo circolarmente che $(i) \Rightarrow (ii) \Rightarrow (iii) \Rightarrow (i)$.

$(i) \Rightarrow (ii)$. Sia L un linguaggio regolare. Per il Teorema di Kleene e il Teorema 2, L è riconosciuto da un automa deterministico $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$. Come si è visto nel Capitolo 6, possiamo supporre, senza perdita di generalità, che tutti gli stati di $\mathcal{A}$ siano accessibili.

Per ogni stato $q \in Q$ consideriamo il linguaggio

$$L_q = \{w \in \Sigma^* \mid \widehat{\delta}(q_0, w) = q\}\,.$$

È chiaro che gli insiemi L_q costituiscono una partizione di Σ^* e che L e l'unione delle classi L_q con $q \in F$. Pertanto, per completare la dimostrazione, è sufficiente mostrare che l'equivalenza $\sim$ associata a tale partizione è una congruenza destra.

Supponiamo quindi $u \sim v$. Allora sono in una stessa classe L_q, $q \in Q$. Pertanto, $\widehat{\delta}(q_0, u) = \widehat{\delta}(q_0, v) = q$ e quindi, per ogni $a \in \Sigma$, $\widehat{\delta}(q_0, ua) = \widehat{\delta}(q_0, va) = \delta(q, a)$. Se ne conclude che ua, va sono entrambi nella classe $L_{\widehat{\delta}(q,z)}$ e quindi $uz \sim vz$.

Questo prova che $\sim$ è una congruenza destra e, pertanto, la (ii) è verificata.

$(ii) \Rightarrow (iii)$. Supponiamo che L sia unione di classi di una congruenza destra $\sim$ su Σ^* di indice finito. Per dimostrare che la Proposizione (iii) è verificata basterà far vedere che ogni classe di $\mathcal{N}_L$ è unione di classi di $\sim$.

In effetti, dato che $\sim$ è una congruenza destra, se $u \sim v$ allora per ogni $a \in \Sigma$ si ha $ua \sim va$. Più in generale, risulterà $uy \sim vy$ per ogni parola $y \in \Sigma^*$. Allora, tenendo conto del fatto che L è unione di classi di $\sim$, o uy e vy sono entrambe in L o nessuna delle due sta in L. In altri termini, si ha $uy \in L \iff vy \in L$ e quindi $u \; \mathcal{N}_L \; v$.

Riassumendo, abbiamo fatto vedere che se $u \sim v$ allora $u \; \mathcal{N}_L \; v$. Questo prova che ogni classe dell'equivalenza $\sim$ è interamente contenuta in una classe di $\mathcal{N}_L$. Pertanto, ogni classe di $\mathcal{N}_L$ è unione di classi di $\sim$ e quindi l'indice di $\mathcal{N}_L$ è necessariamente finito e anzi minore o uguale a quello di $\sim$.

$(iii) \Rightarrow (i)$. Supponiamo che $\mathcal{N}_L$ abbia indice finito. Possiamo allora considerare l'automa $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ in cui:

- gli stati sono le classi dell'equivalenza $\mathcal{N}_L$,
- la funzione di transizione δ è definita nel modo seguente:
  Siano $q \in Q$ e $a \in \Sigma$. Poiché q è una classe dell'equivalenza $\mathcal{N}_L$, tutte le parole ua con $u \in q$ appartengono a un'unica classe p dell'equivalenza $\mathcal{N}_L$. Poniamo allora $\delta(q, a) = p$.
- lo stato iniziale q_0 è la classe d'equivalenza della parola vuota,
- gli stati finali sono le classi d'equivalenza incluse in L.

Tenendo presente che q_0 contiene la parola vuota, si verifica facilmente che per ogni $w \in \Sigma^*$, $\widehat{\delta}(q_0, w)$ contiene la parola w.

Ricordando che L è unione di classi dell'equivalenza $\mathcal{N}_L$, ne segue che se $w \in L$, allora $\widehat{\delta}(q_0, w)$ è incluso in L e quindi è uno stato finale, mentre se $w \notin L$, allora $\widehat{\delta}(q_0, w)$ è incluso nel complemento di L e quindi non è uno stato finale. Se ne conclude che il linguaggio riconosciuto da $\mathcal{A}$ è proprio L. □

L'automa introdotto nella dimostrazione dell'implicazione $(iii) \Rightarrow (i)$ prende il nome di *automa di Nerode* del linguaggio L. Come si è visto, l'automa di Nerode di L ha un numero di stati pari all'indice dell'equivalenza $\mathcal{N}_L$. Dalla precedente dimostrazione possiamo dedurre la seguente notevole proprietà:

Proposizione 3 *Sia L un linguaggio regolare. L'automa di Nerode di L è un automa deterministico che riconosce L col minimo numero di stati possibile.*

DIMOSTRAZIONE: Verificando l'implicazione $(i) \Rightarrow (ii)$ abbiamo osservato che se L è riconosciuto da un automa deterministico $\mathcal{A}$, allora L è unione di classi di una congruenza destra $\sim$ su Σ^* di indice minore o uguale al numero degli stati di $\mathcal{A}$. D'altra parte come osservato verificando l'implicazione $(ii) \Rightarrow (iii)$,

l'indice di $\mathcal{N}_L$ è sicuramente minore o uguale all'indice di $\sim$. Ne concludiamo che l'indice di $\mathcal{N}_L$ è minore o uguale al numero di stati di qualsiasi automa deterministico che riconosce L.

D'altra parte, verificando l'implicazione $(iii) \Rightarrow (i)$, abbiamo verificato che l'automa di Nerode di L riconosce L e ha un numero di stati esattamente uguale all'indice di $\mathcal{N}_L$. Questo conclude la dimostrazione. □

Ora consideriamo il seguente problema (*minimizzazione*): dato un automa a stati finiti deterministico $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, costruire un automa deterministico equivalente ad $\mathcal{A}$ col minimo numero di stati.

Il primo passo consite nel ridursi al caso in cui tutti gli stati siano accessibili. La procedura per ottenere ciò consiste nel selezionare dal grafo di $\mathcal{A}$ gli stati accessibili da q_0 e poi eliminare gli stati non accessibili e restringere di conseguenza la funzione di transizione.

Dalla dimostrazione del Teorema di Nerode, si evince che ciascuna classe della congruenza $\mathcal{N}_L$ è unione di linguaggi L_q, con $q \in Q$. In altri termini, la partizione di Σ^* nelle classi di $\mathcal{N}_L$ induce una partizione di Q. Determinare le classi di tale partizione indotta, che per la finitezza di Q sono semplici liste di stati, ci permetterà di costruire effettivamente l'automa di Nerode.

Osserviamo che la partizione indotta da $\mathcal{N}_L$ su Q gode delle seguenti tre proprietà:

1. l'insieme F è unione di classi,
2. se p e q sono nella medesima classe, allora per ogni lettera $a \in \Sigma$, $\delta(p, a)$ e $\delta(q, a)$ cadono in una stessa classe,
3. ogni partizione di Q che soddisfi le due condizioni precedenti è un raffinamento della partizione indotta da $\mathcal{N}_L$ su Q.

La dimostrazione, peraltro non difficile, delle proprietà suddette è lasciata al lettore.

Il problema della minimizzazione si ridurrà pertanto alla ricerca della meno fine partizione di Q che soddisfa tali condizioni. Tale partizione è approssimata dalla successione di partizioni Π_n di Q definite come segue:

- La partizione Π_0 ha le sole due classi F e $Q - F$.
- per ogni $n \geq 0$, le classi di Π_{n+1} si ottengono spezzando quelle di Π_n in modo da 'separare' le coppie di stati che non soddisfano la Condizione 2.

La costruzione di Π_{n+1} a partire da Π_n è realizzata dalla Procedura Raffina.

Procedura Raffina(Π)

$\Pi_{\text{nuovo}} \leftarrow (\,)$;
per ciascun $S \in \Pi$ **fai**
  per ciascun $q \in S$ **fai**
    per ciascun $a \in \Sigma$ **fai**
      $C_{qa} \leftarrow$ classe di $\delta(q, a)$;
    $C_q \leftarrow (C_{qa})_{a \in \Sigma}$;
  $\mathcal{L} \leftarrow \{C_q \mid q \in S\}$;
  per ciascun $\ell \in \mathcal{L}$ **fai**
    appendi $\{q \mid C_q = \ell\}$ a Π_{nuovo} ;
$\Pi \leftarrow \Pi_{\text{nuovo}}$;

Possiamo applicare questa procedura ripetutamente, partendo da $\Pi_0 = (Q - F, F)$ ottenendo via via le partizioni $\Pi_1, \Pi_2, \ldots$. Poiché Q è finito e quindi anche le sue partizioni sono in numero finito, ci sarà un intero $m \geq 0$ per cui $\Pi_m = \Pi_{m+1}$. La partizione $\Pi = \Pi_m$ è quella cercata (vedi Algoritmo 4).

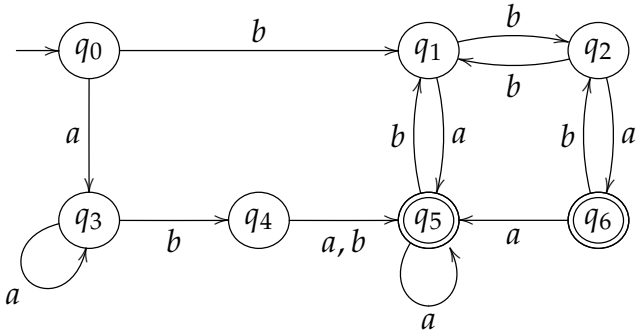
A questo punto l'automa minimo

$$\mathcal{A}' = \langle Q', \Sigma, \delta', \mathcal{C}_0, F' \rangle$$

del linguaggio L si ottiene nel modo seguente

- gli stati sono le classi della partizione Π,
- per ogni classe $\mathcal{C}$ e ogni lettera $a \in \Sigma$, $\delta'(\mathcal{C}, a)$ è la classe $\mathcal{C}'$ che contiene tutti gli stati $\delta(q, a)$ con $q \in \mathcal{C}$,
- lo stato iniziale è la classe $\mathcal{C}_0$ che contiene lo stato iniziale q_0 di $\mathcal{A}$,
- gli stati finali sono le classi contenute in F.

Esempio 24 Si consideri l'automa



Algoritmo 4: Minimizzazione

Ingresso: Un automa deterministico $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$;
Uscita: La partizione Π associata a $\sim$;

$\Pi \leftarrow (Q - F, F)$;
ripeti
  $L \leftarrow \text{Card}(\Pi)$;
  `Raffina` (Π)
finché $L = \text{Card}(\Pi)$;

La partizione iniziale è $\Pi = (Q - F, F) = (\{q_0, q_1, q_2, q_3, q_4\}, \{q_5, q_6\})$. Nella tabella seguente sono riportate le classi di $\delta(q, a), \delta(q, b)$, con $q \in Q$.

	0					1	
	q_0	q_1	q_2	q_3	q_4	q_5	q_6
a	0	1	1	0	1	1	1
b	0	0	0	0	1	0	0

Con l'esecuzione di `Raffina` (Π) la classe $\{q_0, q_1, q_2, q_3, q_4\}$ si 'spezza' nelle tre classi $\{q_0, q_3\}$, $\{q_1, q_2\}$, $\{q_4\}$. Si ottiene così la nuova partizione

$$\Pi = (\{q_0, q_3\}, \{q_1, q_2\}, \{q_4\}, \{q_5, q_6\}).$$

Le classi di $\delta(q, a), \delta(q, b)$, con $q \in Q$ sono riportate nella tabella seguente.

	0		1		2	3	
	q_0	q_3	q_1	q_2	q_4	q_5	q_6
a	0	0	3	3	3	3	3
b	1	2	1	1	3	1	1

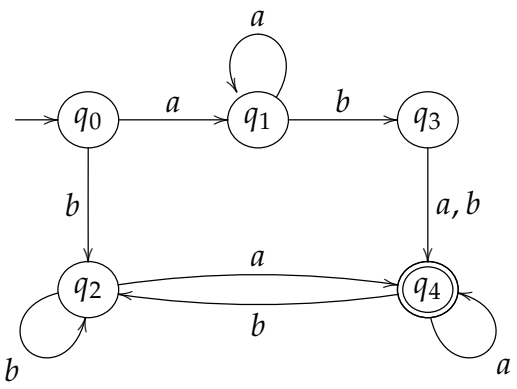
L'ulteriore esecuzione di `Raffina` (Π) 'spezza' la classe $\{q_0, q_3\}$ nei due singleton $\{q_0\}$ e $\{q_3\}$. Si ottiene così la nuova partizione

$$\Pi = (\{q_0\}, \{q_3\}, \{q_1, q_2\}, \{q_4\}, \{q_5, q_6\}).$$

Calcoliamo di nuovo le classi di $\delta(q, a),\ \delta(q, b)$, con $q \in Q$.

	0	1	2		3	4	
	q_0	q_3	q_1	q_2	q_4	q_5	q_6
a	1	1	4	4	4	4	4
b	2	3	2	2	4	2	2

Ora, l'esecuzione di `Raffina` (Π) non modifica ulteriormente Π, che quindi è la partizione cercata. Si ottiene così l'automa



equivalente a quello di partenza.

Capitolo 9

Grammatiche di tipo 3

Ricordiamo (cf. Definizione 12) la definizione di grammatica di tipo 3.

Definizione 27 Una grammatica a struttura di frase $G = \langle V, \Sigma, P, S \rangle$ è *di tipo 3* se tutte le produzioni hanno le forme

$$X \to aY, \quad X \to a, \qquad X, Y \in N,\ a \in \Sigma\,.$$

Nel caso in cui S non compaia nei lati destri delle produzioni è ammessa anche la produzione $S \to \varepsilon$.

Osserviamo che se non fosse ammessa la produzione $S \to \varepsilon$, nessun linguaggio di tipo 3 potrebbe contenere la parola vuota. Ciò darebbe luogo a varie difficoltà tecniche. D'altra parte, la condizione che S non compaia nei lati destri delle produzioni ci assicura che la produzione $S \to \varepsilon$ non può essere utilizzata nella derivazione di qualsiasi parola diversa da ε.

Esempio 25 Si consideri la grammatica $G = \langle V, \Sigma, P, S \rangle$ con $\Sigma = \{a, b\}$, $V = \Sigma \cup \{S, X\}$ e con le produzioni

$$S \to aS\,, \quad S \to bX\,, \quad S \to b\,, \quad X \to aX\,, \quad X \to bS\,, \quad X \to a\,.$$

Si ha ad esempio, $abbab \in L(G)$. Tale grammatica è di tipo 3.

Il nostro prossimo obbiettivo è mostrare che la classe dei linguaggi di tipo 3 coincide con quella dei linguaggi accettati dagli automi a stati finiti.

Teorema 6 *Un linguaggio è accettato da un automa a stati finiti se e soltanto se è generato da una grammatica di tipo 3.*

DIMOSTRAZIONE: Per prima cosa, proveremo che per ogni grammatica $G = \langle V, \Sigma, P, S\rangle$ di tipo 3 esiste un automa a stati finiti (non deterministico) $\mathcal{A}$ tale che

$$L(\mathcal{A}) = L(G)\,.$$

Per semplicità, supponiamo $\varepsilon \notin L(G)$. Costruiamo l'automa $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ nel modo seguente:

- Gli stati sono le variabili della grammatica G e la parola vuota,
- La funzione di transizione è definita da

$$\delta(X, a) = \{Y \in N \cup \{\varepsilon\} \mid X \to aY \text{ in } P\}, \quad \delta(\varepsilon, a) = \emptyset\,,$$

  $X \in N,\ a \in \Sigma$,
- L'unico stato finale è ε.

Iniziamo a verificare l'inclusione $L(\mathcal{A}) \subseteq L(G)$.

Invero, sia w una parola accettata da $\mathcal{A}$. Scriviamo $w = a_1 a_2 \cdots a_n$, con $a_1, a_2, \ldots, a_n \in \Sigma$, $n \geq 1$. Per la Definizione 16, dovranno esistere $X_1, \ldots, X_n \in Q$ tali che

$$X_i \in \delta(X_{i-1}, a_i)\,,\ 1 \leq i \leq n\,, \quad X_0 = S, \quad X_n \in F. \tag{9.1}$$

Dalla definizione di $\mathcal{A}$ segue che in G ci sono le produzioni $X_{i-1} \to a_i X_i$, $1 \leq i \leq n$ e, inoltre, $X_n = \varepsilon$. Si avrà pertanto

$$S = X_0 \Rightarrow a_1 X_1 \Rightarrow a_1 a_2 X_2 \Rightarrow \cdots \Rightarrow a_1 \cdots a_n X_n = w\,. \tag{9.2}$$

e quindi $w \in L(G)$.

Verifichiamo ora l'inclusione $L(G) \subseteq L(\mathcal{A})$. Sia quindi w una parola generata da G. Data la forma delle produzioni delle grammatiche di tipo 3, una derivazione di w dovrà avere necessariamente la forma (9.2) con $X_{i-1} \to a_i X_i$ in P, $1 \leq i \leq n$ e con $X_n = \varepsilon$. Dalla definizione dell'automa $\mathcal{A}$ segue allora che sono verificate le (9.1) e, pertanto, w è accettata da $\mathcal{A}$.

Possiamo quindi concludere che $L(G) = L(\mathcal{A})$.

Nel caso in cui $\varepsilon \in L(G)$, sarebbe necessario modificare l'automa testé costruito in modo da fargli accettare anche la parola vuota. Per fare ciò, in questo caso, è sufficiente aggiungere S agli stati finali.

Abbiamo così dimostrato che ogni linguaggio di tipo 3 è accettato da un automa a stati finiti.

Ora procediamo alla costruzione inversa. Mostreremo che per ogni automa a stati finiti $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ esiste una grammatica di tipo 3 $G = \langle V, \Sigma, P, S\rangle$ tale che

$$L(G) = L(\mathcal{A})\,.$$

Di nuovo, per semplicità, supponiamo $\varepsilon \notin L(\mathcal{A})$. Costruiamo la grammatica G nel modo seguente:

- Le variabili della grammatica G sono gli stati dell'automa $\mathcal{A}$,
- Le produzioni sono:

$$\begin{aligned} &X \to aY, && \text{per ogni } X \in Q,\ a \in \Sigma\,,\ Y \in \delta(X,a)\,, \\ &X \to a\,, && \text{per ogni } X \in Q,\ a \in \Sigma \text{ tali che } \delta(X,a) \cap F \neq \emptyset\,. \end{aligned}$$

- Il simbolo iniziale è lo stato iniziale $S = q_0$.

Iniziamo a verificare l'inclusione $L(\mathcal{A}) \subseteq L(G)$.

Invero, sia w una parola accettata da $\mathcal{A}$. Scriviamo $w = a_1 a_2 \cdots a_n$, con $a_1, a_2, \ldots, a_n \in \Sigma$, $n \geq 1$. Per la Definizione 16, le (9.1) dovranno essere verificate per opportune $X_0, \ldots, X_n \in Q$. Dalla definizione di G segue che in P ci sono le produzioni $X_{i-1} \to a_i X_i$, $1 \leq i \leq n-1$ nonché la produzione $X_{n-1} \to a_n$. Si avrà pertanto

$$S = X_0 \Rightarrow a_1 X_1 \Rightarrow a_1 a_2 X_2 \Rightarrow \cdots \Rightarrow a_1 \cdots a_{n-1} X_n \Rightarrow a_1 \cdots a_n = w\,, \tag{9.3}$$

e quindi $w \in L(G)$.

Verifichiamo ora l'inclusione $L(G) \subseteq L(\mathcal{A})$. Sia quindi w una parola generata da G. Data la forma delle produzioni delle grammatiche di tipo 3, una derivazione di w dovrà avere necessariamente la forma (9.3) con le produzioni $X_{i-1} \to a_i X_i$, $1 \leq i \leq n-1$ e $X_{n-1} \to a_n$ in P. Dalla definizione dell'automa $\mathcal{A}$ segue allora che sono verificate le (9.1) e, pertanto, w è accettata da $\mathcal{A}$.

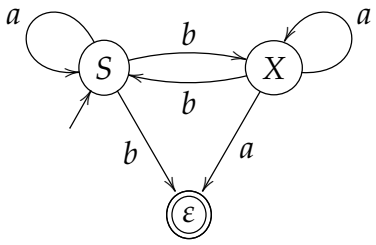
Possiamo quindi concludere che $L(G) = L(\mathcal{A})$.

Nel caso in cui $\varepsilon \in L(\mathcal{A})$, sarebbe necessario modificare la grammatica G in modo da fargli generare anche la parola vuota. Se però S compare nel lato destro di qualche produzione, non è possibile aggiungere la produzione $S \to \varepsilon$. Dovremo pertanto prima introdurre una nuova variabile S' le cui produzioni avranno gli stessi lati destri delle produzioni di S, prendere S' come simbolo iniziale al posto di S e, infine, aggiungere la produzione $S' \to \varepsilon$.

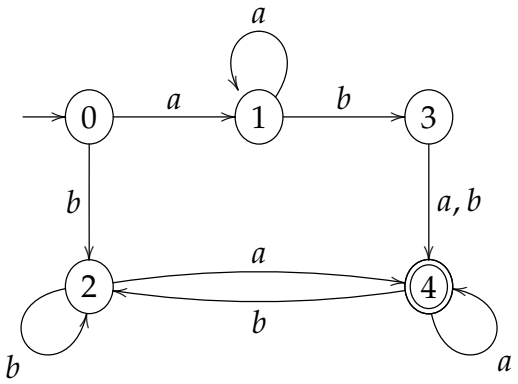
Abbiamo così dimostrato che ogni linguaggio accettato da un automa a stati finiti è di tipo 3. □

Esempio 26 Si consideri la grammatica dell'Esempio 25. Con la costruzione introdotta nella dimostrazione precedente otteniamo l'automa non deterministico

col grafo seguente



Esempio 27 Si consideri l'automa



Ridenominando rispettivamente X_0, X_1, X_2, X_3, X_4 gli stati $0, 1, 2, 3, 4$ e procedendo come nella dimostrazione precedente, si ottiene la grammatica con assioma X_0 e produzioni:

$X_0 \to aX_1$	$X_1 \to bX_3$	$X_2 \to bX_2$	$X_3 \to b$	$X_4 \to a$
$X_0 \to bX_2$	$X_2 \to a$	$X_3 \to a$	$X_3 \to bX_4$	$X_4 \to aX_4$
$X_1 \to aX_1$	$X_2 \to aX_4$	$X_3 \to aX_4$	$X_4 \to bX_2$	

Il linguaggio generato da tale grammatica è esattamente il linguaggio accettato dal nostro automa.

Una nozione più generale è quella di *grammatica lineare destra*. una grammatica si dice *lineare* se tutte le produzioni hanno la forma

$$X \to uYv \quad \text{o} \quad X \to u,$$

con $X, Y \in N$ e $u, v \in T^*$. Una grammatica si dice *lineare destra* se tutte le produzioni hanno la forma

$$X \to uY \quad \text{o} \quad X \to u,$$

con $X, Y \in N$ e $u \in T^*$.

È evidente che tutte le grammatiche di tipo 3 sono lineari destre. Viceversa, si può dimostrare che ogni grammatica lineare destra genera un linguaggio di tipo 3. Vale quindi il seguente

Teorema 7 *Un linguaggio è accettato da un automa a stati finiti se e solo se è generato da una grammatica lineare destra.*

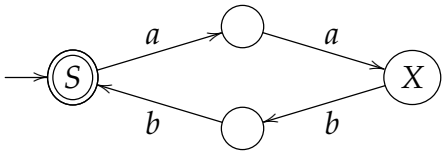
Esempio 28 Si consideri la grammatica con le variabili S e X e le produzioni

$$S \to aaX\,, \quad S \to \varepsilon\,, \quad X \to bbS\,.$$

Questa grammatica genera il linguaggio $L = \{(aabb)^n \mid n \geq 0\}$. Non è difficile verificare che il medesimo linguaggio è generato dalla grammatica di tipo 3 con le variabili S, S', X, Y, Z, simbolo iniziale S' e le produzioni

$$S \to aY\,, \quad Y \to aX\,, \quad S' \to \varepsilon\,, \quad S' \to aY\,, \qquad X \to bZ\,, \quad Z \to bS\,.$$

ed è il linguaggio accettato dall'automa



Capitolo 10

Lemma di iterazione

In questo paragrafo stabiliremo una condizione necessariamente soddisfatta da tutte le parole abbastanza lunghe di un linguaggio regolare.

Teorema 8 (Lemma di iterazione) *Sia L un linguaggio regolare. Esiste un intero n tale che ogni parola $w \in L$ di lunghezza $|w| \geq n$ si può fattorizzare $w = xyz$ con $y \neq \varepsilon$ e $xy^*z \subseteq L$.*

Esempio 29 Si consideri il linguaggio $L = ab^*a$. Ogni parola $w \in L$ di lunghezza $|w| \geq 3$ ha la forma $w = ab^na$ con $n > 0$. Pertanto, posto $x = z = a$, $y = b^n$, si ha $w = xyz$ e $xy^*z = a(b^n)^*a = \{ab^{kn}a \mid k \geq 0\} \subseteq ab^*a$.

Esempio 30 Si consideri il linguaggio $L = \{a^nb^n \mid n \geq 0\}$. Supponiamo che, per opportuni n, x, y, z si abbia

$$a^nb^n = xyz\,, \quad y \neq \varepsilon\,.$$

Se $|x| \geq n$, allora si ha $y = b^k$ per un opportuno $k > 0$ e $xz = a^nb^{n-k} \notin L$. Simmetricamente, se $|z| \geq n$, allora si ha $y = a^k$ e $xz = a^{n-k}b^n \notin L$. Infine, se $|x|, |z| < n$, allora si ha $x = a^{n-h}$, $y = a^hb^k$, $z = b^{n-k}$ per opportuni $h, k > 0$ e quindi $xy^2z = a^{n-h}a^hb^ka^hb^kb^{n-k} = a^nb^ka^hb^n \notin L$. In ogni caso si ha $xy^*z \not\subseteq L$. Per il lemma precedente, concludiamo che L non può essere un linguaggio regolare.

Esempio 31 Si consideri il linguaggio $L = \{a^p \mid p \text{ primo}\}$. Utilizzeremo il Lemma di iterazione per mostrare che L non è un linguaggio regolare. Invero, se L fosse regolare, allora per un primo p sufficientemente grande, si avrebbe

$$a^p = xyz\,, \quad xy^*z \subseteq L\,, \quad y \neq \varepsilon\,,$$

per opportuni $x, y, z \in a^*$. Scriviamo $x = a^i$, $y = a^j$, $z = a^k$, $i, k \geq 0$, $j > 0$. Allora avremmo $xz = a^{i+k} \in L$ e $xy^{i+k}z = a^{(i+k)(j+1)} \in L$. Ne seguirebbe che gli

interi $i+k$ e $(i+k)(j+1)$, sono entrambi primi, il che è un assurdo, visto che $j+1 \geq 2$.

DIMOSTRAZIONE DEL TEOREMA 8: Per il Teorema di Kleene, il linguaggio L è riconosciuto da un automa (che supporremo deterministico) $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$. Poniamo $n = \mathrm{Card}(Q)$.

Sia $w \in L$ una parola di lunghezza $k > n$ e scriviamo $w = a_1 a_2 \cdots a_k$, con $a_1, a_2, \ldots, a_k \in \Sigma$. Poniamo poi

$$q_1 = \delta(q_0, a_1)\,,\ q_2 = \delta(q_1, a_2)\,,\ \ldots\,,\ q_k = \delta(q_{k-1}, a_k)\,. \tag{10.1}$$

Poiché $w \in L$, si ha $q_k = \delta(q_0, w) \in F$. Dato che $k \geq n$, esisteranno degli indici i, j con $0 \leq i < j \leq k$ tali che $q_i = q_j$. Poniamo ora

$$x = a_1 a_2 \cdots a_i\,, \quad y = a_{i+1} a_{i+2} \cdots a_j\,, \quad z = a_{j+1} a_{j+2} \cdots a_k\,.$$

Abbiamo quindi $w = xyz$ e $y \neq \varepsilon$. Per completare la dimostrazione basta quindi far vedere che $xy^*z \subseteq L$. Dalla (10.1) si ottiene facilmente che

$$\delta(q_0, x) = q_i\,, \quad \delta(q_i, y) = q_j = q_i\,, \quad \delta(q_j, z) = q_k\,.$$

Ne segue che per ogni $m \geq 0$,

$$\widehat{\delta}(q_0, xy^m z) = \widehat{\delta}(q_i, y^m z) = \widehat{\delta}(q_i, z) = \widehat{\delta}(q_j, z) = q_k \in F\,.$$

Conseguentemente, $xy^m z \in L$. Questo dimostra che $xy^*z \subseteq L$. □

Capitolo 11

Analisi lessicale

Un codice sorgente è costituito da una sequenza di simboli elementari, detti *lessemi*. I lessemi possono essere raggruppati in categorie lessicali dette *token*.

Per esempio, nel linguaggio C, potremmo distinguere tra gli altri, i token della seguente tabella

Token	Esempi di lessemi
int	int
identificatore	valore, totale, main, ...
costante	3, 2.10, ...
stringa letterale	"digita un numero", "Mario", ...

La prima fase della compilazione è l'*analisi lessicale* nella quale il compilatore riconosce la sequenza dei token che costituisce il codice. Tale compito viene svolto, in genere, da un modulo del compilatore (lo *scanner*). Tale modulo implementa una funzione che, invocata dall'analizzatore sintattico, ritorna il token successivo. Sebbene in linea di principio sarebbe possibile inglobare l'individuazione dei token all'interno dell'analisi sintattica stessa, si preferisce una implementazione separata che assicura semplicità di programmazione, portabilità e, soprattutto, maggiore efficienza, in quanto, come vedremo, per l'analisi lessicale è possibile adottare tecniche specifiche, basate sugli automi a stati finiti, che non si adattano invece all'analisi sintattica.

Come si è visto differenti lessemi identificano uno stesso token. Consideriamo ad esempio il seguente frammento di codice C:

```
int a = 5;
int b = 3.2;
```

Qui i lessemi `a` e `b` corrispondono al token "identificatore" e i lessemi `5` e `3.2` corrispondono al token "costante". Quindi `a` e `b` (come pure `5` e `3.2`) sono perfettamente equivalenti per l'analizzatore sintattico, ma andranno distinti durante l'analisi semantica.

Quando succede questo, è necessario associare al token un attributo (nell'esempio, il nome di un identificatore o il valore di una costante) che potrà essere utilizzato in futuro dal compilatore. Gli attributi vengono immagazzinati nella *tabella dei simboli*.

Vediamo ora, più in dettaglio, qual'è il compito dell'analisi lessicale. I token sono identificati da espressioni regolari, i *pattern*, che identificano l'insieme dei lessemi che vi appartengono.

Ad esempio, se indichiamo con A l'insieme dei caratteri alfabetici e con C l'insieme delle cifre, un identificatore del linguaggio C è descritto dal pattern $A(A + C)^*$.

Vi è inoltre un ordine di priorità fra i pattern. L'analizzatore lessicale deve identificare il più lungo prefisso del testo da analizzare che sia un lessema e ritornare il primo (nell'ordine di priorità stabilito) fra i token corrispondenti a tale lessema; fatto ciò passerà a esaminare il testo rimanente.

Esaminiamo l'espressione C

```
int interesse = 5 ;
```

Il più lungo lessema iniziale è `int`, che corrisponde ai pattern dei token "int" e "identificatore". Evidentemente, "int" avrà priorità maggiore, di modo che l'analizzatore ritorna "int" e, alla successiva attivazione analizzerà

```
interesse = 5 ;
```

Anche questa espressione inizia con il lessema `int`, ma anche con il lessema `interesse`, che corrisponde al token "identificatore" e ha lunghezza maggiore. Pertanto l'analizzatore lessicale deve ritornare "identificatore" e, alla successiva attivazione analizzerà "`= 5 ;`".

Vediamo ora come sia possibile utilizzare la teoria degli automi a stati finiti per realizzare in maniera efficiente l'analisi lessicale. I dati del nostro problema sono una sequenza finita di espressioni regolari $E_1, E_2, \ldots, E_n$ (i pattern) e una parola w (il testo da analizzare). Dobbiamo determinare il più lungo prefisso u di w che appartenga al linguaggio denotato dall'espressione regolare $F = E_1 + E_2 + \cdots + E_n$ nonché il più piccolo indice j tale che u appartiene al linguaggio denotato da E_j.

Come sappiamo (cf. Sez. 7.1), per ognuna delle espressioni E_j, $j = 1, \ldots, n$ è possibile costruire un automa finito

$$\mathcal{A}_j = \langle Q_j, \Sigma, E, \{i_j\}, \{f_j\} \rangle$$

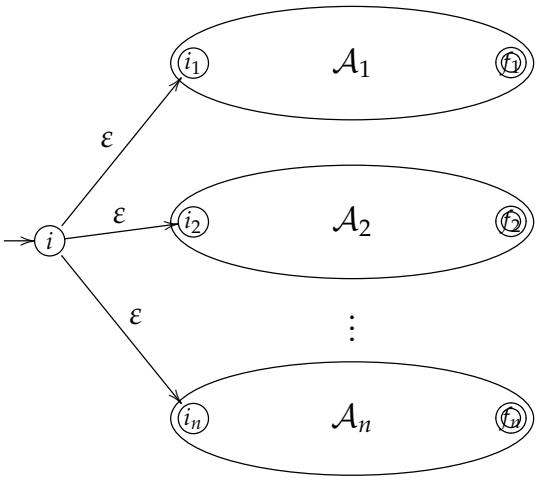
che riconosce il linguaggio denotato da E_j. Possiamo supporre senza perdita di generalità che gli insiemi Q_j siano a due a due disgiunti. Possiamo costruire un'automa che accetta il linguaggio denotato dall'espressione regolare

F adattando la tecnica introdotta nella Sez. 7.1 nel modo seguente: Costruiamo l'automa $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ con

$$Q = \{i_0\} \cup \bigcup_{j=0}^{n} Q_j\,, \quad F = \{f_j \mid 1 \le j \le n\}\,,$$

$$\delta(q,a) = \begin{cases} \delta_j(q,a)\,, & \text{se } q \in Q_j,\ 1 \le j \le n,\ a \in \Sigma \cup \{\varepsilon\}\,, \\ \{i_1, \ldots, i_n\}, & \text{se } q = i_0,\ a = \varepsilon\,, \\ \emptyset\,, & \text{se } q = i_0,\ a \in \Sigma\,. \end{cases}$$

In altri termini, l'automa $\mathcal{A}$ ha gli stati, le transizioni e gli stati finali di $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ e inoltre un nuovo stato iniziale i_0 e delle ε-transizioni da i_0 agli stati iniziali originali degli automi $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$.



L'algoritmo di determinizzazione (cf. Sez. 6.1) ci permette di ottenere da $\mathcal{A}$ un automa deterministico $\mathcal{A}_D = \langle P, \Sigma, \delta, q_0, F' \rangle$ che accetta il linguaggio denotato da F. Inoltre, gli stati di $\mathcal{A}_D$ sono sottoinsiemi di Q e si ha che w sta nel linguaggio E_j se e soltanto se $f_j \in \widehat{\delta}(q_0, u)$.

A questo punto possiamo utilizzare l'automa $\mathcal{A}_D$ per realizzare l'analisi lessicale. In effetti ci basta trovare il più lungo prefisso u di w accettato da $\mathcal{A}_D$ e il più piccolo indice j tale che $f_j \in \widehat{\delta}(q_0, u)$. Osserviamo che se per determinare u fosse necessario scandire l'intero testo w, la procedura sarebbe piuttosto inefficiente. Per fortuna,di solito, questo non è necessario. Invero, gli stati di $\mathcal{A}_D$ sono sottoinsiemi di Q e tra questi c'è l'insieme vuoto. Quando l'automa raggiunge questo stato, non ne uscirà più, quale che sia l'input (come suol dirsi, tale stato funge da *pozzo*). Possiamo quindi concludere che l'ultimo transito da uno stato finale prima dell'ingresso nel pozzo individua il più lungo prefisso di w accettato da $\mathcal{A}_D$.

Algoritmo 5: analisiLessicale

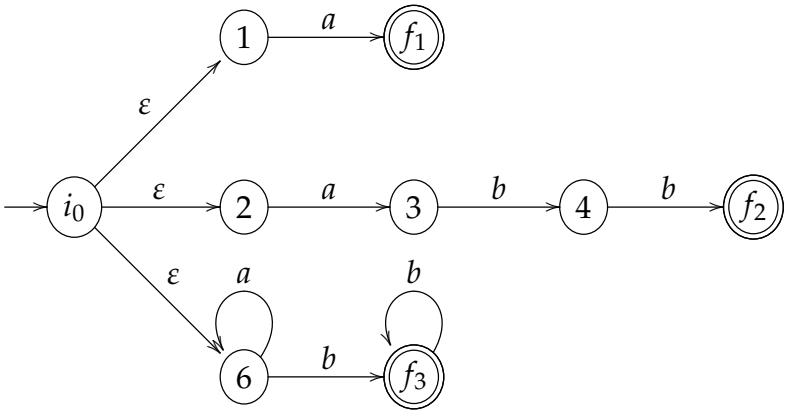
Ingresso: L'automa deterministico $\mathcal{A}_D$ e un puntatore start al testo da analizzare;

Uscita: la massima lunghezza di un prefisso del testo accettato da $\mathcal{A}_D$ e il token corrispondente;

$i \leftarrow$ start;
$q \leftarrow q_0$;
mentre $q \neq \emptyset$ **fai**
    $q \leftarrow \delta(q, *i)$;
    se $q \in F$ **allora**
        token $\leftarrow \min\{j \mid f_j \in q\}$;
        lunghezza $\leftarrow i + 1 -$ start;
    $i \leftarrow i + 1$;
start $\leftarrow$ start + lunghezza; `// ripartirà dalla lettera seguente`

Otteniamo così il seguente algoritmo.

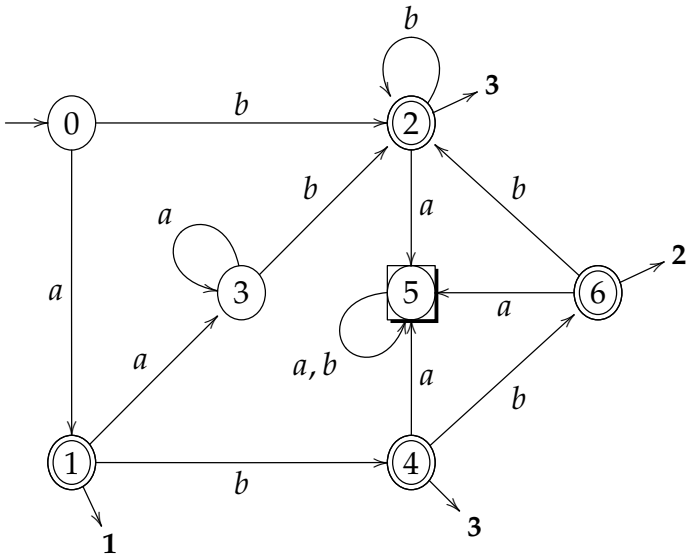
Esempio 32 Consideriamo i pattern $E_1 = a$, $E_2 = abb$, $E_3 = a^*bb^*$. L'automa $\mathcal{A}$ sarà allora



Applichiamo l'algoritmo di determinizzazione. Nell'ultima colonna, per ogni stato finale indichiamo l'indice del pattern corrispondente.

	q	$\delta(q,a)$	$\delta(q,b)$	token
0	i_0, 1, 2, 6	1	2	
1	f_1, 3, 6	3	4	1
2	f_3	5	2	3
3	6	3	2	
4	4, f_3	5	6	3
5	$\emptyset$	5	5	
6	f_2, f_3	5	2	2

Possiamo disegnare il grafo dell'automa, aggiungendo agli stati finali una freccia uscente con l'indicazione del pattern corispondente. Lo stato 5, corrispondente all'insieme vuoto, è denotato da un quadrato.



Per realizzare un analizzatore lessicale ci si può avvalere di programmi come *flex* [1]. Tale programma riceve in input una lista di espressioni regolari e produce il codice C del corrispondente analizzatore lessicale. In pratica, *flex* costruisce l'automa $\mathcal{A}_D$ e il codice C dell'Algoritmo 5.

[1] Il lettore interessato può trovare una guida introduttiva a *flex* all'URL epaperpress.com/lexandyacc/

Parte III

Linguaggi non contestuali

Capitolo 12

Grammatiche non contestuali

Esempio 33 Ricordiamo che una parola $w \in \Sigma$ si dice palindroma se si ha

$$w = a_1 a_2 \cdots a_n = a_n a_{n-1} \cdots a_1$$

per opportuni $n \geq 0$, $a_1, a_2, \ldots, a_n \in \Sigma$. Sia P il linguaggio delle palindrome sull'alfabeto $\Sigma = \{a, b\}$. Osserviamo che:

1. ε, a, b sono palindrome,
2. per ogni palindroma w, le parole awa e bwb sono palindrome,
3. ogni palindroma su Σ si ottiene applicando un numero finito di volte le regole 1. e 2.

Ne segue che il linguaggio P 'soddisfa' l'identità

$$P = \varepsilon + a + b + aPa + bPb \tag{12.1}$$

e anzi tale identità determina univocamente il linguaggio P. Per esempio, se $X \in P$, anche aXa, $aaXaa$, $aabXbaa \in P$ e, in particolare, prendendo $X = a$, $aababaa \in P$.

Le condizioni precedenti da un lato ci forniscono un algoritmo per verificare, ricorsivamente, se una data parola è palindroma o meno, dall'altro, possono essere viste come un metodo per 'generare' tutte le palindrome: riguardiamo X come una 'variabile' e sostituiamo, ripetutamente, X con ε, a, b, aXa o bXb, finchè non otteniamo una parola sull'alfabeto Σ.

Il procedimento descritto non è altro che la generazione di una parola in una grammatica non contestuale. Ricordiamo (cf. Definizione 11) che una grammatica a struttura di frase $G = \langle V, \Sigma, P, S \rangle$ si dice *non contestuale* se tutte le produzioni sono della forma

$$A \to \alpha$$

con $A \in N = V - \Sigma$ e $\alpha \in V^*$. Un linguaggio si dice *non contestuale* se esiste una grammatica non contestuale che lo genera.

Esempio 34 Vogliamo costruire una grammatica non contestuale che generi le palindrome sull'alfabeto $\Sigma = \{a, b\}$. Come si è visto, il linguaggio P delle palindrome su Σ è caratterizzato dall'Equazione (12.1). Pertanto, P è generato dalla grammatica non contestuale $G = \langle V, \Sigma, P, X\rangle$ con $V = \Sigma \cup \{X\}$ e con le produzioni

$$X \to \varepsilon \mid a \mid b \mid aXa \mid bXb\,.$$

In generale, il linguaggio delle palindrome su un alfabeto finito è non contestuale.

Esempio 35 Sia $G = \langle V, \Sigma, P, S\rangle$ la grammatica definita da

$$V = \{a, b, S\}\,, \quad \Sigma = \{a, b\}\,, \quad P = \{S \to \varepsilon,\ S \to aSb\}\,.$$

Si ha allora $S \Rightarrow aSb$, $aSb \Rightarrow aaSbb$, $aaSbb \Rightarrow aaaSbbb$, e quindi $S \Longrightarrow^* aaaSbbb$. Non è difficile verificare che le forme sentenziali di questa grammatica sono tutte le parole della forma a^nSb^n o a^nb^n con $n \geq 0$. Quindi $L(G) = \{a^nb^n \mid n \geq 0\}$. Pertanto il linguaggio $\{a^nb^n \mid n \geq 0\}$ è non contestuale. Come si ricorderà, tale linguaggio non è regolare (cf. Esempio 13).

Le espressioni "grammatica non contestuale" e "linguaggio non contestuale" sono spesso abbreviate con gli acronimi anglosassoni CFG e CFL.

È anche evidente che ogni grammatica di tipo 3 è una grammatica non contestuale. Pertanto, ogni linguaggio regolare è non-contestuale. D'altra parte il linguaggio non-contestuale generato dalla grammatica dell'Esempio 35 non è regolare. Ne deduciamo la seguente

Proposizione 4 *La classe dei linguaggi regolari è strettamente inclusa in quella dei linguaggi non-contestuali.*

Esempio 36 Si consideri la grammatica $G_n = \langle V, \Sigma_n, P, S\rangle$ con

$$\Sigma_n = \{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n\}\,, \quad V = \{S\} \cup \Sigma$$

e con le produzioni:

$$\begin{aligned} &S \to a_iSb_i\,, \quad i = 1, 2, \ldots n\,,\\ &S \to SS\,,\\ &S \to \varepsilon\,. \end{aligned}$$

Il linguaggio D_n generato da G_n è detto il linguaggio *semi-Dick* su n lettere. Se consideriamo ad esempio il caso $n = 2$ e poniamo $a_1 = (\,, b_1 =)\,, a_2 = [\,, b_2 =]\,,$ allora le parole

$$()\,,\ [\,]\,,\ [()]([\,])\,,\ (())[([\,]())]\,,$$

sono elementi di D_2, mentre $)($ non lo è. In effetti, D_2 risulta l'insieme delle sequenze di parentesi ben bilanciate.

Esempio 37 È possibile costruire una grammatica che genera le *espressioni regolari* sull'alfabeto $\{a, b\}$. Ricordiamo (cf. Definizione 23) che

1. $\emptyset$, a, b sono espressioni regolari,
2. Se S_1 e S_2 sono espressioni regolari, allora lo sono anche $(S_1 + S_2)$, $(S_1 S_2)$ e ${S_1}^*$,
3. Tutte le espressioni regolari sull'alfabeto $\{a, b\}$ si ottengono applicando un numero finito di volte le due regole precedenti.

Pertanto, la nostra grammatica $G = \langle V, \Sigma, P, S\rangle$ dovrà avere come alfabeto terminale

$$\Sigma = \{a\,,\ b\,,\ \emptyset\,,\ (\ ,\ )\,,\ +\,,\ ^*\,\}$$

come vocabolario totale $V = \Sigma \cup \{S\}$, e come produzioni

$$\begin{aligned} S &\to a & \qquad S &\to (S + S)\\ S &\to b & \qquad S &\to (SS)\\ S &\to \emptyset & \qquad S &\to S^* \end{aligned}$$

Per maggior coincisione, si usa raggruppare le produzioni di una grammatica non contestuale aventi lo stesso lato sinistro, separando i lati destri con una barra verticale. Ad esempio, le produzioni dell'Esempio 37 si possono indicare

$$S \to a \mid b \mid \emptyset \mid (S + S) \mid (SS) \mid S^*$$

Esempio 38 Si consideri il linguaggio

$$L = \{w \in \Sigma^* \mid |w|_a = |w|_b\}$$

sull'alfabeto $\Sigma = \{a, b\}$. Allo scopo di realizzare una grammatica non contestuale che generi L, consideriamo i linguaggi

$$L_a = \{w \in \Sigma^* \mid |w|_a = |w|_b + 1\}\,, \quad L_b = \{w \in \Sigma^* \mid |w|_b = |w|_a + 1\}\,.$$

Osserviamo che ogni parola $w \in L$ si scrive $w = ub$ con $u \in L_a$ o $w = ua$ con $u \in L_b$ o ancora $w = \varepsilon$.

Invece, ogni parola $w \in L_a$ si può scrivere $w = uau'$ con $u, u' \in L$. Invero, detto u il più lungo prefisso di w tale che $|u|_a \leq |u|_b$, ci si convince facilmente che u è seguito da una a e che $|u|_a = |u|_b$, sicchè $w = uau'$, per un'opportuna parola $u' \in \Sigma^*$. Ancora dalle condizioni $|u|_a = |u|_b$, $|uau'|_a = |uau'|_b + 1$ segue che $|u'|_a = |u'|_b$.

Analogamente, ogni parola $w \in L_b$ si può scrivere $w = ubu'$ con $u, u' \in L$. Pertanto gli insiemi L, L_a, L_b sono caratterizzati dal sistema di identità:

$$L = L_a b + L_b a + \varepsilon\,, \qquad L_a = LaL\,, \qquad L_b = LbL\,.$$

Se associamo ai linguaggi L, L_a, L_b rispettivamente la variabili S, A, B, otteniamo la grammatica $G = \langle V, \Sigma, P, S\rangle$ con $V = \{S, A, B, a, b\}$ e con le produzioni

$$S \to Ab \mid Ba \mid \varepsilon \qquad A \to SaS \qquad B \to SbS$$

Capitolo 13

Alberi di derivazione

In questo paragrafo introduciamo la nozione di albero di derivazione. Consideriamo alberi con radice i cui nodi sono etichettati. I figli di ciascun nodo sono ordinati (da sinistra a destra). Questo ordine si estende a un ordine parziale dei nodi dell'albero: un nodo p è minore ('a sinistra') di un nodo q se esistono un progenitore p' di p e un progenitore q' di q, figli di uno stesso nodo con p' a sinistra di q'.

Definizione 28 Sia $G = \langle V, \Sigma, P, S \rangle$ una grammatica non contestuale e T un albero. Diremo che T è un albero di derivazione della grammatica G se verifica le seguenti condizioni:

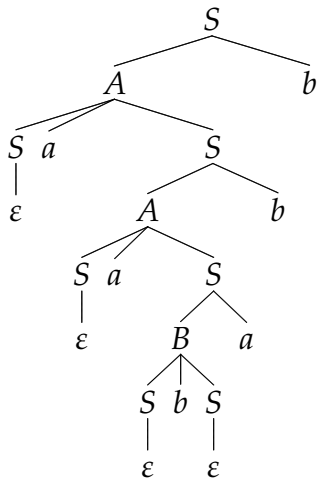
1. le etichette dei nodi interni sono elementi di N,
2. le etichette delle foglie sono elementi di $\Sigma \cup \{\varepsilon\}$,
3. le foglie con etichetta ε sono 'figli unici',
4. se un nodo con etichetta X ha figli etichettati nell'ordine $\alpha_1, \ldots, \alpha_k$, allora $X \to \alpha_1 \cdots \alpha_k$ è una produzione di G.

La parola che si ottiene leggendo, nell'ordine, le etichette delle foglie è la *parola associata* a T.

Esempio 39 Si consideri la grammatica dell'Esempio 38. La parola $w = aababb$ appartiene a $L(G)$. Invero una derivazione di w è

$$\begin{aligned} \underline{S} &\Rightarrow \underline{A}b \Rightarrow Sa\underline{S}b \Rightarrow Sa\underline{A}bb \Rightarrow SaSa\underline{S}bb \Rightarrow SaSa\underline{B}abb \\ &\Rightarrow \underline{S}aSaSbSabb \Rightarrow a\underline{S}aSbSabb \Rightarrow aa\underline{S}bSabb \\ &\Rightarrow aab\underline{S}abb \Rightarrow aababb \end{aligned}$$

ove abbiamo sottolineato la variabile da sostituire. A questa derivazione corrisponde l'albero



Vale la seguente

Proposizione 5 *Sia $G = \langle V, \Sigma, P, S\rangle$ una grammatica non contestuale e siano $A \in N$ e $w \in \Sigma^*$. Si ha $A \Longrightarrow^* w$ se e solo se esiste un albero di derivazione di G associato a w con la radice etichettata A.*

In particolare, w appartiene a $L(G)$ se e solo se esiste un albero di derivazione di G associato a w con la radice etichettata S.

Nel seguito, se non indicato diversamente, considereremo esclusivamente alberi di derivazione in cui l'etichetta della radice è l'assioma della grammatica.

Esempio 40 Si consideri la grammatica $G = \langle V, \Sigma, P, S\rangle$ ove

$$\Sigma = \{0, 1, *, +\}, \quad V = \Sigma \cup \{S\}$$

e con le produzioni

$$S \to S + S \mid S * S \mid 0 \mid 1$$

Si verifica facilmente che il linguaggio generato da questa grammatica è

$$L = \{x_0 o_1 x_1 o_2 \cdots o_n x_n \mid x_i = 0, 1,\ o_j = *, +, 0 \le i \le n,\ 1 \le j \le n,\ n \ge 0\}.$$

La parola $1 + 1 * 0$ ha due alberi di derivazione:

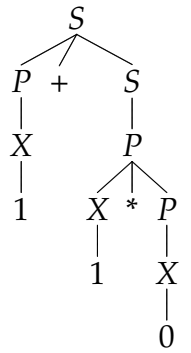
Dal punto di vista 'semantico' il primo corrisponde alla 'computazione' di $1 + (1 * 0)$ e il secondo alla 'computazione' di $(1 + 1) * 0$.

Definizione 29 Una grammatica si dice *non ambigua* se a ogni parola del linguaggio generato corrisponde un'unico albero di derivazione.

Esempio 41 La grammatica dell'Esempio 40 è equivalente alla grammatica $G' = \langle V', \Sigma, P', S \rangle$ ove $V = \Sigma \cup \{S, P, X\}$ con le produzioni

$$S \to P + S \mid P\,, \qquad P \to X * P \mid X\,, \qquad X \to 0 \mid 1\,.$$

Questa grammatica è non ambigua. Ciò segue dal fatto che per derivare una parola $w \in L$ si dovrà necessariamente utilizzare la produzione $S \to P + S$ esattamente tante volte quante sono le occorrenze di $+$ in w e la produzione $P \to X * P$ esattamente tante volte quante sono le occorrenze di $*$ in ciascun 'addendo' dell'espressione w. Ad esempio l'unico albero di derivazione di $1 + 1 * 0$ è



A questo punto ci si può chiedere se, data un grammatica non contestuale ambigua, sia sempre possibile trovare, come nell'esempio precedente, una grammatica non ambigua equivalente. La risposta è, in generale, negativa.

Definizione 30 Un linguaggio non contestuale si dice *inerentemente ambiguo* se non è generato da nessuna grammatica non contestuale non ambigua.

Esempio 42 Il linguaggio $L = \{a^m b^n a^h \mid m, n, h \geq 0,\ m = n \,\mathrm{o}\, n = h\}$ è un linguaggio non contestuale inerentemente ambiguo. Invero L è generato dalla grammatica $G = \langle V, \Sigma, P, S \rangle$ ove $\Sigma = \{a, b\}$, $V = \Sigma \cup \{S, A, X, Y\}$ con le produzioni

$$S \to XA \mid AY\,, \quad X \to aXb \mid \varepsilon\,, \quad A \to aA \mid \varepsilon\,, \quad Y \to bYa \mid \varepsilon\,.$$

La dimostrazione della inerente ambiguità di L è piuttosto complessa ed è pertanto omessa.

Capitolo 14

Semplificazioni

In questo capitolo vedremo come sia possibile trasformare una grammatica non contestuale in una grammatica equivalente in cui le produzioni hanno una forma particolarmente semplice.

14.1 ε-produzioni e produzioni unarie

Definizione 31 Sia $G = \langle V, \Sigma, P, S \rangle$ una grammatica non contestuale. Le produzioni della forma $X \to \varepsilon$ si dicono ε-produzioni. Le produzioni della forma $X \to Y$, $X, Y \in N$ si dicono produzioni 1-arie.

Esempio 43 Si consideri la grammatica $G = \langle V, \Sigma, P, S \rangle$ con $V = \{S, O, P, E, a, b, x\}$, $\Sigma = \{a, b, x\}$ e le produzioni

$$S \to aOb\,, \quad O \to P \mid aOb \mid OO\,, \quad P \to x \mid E\,, \quad E \to \varepsilon\,.$$

La produzione $E \to \varepsilon$ è una ε-produzione, le produzioni $O \to P$ e $P \to E$ sono produzioni unarie.

Osserviamo che in assenza di ε-produzioni e produzioni 1-arie, si ha la certezza che in ogni derivazione diretta $\alpha \to \beta$, o β ha lunghezza maggiore di α o è ottenuto da α sostituendo una variabile con un terminale. Ciò comporta che una parola di lunghezza n può essere generata in $2n - 1$ passi al più.

Proposizione 6 *Ogni linguaggio non contestuale è generato da una grammatica non contestuale priva di ε-produzioni, eccettuata, eventualmente, la produzione $S \to \varepsilon$, ove S è il simbolo iniziale. Inoltre, si può assumere che il simbolo iniziale non compaia nei lati destri delle produzioni.*

DIMOSTRAZIONE: Sia L un linguaggio non contestuale e $G = \langle V, \Sigma, P, S\rangle$ una grammatica non contestuale che genera L. Poniamo $W = \{X \in N \mid X \Longrightarrow^* \varepsilon\}$. Le variabili dell'insieme W saranno dette *variabili annullabili*. Definiamo poi la grammatica $G' = \langle V \cup \{S'\}, \Sigma, P', S'\rangle$ con le produzioni

$$\begin{aligned} &S' \to S \\ &A \to A_1 \cdots A_k \qquad \text{se } A \to \alpha_0 A_1 \alpha_1 \cdots \alpha_{k-1} A_k \alpha_k \text{ in } P, \\ &\qquad\qquad\quad \text{con } \alpha_0, \alpha_1, \ldots, \alpha_k \in W^*,\ A_1, \ldots, A_k \in V,\ k \geq 1. \end{aligned}$$

In altri termini, G' si ottiene da G con le operazioni seguenti:

1. si introduce un nuovo simbolo iniziale S' e la produzione $S' \to S$,
2. si aggiungono tutte le produzioni ottenute cancellando nei lati sinistri delle produzioni di G una o più occorrenze di variabili annullabili,
3. si cancellano le ε-produzioni.

Verifichiamo che G e G' sono equivalenti. Intanto è chiaro che l'introduzione del nuovo simbolo iniziale S' e della produzione $S' \to S$ non modifica il linguaggio generato.

Osserviamo poi che per tutte le altre produzioni $A \to \gamma$ di G' si ha $A \Longrightarrow_G^* \gamma$. Questo implica che $L(G') \subseteq L(G)$. Per mostrare l'inclusione inversa, è sufficiente mostrare che per ogni variabile $A \in N$ e ogni parola non vuota $w \in \Sigma^*$, se $A \Longrightarrow_G^* w$ allora $A \Longrightarrow_{G'}^* w$. Ragioniamo per induzione sulla lunghezza della derivazione. Se la derivazione ha lunghezza 1, allora c'è la produzione $A \to w$ sia in G che in G'. Se la derivazione ha lunghezza maggiore di 1, allora avremo

$$\begin{gathered} A \to A_1 A_2 \cdots A_k \text{ in } P\,, \\ A_1 \overset{*}{\underset{G}{\Longrightarrow}} w_1\,,\ A_2 \overset{*}{\underset{G}{\Longrightarrow}} w_2\,,\ \ldots\,,\ A_k \overset{*}{\underset{G}{\Longrightarrow}} w_k\,, \\ w = w_1 w_2 \cdots w_k\,. \end{gathered}$$

Siano $i_1, \ldots, i_h$ gli indici tali che $w_{i_1}, \ldots, w_{i_h} \neq \varepsilon$. Dall'ipotesi induttiva segue

$$A_{i_1} \overset{*}{\underset{G'}{\Longrightarrow}} w_{i_1}\,,\ \ldots\,,\ A_{i_h} \overset{*}{\underset{G}{\Longrightarrow}} w_{i_h}\,,$$

mentre, per tutti gli indici i diversi da $i_1, \ldots, i_h$ si ha $A_i \Longrightarrow_G^* \varepsilon$ cosicchè $A \to A_{i_1} \cdots A_{i_h}$ è una produzione di P'. In conclusione, si ha

$$A \underset{G'}{\Longrightarrow} A_{i_1} \cdots A_{i_h} \overset{*}{\underset{G'}{\Longrightarrow}} w_{i_1} \cdots w_{i_h} = w\,.$$

Abbiamo così mostrato che se $A \Longrightarrow_G^* w$, allora $A \Longrightarrow_{G'}^* w$. In particolare, se $w \in L(G)$ si ha $S \Longrightarrow_G^* w$ e quindi $S' \Longrightarrow_{G'} S \Longrightarrow_{G'}^* w$, cioè $w \in L(G')$. Questo prova che $L(G) \subseteq L(G')$ e, per quanto visto in precedenza, concludiamo che G e G' sono equivalenti. □

Evidentemente, per poter eliminare le ε-produzioni di una grammatica non contestuale G è necessario determinare effettivamente l'insieme W delle variabili annullabili.

Per fare ciò, osserviamo innanzitutto che

1. se in G c'è la produzione $X \to \varepsilon$, allora X è una variabile annullabile,
2. se in G c'è la produzione $X \to \gamma$ dove γ è una concatenazione di variabili annullabili, allora X è una variabile annullabile,
3. tutte le variabili annullabili della grammatica G si ottengono applicando un numero finito di volte le due regole precedenti.

Consideriamo pertanto gli insiemi W_n di variabili di G definiti come segue:

$$W_1 = \{X \in N \mid X \to \varepsilon \text{ in } P\},$$
$$W_n = \{X \in N \mid \exists\, X \to \gamma \text{ in } P \text{ con } \gamma \in W_{n-1}^*\}\,, \quad n \geq 2.$$

Possiamo osservare che ciascuno degli insiemi W_n è incluso nel successivo e tutti sono inclusi nell'alfabeto finito N. Pertanto a partire da un opportuno indice m, si avrà $W_m = W_{m+1} = W_{m+2} = \cdots$. Ne possiamo concludere che l'insieme delle variabili annullabili è esattamente $W = W_m$.

Da quanto detto si ottiene l'Algoritmo 6 per il calcolo delle variabili annullabili.

Algoritmo 6: variabiliAnnullabili

Ingresso: Una grammatica non contestuale $G = \langle V, \Sigma, S, P\rangle$;
Uscita: L'insieme $W = \{X \in N \mid X \Longrightarrow^* \varepsilon\}$;

$W \leftarrow \{X \in N \mid X \to \varepsilon \text{ in } P\}$;
$N \leftarrow N - W$;
ripeti
  $W' \leftarrow \{X \in N \mid \exists\, X \to \gamma \text{ in } P \text{ con } \gamma \in W^*\}$;
  $W \leftarrow W \cup W'$;
  $N \leftarrow N - W'$;
finché $W' = \emptyset$;

Esempio 44 Si consideri la grammatica dell'Esempio 43. Il calcolo di W dà

$$W_1 = \{E\}\,, \quad W_2 = \{E, P\}\,, \quad W_3 = \{E, P, O\}\,, \quad W_4 = W_3\,,$$

e quindi $W = \{E, P, O\}$. La grammatica senza ε-produzioni equivalente a G, costruita in accordo con la Proposizione 6 avrà le produzioni

$$S' \to S, \qquad S \to aOb \mid ab, \qquad O \to P \mid aOb \mid OO \mid ab \mid O, \qquad P \to x \mid E.$$

Proposizione 7 *Ogni linguaggio non contestuale è generato da una grammatica non contestuale priva di produzioni 1-arie e di ε-produzioni, eccettuata, eventualmente, la produzione $S \to \varepsilon$, ove S è il simbolo iniziale. Inoltre, si può assumere che il simbolo iniziale non compaia nei lati destri delle produzioni.*

DIMOSTRAZIONE: Sia L un linguaggio non contestuale e $G = \langle V, \Sigma, P, S\rangle$ una grammatica non contestuale che genera L. Per la Proposizione 6, possiamo supporre che G sia priva di ε-produzioni, eccettuata, eventualmente, la produzione $S \to \varepsilon$, ove S è il simbolo iniziale e che S non compaia nei lati destri delle produzioni.

Definiamo poi la grammatica $G' = \langle V, \Sigma, P', S\rangle$ con le produzioni

$$A \to \alpha \quad \text{se } \exists\, B \to \alpha \text{ in } P\,,\ A \overset{*}{\underset{G}{\Longrightarrow}} B\,,\ \alpha \in V^* - N\,,$$

In altri termini, G' si ottiene da G con le operazioni seguenti:

1. per ogni coppia di variabili A, B tali che $A \Longrightarrow_G^* B$, si aggiungono i lati destri delle produzioni di B a quelli delle produzioni di A.

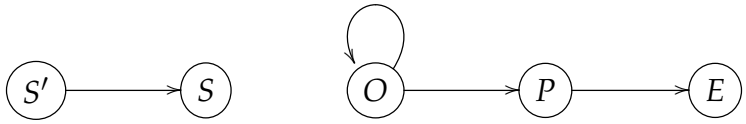
2. si cancellano le produzioni unarie.

Verifichiamo che G e G' sono equivalenti. Osserviamo invero che per tutte le produzioni di $A \to \gamma$ di G' si ha $A \Longrightarrow_G^* \gamma$. Questo implica che $L(G') \subseteq L(G)$. Per mostrare l'inclusione inversa, è sufficiente mostrare che se $A \Longrightarrow_G^* w$ ($A \in N$, $w \in \Sigma^+$) allora $A \Longrightarrow_{G'}^* w$, ragionando come al solito per induzione sulla lunghezza della derivazione. La verifica è lasciata al lettore. □

Come si è visto, per poter eliminare le produzioni unarie, è necessario determinare le coppie di variabili A, B tali che $A \Longrightarrow^* B$. In questo compito siamo facilitati dal fatto che, lavorando su una grammatica priva di ε-produzioni abbiamo la certezza che in una derivazione $A \Longrightarrow^* B$ tutti i passi intermedi devono avere lunghezza 1. Non è difficile convincersi che in questo caso si ha $A \Longrightarrow^* B$ se e solo se c'è un cammino da A a B nel grafo (N, E) con insieme di archi

$$E = \{(X, Y) \in N \times N \mid X \to Y\}\,.$$

Esempio 45 Si consideri la grammaticaottenuta nell'Esempio 44. Essa è priva di ε-produzioni. Il grafo corrispondente alle produzioni unarie è il seguente:



Dovremo quindi aggiungere i lati destri delle produzioni di S a quelli di S', i lati destri delle produzioni di P ed E a quelli di O, i lati destri delle produzioni di E a quelli di P, eliminando contestualmente le produzioni unarie. Otteniamo così una nuova grammatica equivalente a quella di partenza, con le produzioni

$$S' \to aOb \mid ab\,, \quad S \to aOb \mid ab\,, \quad O \to aOb \mid OO \mid ab \mid x\,, \quad P \to x\,.$$

14.2 Variabili improduttive e inaccessibili

Definizione 32 Sia $G = \langle V, \Sigma, P, S\rangle$ una grammatica non contestuale. Una variabile $X \in N = V - \Sigma$ si dice *produttiva* (o *utile*) se esiste $w \in \Sigma^*$ tale che $X \Longrightarrow^* w$. In caso contrario, X si dice *improduttiva* (o *inutile*).

Una variabile $X \in N$ si dice *accessibile* se esistono $\alpha, \beta \in V^*$ tali che $S \Longrightarrow^* \alpha X \beta$. In caso contrario, X si dice *inaccessibile*.

Esempio 46 Si consideri la grammatica $G = \langle V, \Sigma, P, E\rangle$ con $N = \{E, F,\ T, R\}$, $\Sigma = \{+, *, -, a, (,)\}$ e con le produzioni

$$E \to E + E \mid T \mid F\,, \qquad F \to E * E \mid (T) \mid a\,,$$
$$T \to E - T\,, \qquad R \to E - F\,.$$

L'unica produzione che ha T come lato sinistro, contiene T nel lato destro. Pertanto, non è possibile derivare da T una parola che non contenga variabili. Quindi T è una variabile improduttiva.

Invece, R non compare nel lato destro di nessuna produzione. Pertanto nessuna forma senteziale conterrà la variabile R che, quindi, è una variabile inaccessibile.

Definizione 33 Una grammatica non contestuale $G = \langle V, \Sigma, P, S\rangle$ si dice *ridotta* se tutte le variabili $X \neq S$ sono sia produttive che accessibili.

Osserviamo che il simbolo iniziale S è sempre accessibile. D'altra parte, S è produttiva se e soltanto se il linguaggio generato da G è non vuoto.

Evidentemente in una derivazione di una parola di $L(G)$ dal simbolo iniziale non vengono utilizzate né variabili improduttive né variabili inaccessibili.

Pertanto, eliminando da una grammatica G tutte le variabili improduttive e tutte le variabili inaccessibili, nonché le produzioni che contengono tali variabili nel lato destro o nel lato sinistro, si otterrà una grammatica equivalente a G. Abbiamo così dimostrato la seguente

Proposizione 8 *Ogni linguaggio non contestuale è generato da una grammatica non contestuale ridotta.*

Per realizzare tale grammatica, evidentemente è necessario avere una procedura che ci permetta di individuare le variabili improduttive e quelle inaccessibili di una grammatica non contestuale.

Iniziamo considerando gli insiemi W_n delle variabili di G che generano una parola priva di variabili in al più n passi, $n \geq 0$. Ovviamente, $W_0 = \emptyset$. Non è poi difficile verificare che per $n \geq 1$

$$W_n = \{X \in N \mid \exists\, X \to \gamma \text{ in } P \text{ con } \gamma \in (W_{n-1} \cup \Sigma)^*\}$$

Possiamo poi osservare che ciascuno degli insiemi W_n è incluso nel successivo e tutti sono inclusi nell'alfabeto finito N. Pertanto a partire da un opportuno indice m, si avrà $W_m = W_{m+1} = W_{m+2} = \cdots$. L'insieme delle variabili produttive sarà quindi $W = W_m$.

Da quanto detto si ottiene l'Algoritmo 9 per il calcolo delle variabili produttive.

Algoritmo 7: variabiliProduttive

Ingresso: Una grammatica non contestuale $G = \langle V, \Sigma, S, P\rangle$;
Uscita: L'insieme W delle variabili produttive;

$W \leftarrow \emptyset$;
ripeti
    $W' \leftarrow \{X \in N \mid \exists\, X \to \gamma \text{ in } P \text{ con } \gamma \in (W \cup \Sigma)^*\}$;
    $N \leftarrow N - W'$;
    $W \leftarrow W \cup W'$
finché $W' = \emptyset$;

Per determinare le variabili accessibili, osserviamo invece che

1. il simbolo iniziale S è accessibile,
2. se X è una variabile accessibile e $X \to \gamma$ è una produzione, allora tutte le variabili che compaiono in γ sono accessibili,

3. tutte le variabili accessibili si ottengono applicando un numero finito di volte le due regole precedenti.

Per quanto detto, il calcolo delle variabili accessibili si può eseguire nel modo seguente: Consideriamo il grafo orientato (N, E) con insieme di archi

$$E = \{(X, Y) \in N \times N \mid X \to \alpha Y \beta \text{ in } P,\ \alpha, \beta \in V^*\}\,.$$

Le variabili accessibili sono i nodi del grafo accessibili da S. Il calcolo delle variabili accessibili è eseguito dall'Algoritmo 8.

Algoritmo 8: variabiliAccessibili

Ingresso: una grammatica non contestuale $G = \langle V, \Sigma, P, S\rangle$
Uscita: L'insieme W delle variabili accessibili

$W' \leftarrow \{S\}$;
$W \leftarrow W'$;
$N \leftarrow N - W'$;
ripeti
  $W' \leftarrow \{Y \in N \mid X \to \alpha Y \beta \text{ in } P,\ X \in W',\ \alpha, \beta \in V^*\}$;
  $W \leftarrow W \cup W'$;
  $N \leftarrow N - W'$;
finché $W' = \emptyset$;

Per ottenere una grammatica ridotta equivalente a una grammatica data, dovremo quindi

1. determinare le variabili produttive,
2. eliminare le variabili improduttive e le produzioni che contengono tali variabili,
3. determinare le variabili accessibili della grammatica ottenuta,
4. eliminare le variabili inaccessibili e le produzioni che contengono tali variabili.

Osserviamo che non è possibile eseguire prima i passi 3–4 e poi i passi 1–2 in quanto l'eliminazione di variabili improduttive potrebbe creare nuove variabili inaccessibili.

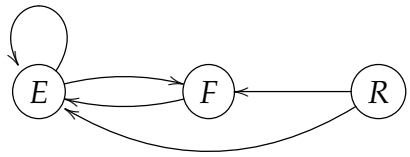
Esempio 47 Consideriamo la grammatica dell'Esempio 46. Si ha

$$W_0 = \emptyset\,, \quad W_1 = \{F\}\,, \quad W_2 = \{F, E\}\,, \quad W_3 = \{F, E, R\}\,, \quad W_4 = W_3\,.$$

Pertanto le variabili produttive sono F, E, R. La grammatica $G' = \langle V', \Sigma, P', E \rangle$ con le variabili F, E, R e le produzioni

$$E \to E + E \mid F\,, \qquad F \to E * E \mid a\,, \qquad R \to E - F\,.$$

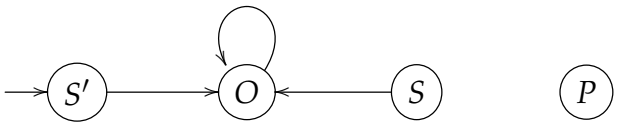
è equivalente a G ma priva di variabili improduttive. Per calcolare le variabili inaccessibili consideriamo il grafo



Le variabili accessibili sono E e F. Otteniamo così la grammatica ridotta $G'' = \langle V'', \Sigma, P'', E \rangle$ equivalente a G con le variabili E e F e le produzioni

$$E \to E + E \mid F\,, \qquad F \to E * E \mid a\,.$$

Esempio 48 Si consideri la grammatica ottenuta nell'Esempio 45. Tutte le variabili sono produttive. Per calcolare le variabili inaccessibili costruiamo il grafo



che ci mostra come le sole variabili accessibili siano S' e O. La grammatica considerata è pertanto equivalente alla grammatica ridotta, priva di ε-produzioni e di produzioni unarie, con le produzioni seguenti:

$$S' \to aOb \mid ab\,, \qquad O \to aOb \mid OO \mid ab \mid x\,.$$

14.3 Forma normale di Chomsky

Definizione 34 Una grammatica non contestuale $G = \langle V, \Sigma, P, S \rangle$ si dice in *forma normale di Chomsky* se ha solo produzioni dei tipi

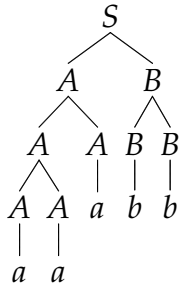
- $X \to YZ$, con $X, Y, Z \in N$,
- $X \to a$, con $X \in N$, $a \in \Sigma$,
- $S \to \varepsilon$, ma solo a condizione che S non compaia nei lati destri delle produzioni.

La caratteristica di una grammatica in forma normale di Chomsky è che negli alberi di derivazione, le foglie sono figli unici mentre i nodi interni costituiscono un albero binario completo.

Esempio 49 La grammatica G con le produzioni

$$S \to AB \mid \varepsilon\,, \quad A \to AA \mid a\,, \quad B \to BB \mid b\,,$$

è in forma normale di Chomsky. Non è difficile verificare che il linguaggio generato è $L = \{\varepsilon\} \cup \{a^m b^n \mid m, n > 0\}$. Un albero di derivazione di *aaabb* è



Vale il seguente

Teorema 9 *Ogni linguaggio non contestuale è generato da una grammatica non contestuale in forma normale di Chomsky.*

Per costruire una grammatica in forma normale di Chomsky equivalente a una data grammatica non contestuale G, innanzitutto ci riduciamo al caso in cui la grammatica data sia priva di ε-produzioni e produzioni unarie. Ciò è sempre possibile come si è visto nella Sez. 14.1.

Il passo successivo consiste nel ridursi a una grammatica che abbia solo produzioni del tipo $X \to \gamma$ con $\gamma \in N^2N^* \cup \Sigma$. Per fare questo è sufficiente introdurre per ogni simbolo terminale $a \in \Sigma$ una nuova variabile A e la produzione $A \to a$ e poi sostituire a con A ogni volta che appare nel lato destro di una produzione $X \to \gamma$, con $\gamma \neq a$.

Esempio 50 Si consideri la grammatica ottenuta nell'Esempio 48. Introducendo le nuove variabili A, B, X e operando come detto sopra, otteniamo una grammatica equivalente con le produzioni

$$S' \to AOB \mid AB\,, \quad O \to AOB \mid OO \mid AB \mid x\,, \quad A \to a\,, \quad B \to b\,.$$

Eliminando la variabile inaccessibile X e ridenominando S , otteniamo la grammatica ridotta con le produzioni

$$S \to AOB \mid AB\,, \quad O \to AOB \mid OO \mid AB \mid x\,, \quad A \to a\,, \quad B \to b\,.$$

A questo punto, le produzioni sono o del tipo $X \to a$, con $X \in N$, $a \in \Sigma$, o del tipo $X \to X_1 \cdots X_k$, con $X, X_1, \ldots, X_k \in N$, $k \geq 2$ (eccetto l'eventuale produzione $S \to \varepsilon$).

Ci resta da modificare la nostra grammatica in modo da ottenerne una equivalente in cui le produzioni $X \to X_1 \cdots X_k$ siano tutte con $k = 2$. Per fare ciò, osserviamo che se $k > 2$, introducendo una nuova variabile Z e sostituendo la produzione $X \to X_1 X_2 \cdots X_k$ con la coppia di produzioni

$$Z \to X_1 X_2 \cdots X_{k-1} \quad \text{e} \quad X \to Z X_k$$

otteniamo una grammatica equivalente. Iterando tale operazione, si raggiunge il risultato cercato.

Esempio 51 Consideriamo la grammatica ottenuta nell'esempio precedente. Per eliminare le produzioni $S \to AOB$ e $O \to AOB$, introduciamo la nuova variabile Z e le produzioni $Z \to AO$, $S \to ZB$ e $O \to ZB$, ottenendo così la grammatica con le produzioni

$$S \to ZB \mid AB, \quad O \to ZB \mid OO \mid AB \mid x, \quad Z \to AO, \quad A \to a, \quad B \to b.$$

che è equivalente a quella di partenza ma è in forma normale di Chomsky.

Concludiamo il capitolo introducendo un'altra forma normale per le grammatiche non contestuali.

Definizione 35 Una grammatica non contestuale G si dice in *forma normale di Greibach* se ha solo produzioni dei tipi

- $X \to a\gamma$, con $a \in \Sigma$ e $\gamma \in N^*$,
- $S \to \varepsilon$, ma solo a condizione che S non compaia nei lati destri delle produzioni.

La caratteristica di una grammatica in forma normale di Greibach G è che ogni derivazione di una parola $w \in L(G)$ è costituita esattamente di $|w|$ passi.

Vale il seguente

Teorema 10 *Ogni linguaggio non contestuale è generato da una grammatica non contestuale in forma normale di Greibach.*

La dimostrazione è omessa.

Capitolo 15

Lemma di Iterazione

Come si è visto (cf. Esempio 30), la proprietà stabilita dal Lemma di Iterazione per i linguaggi regolari non è verificata, in generale, dai linguaggi non contestuali.

È possibile però stabilire una condizione più debole che è soddisfatta necessariamente da ogni parola sufficientemene lunga di un linguaggio non contestuale.

Vale infatti il seguente:

Lemma 5 (di Iterazione) *Sia L un linguaggio non contestuale. Esiste un intero n tale che ogni parola $w \in L$ di lunghezza $|w| > n$ si può fattorizzare $w = xuyvz$ con $uv \neq \varepsilon$ e $xu^k yv^k z \in L$ per ogni $k \geq 0$.*

Il Lemma di Iterazione può essere utile per verificare che un dato linguaggio non è un linguaggio non contestuale, come mostra l'esempio seguente.

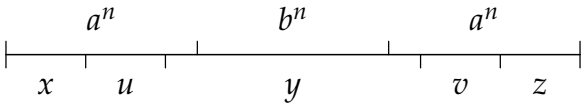
Esempio 52 Si consideri il linguaggio $L = \{a^n b^n a^n \mid n \geq 0\}$. Supponiamo che, per opportuni n, x, u, y, v, z si abbia

$$a^n b^n a^n = xuyvz\,, \quad uv \neq \varepsilon\,.$$

Verifichiamo che $xu^2yv^2z \notin L$. Risulta invero $|xu^2yv^2z| > |xuyvz|$ e quindi $xu^2yv^2z \neq xuyvz = a^n b^n a^n$.

Se $|xu| > n$, allora $a^n b$ è un prefisso di xu e, conseguentemente, anche di xu^2yv^2z. Osserviamo che l'unica parola di L che abbia il prefisso $a^n b$ è proprio $a^n b^n a^n$. Poichè $xu^2yv^2z \neq a^n b^n a^n$, se ne conclude che, in questo caso, $xu^2yv^2z \notin L$. In maniera simmetrica, si verifica che $xu^2yv^2z \notin L$ se $|vz| > n$.

Consideriamo infine il caso in cui $|xu| \leq n$ e $|vz| \leq n$ (vedi fig.).



In questo caso, avremo

$$x = a^i\,, \quad u = a^j\,, \quad v = a^h\,, \quad z = a^k\,, \quad y = a^r b^n a^s\,,$$

per opportuni $i, j, h, k, r, s \geq 0$. Ne segue

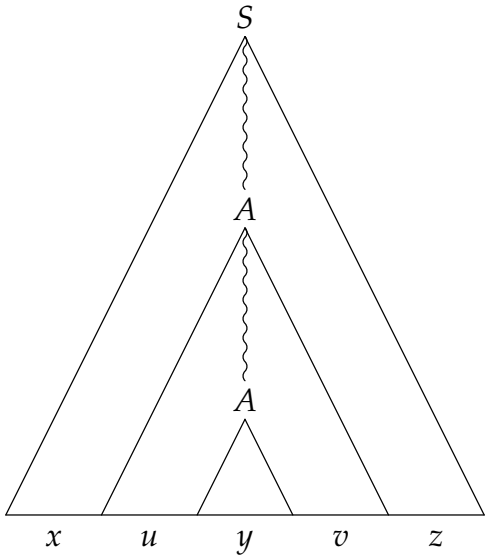
$$xu^2yv^2z = a^{i+j+r} b^n a^{s+h+k}\,,$$

sicché xu^2yv^2z contiene esattamente n occorrenze della lettera b. Poiché l'unica parola di L con tale proprietà è $a^n b^n a^n$ e $xu^2yv^2 \neq a^n b^n a^n$, se ne conclude che, anche in questo caso, $xu^2yv^2z \notin L$.

Abbiamo così verificato che il linguaggio $L = \{a^n b^n a^n \mid n \geq 0\}$ non verifica la proprietà enunciata dal Lemma. Pertanto questo linguaggio non è non contestuale.

DIMOSTRAZIONE DEL LEMMA 5: Sia $G = \langle V, \Sigma, P, S\rangle$ una grammatica non contestuale che genera L. Possiamo supporre, senza perdita di generalità, che G sia priva di ε-produzioni e produzioni unarie. Sia n la massima lunghezza delle parole di L che hanno un albero di derivazione di altezza $\leq \mathrm{Card}(N)$.

Consideriamo quindi una parola $w \in L$ di lunghezza $|w| > n$. L'albero di derivazione di w avrà un'altezza maggiore di $\mathrm{Card}(N)$ e quindi conterrà un cammino che incontra più di $\mathrm{Card}(N)$ nodi interni. Lungo tale cammino, ci saranno quindi due nodi etichettati con una stessa variabile A. La situazione è rappresentata nella figura che segue.



Avremo quindi

$$S \overset{*}{\Longrightarrow} xAz\,, \quad A \overset{*}{\Longrightarrow} uAv\,, \quad A \overset{*}{\Longrightarrow} y\,,$$

per opportuni $x, u, y, v, z \in \Sigma^*$ tali che $w = xuyvz$ e $uv \neq \varepsilon$, poiché la grammatica è priva di ε-produzioni e produzioni unarie. Ne segue

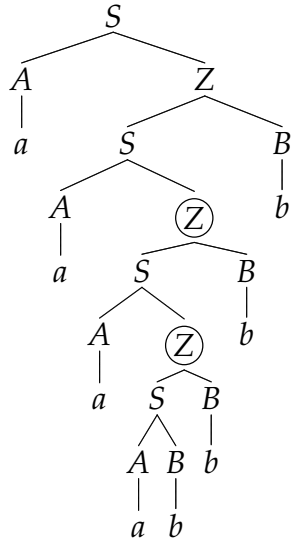
$$S \overset{*}{\Longrightarrow} xAz \overset{*}{\Longrightarrow} xuAvz \overset{*}{\Longrightarrow} xuuAvvz \overset{*}{\Longrightarrow} \cdots \overset{*}{\Longrightarrow} xu^kAv^kz \overset{*}{\Longrightarrow} xu^kyv^kz\,,$$

e quindi $xu^kyv^kz \in L$. □

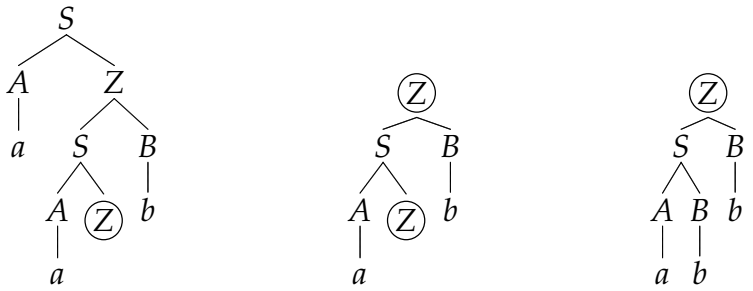
Esempio 53 Consideriamo la grammatica G con le produzioni

$$S \to AZ \mid AB\,, \quad Z \to SB\,, \quad A \to a\,, \quad B \to b\,.$$

L'albero di derivazione di $w = a^4b^4$ è:



Tenendo conto delle due occorrenze di Z indicate dal circoletto, possiamo decomporre l'albero nei tre sottoalberi seguenti:



Si ha quindi

$$S \overset{*}{\Longrightarrow} aaZb\,, \quad Z \overset{*}{\Longrightarrow} aZb\,, \quad Z \overset{*}{\Longrightarrow} abb\,,$$

e pertanto $S \overset{*}{\Longrightarrow} aaa^k(abb)b^kb$. In questo caso, il lemma è soddisfatto con $x = aa$, $u = a$, $y = abb$, $v = b$, $z = \varepsilon$.

Capitolo 16

Algoritmo di Cocke-Kasami-Younger

Sia $G = \langle V, \Sigma, P, S \rangle$ una grammatica non-contestuale. Vogliamo considerare i due seguenti problemi:

1. *Ricognizione*: Data una parola $w \in \Sigma^*$, decidere se $w \in L(G)$.
2. *Parsing*: Data una parola $w \in L(G)$ costruire un albero di derivazione di G associato a w.

Tali problemi sono di grande interesse pratico, in quanto nella compilazione, l'analisi sintattica richiede come passo principale, il parsing del codice sorgente rispetto alle regole grammaticali che definiscono il linguaggio di programmazione.

L'algoritmo di Cocke-Kasami-Younger permette di risolvere in maniera moderatamente efficiente il problema di ricognizione, per qualsiasi linguaggio non contestuale. Tale algoritmo si applica alle grammatiche in forma normale di Chomsky. Il Teorema 9 ci assicura che possiamo sempre ricondurci a questo caso.

Supponiamo quindi che G sia una grammatica in forma normale di Chomsky e sia $w \in \Sigma^*$ una parola non vuota. Possiamo scrivere

$$w = a_1 a_2 \cdots a_n\,, \quad a_1, a_2, \ldots, a_n \in \Sigma,\ n > 0\,.$$

Consideriamo poi gli insiemi

$$N_{ij} = \{X \in N \mid X \stackrel{*}{\Longrightarrow} a_{i+1} a_{i+2} \cdots a_j\}\,, \quad 0 \leq i < j \leq n\,.$$

Allora si ha $N_{0n} = \{X \in N \mid X \Longrightarrow^* w\}$ e pertanto $w \in L(G)$ se e solo se $S \in N_{0n}$. Pertanto per decidere se $w \in L(G)$ è sufficiente calcolare l'insieme

N_{0n} e verificare se tale insieme contiene il simbolo iniziale della grammatica. Vediamo come sia possibile eseguire questo calcolo in maniera efficiente.

Tenendo conto del fatto che la grammatica G è in forma normale di Chomsky, avremo $X \Longrightarrow^* a_j$ se e solo se $X \to a_j$ è una produzione di G. Pertanto

$$N_{j-1\ j} = \{X \in N \mid X \to a_j \text{ in } P\}\,, \quad 1 \leq j \leq n\,, \tag{16.1}$$

Supponiamo ora $j \geq i+2$. Allora si ha $X \Longrightarrow^* a_{i+1}a_{i+2}\cdots a_j$ se e solo se

$$X \to YZ\,, \quad Y \overset{*}{\Longrightarrow} a_{i+1}a_{i+2}\cdots a_h\,, \quad Z \overset{*}{\Longrightarrow} a_{h+1}a_{h+2}\cdots a_j\,,$$

per opportuni $Y, Z \in N$ e $i < h < j$. Otteniamo così

$$N_{ij} = \bigcup_{h=i+1}^{j-1} \{X \in N \mid \exists\, Y \in N_{ih},\ Z \in N_{hj},\ X \to YZ \text{ in } P\}\,.$$

Se poniamo per ogni coppia di variabili $Y, Z \in N$,

$$Y \circ Z = \{X \in N \mid X \to YZ \text{ in } P\}$$

e, più in generale, per ogni coppia di insiemi di variabili $L, M \subseteq N$,

$$L \circ M = \bigcup_{\substack{Y \in L \\ Z \in M}} Y \circ Z\,,$$

e se conveniamo che $N_{ij} = \emptyset$ quando $i \geq j$, l'equazione precedente diventa

$$N_{ij} = \bigcup_{h=0}^{n} N_{ih} \circ N_{hj}\,, \quad 0 \leq i < j \leq n\,, \tag{16.2}$$

formalmente analoga alla formula del prodotto matriciale righe per colonne. È chiaro che le Equazioni (16.1) e (16.2) permettono di calcolare successivamente tutti gli insiemi N_{ij} e quindi, in particolare N_{0n}.

Per esempio, procedendo 'per diagonali', otteniamo l'Algoritmo 9 per la costruzione della matrice N_{ij}.

Esempio 54 Sia G la grammatica con le produzioni

$$S \to SS \mid AA \mid b\,, \quad A \to AS \mid AA \mid a\,.$$

In questo caso, gli insiemi $X \circ Y$, $X, Y \in N$ sono dati dalla seguente tabella:

	S	A
S	S	$\emptyset$
A	A	S, A

Algoritmo 9: Cocke-Kasami-Younger

Ingresso: Una grammatica in f. n. di Chomsky G e una parola
$w = a_1 a_2 \cdots a_n$

Uscita: La matrice N_{ij} associata a w. Ritorna TRUE se $w \in L(G)$, FALSE altrimenti

per $i \leftarrow 0, \ldots, n-1$ **fai**
  $N_{i,i+1} \leftarrow \{X \in N \mid X \to a_{i+1} \text{ in P}\}$;
per $j \leftarrow 2, \ldots, n$ **fai**
  per $i \leftarrow 0, \ldots, n-j$ **fai**

(1) $$N_{i,i+j} \leftarrow \bigcup_{h=i+1}^{i+j-1} N_{ih} \circ N_{h,i+j}$$

se $S \in N_{0,n}$ **allora**
  ritorna TRUE
altrimenti
  ritorna FALSE

Se prendiamo $w = aabb$, otteniamo, successivamente, le matrici

	A			
		A		
			S	
				S

,

	A	S, A		
		A	A	
			S	S
				S

,

	A	S, A	S, A	
		A	A	A
			S	S
				S

,

	A	S, A	S, A	S, A
		A	A	A
			S	S
				S

.

Poiché S compare nella cella in alto a destra, ne deduciamo che $w \in L(G)$.

Analizziamo la complessità dell'algoritmo. Poiché gli insiemi N_{ij} sono tutti inclusi nell'insieme finito N, il tempo necessario per il calcolo dei prodotti $N_{ih} \circ N_{h,i+j}$ può essere maggiorato da una costante. Pertanto, il tempo per l'esecuzione dell'istruzione (1) è maggiorato dal numero dei prodotti da calcolare cioé, in ultima analisi, dalla lunghezza n della parola w. Tenuto conto che tale istruzione viene eseguita un numero di volte pari a $n(n-1)/2$, possiamo concludere che la complessità temporale dell'algoritmo (considerando fissata la grammatica G) è nell'ordine di $O(n^3)$.

Non è difficile integrare l'algoritmo precedente per ottenere il parsing di una parola $w \in L(G)$. In effetti, per ogni variabile $X \in N_{i,j}$, possiamo determinare l'indice h e le variabili $Y \in N_{i,h}$, $Z \in N_{h,j}$ tali che $X \to YZ$ è una produzione di G. Allora una derivazione sinistra di $a_{i+1}a_{i+2}\cdots a_j$ da X si ottiene concatenando la derivazione $X \Longrightarrow YZ$, una derivazione sinistra di $a_{i+1}a_{i+2}\cdots a_h$ da Y e una derivazione sinistra di $a_{h+1}a_{h+2}\cdots a_j$ da Z. Otteniamo così la seguente procedura ricorsiva

---

Procedura Parse(i, j, X)

---

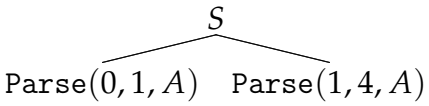
se $j - i = 1$ **allora**
  ritorna $(X \to a_j)$;
trova h, Y, Z tali che $Y \in N_{i,h}$, $Z \in N_{h,j}$, $X \to YZ \in P$;
ritorna $(X \to YZ, \texttt{Parse}(i, h, Y)\,,\ \texttt{Parse}(h, j, Z))$;

La chiamata di $\texttt{Parse}(0, n, S)$ ritorna il parsing di w.

Esempio 55 Consideriamo la grammatica G e la parola w dell'Esempio 54. Poiché $S \in N_{04}$ la chiamata di $\texttt{Parse}(0, 4, S)$ ci porta a cercare una produzione $S \to YZ$ con $Y \in N_{0h}$, $Z \in N_{h4}$, $1 \le h \le 3$. Si ha invero:

$$S \to AA\,, \quad A \in N_{01}\,, \quad A \in N_{14}\,.$$

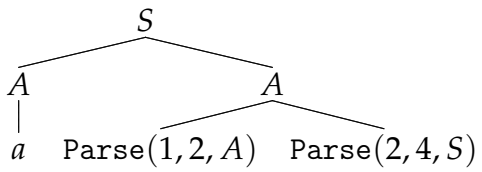
Avremo quindi la chiamata di $\texttt{Parse}(0, 1, A)$ e $\texttt{Parse}(1, 4, A)$. La situazione è rappresentata dal seguente albero:



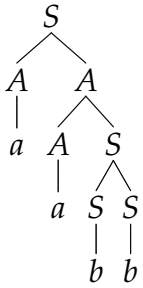
Ora $\texttt{Parse}(0, 1, A)$ ritorna la produzione $A \to a$. Invece, avendosi

$$A \to AS\,, \quad A \in N_{12}\,, \quad S \in N_{24}\,,$$

$\texttt{Parse}(1, 4, A)$ chiamerà $\texttt{Parse}(1, 2, A)$ e $\texttt{Parse}(2, 4, S)$. Si avrà così la situazione seguente:



Continuando in questo modo, si ottiene l'albero


S
A
A
a
A
S
a
S
S
b
b

Capitolo 17

Parsing top-down

L'algoritmo di Cocke-Kasami-Younger risolve, in maniera moderatamente efficiente, il problema del parsing per grammatiche in forma normale di Chomsky. Nel caso in cui si considerino grammatiche che non siano in tale forma, è necessario utilizzare algoritmi differenti. Si possono scegliere due strategie: costruire l'albero di derivazione partendo dalla radice (*top-down parsing*) o partendo dalle foglie (*bottom-up parsing*).

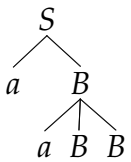
Esempio 56 Si consideri la grammatica G con le produzioni

$$S \to aB \mid bA\,, \quad A \to a \mid aS \mid bAA\,, \quad B \to b \mid bS \mid aBB\,.$$

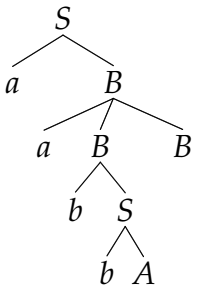
Cerchiamo di fare il parsing della parola $w = aabb$.

Osserviamo che dall'assioma S possiamo derivare aB o bA. D'altra parte, da bA possiamo derivare solo parole che iniziano col terminale b. Quindi necessariamente dobbiamo scegliere la produzione $S \to aB$. Otteniamo così l'albero

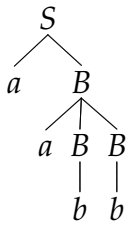
Passiamo ora ad esaminare il nodo con etichetta B. Se utilizzassimo una delle produzioni $B \to b$ o $B \to bS$, otterremmo un albero in cui le foglie più a sinistra sono etichettate con ab e quindi non avremmo più la possibilità di derivare $aabb$. Dobbiamo quindi scegliere la produzione $B \to aBB$ ottenendo l'albero



Consideriamo ora il nodo con etichetta B più a sinistra. La produzione $B \to aBB$ ci darebbe un albero in cui le foglie più a sinistra sono etichettate con aaa e quindi non avremmo più la possibilità di derivare $aabb$. Dobbiamo quindi utilizzare una delle produzioni $B \to b$ o $B \to bS$. Per la verità se scegliamo la seconda produzione e continuiamo nel procedimento, ci troveremo dopo qualche passo nella situazione seguente



dalla quale risulta impossibile proseguire in quanto l'intera w è già presente sulle foglie più a sinistra dell'albero. Utilizzando invece la produzione $B \to b$ e ripetendo lo stesso procedimento per l'altra foglia con etichetta B otteniamo il parsing di w



Cerchiamo di analizzare il procedimento utilizzato nel precedente esempio. Inizialmente il nostro input $aabb$ deve essere derivato dal simbolo iniziale S. Una volta individuata la prima produzione $S \to aB$, otteniamo da S la forma sentenziale aB. Essa è costituita da due parti: il terminale a, che chiameremo *analisi*, che coincide con la prima lettera del nostro input $aabb$ e la variabile B, che chiameremo *predizione*, dalla quale dobbiamo cercare di derivare il resto dell'input, cioè abb.

Al secondo passo, si sostituisce B con aBB. Otteniamo così la forma sentenziale $aaBB$, costituita dall'analisi aa, che è un segmento iniziale dell'input, e dalla predizione BB da cui dobbiamo cercare di derivare il resto dell'input bb. La situazione è schematizzata nella figura seguente:

input	aa	bb
	aa	BB
	analisi	predizione

Ora se sostituiamo la B più a sinistra della predizione con bS otteniamo

aab	b
aab	SB

Al passo successivo dovremo necessariamente sostituire S con bA, ottenendo

$aabb$	
$aabb$	AB

e non possiamo procedere oltre. Invece, se al passo precedente sostituiamo la B più a sinistra della predizione con b otteniamo

aab	b
aab	B

e poi

$aabb$
$aabb$

Abbiamo così determinato, a grandi linee, una procedura per il parsing: a ogni passo

1. sostituiamo la variabile più a sinistra della predizione con il lato sinistro di una sua produzione;
2. verifichiamo che i terminali eventualmente presenti all'inizio della predizione ottenuta siano un fattore iniziale dell'input non ancora analizzato;
3. in caso positivo spostiamo tali terminali nell'input analizzato; in caso negativo l'esecuzione fallisce.

Naturalmente, la procedura delineata è non deterministica: a ogni passo la variabile più a sinistra della predizione può essere sostituita da uno qualunque dei suoi lati destri, ma non tutte le scelte conducono al successo della procedura.

Poiché la nostra procedura è, in generale, non deterministica per realizzarla sarà necessario esplorare tutte le possibili computazioni. Tale esplorazione può essere fatta 'in ampiezza' esaminando parallelamente tutti i possibili sviluppi della computazione, oppure in profondità, esaminando un unico cammino, salvo tornare indietro e provare un'altra strada in caso di fallimento.

Vediamo più in dettaglio un esempio di parsing top-down in ampiezza.

Esempio 57 Si consideri la grammatica non contestuale $G = \langle V, \Sigma, P, S\rangle$ con alfabeto terminale $\Sigma = \{a, b, c\}$, alfabeto non-terminale $N = \{S, A, B, C, D\}$ e produzioni

$$S \to AB \mid DC, \quad A \to a \mid aA, \quad B \to bc \mid bBc, \quad C \to c \mid cC, \quad D \to ab \mid aDb.$$

Per motivi di efficienza, è utile aggiungere alla nostra grammatica un nuovo simbolo terminale # e cercare una derivazione $S\# \Longrightarrow^* w\#$ anzicché $S \Longrightarrow^* w$.

Cerchiamo di fare il parsing della parola *aabc* o meglio cerchiamo una derivazione $S\# \Longrightarrow^* aabc\#$. Come in precedenza, esamineremo l'input da sinistra a destra, una lettera alla volta. Avremo quindi bisogno di tenere traccia

- della lettera che stiamo esaminando (simbolo di *look-ahead*)
- delle produzioni utilizzate in ogni tentativo di parsing,
- delle *predizioni* relative a ciascun tentativo.

Di volta in volta scarteremo le computazioni in cui la prima lettera della predizione è un simbolo terminale diverso dal simbolo di look-ahead. Inizialmente la situazione è la seguente:

	$aabc\#$
	$S\#$

Il simbolo di look-ahead è a. La variabile S può essere sostituita da AB o da DC, sicché le possibili situazioni successive sono rappresentate nel quadro seguente

	$aabc\#$
S_1	$AB\#$
S_2	$DC\#$

dove gli indici 1 e 2 del simbolo S nell'analisi stanno a ricordare che la variabile S è stata sostituita, rispettivamente dal primo e dal secondo dei lati destri delle sue produzioni. Ora la variabile A può essere sostituita rispettivamente da a o aA mentre la variabile D può essere sostituita rispettivamente da ab o aDb. In tutti i casi il primo simbolo della predizione è a: possiamo quindi spostare questo simbolo dal lato 'analisi'. Otteniamo quindi

a	$abc\#$
S_1A_1a	$B\#$
S_1A_2a	$AB\#$
S_2D_1a	$bC\#$
S_2D_2a	$DbC\#$

Il nuovo simbolo di look-ahead è ancora a, ma nella terza riga, la prima lettera della predizione è il terminale b. Possiamo pertanto scartare tale riga.

Effettuando tutte le possibili sostituzioni nelle righe rimanenti otteniamo

a	$abc\#$
$S_1A_1aB_1$	$bc\#$
$S_1A_1aB_2$	$bBc\#$
$S_1A_2aA_1$	$aB\#$
$S_1A_2aA_2$	$aAB\#$
$S_2D_2aD_1$	$abbC\#$
$S_2D_2aD_2$	$aDbbC\#$

Di nuovo, tutte le predizioni iniziano con un terminale. Quindi scartiamo quelle in cui tale terminale è diverso dal simbolo di look-ahead a mentre nelle altre il terminale verrà spostato dal lato 'analisi'.

aa	$bc\#$
$S_1A_2aA_1a$	$B\#$
$S_1A_2aA_2a$	$AB\#$
$S_2D_2aD_1a$	$bbC\#$
$S_2D_2aD_2a$	$DbbC\#$

Ormai il procedimento dovrebbe essere chiaro e pertanto ci limitiamo a riportare i passaggi successivi

aa	$bc\#$
$S_1A_2aA_1aB_1$	$bc\#$
$S_1A_2aA_1aB_2$	$bBc\#$
$S_1A_2aA_2aA_1$	$aB\#$
$S_1A_2aA_2aA_2$	$aAB\#$
$S_2D_2aD_1a$	$bbC\#$
$S_2D_2aD_2aD_1$	$abbbC\#$
$S_2D_2aD_2aD_2$	$aDbbbC\#$

aab	$c\#$
$S_1A_2aA_1aB_1b$	$c\#$
$S_1A_2aA_1aB_2b$	$Bc\#$
$S_2D_2aD_1ab$	$bC\#$

aab	$c\#$
$S_1A_2aA_1aB_1b$	$c\#$
$S_1A_2aA_1aB_2bB_1$	$bcc\#$
$S_1A_2aA_1aB_2bB_2$	$bBcc\#$
$S_2D_2aD_1ab$	$bC\#$

$aabc$	$\#$
$S_1A_2aA_1aB_1bc$	$\#$

$aabc\#$	
$S_1A_2aA_1aB_1bc\#$	

La sequenza finale $S_1A_2aA_1aB_1bc\#$ ci permette di risalire alla derivazione

$$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaB \Rightarrow aabc.$$

Va precisato che non sempre il metodo qui delineato giunge a termine. Il problema sorge quando nella grammatica c'é una derivazione $X \Longrightarrow^* X\alpha$ con $X \in N$ e $\alpha \in V^*$. Si parla, in tal caso, di *recursione destra*. In questa eventualità

si possono ottenere indefinitamente predizioni che iniziano col simbolo X e mai un terminale all'inizio della predizione, sicché la procedura non giungerà mai al termine.

È opportuno rilevare comunque che per ogni grammatica non contestuale, esiste una grammatica equivalente priva di recursione destra. Ad esempio, sono prive di recursione destra le grammatiche in forma normale di Greibach (cf. Sez. 14.3).

L'esplorazione 'in ampiezza' di tutte le possibili computazioni del nostro algoritmo di parsing può risultare molto costosa in termini di utilizzo di memoria. L'alternativa è quella di procedere in profondità, praticando il *backtraking* in caso di fallimento.

Vediamo in dettaglio come avviene il parsing con backtraking.

Esempio 58 Si consideri la grammatica dell'Esempio 57 e cerchiamo nuovamente una derivazione $S\# \Longrightarrow^* aabc\#$.

Inizialmente la situazione è la seguente:

	$aabc\#$
	$S\#$

Il simbolo di look-ahead è a. La variabile S può essere sostituita da AB o da DC. Iniziamo a provare con AB.

	$aabc\#$
S_1	$AB\#$

Ora la variabile A può essere sostituita rispettivamente da a o aA. Di nuovo proviamo la prima possibilità. Otteniamo quindi

a	$abc\#$
S_1A_1a	$B\#$

Adesso proviamo a sostituire B con bc:

a	$abc\#$
$S_1A_1aB_1$	$bc\#$

Il primo simbolo della predizione è un terminale diverso dal simbolo di look-ahead. Pertanto è necessario il backtracking:

a	$abc\#$
S_1A_1a	$B_1\#$

Proviamo la successiva produzione di B, cioè $B \to bBc$.

a	$abc\#$
$S_1A_1aB_2$	$bBc\#$

Di nuovo, il primo simbolo della predizione è un terminale diverso dal simbolo di look-ahead ed è quindi necessario il back-tracking. Stavolta dobbiamo arretrare per più passi, finchè non otteniamo una predizione che inizi con una variabile che ha ancora qualche produzione da provare:

	$aabc\#$
S_1	$A_1B\#$

Adesso proviamo la produzione $A \to aA$:

a	$abc\#$
S_1A_2a	$AB\#$

Ormai il procedimento dovrebbe essere chiaro e pertanto ci limitiamo a riportare i passaggi successivi

aa	$bc\#$
$S_1A_2aA_1a$	$B\#$

aa	$bc\#$
$S_1A_2aA_1aB_1$	$bc\#$

$aabc\#$	
$S_1A_2aA_1aB_1bc\#$	

che ci dà il medesimo risultato ottenuto nell'Esempio 57.

L'implementazione di un algoritmo di parsing top-down con back-tracking per una specifica grammatica può essere realizzato con il metodo della *discesa ricorsiva*.

Un parser a discesa ricorsiva prevede una procedura per ogni variabile che chiama ricorsivamente le procedure relative alle variabili che compaiono nei suoi lati destri, secondo lo schema seguente

Procedura A()

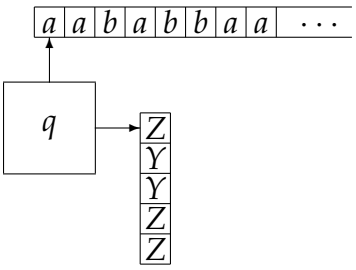
seleziona una produzione $A \to X_1 \cdots X_n$ della variabile A;
per $i \leftarrow 1, \ldots, k$ **fai**
    se X_i *è una variabile* **allora**
        chiama la procedura X_i ( )
    altrimenti se X_i *è il simbolo di look-up* **allora**
        procedere a esaminare il successivo simbolo dell'input
    altrimenti
        /* errore */

La procedura è evidentemente non-deterministica. In una implementazione realistica, l'errore deve corrispondere alla selezione di una ulteriore produzione della variabile A. Solo nel caso in cui la procedura fallisca con tutte le produzioni di A, dovrà restituire il risultato negativo alla procedura che la ha invocata.

Capitolo 18

Automi a pila

Introduciamo ora un nuovo modello di macchina astratta detto *automa a pila*. Possiamo immaginare un automa a pila come un'automa a stati finiti dotato di una struttura dati LIFO (Last In First Out).



A ogni passo di calcolo, la macchina legge un simbolo sul nastro di input e preleva (*pop*) il simbolo in cima alla pila, dopodiché assume un nuovo stato che dipende dallo stato precedente e dai due simboli letti, scrive (*push*) in cima alla pila una stringa di simboli dipendente anch'essa dallo stato precedente e dai due simboli letti, infine sposta la testina di lettura dal nastro di input sulla cella a destra. Sono ammesse anche delle 'ε-transizioni', in cui le operazioni di push e pop sono eseguite senza leggere il nastro di input.

Diamo ora la definizione formale di automa a pila. Per ogni insieme S, denotiamo con $\wp_F(S)$ l'insieme delle parti finite di S.

Definizione 36 Un *automa a pila* è una settupla

$$\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$$

dove

- Q è un insieme finito, detto *insieme degli stati*,

- Σ è un alfabeto, detto *alfabeto di input*,
- Γ è un alfabeto, detto *alfabeto di pila*,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \wp_F(Q \times \Gamma^*)$ è la *funzione di transizione*,
- $q_0 \in Q$ è lo *stato iniziale*,
- $Z_0 \in \Gamma$ è il *simbolo iniziale della pila*,
- $F \subseteq Q$ è l'insieme degli *stati finali*.

L'automa si trova inizialmente nello stato iniziale e la pila contiene il simbolo iniziale della pila. Supponiamo che a un dato istante della computazione l'automa si trova nello stato p, riceve a in input e Z dall'operazione di pop. Se $\delta(p, a, Z)$ contiene una coppia (q, γ), allora l'automa si può portare nello stato q eseguendo il push di γ sulla pila.

Esempio 59 Sia $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ l'automa a pila con

- insieme di stati $Q = \{q_0, p, f\}$,
- alfabeto di input $\Sigma = \{a, b\}$,
- alfabeto di pila $\Gamma = \{Z_0, A\}$,
- insieme degli stati finali $F = \{f\}$,
- funzione di transizione data dalle tabelle seguenti

		a	b	ε
q_0	Z_0	(q_0, aZ_0)	$-$	$-$
	A	(q_0, AA)	(p, ε)	$-$
p	Z_0	$-$	$-$	(f, ε)
	A	$-$	(p, ε)	$-$
f	Z_0	$-$	$-$	$-$
	A	$-$	$-$	$-$

L'automa $\mathcal{A}$, inizialmante nello stato q_0, finchè trova a sul nastro di input le 'copia' nella pila; se trova delle b passa nello stato p e, per ogni b letta 'cancella' una A dalla pila; se trova il simbolo iniziale in cima alla pila passa nello stato finale f e si arresta. Quando ciò avviene, siamo certi che la parola letta sul nastro di input era $a^n b^n$ per un opportuno $n > 0$.

Per descrivere la condizione di un automa a pila in un dato istante, abbiamo bisogno di specificarne lo stato, il contenuto della pila, e ciò che resta ancora da leggere sul nastro di input. Introduciamo pertanto la seguente

Definizione 37 Una *descrizione istantanea* è una tripla $(q,\ u, \gamma) \in Q \times \Sigma^* \times \Gamma^*$. Nell'insieme delle descrizioni istantanee si introduce la relazione $\vdash\!\!\!-_{\mathcal{A}}$ definita, da

$$(q, aw, Z\alpha) \underset{\mathcal{A}}{\vdash\!\!\!-} (p, w, \gamma\alpha) \qquad \text{se} \qquad (p, \gamma) \in \delta(q, a, Z)\,,$$

$q, p \in Q,\ a \in \Sigma,\ w \in \Sigma^*,\ Z \in \gamma,\ \alpha, \gamma \in \Gamma^*$

In altri termini, avremo $D \vdash\!\!\!-_{\mathcal{A}} D'$ se D e D' sono due possibili descrizioni istantanee consecutive di $\mathcal{A}$ in una computazione. Scriveremo poi $D \vdash\!\!\!-^*_{\mathcal{A}} D'$ se esiste una sequenza finita di descrizioni istantanee $D_0, D_1, \ldots, D_n$ tale che

$$D = D_0 \underset{\mathcal{A}}{\vdash\!\!\!-} D_1 \underset{\mathcal{A}}{\vdash\!\!\!-} \cdots \underset{\mathcal{A}}{\vdash\!\!\!-} D_n = D'\,.$$

Si avrà quindi $D \vdash\!\!\!-^*_{\mathcal{A}} D'$ se esiste una computazione che porta l'automa $\mathcal{A}$ dalla descrizione D alla descrizione D'.

Passiamo ora a descrivere quando un input viene accettato dal nostro automa a pila. Abbiamo tre distinte possibilità per definire le parole accettate.

Definizione 38 Sia $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ un automa a pila e $w \in \Sigma^*$. Diremo che una parola $w \in \Sigma^*$ è *accettata per stato finale* dall'automa a pila $\mathcal{A}$ se si ha

$$(q_0, w, Z_0) \overset{*}{\underset{\mathcal{A}}{\vdash\!\!\!-}} (f, \varepsilon, \gamma), \quad f \in F,\ \gamma \in \Gamma^*\,.$$

Invece diremo che w è *accettata per pila vuota* da $\mathcal{A}$ se si ha

$$(q_0, w, Z_0) \overset{*}{\underset{\mathcal{A}}{\vdash\!\!\!-}} (p, \varepsilon, \varepsilon), \quad p \in Q.$$

Infine diremo che w è *accettata per stato finale e pila vuota* da $\mathcal{A}$ se se si ha

$$(q_0, w, Z_0) \overset{*}{\underset{\mathcal{A}}{\vdash\!\!\!-}} (f, \varepsilon, \varepsilon), \quad f \in F\,.$$

L'insieme delle parole accettate per stato finale (risp., per pila vuota, per stato finale e pila vuota) sarà chiamato il *linguaggio riconosciuto da $\mathcal{A}$ per stato finale* (risp., *per pila vuota*, *per stato finale e pila vuota*) e sarà denotato con $L_F(\mathcal{A})$ (risp., $L_P(\mathcal{A})$, $L(\mathcal{A})$).

In sintesi, l'automa $\mathcal{A}$ accetta una parola w se c'è una computazione che termina con la testina di lettura che ha esaurito il nastro di input, e inoltre, a seconda del metodo di accettazione scelto, lo stato raggiunto è uno stato finale e/o la pila risulta vuota.

Va osservato che il modello introdotto è non deterministico e quindi una parola è accettata quando c'è almeno una computazione che verifica la condizione suddetta. Ciò non esclude ovviamente che vi siano altre computazioni che terminano prima di esaurire il nastro di input o che non verifichino la condizione di terminare in uno stato finale o con la pila vuota.

Esempio 60 Sia L il linguaggio delle palindrome pari sull'alfabeto $\Sigma = \{a, b\}$. Vogliamo realizzare un automa a pila che riconosca L per stato finale. Una strategia possibile è quella di iniziare a copiare l'input nella pila poi passare in un nuovo stato e verificare che ciò che rimane sul nastro è uguale a ciò che abbiamo nella pila, letto ovviamente in ordine inverso rispetto a quello di scrittura. Poiché il modello è intrinsecamente non deterministico, non abbiamo bisogno di specificare in quale punto della computazione si deve passare dalla fase di copia nella pila a quella di confronto tra pila e input. Un automa che faccia quanto richiesto è l'automa $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ con $Q = \{q_0, p, f\}$, $\Gamma = \{Z_0, a, b\}$, $F = \{f\}$ e funzione di transizione

		a	b	ε
	Z_0	(q_0, aZ_0)	(q_0, bZ_0)	(p, Z_0)
q_0	a	(q_0, aa)	(q_0, ab)	(p, a)
	b	(q_0, ba)	(q_0, bb)	(p, b)
	Z_0	$-$	$-$	(f, ε)
p	a	(p, ε)	$-$	$-$
	b	$-$	(p, ε)	$-$
	Z_0	$-$	$-$	$-$
f	a	$-$	$-$	$-$
	b	$-$	$-$	$-$

Ad esempio, una computazione su $abba$ è data da

$$(q_0, abba, Z_0) \underset{\mathcal{A}}{\vdash} (q_0, bba, aZ_0) \underset{\mathcal{A}}{\vdash} (q_0, ba, baZ_0) \underset{\mathcal{A}}{\vdash} (p, ba, baZ_0) \underset{\mathcal{A}}{\vdash} (p, a, aZ_0) \underset{\mathcal{A}}{\vdash} (p, \varepsilon, Z_0) \underset{\mathcal{A}}{\vdash} (f, \varepsilon, \varepsilon)\,.$$

Un esempio di computazione su abb è:

$$(q_0, abb, Z_0) \underset{\mathcal{A}}{\vdash} (q_0, bb, aZ_0) \underset{\mathcal{A}}{\vdash} (q_0, b, baZ_0) \underset{\mathcal{A}}{\vdash} (p, b, baZ_0) \underset{\mathcal{A}}{\vdash} (p, \varepsilon, aZ_0)\,.$$

Per poter concludere che *abb* non è accettata dall'automa, dovremmo però verificare che nessuna delle computazioni possibili conduce a uno stato globale del tipo (f, γ), con $\gamma \in \Gamma^*$.

Esempio 61 Vogliamo realizzare un automa a pila che riconosca per pila vuota il linguaggio L delle palindrome sull'alfabeto $\Sigma = \{a, b, c\}$ che contengono una sola occorrenza della lettera c. Tali parole hanno la forma ucu' con $u \in \{a, b\}^*$ e u' uguale alla parola u 'riflessa'. Il nostro automa deve copiare l'input nella pila 'sotto il simbolo iniziale Z', finchè non trova la lettera c; a questo punto cancella il simbolo Z in cima alla pila; poi verifica che ciò che rimane sul nastro è uguale al contenuto della pila, letto ovviamente in ordine inverso rispetto a quello di scrittura. Un automa che faccia quanto richiesto è l'automa $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ con $Q = F = \{q_0\}$, $\Gamma = \{Z, a, b\}$ e funzione di transizione data dalla seguente tabella:

q_0	a	b	c
Z_0	(q_0, Z_0a)	(q_0, Z_0b)	(q_0, ε)
a	(q_0, ε)	$-$	$-$
b	$-$	(p, ε)	$-$

In questo caso, per ogni tripla $(q, x, Z) \in Q \times \Sigma \times \Gamma$ c'è al più un elemento in $\delta(q, x, Z) \cup \delta(q, \varepsilon, Z)$. Pertanto, ogni descrizione istantanea D ammette al massimo una descrizione istantanea successiva. Questo ci permette di verificare 'in tempo reale' se una parola è accettata o meno.

Definizione 39 Un automa a pila $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ è *deterministico* se per ogni $q \in Q$, $a \in \Sigma$, $Z \in \Gamma$ l'insieme $\delta(q, a, Z) \cup \delta(q, \varepsilon, Z)$ contiene al più un elemento.

Un linguaggio accettato per stato finale da un automa a pila deterministico si dirà *deterministico*.

Il nostro prossimo obbiettivo è mostrare che le classi dei linguaggi riconosciuti dagli automi a pila rispettivamente per pila vuota, per stato finale o per stato finale e pila vuota coincidono. Per cominciare, mostriamo che se un linguaggio L è accettato per stato finale e pila vuota da un automa a pila, allora si può costruire un altro automa a pila che riconosce L sia per stato finale, sia per pila vuota, sia per stato finale e pila vuota.

Proposizione 9 *Sia $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ un automa a pila. Esiste effettivamente un automa a pila $\mathcal{A}'$ tale che*

$$L(\mathcal{A}) = L_P(\mathcal{A}') = L_F(\mathcal{A}') = L(\mathcal{A}')\,.$$

DIMOSTRAZIONE: Realizzeremo un automa a pila $\mathcal{A}'$ che si comporta nel modo seguente: scrive un simbolo speciale # all'inizio della pila, poi simula il funzionamento dell'automa $\mathcal{A}$. Se mentre si trova in uno stato finale di $\mathcal{A}$, $\mathcal{A}'$ legge # sulla pila, il che significa che la pila dell'automa simulato si è svuotata, $\mathcal{A}'$ passa nel suo unico stato finale e cancella l'unico simbolo della pila.

Formalmente, $\mathcal{A}' = \langle Q \cup \{q_0', f\}, \Sigma, \Gamma \cup \{\#\}, \delta', q_0', \#, \{f\}\rangle$ ove la funzione di transizione δ' è definita da:

$$\delta'(q,a,Z) = \delta(q,a,Z), \quad \text{per } (q,a,Z) \in Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma,$$

$$\delta'(q,\varepsilon,\#) = \begin{cases} (q_0, Z_0\#) & \text{se } q = q_0', \\ (f,\varepsilon) & \text{se } q \in F, \\ \emptyset & \text{se } q \in Q - F, \end{cases}$$

e $\delta(q,a,Z) = \emptyset$ in tutti i casi rimanenti.

Dalla costruzione risulta chiaro che una computazione di $\mathcal{A}'$ termina con la pila vuota se e soltanto se raggiunge lo stato finale f. Questo implica che

$$L_P(\mathcal{A}') = L_F(\mathcal{A}') = L(\mathcal{A}').$$

Inoltre, tale condizione si verifica se e solo se c'è una computazione di $\mathcal{A}$ con lo stesso input che termina in uno stato di F e con la pila vuota. Ne segue l'asserto. □

Un risultato analogo alla Proposizione 9 vale per i linguaggi accettati per pila vuota.

Proposizione 10 *Sia $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ un automa a pila. Esiste effettivamente un automa a pila $\mathcal{A}'$ tale che*

$$L_P(\mathcal{A}) = L_P(\mathcal{A}') = L_F(\mathcal{A}') = L(\mathcal{A}')\,.$$

DIMOSTRAZIONE: Vista la Proposizione 9, è sufficiente costruire un automa $\mathcal{A}''$ tale che $L(\mathcal{A}'') = L_P(\mathcal{A})$. E in effetti un tale automa si ottiene semplicemente sostituendo in $\mathcal{A}$ l'insieme degli stati finali F con l'insieme Q di tutti gli stati. □

Esempio 62 Consideriamo il linguaggio L dell'Esempio 61. L'automa $\mathcal{A}'$ con insieme di stati $\{q_0', q_0, f\}$, stato iniziale q_0', stato finale f, e la seguente funzione

di transizione

		a	b	c	ε
q_0'	#	–	–	–	$(q_0, Z_0\#)$
	Z_0	–	–	–	–
	a	–	–	–	–
	b	–	–	–	–
q_0	#	–	–	–	(f, ε)
	Z_0	(q_0, Z_0a)	(q_0, Z_0b)	(q_0, ε)	–
	a	(q_0, ε)	–	–	–
	b	–	(q_0, ε)	–	–
f	#	–	–	–	–
	Z_0	–	–	–	–
	a	–	–	–	–
	b	–	–	–	–

riconosce L sia per stato finale, sia per pila vuota, sia per stato finale e pila vuota.

Osserviamo che se $\mathcal{A}$ è un automa a pila deterministico, l'automa $\mathcal{A}'$ che si ottiene dalla Proposizione 9 è anch'esso deterministico. Se ne deriva la seguente

Proposizione 11 *Un linguaggio riconosciuto per pila vuota (o per pila vuota e stato finale) da un automa a pila deterministico è un linguaggio deterministico.*

Osserviamo che, viceversa, non tutti i linguaggi deterministici possono essere riconosciuti per pila vuota da un automa deterministico.

Verifichiamo ora che i linguaggi riconosciuti per stato finale sono anche linguaggi riconosciuti per pila vuota.

Proposizione 12 *Sia $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ un automa a pila. Esiste effettivamente un automa a pila $\mathcal{A}'$ tale che*

$$L_F(\mathcal{A}) = L_P(\mathcal{A}') = L_F(\mathcal{A}') = L(\mathcal{A}')\,.$$

DIMOSTRAZIONE: Tenendo conto della Proposizione 9, è sufficiente mostrare che esiste effettivamente un automa a pila $\mathcal{A}''$ tale che $L_F(\mathcal{A}) = L(\mathcal{A}'')$.

Per fare ciò, ci basterà aggiungere al nostro automa un nuovo stato f e le ε-transizioni necessarie affinchè, arrivato in uno stato finale, passi nello stato f e svuoti la pila.

Formalmente, definiamo $\mathcal{A}'' = \langle Q \cup \{f\}, \Sigma, \Gamma, \delta', q_0, Z_0, F \cup \{f\} \rangle$ ove la funzione di transizione δ' è data da:

$$\delta'(q,a,Z) = \begin{cases} \delta(q,a,Z), & \text{se } q \in Q, \\ \emptyset, & \text{se } q = f, \end{cases}$$

$$\delta'(q,\varepsilon,Z) = \begin{cases} \delta(q,\varepsilon,Z) \cup (f,Z), & \text{se } q \in F, \\ \delta(q,\varepsilon,Z), & \text{se } q \in Q - F, \\ (f,\varepsilon), & \text{se } q = f \end{cases}$$

$(a \in \Sigma,\ Z \in \Gamma)$. □

Dalle proposizioni precedenti possiamo dedurre il seguente importante risultato

Teorema 11 *La famiglia dei linguaggi accettati da automi a pila per pila vuota, la famiglia dei linguaggi accettati da automi a pila per stato finale e la famiglia dei linguaggi accettati da automi a pila per stato finale e pila vuota coincidono.*

Concludiamo la sezione con la seguente

Osservazione 1 Come si è visto, un automa a pila è individuato dalla sua funzione di transizione δ. Tale funzione può essere descritta da una tabella, come negli esempi precedenti. In taluni casi, e specialmente quando in tale tabella risultino molte celle vuote, piuttosto che disegnare la tabella stessa, può risultare più conveniente elencare la lista delle quintuple (q, a, Z, q', γ) tali che $(q', \gamma) \in \delta(q, a, Z)$, $(q, a, Z) \in Q \times \Sigma \cup \{\varepsilon\} \times \Gamma$.

Per esempio, l'automa a pila considerato nell'Esempio 62 può essere più sinteticamente descritto dalla seguente lista di quintuple:

$$(q_0', \varepsilon, \#, q_0, Z_0\#), \quad (q_0, \varepsilon, \#, f, Z_0\#), \quad (q_0, Z_0, a, q_0, Z_0a),$$
$$(q_0, Z_0, b, q_0, Z_0b), \quad (q_0, Z_0, c, q_0, \varepsilon), \quad (q_0, a, a, q_0, \varepsilon),$$
$$(q_0, b, b, q_0, \varepsilon)$$

Un'ulteriore rappresentazione di un automa a pila fa uso di un grafo diretto, analogamente a quanto visto per gli automi a stati finiti. I vertici del grafo saranno, anche in questo caso, gli stati dell'automa. Gli archi del grafo, invece, sono in corrispondenza con le quintuple considerate in precedenza. Più precisamente, per ogni quintupla (q, a, Z, q', γ), nel grafo dell'automa a pila tracceremo l'arco $q \xrightarrow{a, Z/\gamma} q'$.

Capitolo 19

Teorema di caratterizzazione

Definizione 40 Sia data una grammatica non contestuale $G = \langle V, \Sigma, P, S \rangle$. Siano $\alpha, \beta \in V^*$. Scriveremo $\alpha \Longrightarrow_L \beta$ se esistono $u \in \Sigma^*$, $\sigma \in V^*$ e una produzione $X \to \gamma$ di G tali che

$$\alpha = uX\sigma\,, \quad \beta = u\gamma\sigma\,.$$

Una *derivazione sinistra* è una sequenza

$$\alpha = \alpha_0 \underset{L}{\Longrightarrow} \alpha_1 \underset{L}{\Longrightarrow} \cdots \underset{L}{\Longrightarrow} \alpha_n = \beta\,.$$

In altri termini una derivazione sinistra è una derivazione in cui si riscrive sempre la variabile più a sinistra. Dato che in una grammatica non contestuale l'ordine in cui vengono eseguite le derivazioni descritte da un albero di derivazione è inessenziale, possiamo enunciare il seguente

Lemma 6 *Data una grammatica non contestuale G, ogni parola di $L(G)$ si ottiene dall'assioma S con una derivazione sinistra.*

Vogliamo caratterizzare i linguaggi non contestuali mediante gli automi a pila.

Iniziamo con la seguente

Proposizione 13 *Sia $\mathcal{A}$ un automa a pila con un solo stato. Esiste effettivamente una grammatica non contestuale G tale che*

$$L_P(\mathcal{A}) = L(G)\,.$$

Sia $\mathcal{A} = \langle \{q\}, \Sigma, \Gamma, \delta, q, Z_0, \{q\} \rangle$ un automa a pila con un unico stato. Possiamo supporre, senza perdita di generalità, $\Gamma \cap \Sigma = \emptyset$. Definiamo la grammatica G prendendo

- L'alfabeto di pila Γ come insieme delle variabili,

- L'alfabeto di input Σ come insieme dei simboli terminali,
- Il simbolo iniziale della pila Z_0 come assioma,
- Le produzioni

$$Z \to a\gamma, \quad \text{con } (q,\gamma) \in \delta(q,a,Z),\ a \in \Sigma \cup \{\varepsilon\},\ Z \in \Gamma.$$

Dimostreremo che $L_P(\mathcal{A}) = L(G)$. Iniziamo col seguente

Lemma 7 *Sia $a \in \Sigma \cup \{\varepsilon\}$, $u \in \Sigma^*$, $\beta, \beta' \in \Gamma^*$. Si ha*

$$(q, au, \beta) \underset{\mathcal{A}}{\vdash} (q, u, \beta') \qquad \textit{se e solo se} \qquad \beta \underset{L}{\Longrightarrow} a\beta' .$$

DIMOSTRAZIONE: Per la definizione della relazione $\vdash_{\mathcal{A}}$ si ha $(q, au, \beta) \vdash_{\mathcal{A}} (q, u, \beta')$ se e soltanto se

$$\beta = Z\alpha, \quad \beta' = \gamma\alpha, \quad \gamma \in \delta(q,a,Z)\,,$$

$Z \in \Gamma$, $\alpha, \gamma \in \Gamma^*$. D'altra parte, tenendo conto del fatto che β non contiene terminali, si ha $\beta \Longrightarrow_L a\beta'$ se e soltanto se

$$\beta = Z\alpha, \quad \beta' = \gamma\alpha, \quad Z \to a\gamma \text{ in } P\,,$$

$Z \in \Gamma$, $\alpha, \gamma \in \Gamma^*$. Se ne conclude che le due condizioni sono equivalenti. □

Ora siamo pronti per la

DIMOSTRAZIONE DELLA PROPOSIZIONE 13: Dobbiamo verificare che

$$L_P(\mathcal{A}) = L(G).$$

Iniziamo a supporre $w \in L(G)$. Allora per il Lemma 6 ci deve essere una derivazione sinistra di w in G. Tenendo conto del fatto che i lati destri delle produzioni hanno la forma $a\gamma$ con $a \in \Sigma \cup \{\varepsilon\}$ e $\gamma \in \Gamma^*$, tale derivazione avrà la forma:

$$\begin{aligned} S &\underset{L}{\Longrightarrow} a_1\beta_1 \underset{L}{\Longrightarrow} a_1a_2\beta_2 \\ &\underset{L}{\Longrightarrow} \cdots \underset{L}{\Longrightarrow} a_1a_2\cdots a_{n-1}\beta_{n-1} \underset{L}{\Longrightarrow} a_1a_2\cdots a_{n-1}a_n = w\,, \end{aligned} \tag{19.2}$$

$\beta_1, \beta_2, \ldots, \beta_{n-1} \in \Gamma^*$, $a_1, a_2, \ldots, a_n \in \Sigma \cup \{\varepsilon\}$. Si avrà pertanto

$$S \underset{L}{\Longrightarrow} a_1\beta_1, \quad \beta_1 \underset{L}{\Longrightarrow} a_2\beta_2, \quad \ldots, \quad \beta_{n-1} \underset{L}{\Longrightarrow} a_n\,, \tag{19.3}$$

e quindi, applicando il lemma precedente,

$$\begin{aligned}(q, w, S) = (q, a_1 a_2 \cdots a_n, S) &\underset{\mathcal{A}}{\vdash} (q, a_2 \cdots a_n, \beta_1) \\ &\underset{\mathcal{A}}{\vdash} \cdots \underset{\mathcal{A}}{\vdash} (q, a_n, \beta_{n-1}) \underset{\mathcal{A}}{\vdash} (q, \varepsilon, \varepsilon)\,.\end{aligned} \tag{19.5}$$

Questo dimostra che $w = a_1 a_2 \cdots a_n$ è accettata da $\mathcal{A}$ per pila vuota.

Viceversa, supponiamo $w \in L_P(\mathcal{A})$. Allora ci sarà una 'computazione' di $\mathcal{A}$ su w che termina con pila vuota. Varrà cioè l'Equazione (19.5). Dal Lemma precedente segue che valgono le (19.3) e, di conseguenza, la (19.2). Quindi $w \in L(G)$.

Abbiamo così dimostrato che $L_P(\mathcal{A}) = L(G)$. □

La Proposizione 13 mostra che i linguaggi accettati da automi a pila con un solo stato sono non contestuali. Mostreremo ora che tutti i linguaggi non contestuali sono accettati da automi a pila con un solo stato.

Proposizione 14 *Sia G una grammatica non contestuale. Esiste effettivamente un automa a pila $\mathcal{A}$ con un solo stato tale che*

$$L_P(\mathcal{A}) = L(G)\,.$$

DIMOSTRAZIONE: Senza perdita di generalità, possiamo supporre che tutte le produzioni di G abbiano la forma

$$Z \to a\gamma, \qquad Z \in \Gamma,\ a \in \Sigma \cup \{\varepsilon\},\ \gamma \in \Gamma^*.$$

Basta osservare, per esempio, che le produzioni delle grammatiche in forma normale di Chomsky hanno questa forma e applicare il Teorema 9. A questo punto possiamo definire l'automa a pila con un unico stato $\mathcal{A} = \langle \{q\}, \Sigma, \Gamma, \delta, q, Z_0, \{q\} \rangle$ come segue:

- L'alfabeto di input Σ è l'insieme dei simboli terminali di G,
- L'alfabeto di pila Γ è l'insieme delle variabili di G,
- Il simbolo iniziale della pila Z_0 è l'assioma di G,
- La funzione di transizione è definita da

$$\delta(q, a, Z) = \{(q, \gamma) \mid Z \to a\gamma \text{ in } P\}.$$

La Proposizione 13 ci assicura che $L_P(\mathcal{A})$ è generato da una grammatica non contestuale. Ma, ripercorrendo la costruzione di tale grammatica, ci si convince facilmente che essa è esattamente la nostra grammatica G. Ciò dimostra l'asserto. □

Esempio 63 Consideriamo il linguaggio $L = \{a^n b^n \mid n \geq 0\}$. L è generato dalla grammatica con produzioni

$$S \to aSb \mid \varepsilon\,.$$

Tale grammatica è equivalente a quella con le produzioni

$$S \to aSB \mid \varepsilon\,, \qquad B \to b\,,$$

in cui i terminali compaiono nei lati sinistri solo come lettere iniziali. L'automa a pila corrispondente avrà alfabeto di input $\{a, b\}$, alfabeto di pila $\{S, A\}$, simbolo iniziale della pila S e la funzione di transizione data dalla seguente tabella

	a	b	ε
S	SB	$-$	ε
B	$-$	ε	$-$

(Trattandosi di un automa con un unico stato, per semplicità, questo non è riportato nella tabella della funzione di transizione)

I risultati precedenti ci assicurano che un linguaggio L è non contestuale se e solo se è accettato per pila vuota da un automa a pila con un unico stato.

La seguente proposizione mostra che, in realtà, l'ipotesi che l'automa a pila abbia un unico stato non è restrittiva.

Proposizione 15 *Sia $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$ un automa a pila. Esiste effettivamente un automa a pila $\mathcal{A}'$ a un solo stato tale che*

$$L_P(\mathcal{A}) = L_P(\mathcal{A}')\,.$$

DIMOSTRAZIONE: L'idea della dimostrazione è di simulare l'automa dato con un automa a un solo stato che registra nella pila anche lo stato dell'automa simulato. Più precisamente definiamo l'automa a pila

$$\mathcal{A}' = \langle \{q\}, \Sigma, \Gamma', \delta', q, Z_0', \{q\}\rangle$$

con

- alfabeto di pila $\Gamma' = Q \times \Gamma \times Q$,
- simbolo iniziale della pila $Z_0' = (q_0, Z_0, q_0)$,
- funzione di transizione δ' definita come segue: se $(p, \varepsilon) \notin \delta(s, a, Z)$ allora

$$\delta'(a, (s, Z, p)) = \{(t, Z_k, p_{k-1})\,(p_{k-1}, Z_k, p_{k-2}) \cdots (p_1, Z_1, p) \mid (t, Z_k \cdots Z_1) \in \delta(s, a, Z),\ p_1, \ldots, p_{k-1} \in Q\}\,,$$

se invece $(p, \varepsilon) \in \delta(s, a, Z)$ allora

$$\delta'(a, (s, Z, p)) = \{(t, Z_k, p_{k-1})\,(p_{k-1}, Z_k, p_{k-2}) \cdots (p_1, Z_1, p) \mid \\ (t, Z_k \cdots Z_1) \in \delta(s, a, Z),\ p_1, \ldots, p_{k-1} \in Q\} \cup \{\varepsilon\}\,,$$

$s, p \in Q,\ a \in \Sigma \cup \{\varepsilon\},\ Z \in \Gamma$.

Lasciamo al lettore il compito di verificare che c'è una computazione di $\mathcal{A}$ su w che porta nello stato s con la pila contenente la parola $\gamma = Y_n Y_{n-1} \cdots Y_1$ se e solo se per ogni $q_1, \ldots, q_{n-1} \in Q$ c'è una computazione di $\mathcal{A}'$ su w che termina con la pila contenente la parola

$$(s, Y_n, q_{n-1})(q_{n-1}, Y_{n-1}, q_{n-2}) \cdots (q_1, Y_1, q_0)\,,$$

e di dedurre da questo che $L_P(\mathcal{A}) = L_P(\mathcal{A}')$. □

Tenuto conto dei risultati precedenti possiamo enunciare il seguente fondamentale

Teorema 12 *Un linguaggio è non contestuale se e solo se è riconosciuto da un automa a pila.*

Capitolo 20

Parsing top-down deterministico

Si consideri l'Esempio 56. In quel caso, il parsing è notevolmente accelerato dal fatto che, a ogni passo, scartiamo le produzioni in cui la prima lettera del lato sinistro sia un terminale diverso dal simbolo di look-up.

In generale, non sempre i lati destri delle produzioni iniziano con un simbolo terminale. Al fine di ridurre per quanto possibile il non-determinismo, sarebbe opportuno quindi disporre di uno strumento che ci permetta di sapere in anticipo quali sono i lati destri di una produzione da cui è possibile derivare una parola che inizia con un dato simbolo terminale.

Vediamo come formalizzare questa idea. Sia $G = \langle V, \Sigma, P, S \rangle$ una grammatica non-contestuale. Supporremo che G sia priva di variabili inaccessibili o improduttive. Tenuto conto della Proposizione 8, questa ipotesi non è restrittiva.

Per il momento, ci limiteremo a considerare il caso in cui la grammatica G sia priva di ε-produzioni.

Definiamo la funzione FIRST : $V^+ \to \wp(\Sigma)$ ponendo

$$\text{FIRST}(\alpha) = \{a \in \Sigma \mid \alpha \stackrel{*}{\Longrightarrow} a\beta\,,\ \beta \in V^*\}$$

Detto in altri termini, FIRST(α) è l'insieme dei terminali che possono comparire come prima lettera di una conseguenza di α.

Poniamoci ora il problema del parsing di una parola $w \in L(G)$ o, equivalentemente, di determinare una derivazione sinistra

$$S\# = \alpha_0 \underset{L}{\Longrightarrow} \alpha_1 \underset{L}{\Longrightarrow} \alpha_2 \underset{L}{\Longrightarrow} \cdots \underset{L}{\Longrightarrow} \alpha_n = w\#\,.$$

Supponiamo di aver determinato α_i per un certo $i < n$. Possiamo scrivere

$$\alpha_i = uX\beta\,, \quad \alpha_{i+1} = u\gamma\beta\,, \quad X \to \gamma \text{ in } P\,,$$

con $u \in \Sigma^*$, $X \in N$ e $\beta \in V^*$. Inoltre, poiché sappiamo che $\alpha_{i+1} \Longrightarrow_L^* w\#$ avremo anche

$$w\# = uv\,, \quad \gamma\beta \overset{*}{\underset{L}{\Longrightarrow}} v\,, \quad v \in \Sigma^*\,.$$

Inoltre $v \neq \varepsilon$, poiché almeno la lettera # compare in v. Indichiamo con a la prima lettera di v. Dalla condizione $\gamma\beta \Longrightarrow_L^* v$ segue allora che $a \in \text{FIRST}(\gamma)$. Ne concludiamo che la produzione $X \to \gamma$ deve essere tra quelle che soddisfano la condizione

$$a \in \text{FIRST}(\gamma) \tag{20.1}$$

Sarebbe quindi opportuno costruire una *tabella di parsing* della grammatica G in cui a ogni coppia $(X, a) \in N \times \Sigma$ corrispondono le produzioni $X \to \gamma$ per cui è soddisfatta la (20.1).

Definizione 41 Una grammatica non-contestuale G priva di ε-produzioni si dice di classe $LL(1)$ se per ogni coppia di produzioni distinte $X \to \alpha \mid \gamma$ si ha $\text{FIRST}(\alpha) \cap \text{FIRST}(\gamma) = \emptyset$.

In altri termini, una grammatica $LL(1)$ è una grammatica la cui tabella di parsing contiene al più una produzione in ogni ingresso. Questo assicura che il parsing può essere effettuato in maniera deterministica.

Il termine $LL(1)$ indica che è possibile la costruzione efficiente di una derivazione sinistra (Left-most) esaminando l'input da sinistra (Left) a destra e esaminando solo 1 lettera dell'input non ancora presente nell'albero di derivazione.

Una volta creata la tabella di parsing, l'algoritmo di parsing può essere realizzato sostanzialmente con un automa a pila (cf. Algoritmo 14).

Resta da capire come si possa costruire la tabella di parsing. Iniziamo con le seguenti osservazioni:

1. se la prima lettera di α è un simbolo terminale b, allora $\text{FIRST}(\alpha) = b$;
2. se la prima lettera di α è una variabile X, allora per ogni produzione $X \to \beta$, i terminali di $\text{FIRST}(\beta)$ sono anche terminali di $\text{FIRST}(\alpha)$,
3. Tutti gli elementi di $\text{FIRST}(\alpha)$ con α lato destro di qualche produzione, si trovano applicando un numero finito di volte le regole precedenti.

Le regole precedenti ci forniscono una procedura effettiva per il calcolo di $\text{FIRST}(\alpha)$ per tutti gli α che sono lati destri delle produzioni. Inizialmente, poremo $\text{FIRST}(\alpha) = \emptyset$ per tutti gli α. Poi ricalcoliamo tali insiemi usando le regole 1–2 e i valori attuali degli insiemi $\text{FIRST}(\beta)$. Iteriamo quest'ultimo passo, arrestandoci solo quando nessuno degli insiemi $\text{FIRST}(\alpha)$ subisce modifiche.

Algoritmo 10: parsingLL1

Ingresso: La tabella di parsing tabella di una grammatica non contestuale e una parola $w = a_1 a_2 \cdots a_n \#$

Uscita: Le produzioni di una derivazione sinistra di w

```
i ← 1;
push(#, stack);
push(S, stack);
ripeti
    X ← pop(stack);
    se X ∈ Σ allora    // X è un terminale
        se X = a_i allora
            i ← i + 1;
        altrimenti
            Errore
    altrimenti    // X è una variabile
        α = tabella[X, a_i];
        push(α, stack);
        output(X → α);
finché X = #;
```

Esempio 64 Si consideri la grammatica $G = \langle V, \Sigma, P, S \rangle$ con $\Sigma = \{a, b, c, q\}$, $N = \{S, F, C, Q\}$ e con le produzioni

$$S \to AB \mid DC, \quad A \to a \mid aA, \quad B \to bc \mid bBc, \quad C \to c \mid cC, \quad D \to ab \mid aDb.$$

Il calcolo degli insiemi FIRST(α) con α lato destro di una produzione è riportato nella tabella seguente:

	AB	DC	a	aA	bc	bBc	c	cC	ab	aDb
1.	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
2.	$\emptyset$	$\emptyset$	a	a	b	b	c	c	a	a
3.	a	a	a	a	b	b	c	c	a	a
4.	a	a	a	a	b	b	c	c	a	a

Inizialmente assegniamo valore vuoto a tutti gli insiemi.

Al secondo passo, per la regola 1 per tutti i lati destri che iniziano con un terminale, inseriamo tale terminale nell'insieme FIRST.

Al terzo passo, per la regola 2, dobbiamo aggiungere a FIRST(AB) gli elementi di FIRST(a) e FIRST(aA) e a FIRST(DC) gli elementi di FIRST(ab) e FIRST(aDb).

Quest'ultima operazione va ripetuta finché i dati non si stabilizzano, cosa che avviene al passo successivo.

Consideriamo ora il caso in cui la nostra grammatica abbia anche delle ε-produzioni. In tal caso, definiamo le funzioni FIRST : $V^* \to \wp(\Sigma \cup \{\varepsilon\})$ e FOLLOW : $N \to \wp(\Sigma)$ ponendo FIRST$(\varepsilon) = \{\varepsilon\}$ e

$$\begin{aligned}
\text{FIRST}(\alpha) &= \{a \in \Sigma \mid \alpha \overset{*}{\Longrightarrow} a\beta\,,\ \beta \in V^*\}\,, && \alpha \in V^*,\ \alpha \not\Longrightarrow^* \varepsilon\,,\\
\text{FIRST}(\alpha) &= \{a \in \Sigma \mid \alpha \overset{*}{\Longrightarrow} a\beta\,,\ \beta \in V^*\} \cup \{\varepsilon\}\,, && \alpha \in V^*,\ \alpha \overset{*}{\Longrightarrow} \varepsilon\,,\\
\text{FOLLOW}(X) &= \{a \in \Sigma \mid S\# \overset{*}{\Longrightarrow} \beta X a \gamma\,,\ \beta, \gamma \in V^*\}\,, && X \in N\,.
\end{aligned}$$

Detto in altri termini, FIRST(α) è l'insieme dei terminali che possono comparire come prima lettera di una conseguenza di α, aumentato della parola ε nel caso in cui α sia annullabile. Invece, FOLLOW(X) è l'insieme dei terminali che possono seguire la variabile X in una forma sentenziale.

Vediamo come utilizzare tali funzioni nel parsing top-down. Come nel caso precedente, vogliamo determinare una derivazione sinistra

$$S\# = \alpha_0 \underset{L}{\Longrightarrow} \alpha_1 \underset{L}{\Longrightarrow} \alpha_2 \underset{L}{\Longrightarrow} \cdots \underset{L}{\Longrightarrow} \alpha_n = w\#\,.$$

Supponiamo di aver determinato α_i per un certo $i < n$. Possiamo scrivere

$$\alpha_i = uX\beta\,, \quad \alpha_{i+1} = u\gamma\beta\,, \quad X \to \gamma \text{ in } P\,,$$

con $u \in \Sigma^*$, $X \in N$ e $\beta \in V^*$. Inoltre, poiché sappiamo che $\alpha_{i+1} \Longrightarrow_L^* w\#$ avremo anche

$$w\# = uv\,, \quad \gamma\beta \underset{L}{\overset{*}{\Longrightarrow}} v\,, \quad v \in \Sigma^*\,.$$

Inoltre $v \neq \varepsilon$, poiché almeno la lettera # compare in v. Indichiamo con a la prima lettera di v. Dalla condizione $\gamma\beta \Longrightarrow_L^* v$ segue allora che o $a \in \text{FIRST}(\gamma)$ o $\gamma \Longrightarrow^* \varepsilon$ e $\beta \Longrightarrow^* v$. Nel secondo caso, si ha anche $S \Longrightarrow^* uX\beta \Longrightarrow^* uXv$ e quindi $a \in \text{FOLLOW}(X)$. Concludiamo che la produzione $X \to \gamma$ deve essere tra quelle che soddisfano una delle condizioni

$$a \in \text{FIRST}(\gamma) \quad \text{oppure} \quad \varepsilon \in \text{FIRST}(\gamma) \text{ e } a \in \text{FOLLOW}(X)\,. \tag{20.2}$$

Dovremo quindi costruire la *tabella di parsing* della grammatica G in cui a ogni coppia $(X, a) \in N \times \Sigma$ corrispondono le produzioni $X \to \gamma$ per cui è soddisfatta la (20.2).

Definizione 42 Una grammatica non-contestuale G si dice di classe $LL(1)$ se per ogni coppia di produzioni distinte $X \to \alpha$, $X \to \gamma$ sono soddisfatte le seguenti condizioni:

1. $\text{FIRST}(\alpha) \cap \text{FIRST}(\gamma) = \emptyset$.

2. Se $\alpha \Longrightarrow^* \varepsilon$, allora $\gamma \not\Longrightarrow^* \varepsilon$ e $\text{FOLLOW}(X) \cap \text{FIRST}(\gamma) = \emptyset$.

Per poter costruire la tabella di parsing nel caso in cui siano presenti anche delle ε-transizioni, dobbiamo calcolare le funzioni FIRST and FOLLOW in questo caso più generale. Iniziamo con le seguenti osservazioni:

1. se la prima lettera di α è un simbolo terminale b, allora $\text{FIRST}(\alpha) = b$;

2. se la prima lettera di α è una variabile X, allora per ogni produzione $X \to \beta$, i terminali di $\text{FIRST}(\beta)$ sono anche terminali di $\text{FIRST}(\alpha)$,

3. se la prima lettera di α è una variabile annullabile, allora le due condizioni precedenti sono verificate anche dalla seconda lettera di α; se quest'ultima è una variabile annullabile, allora anche dalla terza, e così via;

4. se tutte le lettere di α sono variabili annullabili, allora $\varepsilon \in \text{FIRST}(\alpha)$;

5. Tutti gli elementi di $\text{FIRST}(\alpha)$ con α lato destro di qualche produzione, si trovano applicando un numero finito di volte le regole precedenti.

Le regole precedenti ci forniscono quindi una procedura effettiva per il calcolo di $\text{FIRST}(\alpha)$ per tutti gli α che sono lati destri delle produzioni. Inizialmente, porremo $\text{FIRST}(\alpha) = \emptyset$ per tutti gli α. Poi ricalcoliamo tali insiemi usando le regole 1–4 e i valori attuali degli insiemi $\text{FIRST}(\beta)$. Iteriamo quest'ultimo passo, arrestandoci solo quando nessuno degli insiemi $\text{FIRST}(\alpha)$ subisce modifiche.

Esempio 65 Si consideri la grammatica $G = \langle V, \Sigma, P, S \rangle$ con $\Sigma = \{a, b, c, q\}$, $N = \{S, F, C, Q\}$ e con le produzioni

$$S \to FQ \mid aSbS, \quad F \to CF \mid \varepsilon, \quad C \to c, \quad Q \to q.$$

Come si verifica facilmente, l'unica variabile annullabile è F. Il calcolo degli insiemi $\text{FIRST}(\alpha)$ con $\alpha = FQ, aSbS, CF, \varepsilon, c, q$ è riportato nella tabella seguente:

	FQ	$aSbS$	CF	ε	c	q
1.	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
2.	$\emptyset$	a	$\emptyset$	ε	c	q
3.	q	a	c	ε	c	q
4.	c, q	a	c	ε	c	q
5.	c, q	a	c	ε	c	q

Inizialmente assegniamo valore vuoto a tutti gli insiemi.

Al secondo passo, dalla regola 1 si ottiene FIRST$(aSbS) = \{a\}$, FIRST$(c) = \{c\}$ e FIRST$(q) = q$. Gli insiemi FIRST(FQ) e FIRST(CF) vanno calcolati usando le regole 2 e 3, ma, avendosi al passo precedente FIRST(F) = FIRST(C) = FIRST$(Q) = \emptyset$, risultano entrambi vuoti.

Al terzo passo, FIRST$(aSbS)$, FIRST(c) e FIRST(q) saranno invariati. L'insieme FIRST(FQ) viene posto uguale ai terminali che compaiono nei FIRST dei lati destri delle produzioni di F (per la regola 2) e di Q (per la regola 3), cioè tutti i terminali che compaiono in FIRST(CF), FIRST(ε) e FIRST(q)). L'insieme FIRST(CF) dovrà invece essere posto uguale ai terminali che compaiono in FIRST(c) (regola 2).

Quest'ultima operazione va ripetuta finché i dati non si stabilizzano, cosa che avviene al passo 5.

Per il calcolo di FOLLOW iniziamo con le seguenti osservazioni:

1. L'insieme FOLLOW(S) contiene #.
2. Se c'è una produzione $X \to \alpha Y \beta$ con $X, Y \in N$, $\alpha, \beta \in V^*$, allora tutti i terminali contenuti in FIRST(β) sono contenuti anche in FOLLOW(Y).
3. Se inoltre $\varepsilon \in$ FIRST(β), cioè $\beta \Longrightarrow^* \varepsilon$, allora tutti i terminali contenuti in FOLLOW(X) sono contenuti anche in FOLLOW(Y).
4. Tutti gli elementi di FOLLOW(Y), $Y \in N$, si trovano applicando un numero finito di volte le regole precedenti.

Anche qui, le regole enunciate ci forniscono una procedura effettiva per il calcolo di FOLLOW(Y) per tutte le variabili Y. Inizialmente poniamo gli insiemi FOLLOW(Y) uguali a ciò che si ottiene applicando le regole 1. e 2. Poi aggiungiamo a tali insiemi i terminali ottenuti usando la regola 3 e i valori attuali degli insiemi FOLLOW(Y). Iteriamo quest'ultimo passo, finché i valori non si stabilizzano.

Esempio 66 Calcoliamo la funzione FOLLOW sulle variabili della grammatica introdotta nell'Esempio 65.

Il calcolo di FOLLOW(Y), $Y = S, F, C, Q$ è qui riportato.

	S	F	C	Q
1.	$\#, b$	q	c	$\emptyset$
2.	$\#, b$	q	c, q	$\#$
3.	$\#, b$	q	c, q	$\#$

Al primo passo, poniamo # in FOLLOW(S) per la regola 1 mentre la regola 2 ci impone di aggiungere a FOLLOW(S) i terminali di FIRST(bS), a FOLLOW(F)

i terminali di FIRST(Q), a FOLLOW(C) i terminali di FIRST(F). Il calcolo di FIRST(bS), FIRST(Q) e FIRST(F) si esegue applicando le regole 1–5 precedentemente enunciate, tenendo conto che per ogni produzione $X \to \beta$, i terminali di FIRST(β) sono già stati calcolati nell'Esempio 65. Si ottiene quindi FIRST$(bS) = \{b\}$, FIRST$(Q) = \{q\}$ e FIRST$(F) = \{c, \varepsilon\}$.

Al secondo passo, la regola 3 ci impone di aggiungere a FOLLOW(C) i terminali contenuti in FOLLOW(F) e a FOLLOW(Q) i terminali contenuti in FOLLOW(S).

Poiché ci sono state modifiche, tale passo và ripetuto. Dato che la ripetizione non determina alcuna modifica, il computo termina.

Esempio 67 Costruiamo ora la tabella di parsing della grammatica introdotta nell'Esempio 65.

In questa tabella le linee corrispondono alle variabili e le colonne ai terminali. Nella cella corrispondente alla generica coppia (X, a), dobbiamo inserire i lati destri delle produzioni $X \to \gamma$ che soddisfano la Condizione (20.2):

	a	b	c	q	#
S	$aSbS$		FQ	FQ	
F			CF	ε	
C			c		
Q				q	

La presenza di una produzione al più in ciascuna cella assicura che la grammatica è di tipo LL(1). Ciò permetterà l'esecuzione efficiente del parsing.

Capitolo 21

Proprietà di chiusura

Stabiliremo ora alcune proprietà di chiusura della classe dei linguaggi non contestuali. Iniziamo con le cosiddette operazioni regolari.

Proposizione 16 *La classe dei linguaggi non contestuali è chiusa per unione, concatenazione e chiusura di Kleene.*

DIMOSTRAZIONE: Siano $L_1, L_2 \subseteq \Sigma^*$ due linguaggi non contestuali e siano rispettivamente

$$G_1 = \langle V_1, \Sigma, P_1, S_1 \rangle \quad \text{e} \quad G_2 = \langle V_2, \Sigma, P_2, S_2 \rangle \,,$$

due grammatiche non contestuali che li generano. Senza perdita di generalità, possiamo supporre che G_1 e G_2 non abbiano variabili comuni. Consideriamo ora la grammatica non contestuale $G = \langle V, \Sigma, P, S \rangle$ con vocabolario $V = V_1 \cup V_2 \cup \{S\}$, ove S è una nuova variabile che non compare nè in G_1 nè in G_2, con le produzioni di G_1 e di G_2 e inoltre le produzioni

$$S \to S_1 \mid S_2 \,. \tag{21.1}$$

Non è difficile convincersi che il linguaggio generato da questa grammatica è l'unione $L_1 \cup L_2$, che pertanto è un linguaggio non contestuale.

Se poi sostituiamo le produzioni (21.1) con la produzione

$$S \to S_1 S_2 \,.$$

si ottiene una grammatica che genera la concatenazione $L_1 L_2$ dei due linguaggi considerati. Quindi anche questa è un linguaggio non contestuale.

Consideriamo infine la grammatica non contestuale $G' = \langle V', \Sigma, P', S \rangle$ con vocabolario $V = V_1 \cup \{S\}$, con le produzioni di G_1 e inoltre le produzioni

$$S \to S_1 S \mid \varepsilon \,.$$

Non è difficile convincersi che il linguaggio generato da questa grammatica è la chiusura di Kleene L_1^*, che pertanto è anch'essa un linguaggio non contestuale. □

Esempio 68 Come sappiamo, i linguaggi

$$L_1 = \{a^n b^n \mid n \geq 0\}, \qquad L_2 = \{b^n c^n \mid n \geq 0\},$$

sono non contestuali, così come i linguaggi regolari a^* e c^*. Per la proposizione precedente, saranno quindi non contestuali anche i linguaggi

$$L_3 = L_1 c^* = \{a^n b^n c^m \mid m, n \geq 0\}, \quad L_4 = a^* L_2 = \{a^m b^n c^n \mid m, n \geq 0\}.$$

D'altra parte l'intersezione di L_3 e L_4 è il linguaggio

$$L = L_3 \cap L_4 = \{a^n b^n c^n \mid n \geq 0\},$$

che, come abbiamo visto in precedenza (cf. Esempio 52) non è non contestuale.

L'esempio precedente ci mostra che la famiglia dei linguaggi non contestuali non è chiusa per intersezione. Osserviamo poi che tale famiglia non è chiusa nemmeno per complementazione. Invero è ben noto che l'intersezione si può esprimere mediante le operazioni di unione e complementazione. Pertanto, se la famiglia dei linguaggi non contestuali fosse chiusa per complementazione, essendo chiusa anche per unione, risulterebbe chiusa per intersezione, ma come abbiamo appena mostrato questo non avviene.

Sebbene i linguaggi non contestuali non siano chiusi per intersezione, vale la seguente proprietà più debole:

Proposizione 17 *L'intersezione di un linguaggio non contestuale con un linguaggio regolare è un linguaggio non contestuale.*

Senza volerci addentrare nei dettagli della dimostrazione di questa proposizione, ci limitiamo a osservare che un linguaggio regolare R è accettato da un automa a stati finiti $\mathcal{A}_1$ mentre un linguaggio non contestuale L è accettato da un automa a pila $\mathcal{A}_2$ con un solo stato. L'intersezione di tali linguaggi sarà quindi accettata, per stato finale e pila vuota, da un automa a pila in cui le transizioni di stato simulano il funzionamento di $\mathcal{A}_1$ mentre l'evoluzione della pila simula il funzionamento di $\mathcal{A}_2$.

Esempio 69 Sia D_1 il linguaggio semi-Dick su una lettera (cf. Esempio 36). Si ha $D_1 \cap a_1^* b_1^* = \{a_1^n b_1^n \mid n \geq 0\}$ e, come sappiamo, questo linguaggio è non contestuale.

Invece il linguaggio $L = D_2 \cap \{a_1, a_2\}^*\{b_1, b_2\}^*$ è l'insieme delle parole uv in cui $u \in \{a_1, a_2\}^*$ e v è ottenuta da u sostituendo le lettere a_1 e a_2 rispettivamente con b_1 e b_2 e capovolgendo la parola ottenuta. Pertanto il linguaggio delle palindrome pari su $\{a_1, a_2\}$ si ottiene da L sostituendo, in ogni parola, le lettere b_1 e b_2 rispettivamente con a_1 e a_2.

Siano Σ_1 e Σ_2 due alfabeti e si consideri una funzione $f : \Sigma_1 \to \Sigma_2^*$. È possibile estendere tale funzione a Σ_1^* ponendo

$$f(\varepsilon) = \varepsilon, \quad f(a_1 a_2 \cdots a_n) = f(a_1) f(a_2) \cdots f(a_n),$$

$a_1, a_2, \ldots, a_n \in \Sigma$, $n \geq 2$. Una funzione così definita si dice un *omomorfismo* (o sostituzione univoca).

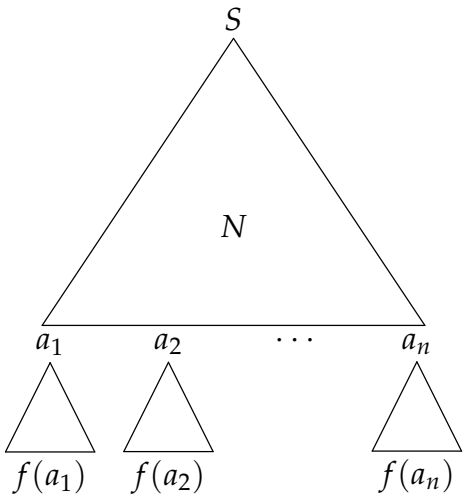
Proposizione 18 *La classe dei linguaggi non contestuali è chiusa per omomorfismi.*

DIMOSTRAZIONE: Siano $L \subseteq \Sigma_1^*$ un linguaggio non contestuale e $f : \Sigma_1^* \to \Sigma_2^*$ un omomorfismo. Ci sarà allora una grammatica non contestuale $G = \langle V, \Sigma_1, P, S \rangle$ che accetta L. Senza perdita di generalità, possiamo supporre che $V \cap \Sigma_2 = \emptyset$.

Costruiamo una grammatica che genera $f(L_1)$. Invero è sufficiente considerare la grammatica $G' = \langle V \cup \Sigma_2, \Sigma_2, P', S \rangle$ le cui variabili sono tutte le lettere di V (variabili e terminali), con le produzioni di G e inoltre le produzioni

$$a \to f(a)\,, \qquad a \in \Sigma_1\,.$$

Analizzando la struttura degli alberi di derivazione in questa nuova grammatica, non è difficile convincersi che essa genera proprio il linguaggio $f(L)$. □



Le Proposizioni 17 e 18 ci assicurano che intersecando un linguaggio semi-Dick con un linguaggio regolare e applicando un morfismo, otteniamo un linguaggio non contestuale. Il seguente Teorema di Rappresentazione ci assicura che in questo modo possiamo ottenere tutti i linguaggi non contestuali.

Teorema 13 (Chomsky, Schützenberger) *Un linguaggio L è non contestuale se e soltanto se esistono un intero $k > 0$, un linguaggio regolare R e un morfismo f tali che*

$$L = f(D_k \cap R)\,.$$