Riassunto delle Puntate Precedenti

Linguaggi di Programmazione

- Diversi linguaggi di programmazione usano diversi paradigmi (programming style)
- Non è facile valutare la qualità di un linguaggio di programmazione
- Non è possibile stabilire la correttezza di un programma tramite testing
- È impossibile verificare in modo consistente e completo qualsiasi proprietà di un programma che dipenda dall'esito della computazione

← Precedente Successiva →

Definire un Linguaggio

Cosa hanno in comune diversi linguaggi?

Aspetti importanti di un linguaggio di programmazione:

Sintassi

- Come si presenta il codice? Come è fatto?
- Forma delle sue espressioni, istruzioni ecc
- Specificata da una grammatica formale (BNF)
- Definisce il parser del co

Semantica

- Cosa succede durante l'esecuzione?
- Significato delle sue espressioni, istruzioni ecc
- Non esiste uno standard de facto
- Definisce le possibili tracce di esecuzione

← Precedente Successiva →

Sintassi e Semantica: Esempio

Ciclo in Java

```
while (<condizione>) <istruzione>
```

Sintassi: while (<condizione>) <istruzione>

Semantica: valuta la condizione, se è falsa termina, altrimenti esegui l'istruzione e ricomincia l'esecuzione del while

N.B. In un buon LdP la sintassi deve suggerire la semantica.

← Precedente Successiva →

Linguaggio di Programmazione

Definizione

Un linguaggio di programmazione è un insieme di regole che definiscono un insieme di stringhe (programmi) legali, attribuendo ad ogni stringa un preciso significato.

- Un linguaggio è definito prescindendo dai dettagli del dispositivo che eseguirà i programmi
- Alcune caratteristiche implementative possono condizionare i risultati durante l'esecuzione
- **Esempio:** il massimo intero rappresentabile varia da macchina a macchina

Problema: come dare una descrizione finita a un linguaggio che ha un numero infinito di frasi?

← Precedente Successiva →

Regole per la Definizione di un Linguaggio

Regole Lessicali

Descrivono quali sequenze di simboli costituiscono le parole del linguaggio

Esempio: un intero è una sequenza di caratteri numerici, non separati da spazi

Relazione tra simboli dell'alfabeto

Regole Sintattiche

Descrivono come le parole possono essere combinate per formare istruzioni legali

Esempio: un'espressione può essere costituita da un'espressione seguita da '+' o '-' e da un'altra espressione

Relazione tra parole

Regole Semantiche

Descrivono il significato di u

Relazione tra parole e sign

← Precedente Successiva →

Classificazione dei Linguaggi (Chomsky 1956-1959)

Classificazione basata sulla complessità delle produzioni delle grammatiche che li generano:

- **Linguaggi di tipo 0** (a struttura di fase)
- **Linguaggi di tipo 1** (dipendenti da contesto)
- **Linguaggi di tipo 2** (libere da contesto)
- **Linguaggi di tipo 3** (regolari)

Importante

I linguaggi liberi da contesto e i linguaggi regolari sono utilizzati per definire la sintassi dei linguaggi di programmazione.

BNF (Backus e Naur) e (Chomsky) sono strumenti teorici simil 2.

← Precedente Successiva →

Analisi Lessicale

- Le unità sintattiche più piccole sono chiamate **lessemi**
- Un **token** è una categoria di lessemi (es. identificatori, costanti intere ecc)

Esempio: index = 2 * count + 3

Lessemi	Tokens
index	identificatore
=	operatore
2	costante
*	operatore
count	identificatore
+	
3	

← Precedente Successiva →

Espressioni Regolari

Per descrivere la forma dei tokens possono essere utilizzate le espressioni regolari:

- Espressioni che descrivono stringhe
- `|` e `(  )` per indicare alternative
- `[:  :]` per indicare insiemi di caratteri
- * + per indicare ripetizioni

Esempi

```
prim(a|o) → descrive le stringhe "prima" e "primo"
[:digit:]+ → descrive tutte le stringhe di naturali con almeno una cifra
// [:alnum:]* \n → descrive una stringa che comincia con //, continua con
una sequenza qualsiasi di caratteri alfanumerici, e termina con newline
```

← Precedente Successiva →

Grammatica

Una grammatica è un metodo generativo per descrivere linguaggi.

Definizione Formale

Una grammatica è una quadrupla **G = (T, NT, P, S)**, dove:

- **T** è l'insieme di simboli terminali (di solito indicati con a, b, ...)
- **NT** è l'insieme finito di simboli non terminali o variabili (di solito indicati con A, B, ...)
- **P** è l'insieme delle produzioni dove ogni coppia $\langle \alpha, \beta \rangle \in P$ viene scritta come $\alpha \rightarrow \beta$
- **S ∈ NT** è il simbolo iniziale

← Precedente Successiva →

Esempio di Grammatica

Grammatica per stringhe palindrome su {a, b, c}

G = (T, NT, P, S) dove:

- **T** = {a, b, c}
- **NT** = {S}
- **P** = {S → aSa, S → bSb, S → cSc, S → ε, S → a, S → b, S → c}

Esempio di derivazione (generazione di programma)

```
S ⇒ aSa ⇒ abSba ⇒ abba
```

← Precedente Successiva →

Come Descrivere Formalmente un Linguaggio?

Distinguiamo:

Backus-Naur Form

Metodo più diffuso per descrivere la sintassi dei linguaggi di programmazione

Extended BNF

Migliora la leggibilità della BNF

Grafi Sintattici

- Introdotti per ALGOL 60 (Taylor, 1961)
- Ogni regola è descritta da un grafo orientato
- Simboli non terminali in rettangoli
- Simboli terminali in rettangoli arrotondati
- Ogni stringa valida percorre i grafi opportunamente

← Precedente Successiva →

Semantica di un Linguaggio

Definisce il significato di ogni costrutto sintattico legale del linguaggio

Semantica Statica

Può essere controllata **prima** dell'esecuzione del programma, perché non dipende dall'esito del calcolo

Semantica Dinamica

Deve essere controllata **durante** l'esecuzione del programma, perché specifica il comportamento atteso durante l'esecuzione

← Precedente Successiva →

Grammatiche ad Attributi

Definizione

Una grammatica ad attributi è una tripla ⟨G, A, E⟩ dove:

- **G** è una grammatica libera da contesto
- **A** è un insieme di attributi associati ai simboli della grammatica G
- **E** è un insieme di equazioni per il calcolo dei valori dei vari attributi (regole semantiche)

Tipi di Attributi

- **Sintetizzato:** calcolato dalle foglie alla radice (ascendente)
- **Ereditato:** calcolato dall
- **Intrinseco:** prodotto da

← Precedente Successiva →

Come Definire la Semantica Dinamica?

Non esiste uno standard come per la sintassi

La descrizione formale permette di verificare formalmente la correttezza di un programma

Principali Formalismi

Semantica Operazionale

Definisce un interprete del linguaggio su una macchina virtuale

Semantica Denotazionale

Definisce la semantica in termini di funzioni ricorsive

Semantica Assior

Fondata sulla logica matematica e sul concetto di asserzione

← Precedente Successiva →

Semantica Operazionale

Ottenuta definendo un interprete del linguaggio L su di una macchina virtuale i cui componenti sono descritti in modo "matematico"

Il significato di un'istruzione consiste nella variazione dello stato della macchina virtuale (registri, program counter, istruzione corrente, etc.) causato dall'esecuzione dell'istruzione stessa

Vantaggi (+)

- Vicina all'intuizione di un programmatore
- Simile ad un manuale
- Usata anche in pratica (per PL/I)

Svantaggi (-)

- Descrizioni molto complesse
- La macchina virtuale è spesso un altro linguaggio che necessita di essere definito formalmente

← Precedente Successiva →

Semantica Denotazionale

Metodo introdotto da Scott e Strachey nel 1971. È il metodo finora più utilizzato.

Definisce la semantica di un linguaggio in termini di **funzioni ricorsive**: per ogni costrutto del linguaggio si definisce un oggetto matematico che lo rappresenti e una funzione che collega istanze dello stesso costrutto allo stesso oggetto matematico.

Esempio: assegnamento x := E

$$M_a(i_k := E, s)\Delta =$$
$$\{ \text{ error if } M_e(E, s) = \text{error}$$
$$\{\langle i_1, \text{VARMAP}(i_1, s)\rangle, ..., \langle i_k, M_e(E, s)\rangle, ...\} \text{ otherwise } \}$$

Vantaggi (+)

(-)

- Matematicamente rigo essa

← Precedente Successiva →

Semantica Assiomatica

Introdotta da Floyd nel 1967 e raffinata da Hoare nel 1969.

Fondata sulla logica matematica e sul concetto di **asserzione**

Definizioni

- **Asserzione:** un'espressione logica (o predicato)
- **Postcondition:** descrive vincoli dopo un'istruzione
- **Precondition:** descrive vincoli prima di un'istruzione
- **Weakest precondition:** la precondition meno restrittiva che garantisce la postcondition

Esempio: Costrutto if-then-else

← Precedente Successiva →

Regola di inferenza:

Compilatori ed Interpreti

Importante

Alcune proprietà (⇒ siamo ancora nell'ambito della semantica di un linguaggio) degli elementi di un programma sono determinate dall'implementazione del linguaggio.

Classificazione dei Costrutti

- **Costrutti Safe:** il loro corretto utilizzo (semantica) può essere determinato automaticamente dal compilatore
- **Costrutti Unsafe:** il loro corretto utilizzo è a carico del programmatore

Per capire cosa si può valutare automaticamente utilizzando un particolare linguaggio dobbiamo avere una sensib[illegible]ento del relativo compilatore/interprete

← Precedente Successiva →