Lezione 1 – Sintassi e Semantica

Programmazione

Modulo 1 - Fondamenti di Programmazione

Unità didattica 4 – Linguaggi e Grammatica

Marco Anisetti

Università degli Studi di Milano - SSRI - CdL Online

Sintassi e Semantica(1)

Sintassi: descrive come viene scritta una frase

Semantica: descrive il significato dello scritto

Un linguaggio di programmazione è quasi sempre descritto da una sintassi formale e da una semantica informale

Esempio:

- **Sintassi:** la data è un insieme di caratteri *D* e / : 01/03/2013 è una data
- **Semantica:** Il giorno a cui si riferisce non è identificabile considerando solo la sintassi

Introduzione alla Grammatica

Tecnica generativa per definire un linguaggio

Include tutte le regole necessarie per **generare** parole di un linguaggio

- Non i passaggi per **riconoscere** una parole come nel caso degli automi a stati (sistema riconoscitivo)

La sintassi può essere formalmente definita da una grammatica

- Esempio le regole per scrivere una data
  - gg/mm/aaaa

Notazione delle espressioni matematiche (1)

Espressioni tipo $a - b + c$ hanno un formato molto familiare

- Usano una notazione a cui siamo abituati

Linguaggi di programmazione usano dei mix di notazioni differenti

- **Prefissa**: l'operatore è indicato prima degli operandi $+a\ b$
- **Postfissa**: l'operatore è indicato dopo gli operandi $a\ b+$
- **Infissa**: l'operatore è indicato tra gli operandi $a + b$

Le notazioni prefissa e post fissa non richiedono parentesi per essere eseguite correttamente (nell'ordine corretto)

Notazione delle espressioni matematiche (2)

Esempi di notazione:

- **Prefissa**: $*+10\,20\,30 = *30\,30 = 900$ similmente $*10+20\,30 = *10\,50$
- **Postfissa**: $10\,20+30* = 30\,30* = 900$ similmente $10\,20\,30+* = 10\,50*$

La notazione **infissa** richiede delle regole di precedenza o l'uso di parentesi

- Associazione sinistra in caso di pari precedenza

Il numero di operandi di un operatore è detto arità

- Funzioni tipo *max(x,y)* sono prefisse e hanno l'arità che dipendente da quello che c'è in parentesi

Abstract Syntax Tree (AST)

Una modalità grafica ad albero per evidenziare i componenti più significativi di un linguaggio

- Mostra la struttura sintattica non la notazione

Abstract Syntax Tree (AST)

Una modalità grafica ad albero per evidenziare i componenti più significativi di un linguaggio

- Mostra la struttura sintattica non la notazione

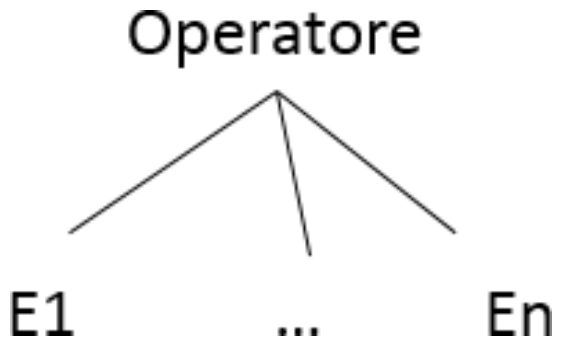
Esempio: Espressioni matematiche (E) semplici costituite da **operatore** e operandi

Abstract Syntax Tree (AST)

Una modalità grafica ad albero per evidenziare i componenti più significativi di un linguaggio

- Mostra la struttura sintattica non la notazione

Esempio: Espressioni matematiche (E) semplici costituite da **operatore** e operandi



Abstract Syntax Tree (AST)

Una modalità grafica ad albero per evidenziare i componenti più significativi di un linguaggio

- Mostra la struttura sintattica non la notazione

Esempio: Espressioni matematiche (E) semplici costituite da **operatore** e operandi

Ci dice che l'operatore riferisce operandi che possono essere anch'essi espressioni E

AST: esercizi

Esercizio: Disegnare l'AST delle espressioni nelle forme prefissa $+a\,b$ infissa $a+b$ e postfissa $a\,b+$

AST: esercizi

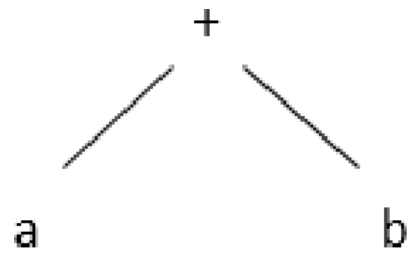
Esercizio: Disegnare l'AST delle espressioni nelle forme prefissa $+a\,b$ infissa $a+b$ e postfissa $a\,b+$

L'AST è indipendente dalla notazione e quindi dalla **grammatica**

AST: esercizi

Esercizio: Disegnare l'AST delle espressioni nelle forme prefissa $+a\ b$ infissa $a + b$ e postfissa $a\ b+$

L'AST è indipendente dalla notazione e quindi dalla **grammatica**



AST: esercizi

Esercizio: Disegnare l'AST delle espressioni nelle forme prefissa $+a\ b$ infissa $a + b$ e postfissa $a\ b+$

L'AST è indipendente dalla notazione e quindi dalla **grammatica**

***Concrete syntax tree* (parse tree)** è la versione specifica dell'AST per una data grammatica

AST: esercizi

Esercizio: Disegnare l'AST delle espressioni nelle forme prefissa $+a\ b$ infissa $a + b$ e postfissa $a\ b+$

L'AST è indipendente dalla notazione e quindi dalla **grammatica**

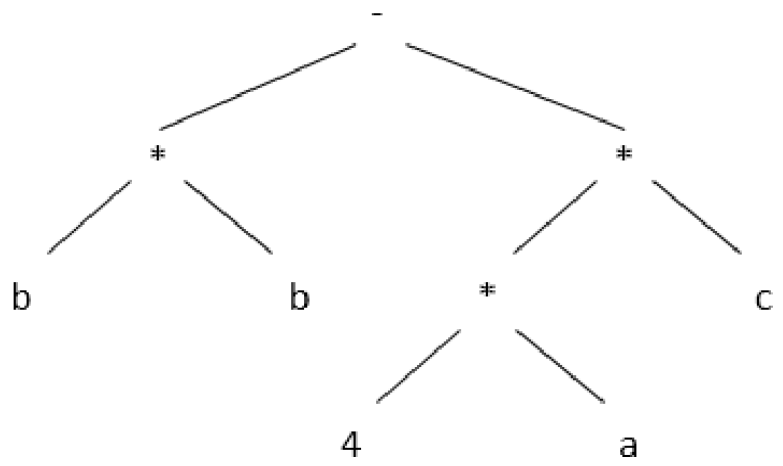
Esercizio: Disegnare l'AST dell'espressione $-*b\ b\ **4\ a\ c$

AST: esercizi

Esercizio: Disegnare l'AST delle espressioni nelle forme prefissa $+a\ b$ infissa $a + b$ e postfissa $a\ b+$

L'AST è indipendente dalla notazione e quindi dalla **grammatica**

Esercizio: Disegnare l'AST dell'espressione $-*b\ b\ **4\ a\ c$



FINE

Lezione 2 – Grammatica

Programmazione

Modulo 1 - Fondamenti di Programmazione

Unità didattica 4 – Linguaggi e Grammatica

Marco Anisetti

Università degli Studi di Milano - SSRI - CdL Online

Linguaggio

Alfabeto Σ

Σ^* denota l'insieme di tutte le parole *w* composte da elementi di Σ, compresa ϵ.

Un linguaggio *L* è un sottoinsieme delle parole costruibili su un alfabeto Σ, $L \subseteq \Sigma^*$

token o terminale = parola che appartiene al linguaggio

Grammatica(1)

Una grammatica è una quartupla $G = (N, \Sigma, R, S)$

N insieme dei simboli non terminali, o *metasimboli*, cioè che non possono comparire negli enunciati del linguaggio ma che ci servono per denotare elementi di un enunciato $N \cap \Sigma = \emptyset$

Σ alfabeto del linguaggio costituito da simboli terminali (token)

R insieme finito delle regole di produzione nella forma $\alpha \rightarrow \beta$ con $\alpha \in N^* \setminus \{\epsilon\}$ e $\beta \in (N \cup \Sigma)^*$. Quando la regola viene applicata, un'istanza di una stringa α può essere riscritta in una istanza della stringa β

S simbolo non terminale speciale, $S \in N$ ed è il punto di partenza che denota un enunciato valido.

Grammatica(2)

[Relazione di produzione e derivazione]

Produzione: $\Rightarrow_G \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^* : \gamma \Rightarrow_G \delta$ sse δ si ottiene da γ mediante l'applicazione di una singola regola di produzione di R nella grammatica G.

- **Regole di riscrittura dei non terminali**

Derivazione: $\Rightarrow^*_G : \gamma \Rightarrow^*_G \delta$ sse δ si ottiene da γ mediante l'applicazione di zero o più regole di produzione di R nella grammatica G

- **Regole da applicare per verificare la derivabilità** rispetto a delle produzioni partendo dal simbolo iniziale

Grammatica(3)

Linguaggio generato da *G* denominato *L*(*G*)

L'insieme di tutte le sequenze di simboli terminali ottenibili applicando le regole di produzione dell'insieme *R*, a partire dal simbolo iniziale *S*.

$L(G) = \{w : w \in \Sigma^* \wedge S \Rightarrow^*_G w\}$

Grammatica(4)

Si può dire che la grammatica di un linguaggio **impone** una struttura gerarchica (**parse tree**) agli enunciati definiti con quel linguaggio

La struttura gerarchica è data dalle produzioni

Grammatica(5)

Le foglie della gerarchia sono i token o terminali

I nodi contengono i simboli non terminali

Ogni nodo è generato da una produzione

- *regola che definisce un simbolo non terminale in termini di una sequenza di altri non terminali o terminali*

Parse Tree

E’ un albero ordinato con radice, che rappresenta la struttura sintattica di una stringa relativamente ad una grammatica formale

Si differenzia dall’ AST perchè i suoi elementi riflettono più concretamente la sintassi di un linguaggio in input

Una grammatica per un linguaggio **impone** un *parse tree* su quanto scritto in quel linguaggio

AST e Parse Tree

La grammatica viene scritta per

- **riflettere la sintassi astratta**

Significa che le regole di produzione sono fatte per far in modo che il parse tree sia il più possibile simile all' AST

Il parse tree rappresenta graficamente una **derivazione**

- Ogni nodo interno è una regola, ogni foglia è un terminale
- Descrive in dettaglio la sintassi

L'AST

- Non descrive in dettaglio la sintassi

Grammatica: Esempio

Consideriamo una espressione *8*(3-2)*

Grammatica: Esempio

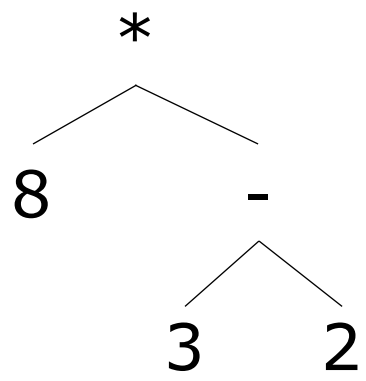
Consideriamo una espressione *8*(3-2)*

Come sarebbe il suo AST

Grammatica: Esempio

Consideriamo una espressione *8*(3-2)*

AST

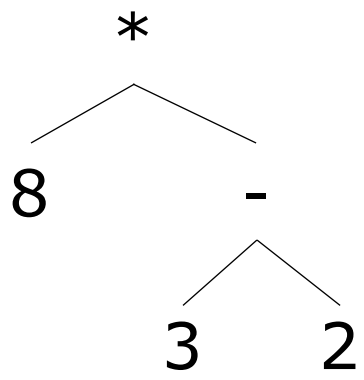


Grammatica: Esempio

Consideriamo una espressione *8*(3-2)*

Proviamo per intuito a pensare alla grammatica e al parse tree

AST

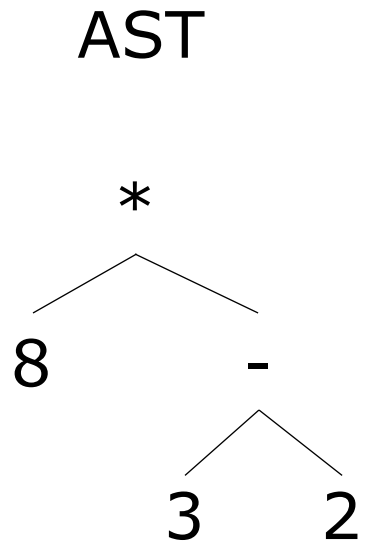


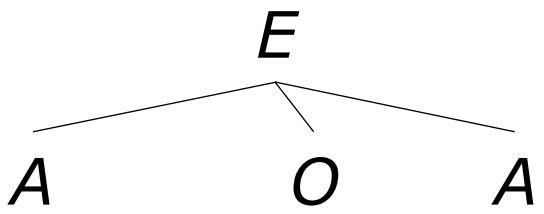
E

Grammatica: Esempio

Consideriamo una espressione *8*(3-2)*

Proviamo per intuito a pensare alla grammatica e al parse tree

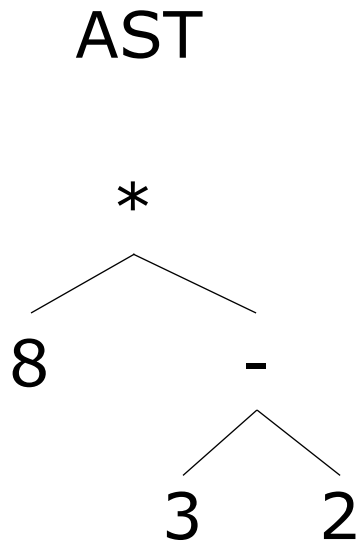


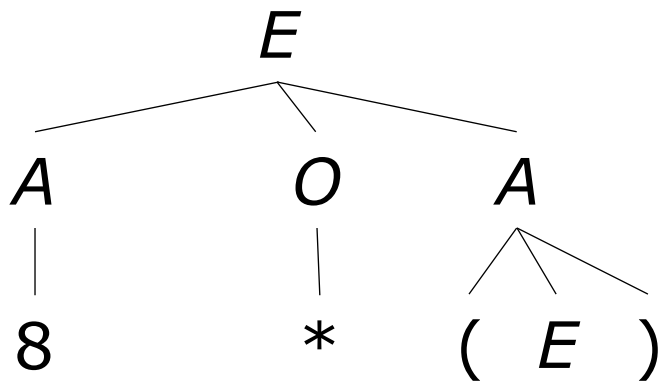


Grammatica: Esempio

Consideriamo una espressione *8*(3-2)*

Proviamo per intuito a pensare alla grammatica e al parse tree

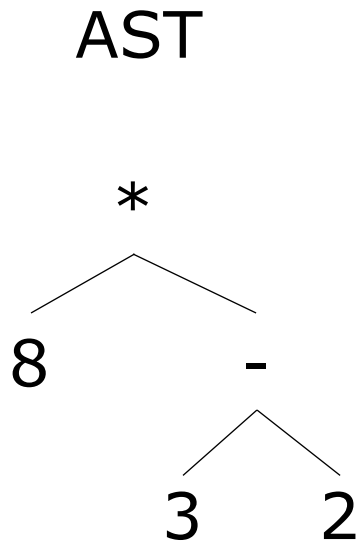


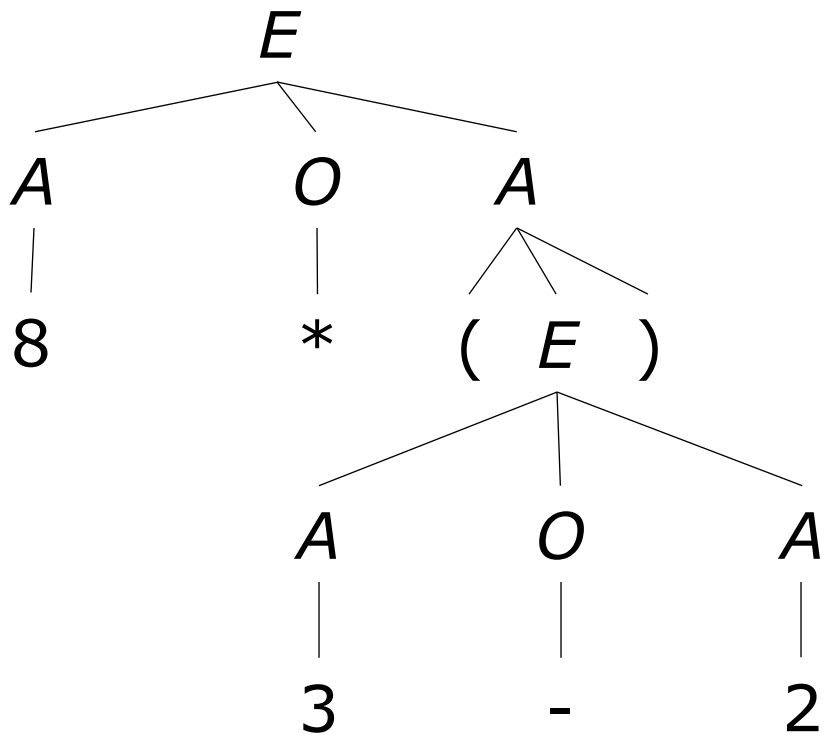


Grammatica: Esempio

Consideriamo una espressione *8*(3-2)*

Proviamo per intuito a pensare alla grammatica e al parse tree



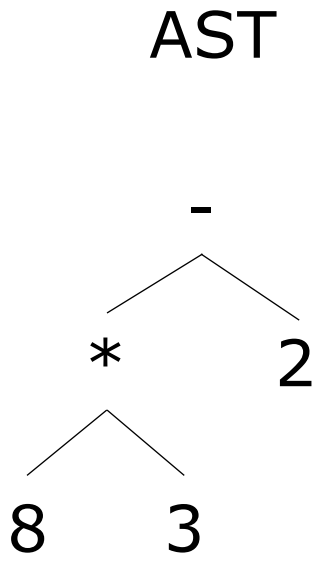


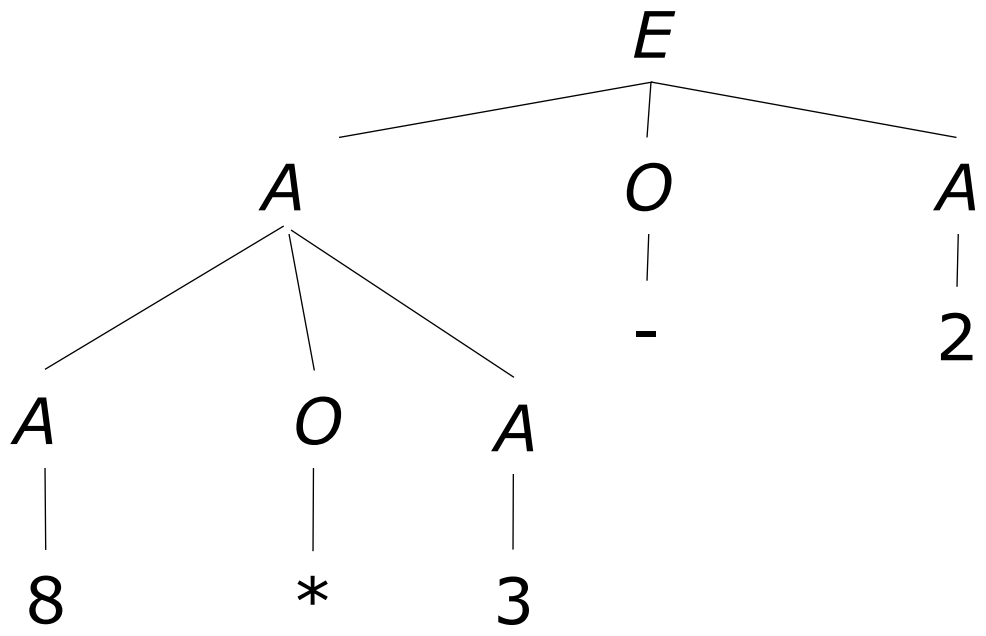
Grammatica: Esempio

Se fosse stato 8*3-2 come sarebbero stati AST e Parse Tree?

Grammatica: Esempio

Se fosse stato 8*3-2 come sarebbero stati AST e Parse Tree?





Formalizziamo l'esempio

Linguaggio delle espressioni aritmetiche

$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, \times, /, (,)\}$

Non Terminali (E,A,O,N,I,M,C):

- *E* simbolo di partenza
- *A* per ″argomento″
- *O* per ″operazione″
- *N* per ″numero naturale″
- *I* per ″cifra iniziale di un numero naturale″
- *M* per ″sequenza delle eventuali cifre successive di numero naturale″
- *C* denota una qualsiasi cifra decimale.

Formalizziamo l'esempio

Linguaggio delle espressioni aritmetiche

$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, \times, /, (,)\}$

Non Terminali (E,A,O,N,I,M,C):

- *E* simbolo di partenza
- *A* per "argomento"
- *O* per "operazione"
- *N* per "numero naturale"
- *I* per "cifra iniziale di un numero naturale"
- *M* per "sequenza delle eventuali cifre successive di numero naturale"
- *C* denota una qualsiasi cifra decimale.

Produzioni

$$E \to AOA \qquad O \to + \qquad I \to 1$$
$$A \to N \qquad O \to - \qquad I \to 2$$
$$A \to (E) \qquad O \to \times \qquad I \to 3$$
$$N \to C \qquad O \to / \qquad I \to 4$$
$$N \to IM \qquad C \to 0 \qquad I \to 5$$
$$M \to C \qquad C \to I \qquad I \to 6$$
$$M \to CM \qquad I \to 7$$
$$I \to 8$$
$$I \to 9$$

Formalizziamo l’ esempio

Esempio: derivazione per la stringa $2\times(3+4)$

$$
\begin{aligned}
E \;\Rightarrow\; & AOA \Rightarrow NOA \Rightarrow NO(E) \Rightarrow NO(AOA) \Rightarrow IO(AOA) \Rightarrow 2O(AOA) \Rightarrow \\
& 2 \times (AOA) \Rightarrow 2 \times (A + A) \Rightarrow 2 \times (N + A) \Rightarrow 2 \times (N + N) \Rightarrow \\
& 2 \times (I + N) \Rightarrow 2 \times (I + I) \Rightarrow 2 \times (3 + I) \Rightarrow 2 \times (3 + 4),
\end{aligned}
$$

Quindi si ricava che $E \Rightarrow^* 2\times(3+4)$ quindi $2\times(3+4)$ appartiene al linguaggio.

Esercizio

Data la seguente grammatica (minuscoli i terminali):

$S \Rightarrow CVRT$

$C \Rightarrow T,\ C \Rightarrow R,\ C \Rightarrow h$

$V \Rightarrow a,\ V \Rightarrow i,\ V \Rightarrow u$

$T \Rightarrow p,\ T \Rightarrow t,\ T \Rightarrow k$

$R \Rightarrow n,\ R \Rightarrow l,\ R \Rightarrow r$

Esercizio

Data la seguente grammatica:

$S \Rightarrow CVRT$

$C \Rightarrow T,\ C \Rightarrow R,\ C \Rightarrow h$

$V \Rightarrow a,\ V \Rightarrow i,\ V \Rightarrow u$

$T \Rightarrow p,\ T \Rightarrow t,\ T \Rightarrow k$

$R \Rightarrow n,\ R \Rightarrow l,\ R \Rightarrow r$

Quali delle seguenti espressioni fa parte del linguaggio?

- tank
- tar
- bin
- leak

Lezione 3 – BNF e Carte sintattiche

Programmazione

Modulo 1 - Fondamenti di Programmazione

Unità didattica 4 – Linguaggi e Grammatica

Marco Anisetti

Università degli Studi di Milano - SSRI - CdL Online

Grammatiche non contestuali

La grammatica fin ora vista è detta non contestuale

- Presenta produzioni destre e sinistre
  - S ⇒ aB, B ⇒Sb (*minuscolo è un terminale*)
  - Se solo destre o sinistre si chiama regolare
    - **Espressioni regolari**

Nella grammatica contestuale le sostituzioni di un non terminale dipendono dal contesto (sinistro e destro)

- Utili per linguaggi naturali in cui una parola è appropriata in un certo contesto

Gerarchia di Chomsky

Studio della proprietà sintattiche dei programmi e dei linguaggi di programmazione

Tipo 0: senza restrizioni: include tutte le grammatiche

- Macchine di Turing

Tipo 1: contestuali: Grammatiche contestuali

- Macchine di Turing non deterministiche.

Tipo 2: non-contestuali: Grammatica non contestuale, descrizione formale dei linguaggi di programmazione

- Automi a pila non deterministici

Tipo 3: regolari: Grammatica regolare, componenti sintattici più elementari

- Automi a Stati Finiti

Formato di Backus e Naur BNF(1)

Ci concentriamo su grammatiche non contestuali

BNF: Formalismo utilizzato nell'informatica

- → delle regole viene sostituita da ::=
- **Simboli non terminali** racchiusi tra parentesi angolari
  - <espressione>
- **Simboli terminali (token)** racchiusi tra virgolette (' ' o " ")

Notazione compatta per più regole con lo stesso membro sinistro

<cifra iniziale> ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

Esempio: numero reale

Il simbolo ::= può essere letto come può essere

Il simbolo | come oppure

< real –number >::=*< integer –part >*”.”*< fraction >*

< integer –part >::=*< digit >* | *< integer –part >< digit >*

< fraction >::=*< digit >* | *< digit >< fraction >*

< digit >::=”0”|”1”|”2”|”3”|”4”|”5”|”6”|”7”|”8”|”9”

Formato di Backus e Naur BNF(1)

Esistono alcune varianti a seconda dell'autore

- Esempio: Extended BNF

Elementi opzionali racchiusi tra parentesi tonde o quadre

Elementi che possono essere ripetuti zero o più volte racchiusi tra parentesi graffe

BNF: Esercizio

Ridefinire in BNF la grammatica delle espressioni matematiche

BNF: Esercizio

Ridefinire in BNF la grammatica delle espressioni matematiche

<espr>::=*<num>* | ′(′*<espr>*′)′ | *<espr><op><espr>* |
<espr>′=′*<espr>*
<num>::=*<cifra>* | *<num><cifra>*
<cifra>::=′0′|′1′|…|′9′
<op>::=′+′|′−′|′·′|′/′

BNF: Esercizio

Ridefinire in BNF la grammatica delle espressioni matematiche

Ecco una versione più precisa con la stessa nomenclatura dell'esempio precedente

<E>::=<N> |'('<E>')'| <A><O><A> |
<E>'='<E>
<N>::=<C> |<I><M>
<C>::='0'|<I>
<I>::='1'| . . . |'9'
<M>::= <C>|<C><M>
<O>::='+'|'−'|'·'|'/'

BNF: Esercizio

Si consideri la seguente definizione in BNF di un linguaggio:

<expr>::=*<const>* | *<fn>* "(" *<args>* ")"
<args>::=*<expr>* | *<expr>* "," *<args>*
<const>::="a"|"b"|"c"|"d"|"e"
<fn>::="f"|"g"|"h"

Quale delle seguenti espressioni fa parte del linguaggio?

a|*e*(*f*,*h*)
g(*a*)
g()
g

Esempio: BNF linguaggio naturale

Σ = {il, lo, la, cane, mela, gatto, mangia, graffia, ,}

N = {frase, soggetto, verbo, complemento, articolo, nome}

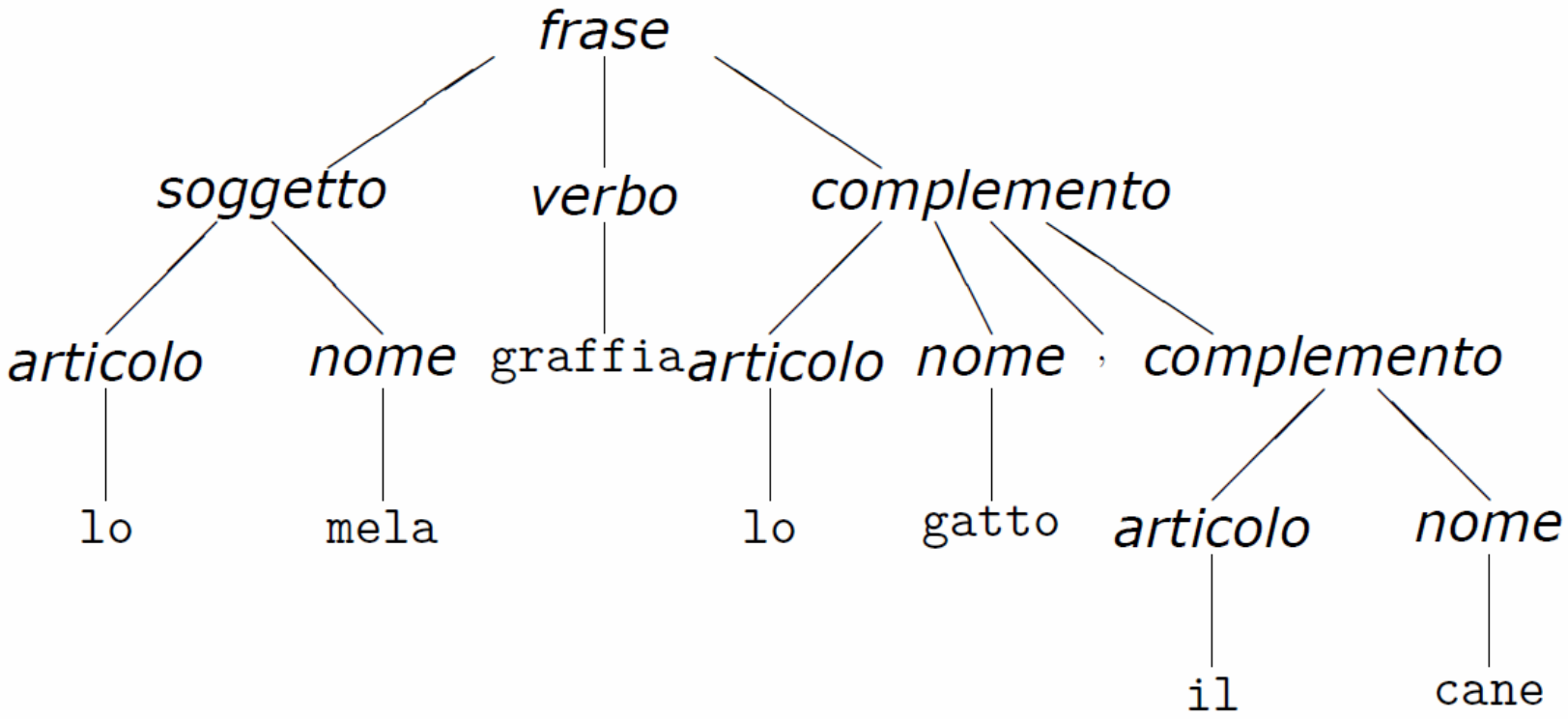
P, regole espresse in *BNF* (forma di *Backus-Naur*):

<frase> ::= <soggetto>" "<verbo>" "<complemento>
<soggetto> ::= <articolo>" "<nome>
<articolo> ::= "il" | "la" | "lo"
<nome> ::= "cane" | "mela" | "gatto"
<verbo> ::= "mangia" | "graffia"
<complemento> ::= <articolo>" "<nome> | <articolo>" "<nome> "," <complemento>

S = frase

Esempio tratto dal libro: Pighizzini, Ferrari, "Dai fondamenti agli oggetti"

Esempio: BNF linguaggio naturale



Esempio tratto dal libro: Pighizzini, Ferrari, "Dai fondamenti agli oggetti"

Ambiguità sintattiche

Se una stringa ha più di un parse tree allora la grammatica associata è ambigua

Un linguaggio di programmazione va sempre descritto con una grammatica non ambigua, quindi serve disambiguare dove necessario

- Esempio: E::= E - E | 0 | 1

Ambiguità sintattiche

Se una stringa ha più di un parse tree allora la grammatica associata è ambigua

Un linguaggio di programmazione va sempre descritto con una grammatica non ambigua, quindi serve disambiguare dove necessario

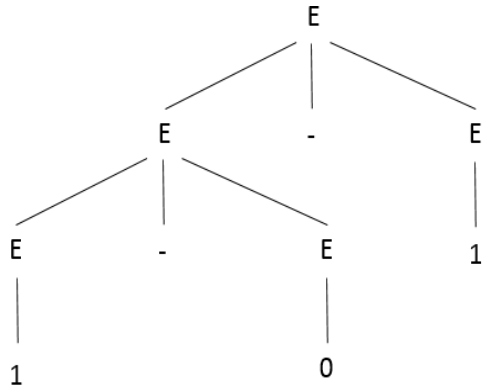
- Esempio: E::= E - E | 0 | 1
- La stringa 1 - 0 - 1 ha due parse tree quindi è ambigua la grammatica

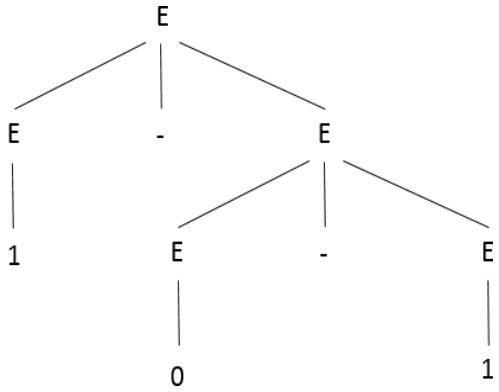
Ambiguità sintattiche

Se una stringa ha più di un parse tree allora la grammatica associata è ambigua

Un linguaggio di programmazione va sempre descritto con una grammatica non ambigua, quindi serve disambiguare dove necessario

- Esempio: E::= E - E | 0 | 1
- La stringa 1 - 0 - 1 ha due parse tree quindi è ambigua la grammatica





Grammatiche regolari e linguaggi regolari

Tipo 3 di Chomsky

Produzioni nella forma $A \rightarrow \beta$ $A \in N$ e $\beta \in (N \cup \Sigma)^*$

Produzione destra $A \rightarrow aB$ $B \rightarrow b$

Produzione sinistra $B \rightarrow Ab$ $A \rightarrow a$

con A,B ∈ N e a,b ∈ Σ

Un linguaggio regolare è un linguaggio le cui stringhe sono implicate da un' espressione regolare

Uso delle espressioni regolari

Grammatiche di tipo 3 sono adatte a rappresentare un ristrettissimo insieme di linguaggi formali

Le espressioni regolari vengono utilizzate come linguaggio di input per vari sistemi che trattano stringhe

Comandi UNIX (grep) per la ricerca di stringhe (metacaratteri)

Generatori di analizzatori lessicali, tipo Lex (Lexical analyzer generator) e Flex (Fast Lex))

Espressioni regolari: costruzione

Utilizzano le operazioni tipiche di un linguaggio formale *L*

Alfabeto Σ , unione $\Sigma \cup \Sigma$, concatenazione $\Sigma \cdot \Sigma$, Σ^*, Σ^+

Partendo da un linguaggio formato da stringhe di lunghezza unitaria se ne creano altri di lunghezza 2, 3, ...

Espressioni regolari: definizione

Espressione regolare r ,definita su Σ e su un insieme di metasimboli $+, *, (,), \cdot, \emptyset$ non appartenenti a Σ , è una stringa r appartenente all'alfabeto $(\Sigma \cup +, *, (,), \cdot, \emptyset)$ tale che valga una delle seguenti condizioni

- $r = \emptyset$
- $r = x, x \in \Sigma$
- $r = (s + t)$, s, t espressioni regolari su Σ e $+$ è l'unione
- $r = s \cdot t$, s,t espressioni regolari su Σ e $\cdot$ è la concatenazione
- $r = s^*$, s espressione regolare su Σ e $*$ è la chiusura di Kleene

Esempio:

- Espressione regolare per $L = \{w \in \{0,1\}^* : 0 \text{ e } 1 \text{ alternati in } w\}$

$$(01)^* + (10)^* + 0(10)^* + 1(01)^*$$

Espressioni regolari: nomi varabili

Nomi di variabili di un linguaggio di programmazione:

Stringhe che iniziano con una lettera e che possono contenere caratteri alfanumerici

caratteri = $A|B|C|\ldots|Z|a|b|\ldots|z|$

numeri = $0|1|2|3|4|5|6|7|8|9$

Espressione regolare:

variabile = *caratteri*(*caratteri*|*numeri*)$*$

- $L($*variabile*$) =$ *A,B,. . . ,a,. . . ,z,AA,. . . ,X*1*,. . . ,j*1*,. . . ,contatore,. . .*

Carta sintattica

Le carte sintattiche sono dei diagrammi che esprimono le regole di una grammatica in forma grafica.

- Un diagramma per ciascun simbolo non terminale

In una carta sintattica:

- I rettangoli indicano simboli non terminali (che andranno espansi con le carte sintattiche corrispondenti)
- Gli ovali o rettangoli con gli angoli arrotondati, indicano simboli terminali, che quindi non devono essere espansi ulteriormente
- Le frecce sono definite in modo tale che, seguendo i percorsi da esse delineati, sia possibile ricostruire una sequenza lecita di simboli
- Ogni biforcazione indica un'alternativa

Carta sintattica esempio

Carta sintattica per l'esempio delle espressioni matematiche