Lezione 18 – automi a pila e macchine di Turing; alberi sintattici e ambiguità

Lezione del 11/04/2025

Grammatiche di tipo 2: automi a pila

- Le due modalità di accettazione sono, sostanzialmente, equivalenti, come specificato nel seguente teorema

TEOREMA G.10: per ogni linguaggio L, esiste un PDA M che accetta L per pila vuota se e soltanto se esiste un PDA M' che accetta L per stato finale

- Come conseguenza del precedente teorema, possiamo parlare di ***insieme dei linguaggi accettati da automi a pila*** indipendentemente dalla modalità di accettazione

- Infine, il prossimo teorema mostra che tale insieme coincide con l'insieme dei linguaggi context-free

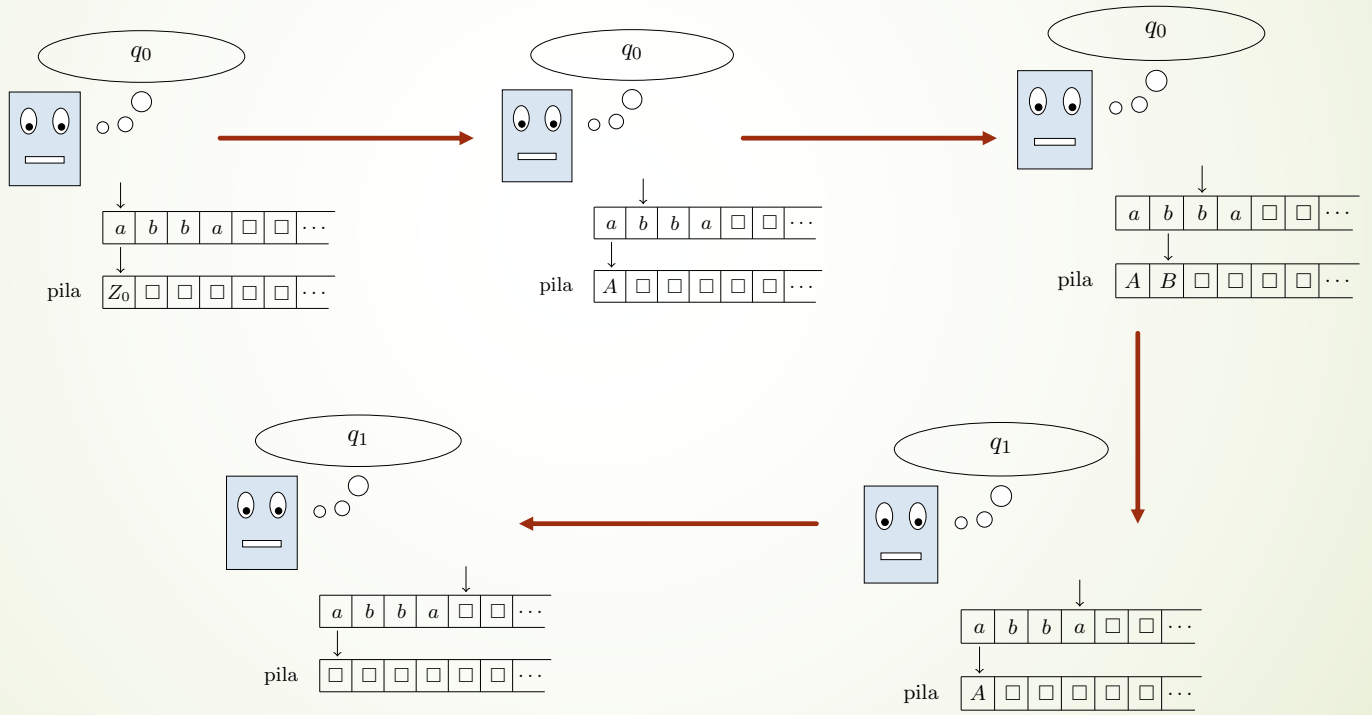
TEOREMA G.11: un linguaggio L è context-free se e soltanto se esiste un PDA M che accetta L

Grammatiche di tipo 2: automi a pila

- **ESEMPIO**: vediamo un PDA $\langle \{a,b\}, \{Z_0, A, B\}, Z_0 , \{q_0 , q_1 \} , \emptyset, q_0 , \delta \rangle$ che riconosce per pila vuota il linguaggio L_{PPAL} delle parole palindrome e di lunghezza pari sull'alfabeto {a,b}
  - definiamo la funzione δ:
    - $\delta(q_0, a, Z_0) = \{(q_0 ,A)\}$, $\delta(q_0, b, Z_0) = \{(q_0 , B)\}$,
    - $\delta(q_0, a, A) = \{(q_0 ,AA), (q_1 ,\varepsilon)\}$, $\delta(q_0, b, A) = \{(q_0 ,AB)\}$
    - $\delta(q_0, a, B) = \{(q_0 ,BA)\}$, $\delta(q_0, b, B) = \{(q_0 ,BB), (q_1 ,\varepsilon)\}$
    - $\delta(q_1, a, A) = \delta(q_1, b, B) = \{(q_1 ,\varepsilon)\}$
    - osserviamo esplicitamente che $\delta(q_1, a, B)$ e $\delta(q_1, b, A)$ non sono definite
  - intuitivamente, mentre scandisce i caratteri contenuti nella prima metà della parola input l'automa ***può*** rimanere nello stato q_0 accumulando nella pila i simboli letti, e mentre scandisce i caratteri contenuti nella seconda metà della parola input l'automa ***può*** entrare e rimanere nello stato q_1 rimuovendo un simbolo dalla pila se esso corrisponde al carattere letto
    - e come fa a capire se si trova nella prima o nella seconda metà della parola? Non lo capisce! **A questo serve il non determinismo**!
  - in questo modo la pila *ha la possibilità di svuotarsi* solo se la parola in input è palindroma

Grammatiche di tipo 2: automi a pila

- **ESEMPIO[continuazione]**: vediamo ***una*** computazione (accettante) del PDA appena descritto sull'input abba

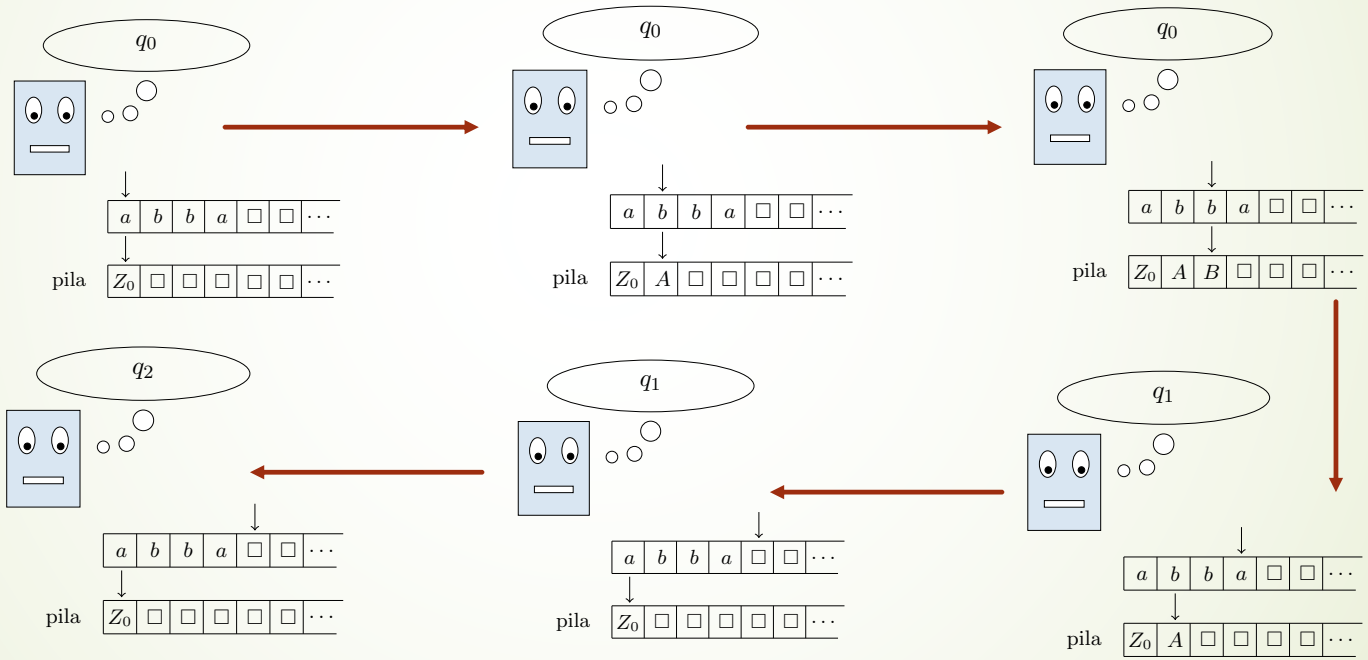


Grammatiche di tipo 2: automi a pila

- **ESEMPIO**: vediamo un PDA $\langle \{a,b\}, \{Z_0, A, B\}, Z_0 , \{q_0 , q_2 , q_1 \} , \{q_2\}, q_0 , \delta \rangle$ che riconosce per stato finale il linguaggio L_{PPAL} delle parole palindrome e di lunghezza pari sull'alfabeto {a,b}
  - definiamo la funzione δ:
    - $\delta(q_0, a, Z_0) = \{(q_0 , Z_0A)\}$, $\delta(q_0, b, Z_0) = \{(q_0 , Z_0B)\}$,
    - $\delta(q_0, a, A) = \{(q_0 , AA), (q_1 , \varepsilon)\}$, $\delta(q_0, b, A) = \{(q_0 , AB)\}$
    - $\delta(q_0, a, B) = \{(q_0 , BA)\}$, $\delta(q_0, b, B) = \{(q_0 , BB), (q_1 , \varepsilon)\}$
    - $\delta(q_1, a, A) = \delta(q_1, b, B) = \{(q_1 , \varepsilon)\}$ $\delta(q_1, \varepsilon, Z_0) = \{(q_2 , Z_0)\}$
  - come nel caso precedente, mentre scandisce i caratteri contenuti nella prima metà della parola input l'automa ***può*** rimanere nello stato q_0 accumulando nella pila i simboli letti e mentre scandisce i caratteri contenuti nella seconda metà della parola input l'automa ***può*** entrare e rimanere nello stato q_1 rimuovendo un simbolo dalla pila se esso corrisponde al carattere letto
  - in questo modo la pila *ha la possibilità di svuotarsi fino a contenere il solo carattere Z_0* solo se la parola in input è palindroma
  - e, a questo punto, entra nello stato finale q_2 accettando l'input

Grammatiche di tipo 2: automi a pila

- **ESEMPIO[continuazione]**: vediamo ***una*** computazione (accettante) del PDA appena descritto sull'input abba



Macchina di Turing vs Automa a pila

- In effetti, un PDA ''assomiglia'' a una MdT che non può scrivere sul primo nastro e ha una gestione a pila del secondo nastro
- Mostriamo ora come simulare un PDA mediante una macchina di Turing
  - a titolo di esempio, mostriamo la simulazione di un PDA che accetta per pila vuota
- Sia A= $\langle \Sigma , \Gamma , Z_0 , Q , \emptyset, q_0 , \delta \rangle$ un PDA che accetta per pila vuota
- Costruiamo $T = \langle \Sigma \cup \Gamma , Q_T, q_0 , \{q_A, q_R\}, P \rangle$ inizializzando $Q_T = Q \cup \{q_A\}$ e P all'insieme vuoto e poi, per ogni $q \in Q$, $a \in \Sigma$, $Z \in \Gamma$:
  - se $(q_2,\varepsilon) \in \delta(q_1,a,Z)$ inseriamo in P la quintupla $\langle q_1, (a,Z), (a,\square), q_2, (D,S) \rangle$
  - se $(q_2,Z_1Z_2...Z_k) \in \delta(q_1,a,Z)$, con k>0 e $Z_1, Z_2, ..., Z_k \in \Gamma$, inseriamo in Q_T gli stati $q(q_2, Z_2Z_3...Z_k)$, $q(q_2, Z_3Z_4...Z_k)$, ..., $q(q_2, Z_k)$ e in P le quintuple

    $\langle q_1, (a,Z), (a,Z_1), q(q_2, Z_2Z_3...Z_k), (F,D) \rangle$, $\langle q(q_2, Z_2...Z_k), (a, \square), (a,Z_2), q(q_2, Z_3...Z_k), (F,D) \rangle$

    , ..., $\langle q(q_2, Z_{k-1}Z_k), (a, \square), (a,Z_{k-1}), q(q_2,Z_k), (F,D) \rangle$, $\langle q(q_2, Z_k), (a, \square), (a,Z_k), q_2, (D,F) \rangle$

  - [... continua ...]

Macchina di Turing vs Automa a pila

- Sia A= $\langle \Sigma , \Gamma , Z_0 , Q , \emptyset, q_0 , \delta \rangle$ un PDA che accetta per pila vuota
- Costruiamo $T = \langle \Sigma \cup \Gamma , Q_T, q_0 , \{q_A, q_R\}, P \rangle$ inizializzando $Q_T = Q \cup \{q_A\}$ e P all'insieme vuoto e poi, per ogni $q \in Q$, $a \in \Sigma$, $Z \in \Gamma$:
  - [... segue...]
  - se $(q_2,\varepsilon) \in \delta(q_1, \varepsilon, Z)$ inseriamo in P le quintuple
    $\langle q_1, (a,Z), (a,\square), q_2, (F,S) \rangle$ **per ogni** $a \in \Sigma$
  - se $(q_2,Z_1Z_2...Z_k) \in \delta(q_1, \varepsilon, Z)$, con k>0 e $Z_1, Z_2, ..., Z_k \in \Gamma$, inseriamo in Q_T gli stati $q(q_2, Z_2Z_3...Z_k)$, $q(q_2, Z_3Z_4...Z_k)$, ..., $q(q_2, Z_1)$ e in P, **per ogni** $a \in \Sigma$, le quintuple

    $\langle q_1, (a,Z), (a,Z_1), q(q_2, Z_2Z_3...Z_k), (F,D) \rangle$, $\langle q(q_2, Z_2...Z_k), (a, \square), (a,Z_2), q(q_2, Z_3...Z_k), (F,D) \rangle$

    , ..., $\langle q(q_2, Z_{k-1}Z_k), (a, \square), (a,Z_{k-1}), q(q_2,Z_k), (F,D) \rangle$, $\langle q(q_2, Z_k), (a, \square), (a,Z_k), q_2, (F,F) \rangle$
  - infine, per ogni q ∈ Q, inseriamo in P la quintupla $\langle q, (\square, \square), (\square, \square), q_A, (F,F) \rangle$

- ESERCIZIO: dimostrare l'equivalenza di A e T

Grammatiche context-free: automi a pila

- Fino ad ora abbiamo descritto automi a pila non deterministici
  - infatti, la funzione di transizione δ associa a uno stato interno, un simbolo sul nastro (o nessun simbolo sul nastro) e un simbolo sulla pila *un insieme di azioni* fra le quali *scegliere* quella da eseguire
  - precisamente, trovandosi nello stato interno q_1, leggendo a sul nastro e Z sulla pila, l'azione da eseguire deve essere scelta nell'insieme $\delta(q_1, a, Z) \cup \delta(q_1, \varepsilon, Z)$
  - che, in particolare, significa che trovandosi nello stato interno q_1, leggendo a sul nastro e Z sulla pila si può scegliere di eseguire un'azione che ''consuma'' a (scegliendo l'azione in $\delta(q_1, a, Z)$) oppure di eseguire un'azione che ''non consuma'' a (scegliendo l'azione in $\delta(q_1, \varepsilon, Z)$)
- Affinché un automa a pila sia deterministico è necessario che per ogni stato interno, per ogni simbolo sul nastro e per ogni simbolo sulla pila l'insieme di azioni fra le quali scegliere si riduca a un'unica azione:

  un automa a pila $\mathcal{M} = \langle \Sigma, \Gamma, Z_0, Q, Q_F, q_0, \delta \rangle$ è **deterministico** se per ogni $q \in Q$, per ogni $a \in \Sigma$, e per ogni $Z \in \Gamma$ accade che

$$|\delta(q_1, a, Z)| + |\delta(q_1, \varepsilon, Z)| = 1$$

Grammatiche context-free: automi a pila

- Come sappiamo, dal punto di vista della calcolabilità la Macchina di Turing non deterministica è equivalente alla Macchina di Turing deterministica
  - ossia, tutto ciò che possiamo decidere con una macchina di Turing non deterministica possiamo deciderlo anche usando una macchina di Turing deterministica
- Di contro, le cose funzionano diversamente nel caso degli automi a pila: esistono linguaggi context-free che non sono accettati da automi a pila deterministici

TEOREMA G.12: l'insieme dei linguaggi accettati da automi a pila deterministici è un sottoinsieme proprio dei linguaggi context-free

- in altri termini, gli automi a pila deterministici sono ''strettamente meno potenti'' di quelli non deterministici

Macchina di Turing vs Automa a pila

- Ricapitoliamo: abbiamo definito PDA non deterministici e PDA deterministici
- Poi, abbiamo visto che i PDA non deterministici sono strettamente più potenti dei PDA deterministici
- e avevamo anche visto che un PDA non deterministico può essere simulato da una macchina di Turing
  - e noi sappiamo bene come simulare una macchina di Turing non deterministica mediante una macchina di Turin deterministica!
- La domanda sorge allora spontanea: ma, allora, perché i PDA non deterministici non sono simulabili da PDA deterministici se, comunque, tutto si riconduce a macchine di Turing deterministiche?!
- ATTENZIONE! La risposta è semplice:
  - prendiamo un PDA non deterministico NA,
  - lo trasformiamo in una macchina di Turing non deterministica NT (come abbiamo visto),
  - trasformiamo NT in una macchina deterministica T e...
  - ... e basta! Non lo sappiamo mica come trasformare T in un PDA deterministico!!!

Macchina di Turing vs Automa a pila

- Perché i PDA non deterministici non sono simulabili da PDA deterministici se, comunque, tutto si riconduce a macchine di Turing deterministiche?!
  - Prendiamo un PDA non deterministico NA,
  - lo trasformiamo in una macchina di Turing non deterministica NT,
  - trasformiamo NT in una macchina deterministica T e...
  - ... e basta! Non lo sappiamo mica come trasformare T in un PDA deterministico!!!
- Perciò: abbiamo un linguaggio L context-free

  che è accettato da un PDA non deterministico

  che simuliamo mediante una macchina non deterministica NT

  che simuliamo mediante una macchina deterministica T
- e quindi concludiamo che L è accettato da una macchina di Turing deterministica T
  - e ci voleva tanto?! Lo sappiamo che i linguaggi context-free sono decidibili!
- ma, poiché non possiamo simulare T mediante un PDA deterministico, nulla possiamo concludere circa l'accettabilità di L da PDA deterministici

Grammatiche context-free: alberi sintattici

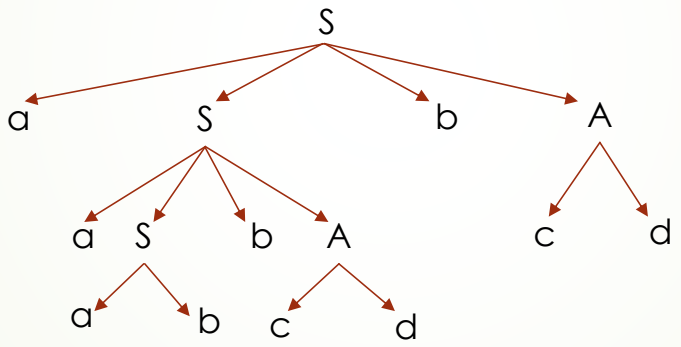
- Caratteristica delle grammatiche di tipo 2 è che la derivazione di una parola generata da una grammatica di tipo 2 può essere rappresentata mediante un ***albero sintattico***
- Un ***albero sintattico*** associato a una grammatica G = $\langle V_T, V_N, P, S \rangle$ è un albero i cui nodi sono etichettati:
  - l'etichetta della radice è S
  - le etichette dei nodi interni sono gli elementi di V_N
  - le etichette delle foglie sono gli elementi di V_T
  - se un nodo interno è etichettato con un terminale A e i suoi figli sono etichettati con i caratteri $x_1, x_2, \ldots, x_h$, dove $x_i \in V_T \cup V_N$ per ogni $i = 1, 2, \ldots, h$, allora $A \longrightarrow x_1 x_2 \ldots x_h$ è una produzione in P
- Gli alberi sintattici forniscono una descrizione sintetica della *struttura sintattica* di una parola
  - ossia, delle produzioni che hanno permesso di generarla

Grammatiche context-free: alberi sintattici

- ESEMPIO: sia $G = \langle \{a,b,c,d\}, \{S,A\}, P, S \rangle$ la grammatica context-free definita dall'insieme di produzioni $P = \{ S \longrightarrow aSbA,\ S \longrightarrow ab,\ A \longrightarrow cAd\ ,\ A \longrightarrow cd \}$
  - la parola aaabbcdbcd può essere generata nel modo seguente:

  $S \Longrightarrow aSbA \Longrightarrow aaSbAbA \Longrightarrow aaabbAbA \Longrightarrow aaabbcdbA \Longrightarrow aaabbcdbcd$

  - l'albero sintattico corrispondente a questa derivazione è



  - osserviamo che questo albero sintattico corrisponde anche alla derivazione

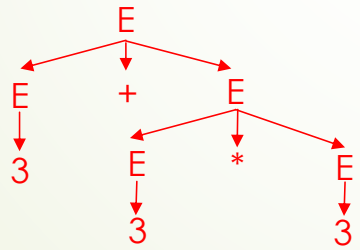
  $S \Longrightarrow aSbA \Longrightarrow aSbcd \Longrightarrow aaSbAbcd \Longrightarrow aaSbcdbcd \Longrightarrow aaabbcdbcd$

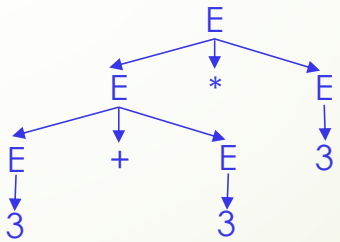
  della parola aaabbcdbcd

Grammatiche context-free: alberi sintattici

- Dunque, uno stesso albero sintattico può corrispondere a più di una derivazione
  - e questo non è un grosso problema
- Ma è vero anche che ad una stessa parola possono corrispondere più alberi sintattici
- **ESEMPIO**: sia G = ⟨ {3,+,*}, {E}, P, E ⟩ la grammatica che permette di derivare le espressioni sul solo numero 3 contenenti solo somme e prodotti, con P = {E ⟶ a, E ⟶ E+E, E ⟶ E*E, E ⟶ (E)}
  - ad esempio 3+(3*3) oppure 3+3+3+3*3*3

  e consideriamo l'espressione 3+3*3: ad essa corrispondono i due alberi





Grammatiche context-free: ambiguità

- Ma è vero anche che ad una stessa parola possono corrispondere più alberi sintattici
- Quando in una grammatica si verifica questo fenomeno la grammatica è detta ***ambigua***
- L'ambiguità di una grammatica può causare problemi nei casi in cui l'albero sintattico è usato per *interpretare semanticamente* la parola
  - ossia, per associare un significato alla parola
- Nell'esempio appena visto l'albero sintattico può indicare l'ordine nel quale sono eseguite le operazioni: ad esempio eseguendole dal basso verso l'alto
  - in tal caso, nell'albero rosso viene eseguita prima l'operazione 3*3 = 9 e poi il risultato è sommato a 3, così che viene calcolato 12
  - in tal caso, nell'albero blu viene eseguita prima l'operazione 3+3 = 6 e poi il risultato è moltiplicato per 3, così che viene calcolato 18
- alla luce di ciò, quale significato dobbiamo associare all'espressione 3+3*3?

Grammatiche context-free: ambiguità

- Dunque, l'ambiguità è una caratteristica poco desiderabile di una grammatica
- e, quindi, ci piacerebbe poter capire se una grammatica è ambigua
  - così da poterla escludere dalla nostra considerazione, ad esempio

- Ma neanche questo funziona... Infatti:
- sia L_A l'insieme delle grammatiche di tipo 2 ambigue

  TEOREMA G.13: il linguaggio L_A è non decidibile

- Ciò significa che non esiste un algoritmo che, data una grammatica G di tipo 2 decide se G è ambigua

Grammatiche e linguaggi di programmazione

- Le grammatiche formali che stiamo considerando furono introdotte intorno agli anni '70 da Noam Chomsky allo scopo di rappresentare i procedimenti sintattici elementari alla base della costruzione delle frasi nei linguaggi naturali (in particolare, nella lingua inglese)
  - ma si rivelarono ben presto uno strumento inadatto allo scopo
- Tuttavia, le grammatiche formali sono uno strumento fondamentale per lo studio delle proprietà sintattiche dei programmi e dei linguaggi di programmazione
- Infatti, la sintassi di un qualunque linguaggio di programmazione $\mathcal{P}$ può essere descritta da una grammatica $G_{\mathcal{P}}$
- così che un programma sintatticamente corretto nel linguaggio di programmazione $\mathcal{P}$ altri non è che una parola appartenente a $L(G_{\mathcal{P}})$ (il linguaggio generato da $G_{\mathcal{P}}$)

Grammatiche e linguaggi di programmazione

- I linguaggi di programmazione di interesse pratico sono ''grosso modo'' linguaggi di tipo 2
  - ''grosso modo'': alcune caratteristiche (ad esempio l'unicità della dichiarazione di una variabile) li rendono, in effetti, di tipo 1
  - ma è possibile isolare e gestire a parte gli aspetti contestuali (cioè, di tipo 1) di un programma
  - e, dunque, l'analisi sintattica di un programma è suddivisa in due parti (non necessariamente indipendenti): quella che si occupa degli aspetti di tipo 1 e quella che si occupa della struttura generale di tipo 2

- Come sappiamo, il processo di verifica della correttezza di un programma è preliminare alla sua traduzione in un programma oggetto – il codice eseguibile
  - per questo, durante la verifica di correttezza (detta *analisi sintattica* o *parsing*) oltre a decidere se il programma è una parola del linguaggio generato dalla grammatica viene derivato anche l'albero sintattico
  - e proprio utilizzando l'albero sintattico viene generato il codice oggetto