Argomenti trattati durante la lezione:

- Linguaggi context-free. Grammatiche
- Grammatiche: ambiguità e forme normali

CF: Linguaggio naturale/parsing/interprete

La grammatica G_1:

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

- insieme di regole di sostituzione (o produzioni)
- variabili: A, B
- terminali (simboli dell'alfabeto): $0, 1, \#$
- variabile iniziale: A

Definition

Una grammatica context-free è una quadrupla (V, Σ, R, S), dove

- V è un insieme finito di variabili
- Σ è un insieme finito di terminali disgiunto da V
- R è un insieme di regole, dove ogni regola è una variabile e una stringa di variabili e terminali
- $S \in V$ è la variabile iniziale

Una grammatica genera stringhe nel seguente modo:

1. Scrivi la variabile iniziale
2. Trova una variabile che è stata scritta e una regola che inizia con quella variabile. Sostituisci la variabile con il lato destro della regola
3. Ripeti 2 fino a quando non ci sono più variabili

Esempio per G_1:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000\#111$$

La sequenza di sostituzioni si chiama derivazione di $000\#111$

Molti linguaggi sono unione di linguaggi più semplici

- Se il linguaggio è regolare, possiamo trovare un DFA che lo riconosce

(1) Unione di linguaggi più semplici

- Molti linguaggi sono unione di linguaggi più semplici
- Idea:
  - costruisci grammatiche separate per ogni componente
  - unisci le grammatiche con una nuova regola iniziale

$$S \to S_1 \mid S_2 \mid \cdots \mid S_k$$

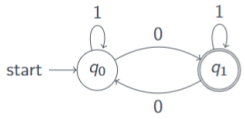
dove $S_1, S_2, \ldots, S_k$ sono le regole iniziali delle componenti

Esempio: grammatica per $\{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$

- Grammatica per 0^n1^n: $S_1 \to 0S_11 \mid \varepsilon$
- Grammatica per 1^n0^n: $S_2 \to 1S_20 \mid \varepsilon$
- Unione: $S \to S_1 \mid S_2$

(2) Trasformare un DFA in CFG

- DFA per $\{w \in \{0,1\}^* \mid w \text{ contiene un numero dispari di } 0\}$:



- Grammatica context-free:

$$R_0 \to 0R_1 \mid 1R_0$$
$$R_1 \to 0R_0 \mid 1R_1 \mid \varepsilon$$

Vediamo subito alcuni esempi:

Problem 1 Give a context-free grammar that generates the following language over $\{0,1\}^*$:

$$L = \{w | w \text{ contains more 1s than 0s}\}$$

Idea: this is similar to the language where the number of 0s is equal to the number of 1s, except we must ensure that we generate at least one 1, and we must allow an arbitrary number of 1s to be generated anywhere in the derivation. The following grammar accomplishes this task:

$$\begin{aligned} S &\to S_1 1 S_1 \\ S_1 &\to 0S_1 1 | 1S_1 0 | S_1 S_1 | 1S_1 | \epsilon \end{aligned}$$

Problem 2 Give a context-free grammar generating the language

$$L = \text{ the complement of the language } \{a^n b^n | n \geq 0\}.$$

Idea: we can break this language into the union of several simpler languages: $L = \{a^i b^j | i > j\} \cup \{a^i b^j | i < j\} \cup (a \cup b)^* b (a \cup b)^* a (a \cup b)^*$. That is, all strings of a's followed by b's in which the number of a's and b's differ, unioned with all strings *not* of the form $a^i b^j$.

First, we can achieve the union of the CFGs for the three languages:

$$S \to S_1 | S_2 | S_3$$

Now, the set of strings $\{a^i b^j | i > j\}$ is generated by a simple CFG:

$$S_1 \to aS_1 b | aS_1 | a$$

Similarly for $\{a^i b^j | i < j\}$:

$$S_2 \to aS_2 b | S_2 b | b$$

Finally, $(a \cup b)^* b (a \cup b)^* a (a \cup b)^*$ is easily generated as follows:

$$\begin{aligned} S_3 &\to XbXaX \\ X &\to aX | bX | \epsilon \end{aligned}$$

2.4 Give context-free grammars that generate the following languages. In all parts, the alphabet Σ is {0,1}.

A a. $\{w |$ w contains at least three 1s$\}$

A d. $\{w |$ the length of w is odd and its middle symbol is a 0$\}$

2.4 **(a)** $S \to R1R1R1R$
$R \to 0R \mid 1R \mid \varepsilon$

(d) $S \to 0 \mid 0S0 \mid 0S1 \mid 1S0 \mid 1S1$

2.6 Give context-free grammars generating the following languages.

A a. The set of strings over the alphabet {a,b} with more a's than b's

A c. $\{w\#x |$ $w^{\mathcal{R}}$ is a substring of x for $w, x \in \{0,1\}^*\}$

2.6 **(a)** $S \to TaT$
$T \to TT \mid aTb \mid bTa \mid a \mid \varepsilon$
T generates all strings with at least as many a's as b's, and S forces an extra a.

(c) $S \to TX$
$T \to 0T0 \mid 1T1 \mid \#X$
$X \to 0X \mid 1X \mid \varepsilon$

Con le regole, riusciamo a costruire un albero sintattico (parsing tree):

- nodi interni sono variabili
- foglie sono variabili, simboli terminali o ϵ

Esistono delle grammatiche *ambigue*, dato che è possibile generare una stessa stringa *in modi diversi*. Occorre quindi definire un *ordine* di derivazione.

- Derivazione a sinistra (leftmost derivation): ad ogni passo, sostituisco la variabile che si trova più a sinistra.

- Una stringa w è derivata ambiguamente dalla grammatica G se esistono due o più alberi sintattici che la generano
- Equivalente: Una stringa w è derivata ambiguamente dalla grammatica G se esistono due o più derivazioni a sinistra che la generano
- Una grammatica è ambigua se genera almeno una stringa ambiguamente

Problem 3) Give a CFG to generate

$$A = \{a^i b^j c^k | i, j, k \geq 0 \text{ and either } i = j \text{ or } j = k\}.$$

Is the grammar ambiguous? Why or why not?

Idea: this language is simply the union of $A_1 = \{a^i b^j c^k | i, j, k \geq 0, i = j\}$ and $A_2 = \{a^i b^j c^k | i, j, k \geq 0, j = k\}$. We can create simple grammars for the separate languages and union them:

$$S \to S_1 | S_2$$

For A_1, we simply ensure that the number of a's equals the number of b's:

$$\begin{aligned} S_1 &\to S_1 c | A | \epsilon \\ A &\to aAb | \epsilon \end{aligned}$$

Similarly for ensuring that the number of b's equals the number of c's:

$$\begin{aligned} S_2 &\to aS_2 | B | \epsilon \\ B &\to bBc | \epsilon \end{aligned}$$

This grammar is ambiguous. For $x = a^n b^n c^n$, we may use either S_1 or S_2 to generate x.

***2.27** Let $G = (V, \Sigma, R, \langle \text{STMT} \rangle)$ be the following grammar.

$$\begin{aligned} \langle \text{STMT} \rangle &\to \langle \text{ASSIGN} \rangle \mid \langle \text{IF-THEN} \rangle \mid \langle \text{IF-THEN-ELSE} \rangle \\ \langle \text{IF-THEN} \rangle &\to \texttt{if condition then } \langle \text{STMT} \rangle \\ \langle \text{IF-THEN-ELSE} \rangle &\to \texttt{if condition then } \langle \text{STMT} \rangle \texttt{ else } \langle \text{STMT} \rangle \\ \langle \text{ASSIGN} \rangle &\to \texttt{a:=1} \end{aligned}$$

$$\begin{aligned} \Sigma &= \{\texttt{if}, \texttt{condition}, \texttt{then}, \texttt{else}, \texttt{a:=1}\} \\ V &= \{\langle \text{STMT} \rangle, \langle \text{IF-THEN} \rangle, \langle \text{IF-THEN-ELSE} \rangle, \langle \text{ASSIGN} \rangle\} \end{aligned}$$

G is a natural-looking grammar for a fragment of a programming language, but G is ambiguous.

a. Show that G is ambiguous.

b. Give a new unambiguous grammar for the same language.

a. We can easily prove by driving the sentence:
`if condition then if condition then a := 1 else a := 1`
Using two different leftmost derivation, as in the following:

First One:

```
STMT => IF-THEN-ELSE
     => if condition then STMT else STMT
     => if condition then IF-THEN  else STMT
      => if condition then if condition then STMT else STMT
     => if condition then if condition then ASSIGN else STMT
     => if condition then if condition then a := 1 else STMT
     => if condition then if condition then a := 1 else ASSIGN
     => if condition then if condition then a := 1 else a := 1
```

Second One:

```
STMT => IF-THEN
     => if condition then STMT
     => if condition then IF-THEN-ELSE
     => if condition then if condition then STMT else STMT
     => if condition then if condition then ASSIGN else STMT
     => if condition then if condition then a := 1 else STMT
     => if condition then if condition then a := 1 else ASSIGN
     => if condition then if condition then a := 1 else a := 1
```

b.

⟨STMT⟩ → ⟨ASSIGN⟩ | ⟨IF-THEN-ELSE⟩
⟨IF-THEN-ELSE⟩ → if condition then ⟨T⟩ else ⟨STMT⟩ | T
T → ⟨IF-THEN⟩
⟨IF-THEN⟩ → if condition then ⟨STMT⟩
⟨ASSIGN⟩ → a:=1

È spesso conveniente avere le grammatiche in una forma semplificata. Una delle forme più semplici e utili è la Forma Normale di Chomsky.

Definition

Una grammatica context-free è in forma normale di Chomsky se ogni regola è della forma

$$A \to BC$$
$$A \to a$$

dove a è un terminale, B, C non possono essere la variabile iniziale. Inoltre, ci può essere la regola $S \to \varepsilon$ per la variabile iniziale S

Theorem

Ogni linguaggio context-free è generato da una grammatica in forma normale di Chomsky

Idea: possiamo trasformare una grammatica G in forma normale di Chomsky:

1. aggiungiamo una nuova variabile iniziale
2. eliminiamo le ε-regole $A \to \varepsilon$
3. eliminiamo le regole unitarie $A \to B$
4. trasformiamo le regole rimaste nella forma corretta

1. The original CFG G_6 is shown on the left. The result of applying the first step to make a new start variable appears on the right.

$$\begin{aligned} S &\to ASA \mid \mathtt{a}B \\ A &\to B \mid S \\ B &\to \mathtt{b} \mid \varepsilon \end{aligned} \qquad \begin{aligned} \boldsymbol{S_0} &\to \boldsymbol{S} \\ S &\to ASA \mid \mathtt{a}B \\ A &\to B \mid S \\ B &\to \mathtt{b} \mid \varepsilon \end{aligned}$$

2. Remove ε-rules $B \to \varepsilon$, shown on the left, and $A \to \varepsilon$, shown on the right.

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathbf{a} \\ A &\to B \mid S \mid \boldsymbol{\varepsilon} \\ B &\to \mathtt{b} \mid \varepsilon \end{aligned} \qquad \begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \mid \boldsymbol{S} \\ A &\to B \mid S \mid \varepsilon \\ B &\to \mathtt{b} \end{aligned}$$

3a. Remove unit rules $S \to S$, shown on the left, and $S_0 \to S$, shown on the right.

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \mid S \\ A &\to B \mid S \\ B &\to \mathtt{b} \end{aligned} \qquad \begin{aligned} S_0 &\to S \mid \boldsymbol{ASA} \mid \mathbf{a}\boldsymbol{B} \mid \mathbf{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A &\to B \mid S \\ B &\to \mathtt{b} \end{aligned}$$

3b. Remove unit rules $A \to B$ and $A \to S$.

$$\begin{aligned} S_0 &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A &\to B \mid S \mid \mathbf{b} \\ B &\to \mathtt{b} \end{aligned} \qquad \begin{aligned} S_0 &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A &\to S \mid \mathtt{b} \mid \boldsymbol{ASA} \mid \mathbf{a}\boldsymbol{B} \mid \mathbf{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ B &\to \mathtt{b} \end{aligned}$$

4. Convert the remaining rules into the proper form by adding additional variables and rules. The final grammar in Chomsky normal form is equivalent to G_6. (Actually the procedure given in Theorem 2.9 produces several variables U_i and several rules $U_i \to \mathtt{a}$. We simplified the resulting grammar by using a single variable U and rule $U \to \mathtt{a}$.)

$$\begin{aligned} S_0 &\to AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\ S &\to AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\ A &\to \mathtt{b} \mid AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\ A_1 &\to SA \\ U &\to \mathtt{a} \\ B &\to \mathtt{b} \end{aligned}$$

Problem 5 Convert the following CFG into Chomsky normal form.

$$\begin{aligned} A &\rightarrow BAB|B|\epsilon \\ B &\rightarrow 00|\epsilon \end{aligned}$$

1. First add a new start symbol S_0 and the rule $S_0 \rightarrow A$:

$$\begin{aligned} S_0 &\rightarrow A \\ A &\rightarrow BAB|B|\epsilon \\ B &\rightarrow 00|\epsilon \end{aligned}$$

2. Next eliminate the ϵ-rule $B \rightarrow \epsilon$, resulting in new rules corresponding to $A \rightarrow BAB$:

$$\begin{aligned} S_0 &\rightarrow A \\ A &\rightarrow BAB|BA|AB|A|B|\epsilon \\ B &\rightarrow 00 \end{aligned}$$

3. Now eliminate the redundant rule $A \rightarrow A$ and the ϵ-rule $A \rightarrow \epsilon$:

$$\begin{aligned} S_0 &\rightarrow A|\epsilon \\ A &\rightarrow BAB|BA|AB|BB|B \\ B &\rightarrow 00 \end{aligned}$$

4. Now remove the unit rule $A \rightarrow B$:

$$\begin{aligned} S_0 &\rightarrow A|\epsilon \\ A &\rightarrow BAB|BA|AB|BB|00 \\ B &\rightarrow 00 \end{aligned}$$

5. Then remove the unit rule $S_0 \rightarrow A$:

$$\begin{aligned} S_0 &\rightarrow BAB|BA|AB|BB|00|\epsilon \\ A &\rightarrow BAB|BA|AB|BB|00 \\ B &\rightarrow 00 \end{aligned}$$

6. Finally convert the 00 and BAB rules:

$$\begin{aligned} S_0 &\rightarrow BA_1|BA|AB|BB|N_0N_0|\epsilon \\ A &\rightarrow BA_1|BA|AB|BB|N_0N_0 \\ A_1 &\rightarrow AB \\ B &\rightarrow N_0N_0 \\ N_0 &\rightarrow 0 \end{aligned}$$

This grammar satisfies all the requirements for Chomsky Normal Form.

Rimane possibile anche dimostrare “formalmente” che un linguaggio è context-free.

2. (8 punti) *Per ogni linguaggio L, sia* $\text{prefix}(L) = \{u \mid uv \in L \text{ per qualche stringa } v\}$. *Dimostra che se L è un linguaggio context-free, allora anche* $\text{prefix}(L)$ *è un linguaggio context-free.*

Se L è un linguaggio context-free, allora esiste una grammatica G *in forma normale di Chomski* che lo genera. Possiamo costruire una grammatica G' che genera il linguaggio $\text{prefix}(L)$ in questo modo:

- per ogni variabile V di G, G' contiene sia la variabile V che una nuova variabile V'. La variabile V' viene usata per generare i prefissi delle parole che sono generate da V;
- tutte le regole di G sono anche regole di G';
- per ogni variabile V di G, le regole $V' \to V$ e $V' \to \varepsilon$ appartengono a G;
- per ogni regola $V \to AB$ di G, le regole $V' \to AB'$ e $V' \to A'$ appartengono a G';
- se S è la variabile iniziale di G, allora S' è la variabile iniziale di G'.

Oppure, potremmo aver bisogno di altro…

Accenni generici ad altri concetti:

- Linguaggi non context free → Pumping Lemma per Linguaggi Context Free

Problem 15 Let $L_4 = \{w\#x \mid w \text{ is a substring of } x\}$. Show that L_4 is not a context-free language (CFL).

Solution: In order to show that L_4 is not a CFL, we will proceed by contradiction. Assume L_4 is a CFL. Let p be the pumping length given by the pumping lemma and let $s = a^p b^p \# a^p b^p$. Because s is a member of L_4 and $|s| > p$, the pumping lemma says that s can be split into $uvxyz$ satisfying the following conditions:

1. for each $i \geq 0$, $uv^i xy^i z \in L_4$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

For convenience, we also write s as $L\#R$, where L and R stand for the strings to the left and to the right of $\#$, respectively.

First of all, we can note that vy cannot contain $\#$, since uv^2xy^2z would contain more than one $\#$ and would not be in L_4. Then, we can think of three possible cases for the string vy:

- vy contains more symbols from L than from R.
  In this case, $uv^2xy^2z = L'\#R'$ where $|L'| > |R'|$. Therefore, L' cannot be a substring of R' and the entire string is not in L_4.
- vy contains more symbols from R than from L.
  In this case (pumping down), $uv^0xy^0z = L'\#R'$ where $|L'| > |R'|$. Therefore, L' cannot be a substring of R' and the entire string is not in L_4.
- vy contains an equal number of symbols from both L and R.
  In this case, because of conditions 2 and 3 of the pumping lemma, $v = b^j$ and $w = a^j$ for some j such that $1 \leq j \leq p/2$. Therefore $uv^2xy^2z = a^p b^{p+j} \# a^{p+j} b^p$ is not in the language.

Because every possible way of splitting the input into $uvxyz$ yields a contradiction, the initial assumption that L_4 is a CFL is false and the proof is complete.

- Automi a pila

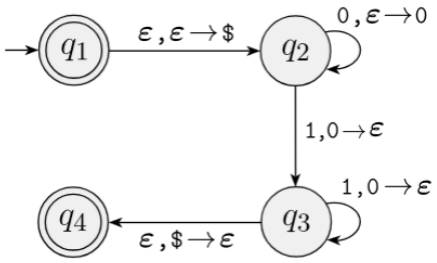
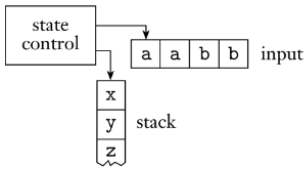


FIGURE 2.15
State diagram for the PDA M_1 that recognizes $\{0^n 1^n | \, n \geq 0\}$

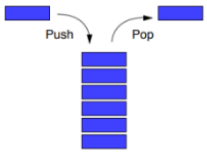
. agli Automi a Pila (PDA)

- Input: stringa di caratteri dell'alfabeto
- Memoria: stati + pila
- Funzione di transizione: dato lo stato corrente, un simbolo di input ed il simbolo in cima alla pila, stabilisce quali possono essere gli stati successivi e i simboli da scrivere sulla pila

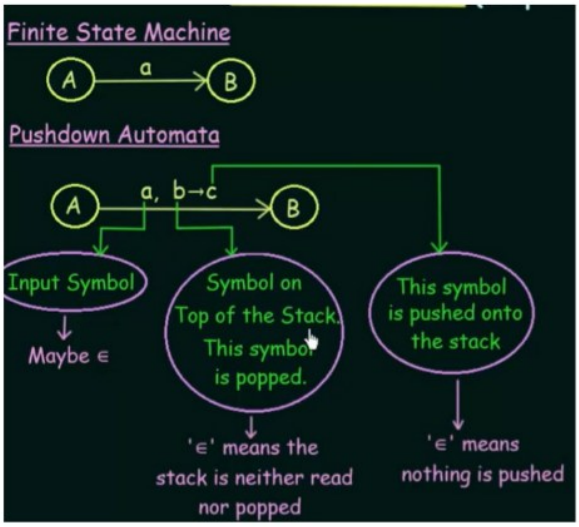


La pila è un dispositivo di memoria last in, first out (LIFO):

- Push: scrivi un nuovo simbolo in cima alla pila e "spingi giù" gli altri
- Pop: leggi e rimuovi il simbolo in cima alla pila (top)



La pila permette di avere memoria infinita (ad accesso limitato)



Il precedente infatti può essere risolto come:

2.25 For any language A, let $\mathbf{SUFFIX}(A) = \{v | uv \in A$ for some string $u\}$. Show that the class of context-free languages is closed under the SUFFIX operation

Let A be a context-free language recognized by the PDA M_1. We are going to construct a PDA M recognizing the language SUFFIX(A) to show that the language SUFFIX(A) is also context-free.

Let M_2 be identical to M_1. We update the transitions of M_2 so that the transitions are performed without consuming any input symbol. That is, any transition of the form $a, b \to c$ is replaced with $\varepsilon, b \to c$. We add the transition $\varepsilon, \varepsilon \to \varepsilon$ from each state in M_2 to the corresponding state in M_1. M_1 and M_2 together form the PDA M. Start state of M is the start state of M_2.

When M starts reading a string v, it will construct the stack as it was reading u without reading u and without consuming any input symbol. This is possible since we updated all transitions in M_2 accordingly. Then at some point, by following the transition $\varepsilon, \varepsilon \to \varepsilon$, the computation continues in M_1 by reading the string v. So, the only accepted strings are those which are the suffixes of the strings in A.

We conclude that the set of context-free languages is closed under the SUFFIX operation.