

Linguaggi formali e compilazione

Corso di Laurea in Informatica

A.A. 2015/2016

Analisi sintattica (parte II)

Alberi di derivazione (parse tree) e ambiguità

Grammatiche di tipo 1 e di tipo 3

Analisi sintattica (parte II)

Alberi di derivazione (parse tree) e ambiguità

Grammatiche di tipo 1 e di tipo 3

Equivalenza di grammatiche

- ▶ In Informatica (e non solo, naturalmente) esistono sempre molti modi per risolvere un problema, alcuni più efficienti altri meno.
- ▶ Per definire un linguaggio, ad esempio, si possono usare grammatiche equivalenti (cioè che generano lo stesso insieme di stringhe terminali) anche molto diverse.
- ▶ Le seguenti grammatiche definiscono $L_5 = a^*b^*$:

$$\begin{array}{ll}
 S \rightarrow \epsilon \mid A & S \rightarrow AB \\
 A \rightarrow a \mid aA \mid B & A \rightarrow \epsilon \mid aA \\
 B \rightarrow bB \mid b & B \rightarrow \epsilon \mid bB
 \end{array}$$

- ▶ Le seguenti grammatiche definiscono lo stesso semplice linguaggio per le espressioni aritmetiche:

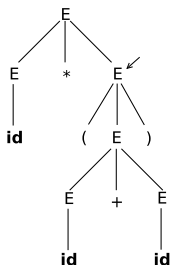
$$\begin{array}{ll}
 E \rightarrow E+E \mid E * E & E \rightarrow E+T \mid T \\
 E \rightarrow (E) \mid \text{number} & T \rightarrow T * F \mid F \\
 & F \rightarrow \text{number} \mid (E)
 \end{array}$$

Quale grammatica usare?

- ▶ Poiché l'obiettivo è di costruire un riconoscitore (parser), il criterio che deve guidare il progettista è proprio quello di rendere possibile, e agevole, il parsing.
- ▶ Fondamentalmente un parser deve “trovare” una derivazione dall'assioma alla stringa in input.
- ▶ In generale, se una stringa appartiene al linguaggio, essa può essere derivata in molti modi possibili.
- ▶ Tale ridondanza non è un fatto positivo e può avere ripercussioni sull'interpretazione da dare alla stringa (si ricordi che le “stringhe” più importanti per noi sono programmi in un qualche linguaggio di programmazione).
- ▶ Cominceremo a lavorare proprio su questo.

Alberi di derivazione (parse tree)

- ▶ Ci sono derivazioni diverse che possono essere descritte mediante uno stesso *parse tree*.
- ▶ Ad esempio, il seguente parse tree descrive “molte” possibili derivazioni per la stringa **number * (number + number)** nella grammatica G_1 :



- ▶ Infatti il parse tree lascia indeterminato l'ordine di applicazione di tutte quelle riscritture che non coinvolgano, nell'albero stesso, nodi interni su uno stesso cammino radice-foglia.

Alberi di derivazione (parse tree)

Analisi sintattica (parte II)

Alberi di derivazione (parse tree) e ambiguità

Grammatiche di tipo 1 e di tipo 3

- ▶ Formalmente, un parse tree per una grammatica libera $G = (\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ è un albero radicato ed etichettato che soddisfa le seguenti proprietà:
 - ▶ i nodi interni sono etichettati con simboli di \mathcal{N} e, in particolare, la radice è etichettata con l'assioma;
 - ▶ le foglie sono etichettate con simboli di \mathcal{T} o con il simbolo ϵ ;
 - ▶ se $A \in \mathcal{N}$ etichetta un nodo interno e X_1, \dots, X_k sono le etichette dei figli di (il nodo con etichetta) A , con $X_i \in \mathcal{V}$, allora deve esistere in \mathcal{P} la produzione $A \rightarrow X_1 X_2 \dots X_k$;
 - ▶ Se $A \in \mathcal{N}$ etichetta un nodo interno il cui unico figlio è un nodo con etichetta ϵ , allora deve esistere in \mathcal{P} la produzione $A \rightarrow \epsilon$.

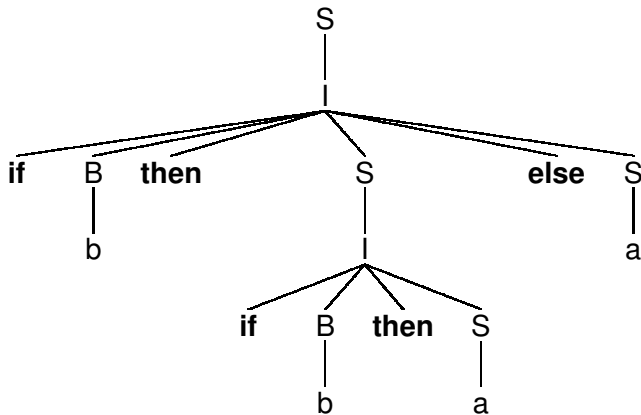
Un secondo esempio

- Un parse tree per “**if b then if b then a else a**” nella grammatica G_I :

$$S \rightarrow I \mid A, \quad A \rightarrow a, \quad B \rightarrow b$$

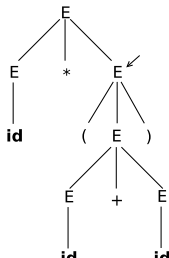
$$I \rightarrow \text{if } B \text{ then } S \mid \text{if } B \text{ then } S \text{ else } S$$

è



Alberi di derivazione (parse tree)

- ▶ I parse tree sono importanti perché impongono una struttura sintattica (gerarchica) alle frasi e ciò costituisce un primo passo per attribuire ad esse un significato.
- ▶ La struttura gerarchica consente di decomporre una frase in sottofrasi, corrispondenti ai sottoalberi e al simbolo non terminale che ne rappresenta la radice.
- ▶ Nel parse tree per “**number** * (**number** + **number**)”, il sottoalbero indicato con la freccia rappresenta la frase, che potremmo definire “di senso compiuto”, (**number** + **number**):



Derivazioni canoniche

- ▶ Le derivazioni distinte alle quali corrisponde uno stesso parse tree sono in qualche modo “la stessa derivazione”, proprio perché impongono alla frase derivata la stessa struttura.
- ▶ Per tutte queste derivazioni possiamo risolvere il problema della “ridondanza” considerando solo derivazioni cosiddette *canoniche*.
- ▶ $S \Rightarrow_G^* \alpha$ è una *derivazione canonica sinistra/destra* della frase α in G , se ad ogni passo viene riscritto il simbolo non terminale più a sinistra/destra.
- ▶ Ad esempio, delle due seguenti derivazioni per **number + (number)** in G_1 :

$$E \Rightarrow_{G_1} E+E \Rightarrow_{G_1} E+(E) \Rightarrow_{G_1} \text{number}+(E) \Rightarrow_{G_1} \text{number}+(\text{number})$$

e

$$E \Rightarrow_{G_1} E+E \Rightarrow_{G_1} E+(E) \Rightarrow_{G_1} E+(\text{number}) \Rightarrow_{G_1} \text{number}+(\text{number})$$

la seconda è una derivazione (canonica) destra,

Derivazioni canoniche

- ▶ È tuttavia possibile che, per una data frase, esistano non solo derivazioni differenti (circostanza molto probabile) bensì parse-tree differenti e quindi derivazioni (canoniche) destre o sinistre differenti.
- ▶ Ad esempio, in G_1 la frase **number + number + number** può essere derivata così

$$\begin{aligned}
 E &\Rightarrow_{G_1} E + \underline{E} \\
 &\Rightarrow_{G_1} E + E + \underline{E} \\
 &\Rightarrow_{G_1} E + \underline{E} + \text{number} \\
 &\Rightarrow_{G_1} \underline{E} + \text{number} + \text{number} \\
 &\Rightarrow_{G_1} \text{number} + \text{number} + \text{number}
 \end{aligned}$$

o così

$$\begin{aligned}
 E &\Rightarrow_{G_1} E + \underline{E} \\
 &\Rightarrow_{G_1} \underline{E} + \text{number} \\
 &\Rightarrow_{G_1} E + \underline{E} + \text{number} \\
 &\Rightarrow_{G_1} \underline{E} + \text{number} + \text{number} \\
 &\Rightarrow_{G_1} \text{number} + \text{number} + \text{number}
 \end{aligned}$$

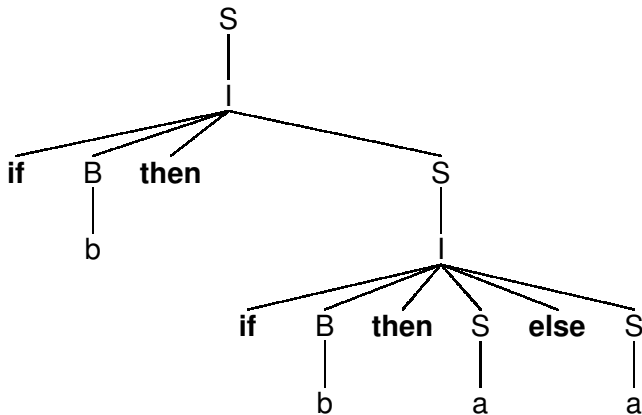
- ▶ Il fatto che esistano più derivazioni canoniche per una frase, e quindi parse tree diversi, è un fenomeno che viene definito con il termine di *ambiguità* della grammatica.
- ▶ Si definisce cioè *ambigua* una grammatica in cui almeno una frase ammetta differenti parse tree.
- ▶ L'ambiguità è un ostacolo alla realizzazione di un parser e, prima ancora, alla attribuzione di un significato alla frase, visto che alberi diversi impongono strutture sintattiche diverse.
- ▶ Vediamo subito due esempi importanti.

Esempio 1

- La grammatica G_1 è ambigua in quanto alla frase

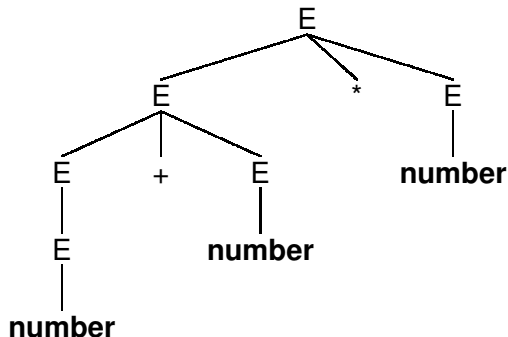
if b then if b then a else a

corrisponde un parse tree diverso da quello già visto,
e precisamente



Esempio 2

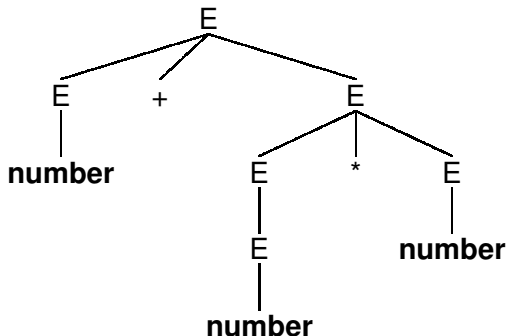
- ▶ Alla stringa **number + number * number** corrisponde in G_1 un primo parse tree illustrato di seguito:



- ▶ Si noti come esso suggerisca un'interpretazione dell'espressione, e precisamente **(number + number) * number**, che non coincide con quella che universalmente si dà alla frase in Matematica.

Esempio 2

- ▶ Il secondo parse tree che corrisponde a **number + number * number** in G_1 è:



- ▶ Questa volta l'albero suggerisce l'interpretazione "corretta" **number + (number * number)**.

- ▶ Come già accennato, se una grammatica è ambigua il parsing (e poi l'interpretazione) diviene problematico.
- ▶ È quindi importante che la grammatica non sia ambigua e, nel caso lo sia, sapere come disambiguare le “frasi ambigue”.
- ▶ Per certe classi di grammatiche l'eventuale ambiguità può essere stabilita (e rimossa) mediante un algoritmo.
- ▶ Si tratta però di un approccio che, quand'anche possibile, risulta macchinoso.
- ▶ In certi casi si preferisce utilizzare una grammatica ambigua e implementare una regola di disambiguazione direttamente nel parser.

Esempio

- ▶ Riconsideriamo il frammento di grammatica:

$$S \rightarrow I \mid A$$

$$I \rightarrow \mathbf{if\ B\ then\ S \mid if\ B\ then\ S\ else\ S}$$

- ▶ Questa grammatica è semplice ma, come sappiamo, ambigua per **if b then if b then a else a**.
- ▶ Una grammatica non ambigua che “forza” l’interpretazione classica è:

$$S \rightarrow O \mid C$$

$$C \rightarrow \mathbf{if\ B\ then\ C\ else\ C \mid A}$$

$$O \rightarrow \mathbf{if\ B\ then\ S \mid if\ B\ then\ C\ else\ O}$$

- ▶ L’unica derivazione possibile è:

$$S \Rightarrow O \Rightarrow \mathbf{if\ B\ then\ S}$$

$$\Rightarrow \mathbf{if\ B\ then\ C} \Rightarrow \mathbf{if\ B\ then\ if\ B\ then\ C\ else\ C}$$

$$\stackrel{*}{\Rightarrow} \mathbf{if\ b\ then\ if\ b\ then\ a\ else\ a}$$

Esempio (continua)

Analisi sintattica (parte II)

Alberi di derivazione (parse tree) e ambiguità

Grammatiche di tipo 1 e di tipo 3

- ▶ La grammatica modificata è più ostica da usare.
- ▶ Si preferisce quindi lasciare la sintassi ambigua e fare in modo che il parser risolva i conflitti (associando ogni `else` all'ultimo `then` incontrato).

Altri aspetti problematici

Analisi sintattica (parte II)

Alberi di derivazione (parse
tree) e ambiguità

Grammatiche di tipo 1 e di
tipo 3

- ▶ Oltre all'ambiguità, esistono altre caratteristiche che, per una data grammatica libera, possono ostacolare il processo di parsing.
- ▶ I più importanti sono:
 - ▶ presenza di cicli (derivazioni del tipo $A \Rightarrow^+ A$) o comunque di *left recursion* (derivazioni del tipo $A \Rightarrow^+ A\alpha$, con $\alpha \neq \epsilon$);
 - ▶ presenza di *prefissi comuni* a due o più produzioni relative allo stesso non terminale, ad esempio $A \rightarrow \alpha\beta_1$ e $A \rightarrow \alpha\beta_2$.

Esempi

- ▶ La grammatica

$$S \rightarrow S_a \mid b$$

presenta una ricorsione immediata a sinistra in quanto $S \rightarrow S_a$.

- ▶ La grammatica

$$\begin{aligned} S &\rightarrow A \mid b \\ A &\rightarrow A_a \mid S_a \end{aligned}$$

presenta una ricorsione a sinistra in quanto $S \Rightarrow^+ S_a$.

- ▶ La grammatica G_l (oltre ad essere ambigua) presenta un prefisso comune alla due produzioni che riscrivono il simbolo l .

- ▶ Nella grammatica

$$S \rightarrow SA$$

$$A \rightarrow a \mid \epsilon$$

c'è una produzione $A \rightarrow \epsilon$ e questa provoca il ciclo $S \Rightarrow^+ S$.

Precedenza degli operatori

- ▶ Come sappiamo, la grammatica G_1 :

$$E \rightarrow E+E \mid E * E \mid \mathbf{number} \mid (E)$$

è ambigua

- ▶ In questo caso per eliminare l'ambiguità è sufficiente introdurre un nuovo simbolo non terminale E'

$$\begin{aligned} E &\rightarrow E+E' \mid E * E' \mid E' \\ E' &\rightarrow \mathbf{number} \mid (E) \end{aligned}$$

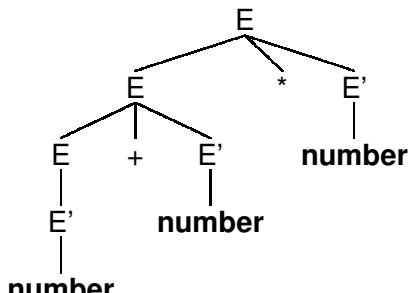
- ▶ La nuova grammatica non è più ambigua ma, come anche la stessa grammatica G_1 di partenza, presenta un problema legato alla precedenza degli operatori.

Precedenza degli operatori

- Ad esempio, per la frase **number + number * number** l'unica derivazione canonica destra possibile è:

$$\begin{aligned}
 E &\Rightarrow E * E' \Rightarrow E * \mathbf{number} \\
 &\Rightarrow E + E' * \mathbf{number} \Rightarrow E + \mathbf{number} * \mathbf{number} \\
 &\Rightarrow \mathbf{number} + \mathbf{number} * \mathbf{number}
 \end{aligned}$$

alla quale corrisponde l'albero di derivazione "erroneo":



Precedenza degli operatori (continua)

- ▶ Le “usuali” precedenze di operatore possono essere forzate introducendo differenti simboli non terminali per i diversi livelli di precedenza.
- ▶ Ad esempio, nel caso di somma e prodotto possiamo introdurre due livelli di precedenza, rappresentati dai non terminali E e T , oltre ad un simbolo (useremo F) per la categoria delle espressioni di base (nel nostro caso identificatori ed espressioni tra parentesi):

$$E \rightarrow E+T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow \mathbf{number} \mid (E)$$

Precedenza degli operatori (continua)

- ▶ La grammatica precedente (con le ovvie estensioni) “gestisce” senza ambiguità anche il caso degli operatori di divisione e sottrazione, inseriti nelle giuste categorie sintattiche:

$$E \rightarrow E+T \mid E-T \mid T$$

$$T \rightarrow T*F \mid T/F \mid F$$

$$F \rightarrow \mathbf{number} \mid (E)$$

- ▶ Domanda: perché scriviamo (ad esempio)

$$T \rightarrow T*F$$

e non, invece,

$$T \rightarrow F*T \quad ?$$

Precedenza degli operatori (continua)

- ▶ Volendo inserire anche l'operatore di esponenziazione (che ha precedenza maggiore), è necessario prevedere un'ulteriore variabile sintattica e ricordarsi che l'operatore di esponenziazione (qui useremo il simbolo \wedge) è associativo a destra:

$$E \rightarrow E+T \mid E-T \mid T$$

$$T \rightarrow T*P \mid T/P \mid P$$

$$P \rightarrow F^P \mid F$$

$$F \rightarrow \mathbf{number} \mid (E)$$

Esercizi

- ▶ Si consideri il linguaggio $L_{a>b}$ delle stringhe su $\{a, b\}$ che contengono più a che b . Si dica, giustificando la risposta, se la seguente grammatica libera genera $L_{a>b}$

$$S \rightarrow a \mid aS \mid Sa \mid abS \mid aSb \mid Sab \mid baS \mid bSa \mid Sba$$

- ▶ Si fornisca una grammatica libera per il linguaggio su \mathcal{B} contenente tutte e sole le stringhe con un diverso numero di 0 e 1.
- ▶ Si dica qual è il linguaggio generato dalla grammatica

$$\begin{aligned} S &\rightarrow aA \\ A &\rightarrow AB \mid B \\ B &\rightarrow b \end{aligned}$$

Automati finiti e grammatiche lineari destre

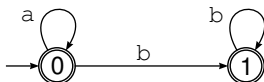
Analisi sintattica (parte II)

Alberi di derivazione (parse tree) e ambiguità

Grammatiche di tipo 1 e di tipo 3

- ▶ Dato un automa a stati finiti M che riconosce il linguaggio L è immediato definire una grammatica lineare destra G_M che genera lo stesso linguaggio, cioè tale che $L(G_M) = L$.
- ▶ La costruzione è molto semplice:
 - ▶ per ogni stato q dell'automata la grammatica conterrà un simbolo non terminale A_q ;
 - ▶ l'assioma iniziale è A_{q_0} ;
 - ▶ per ogni transizione $\delta(q, a) = q'$, la grammatica conterrà la produzione $A_q \rightarrow aA_{q'}$;
 - ▶ se q' è uno stato finale, la grammatica conterrà la produzione $A_{q'} \rightarrow \epsilon$.

- ▶ La grammatica corrispondente all'automa M_5 :



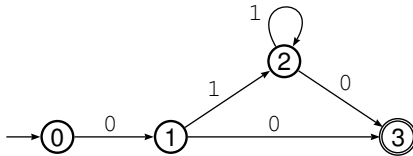
è

$$A_{q_0} \rightarrow aA_{q_0} \mid bA_{q_1} \mid \epsilon$$

$$A_{q_1} \rightarrow bA_{q_1} \mid \epsilon$$

Esempio

- La grammatica corrispondente all'automa M_3 :



è

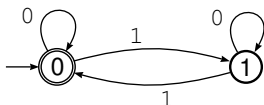
$$A_{q_0} \rightarrow 0A_{q_1}$$

$$A_{q_1} \rightarrow 1A_{q_2} \mid 0A_{q_3}$$

$$A_{q_2} \rightarrow 1A_{q_2} \mid 0A_{q_3}$$

$$A_{q_3} \rightarrow \epsilon$$

- ▶ La grammatica corrispondente all'automa M_{parity} :



è

$$A_{q_0} \rightarrow 0A_{q_0} \mid 1A_{q_1} \mid \epsilon$$

$$A_{q_1} \rightarrow 1A_{q_0} \mid 0A_{q_1}$$

Automati finiti e grammatiche lineari destre (continua)

- ▶ La costruzione appena vista è completamente reversibile.
- ▶ Data una grammatica lineare destra G , possiamo facilmente costruire un automa non deterministico M che riconosce il linguaggio generato da G .
 - ▶ per ogni non terminale A della grammatica definiamo uno stato q_A dell'automa;
 - ▶ per ogni produzione $A \rightarrow aB$ inseriamo la transizione $\delta(q_A, a) = q_B$ e per ogni produzione $A \rightarrow B$ inseriamo la transizione $\delta(q_A, \epsilon) = q_B$
 - ▶ lo stato iniziale è q_S ;
 - ▶ si definisce uno stato finale q_f , non corrispondente ad alcun simbolo non terminale;
 - ▶ per ogni produzione $A \rightarrow a$ si pone $\delta(q_A, a) = q_f$ e, analogamente, per ogni produzione $A \rightarrow \epsilon$ si pone $\delta(q_A, \epsilon) = q_f$.

- L'automa corrispondente alla grammatica:

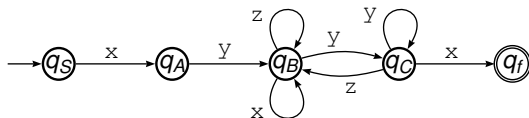
$$S \rightarrow xA$$

$$A \rightarrow yB$$

$$B \rightarrow xB \mid zB \mid yC$$

$$C \rightarrow x \mid zB \mid yC$$

è

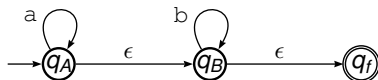


- L'automa corrispondente alla grammatica:

$$A \rightarrow aA \mid B$$

$$B \rightarrow bB \mid \epsilon$$

è



Grammatiche dipendenti dal contesto

Analisi sintattica (parte II)

Alberi di derivazione (parse tree) e ambiguità

Grammatiche di tipo 1 e di tipo 3

- ▶ La forma generale delle produzioni di una grammatica di tipo 1 (dipendente dal contesto) è:

$$\alpha A \gamma \rightarrow \alpha \beta \gamma$$

dove $\alpha, \gamma \in \mathcal{V}^*$, $\beta \in \mathcal{V}^+$ e $A \in \mathcal{N}$.

- ▶ La forma delle produzioni “spiega” il nome di queste grammatiche.
- ▶ Infatti, il simbolo A può essere riscritto come β solo se appare nel *contesto* rappresentato dalle stringhe α e γ .

Linguaggi dipendenti dal contesto

- ▶ Sono così detti i linguaggi generabili da grammatiche dipendenti dal contesto.
- ▶ Poiché la forma generale delle produzioni di una grammatica context-dependent sono più generali di quelle di una grammatica context-free, l'insieme dei linguaggi dipendenti dal contesto “contiene” l'insieme dei linguaggi liberi.
- ▶ Viceversa, ci sono linguaggi dipendenti dal contesto che non sono liberi.
- ▶ Un esempio è il linguaggio

$$L_{13} = \{a^n b^n c^n \mid n \geq 0\}$$

Costrutti non liberi nei linguaggi di programmazione

Analisi sintattica (parte II)

Alberi di derivazione (parse tree) e ambiguità

Grammatiche di tipo 1 e di tipo 3

- ▶ Alcuni aspetti della sintassi dei linguaggi di programmazione non sono catturabili da produzioni di grammatiche libere.
- ▶ Ad esempio, il fatto che una variabile debba essere dichiarata prima dell'uso non è esprimibile mediante una grammatica libera.
- ▶ Tale caratteristica è catturata dal cosiddetto *linguaggio delle repliche* (che non è libero):

$$L_R = \{\alpha\alpha \mid \alpha \in \mathcal{T}^*\}.$$

Costrutti non liberi nei linguaggi di programmazione

Analisi sintattica (parte II)

Alberi di derivazione (parse tree) e ambiguità

Grammatiche di tipo 1 e di tipo 3

- ▶ Un altro aspetto non esprimibile con grammatiche libere è che il numero e il tipo degli argomenti di una funzione coincida ordinatamente con il numero e il tipo dei parametri formali.
- ▶ Anziché utilizzare grammatiche più potenti (che renderebbero difficoltoso, quando non impossibile, il parsing), l'approccio consiste nell'ignorare il problema a livello sintattico.
- ▶ La verifica di correttezza viene completata invece durante l'*analisi semantica*