Corso di Linguaggi di Programmazione

Lezione 13

Alberto Ceselli
alberto.ceselli@unimi.it

Dipartimento di Informatica
Università degli Studi di Milano

23 Aprile 2013

Università degli Studi di Milano

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.

Università degli Studi di Milano

Cosa hanno in comune diversi linguaggi?

Aspetti importanti di un linguaggio di programmazione (sia da parte di chi lo deve costruire/estendere, sia da parte di chi lo deve utilizzare):

- *sintassi*: come si presenta il codice? come è fatto? (forma della sue espressioni, istruzioni ecc)
  - specificata da una grammatica formale (BNF)
  - definisce il *parser* del compilatore
- *semantica*: cosa succede durante l'esecuzione di un programma? (significato delle sue espressioni, istruzioni ecc)
  - non esiste uno standard de facto per la descrizione della semantica di un linguaggio
  - definisce le possibili tracce di esecuzione del programma

Sintassi e semantica: esempio

Ciclo in Java:

- sintassi: `while (<condizione>) <istruzione>`
- semantica: valuta la condizione, se è falsa termina, altrimenti esegui l'istruzione e ricomincia l'esecuzione del while

N.B. I nun buon LdP la *sintassi* deve suggerire la *semantica*.

Linguaggio di Programmazione

Definition

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
  - **Es:** 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?

Regole per la definizione di un linguaggio

- **Lessicali**, che descrivono quali *sequenze di simboli* costituiscono le parole del linguaggio (i.e., come sono fatti i token del linguaggio - identificatori, parole chiave, ...)
  - esempio: un intero è una sequenza di caratteri numerici, non separati da spazi.
  - relazione tra simboli dell'alfabeto.
- **Sintattiche**, che descrivono come le parole del linguaggio possono venire combinate tra di loro per formare istruzioni legali, ovvero quali *sequenze di parole* sono corrette
  - esempio: un'espressione può essere costituita da un'espressione seguita dal simbolo '+' o '-' e da un'altra espressione.
  - relazione tra parole.
- **Semantiche**, che descrivono il *significato di una frase corretta*
  - relazione tra parole e significati.

Classificazione dei Linguaggi (Chomsky 1956-1959)

È una 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 *grammatiche libere da contesto* (Chomsky) sono due strumenti teorici molto simili per descrivere Linguaggi di tipo 2. Espressioni regolari e *grammatiche regolari* sono strumenti per descrivere Linguaggi di tipo 3.

Regole lessicali

- Le unità sintattiche più piccole sono chiamate *lessemi*
- In un linguaggio, un *token* è una categoria di lessemi (es. identificatori, costanti intere ecc)
- Esempio: `index = 2 * count + 3`

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

Università degli Studi di Milano

Espressioni regolari

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

- espressioni che descrivono *stringhe*
- esempio: `|` e `( )` per indicare alternative, `[: :]` per indicare insiemi di caratteri, `*` `+` per indicare ripetizioni
- esempio: `prim(a|o)` descrive le strighe `prima` e `primo`
- esempio: `[:digit:]+` descrive tutte le stringhe di naturali con almeno una cifra
- esempio: `// [:alnum:]* \n` descrive una stringa che comincia con `//`, continua con una sequenza qualsiasi di caratteri alfanumerici, e termina con newline

Università degli Studi di Milano

Sintassi

Definition

Sintassi. "La parte della grammatica che contiene le regole di combinazione degli elementi lessicali e significativi, e quindi di formazione delle frasi" (Zingarelli).

- In informatica: l'insieme delle regole per la corretta costruzione delle istruzioni in un linguaggio di programmazione.
- La sintassi fornisce le informazioni necessarie per comprendere la struttura (e quindi parte del significato) di un programma.
- Identificare la struttura sintattica di un programma è il primo passo sia in caso venga interpretato, sia in caso venga compilato.

UNIVERSITÀ DEGLI STUDI DI MILANO

Sintassi

Come definire la *sintassi* di un LdP?

- Descrivendo come *riconoscere* programmi sintatticamente validi in quel LdP (analizzatori sintattici, o *parser*)
- Descrivendo come *generare* programmi sintatticamente validi in quel LdP (generalmente, più espressivo)

(anche se nessuno dei due modi è utile per chi deve *imparare* ad utilizzare il linguaggio!)

Grammatica

Una *grammatica* è un *metodo generativo* per descrivere linguaggi. Formalmente, una *grammatica* è una quadrupla $G = (T, NT, P, S)$, dove:

- T è l'insieme di **simboli terminali**
  - di solito indicati con $a, b, \ldots$
- NT è l'insieme finito di **simboli non terminali** o variabili
  - di solito indicati con $A, B, \ldots$
- $P = \{\langle\alpha_1, \beta_1\rangle, \ldots, \langle\alpha_n, \beta_n\rangle\}$ è l'insieme delle **produzioni**
  - dove α e β indicano stringhe miste, e α_j contiene almeno un simbolo non terminale per $1 \leq j \leq n$
  - ogni coppia $\langle\alpha, \beta\rangle \in P$ di solito viene scritta come $\alpha \rightarrow \beta$
- $S \in NT$ è il **simbolo iniziale**.

Esempio (1)

Una grammatica per stringhe di palindrome su $\{a, b, c\}$.
$G = (T, NT, P, S)$ dove:

- $T = \{a, b, c\}$,
- $NT = \{S\}$,
- $P = \{S \rightarrow aSa, S \rightarrow bSb, S \rightarrow cSc,$
  $S \rightarrow \epsilon, S \rightarrow a, S \rightarrow b, S \rightarrow c\}$.

Un esempio di **derivazione** (generazione di programma) è
$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$.

Come descrivere formalmente un linguaggio?

Distinguiamo:

- Bakus-Naur Form e Grammatiche libere da contesto
  - Metodi piú diffusi per descrivere la sintassi dei linguaggi di programmazione
- Extended BNF
  - Migliora la leggibilitá della BNF
- Grafi sintattici
  - Introdotti per descrivere l'ALGOL 60 (Taylor, 1961)
  - Ogni regola è descritta da un grafo orientato, dove i simboli non terminali sono racchiusi in rettangoli e i simboli terminali sono racchiusi in rettangoli arrotondati.
  - Ogni stringa valida può essere ottenuta percorrendo i grafi in modo opportuno.

Semantica di un linguaggio

- Definisce il *significato* di ogni costrutto sintattico legale del linguaggio;
- Distinguiamo semantica **statica** da semantica **dinamica**:
  - la semantica *statica* può essere controllata *prima* dell'esecuzione del programma, perchè non dipende dall'esito del calcolo
  - la semantica *dinamica* deve essere controllata *durante* l'esecuzione del programma, perchè specifica il comportamento atteso durante l'esecuzione.

Università degli Studi di Milano

Semantica statica e BNF

- Alcune regole semantiche possono essere specificate anche in BNF (non tutte)
- Regola: Un valore floating point non può essere assegnato ad una variabile di tipo int
- ... può essere specificata in BNF, introducendo nuovi nonterminali e nuove regole.
- Regola: Nessuna variabile può essere utilizzata prima della sua dichiarazione
- ... non può essere specificata in BNF.

Semantica statica: Grammatica ad attributi

Una **grammatica ad attributi** è una tripla $\langle G, A, E \rangle$ dove:

- G è una grammatica libera da contesto
- A è un insieme di *attributi* associati ai simboli non-terminali e terminali della grammatica G
- E è un insieme di equazioni per il calcolo dei valori dei vari attributi, dette *regole semantiche*, associate alle regole di produzione della grammatica G.
  - Le regole semantiche sono scritte in un'opportuna notazione, detta *metalinguaggio semantico*, ad esempio notazioni matematiche, logiche o imperative

Grammatica ad attributi - Attributi (1)

Sia $X_0 \rightarrow X_1 \ldots X_n$ una regola sintattica, ad ogni simbolo non terminale X_i si associano uno o più attributi, $A(X_i)$.
Un attributo di X ha un dominio (ad esempio può assumere valori interi o stringhe) e può essere:

- **Sintetizzato**
- **Ereditato**
- **Intrinseco**

Grammatica ad attributi - Attributi (2)

- **Sintetizzato** $S(X_0)$
  - il suo valore dipende da $X_1, \ldots, X_n$: un attributo del padre è calcolato in funzione dei valori degli attributi dei figli.
  - $S(X_0) = f(A(X_1), \ldots, A(X_n))$
  - il calcolo va dalle foglie alla radice (ascendente)
- **Ereditato** $I(X_j)$
  - il suo valore dipende da $X_0, X_1, \ldots, X_n$: un attributo di un nodo è calcolato in funzione dal valore degli attributi del suo padre e dei fratelli (a volte solo quelli left-most per non avere cicli).
  - $I(X_j) = f(A(X_0), \ldots, A(X_n))$, $1 \leq j \leq n$
  - non c'è un ordine particolare per il calcolo, tendenzialmente dalla radice alle foglie (discendente)
- **Intrinseco** $V(X)$
  - prodotto dall'esterno (es. analizzatore lessicale o lettura da tastiera).

Grammatica ad attributi - Notazioni

- $\langle A \rangle.a$ indica l'attributo a del simbolo non terminale $\langle A \rangle$
- I simboli non terminali che compaiono più volte nella parte destra di una regola di produzione sono indicizzati per distinguere le occorrenze multiple
  - $\langle expr \rangle \rightarrow \langle T \rangle + \langle T \rangle$ diventa $\langle expr \rangle \rightarrow \langle T \rangle[1] + \langle T \rangle[2]$
  - oppure, $E \rightarrow T + T$ diventa $E \rightarrow T_1 + T_2$

Come definire formalmente la semantica (dinamica) di un linguaggio?

- Non esiste uno standard come nel caso della descrizione formale della sintassi;
- La descrizione formale rispetto al linguaggio naturale permette di verificare formalmente la correttezza di un programma (vedi theorem prover basati su semantica assiomatica);
- Principali formalismi utilizzati:
  - semantica operazionale
  - semantica denotazionale
  - semantica assiomatica

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.
- Esempio: pag 141 Sebesta

Semantica Operazionale: valutazione

- Descrizioni molto complesse
- Spesso la macchina virtuale è un *linguaggio di programmazione* più semplice ... che necessita di essere definito formalmente
- ... infatti, la macchina fisica è macchina virtuale per il linguaggio macchina

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

Università degli Studi di Milano

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 di uno stesso costrutto allo stesso oggetto matematico.
- Lo stato delle variabili viene esaminato e modificato tramite funzioni.

Semantica Denotazionale: esempio

Esempio: assegnamento $x := E$ con associata una funzione $M_a()$

$$M_a(i_k := E, s)\Delta = \begin{cases} \textbf{error} & \textit{if } M_e(E, s) = \textbf{error} \\ \{< i_1, VARMAP(i_1, s) >, \ldots, < i_k, M_e(E, s) >, \ldots, < i_n, VARMAP(i_n, s) >\} & \textit{otherwise} \end{cases}$$

Semantica Denotazionale: valutazione

+ Matematicamente rigorosa
+ Può aiutare il *progettista* del linguaggio (se un costrutto richiede descrizione complessa, probabilmente risulterà oscuro anche ad un programmatore, ed andrà quindi riprogettato).
- Molto complessa
- Poco utile per comprendere il linguaggio

Semantica Assiomatica

- Introdotta da Floyd nel 1967 e raffinata da Hoare nel 1969.
- Fondata sulla logica matematica e sul concetto di **asserzione**
- Ad ogni istruzione sono associati vincoli sul valore che devono assumere le variabili *prima* e *dopo* l'esecuzione dell'istruzione

Semantica Assiomatica: definizioni

- Un'*asserzione* è un'espressione logica (o predicato)
- Se descrive vincoli sul valore delle variabili *dopo* un'istruzione l'asserzione è una *postcondition*.
- Se descrive vincoli sul valore delle variabili *prima* di un'istruzione l'asserzione è una *precondition*.
- Esempio: `sum = 2 * x + 1`
- ... pongo come postcondition: $\{\text{sum} > 1\}$
- ... ne deriva una precondition,
  tipo: $\{x > 100\}$ (oppure $\{x > 20\}$)
- Una postcondition diventa la precondition dell'istruzione successiva

Semantica Assiomatica: weakest preconditions

- Data una postcondition
- ... tra tutte le preconditions cerchiamo la *meno restrittiva* che garantisce di rispettare la postcondition
- ... nell'esempio precedente la weakest precondition è $\{x > 0\}$
- La postcondition dell'ultima istruzione è il risultato del programma
- Per valutare semanticamente il programma partiamo dalla postcondition dell'ultima istruzione e calcoliamo all'indietro le weakest preconditions (che diventano le postconditions dell'istruzione precedente)

Università degli Studi di Milano

Semantica Assiomatica: esempio

Costrutto di selezione if-then-else

- {P} if B then S1 else S2 {Q}
- Regola di inferenza di preconditions (P) data una postcondition (Q):

$$\frac{\{B \ and \ P\}S1\{Q\}, \{(not \ B) \ and \ P\}S2\{Q\}}{\{P\}if \ \ B \ \ then \ S1 \ else \ S2\{Q\}}$$

- Esempio: if (x > 0) then y := y-1 else y := y+1 (alla lavagna, Sebesta pag 148)

- Poco chiara come strumento di definizione.
- Difficile definire regole di inferenza (a meno di progettare l'intero linguaggio con semantica assiomatica in testa)
- Poco utilizzata, anche durante il progetto di compilatori

\+ In alcuni casi (programmi piccoli e semplici) permette di provare formalmente la correttezza di un programma.

\+ Può aiutare *durante lo sviluppo di programmi*, per inserire *assertions* nel programma stesso

Motivation

Importante

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

Nelle prossime lezioni, analizziamo diversi aspetti dei linguaggi di programmazione, cercando di capire cosa si può valutare in modo automatico, come e perchè.
In particolare, parleremo di

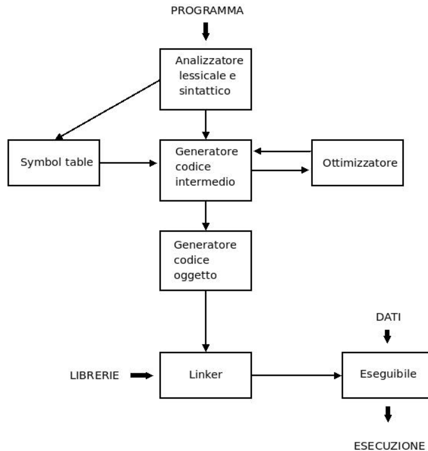
- costrutti o feature **safe** se il loro corretto utilizzo (semantica) può essere determinato **automaticamente dal compilatore**
- costrutti o feature **unsafe** se il loro corretto utilizzo è **a carico del programmatore**

Linguaggi e compilatori

Per capire cosa si può valutare automaticamente utilizzando un particolare linguaggio – e come – dobbiamo avere una sensibilità "a basso livello" del funzionamento del relativo compilatore / interprete ...

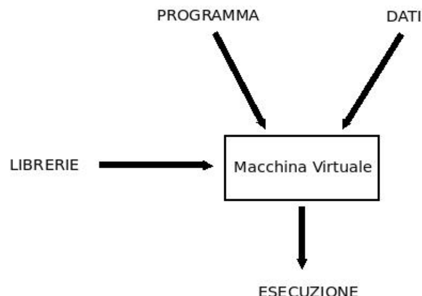
Università degli Studi di Milano

Struttura di un compilatore



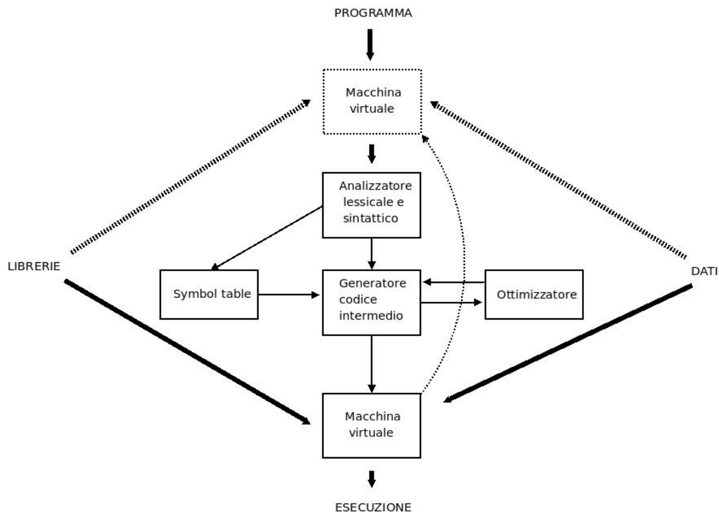
Vantaggi: massima efficienza! Es. C, C++

Struttura di un interprete



Vantaggi: massima flessibilità!
Es. Scheme, Prolog

Sistema ibrido



Generazione di codice intermedio

- un *compilatore* genera solo codice *intermedio*
- una *macchina virtuale* interpreta ed esegue il codice intermedio

Es. Java

UNIVERSITÀ DEGLI STUDI DI MILANO

Compilatori just-in-time

- una *macchina virtuale* riceve il codice
- quando una porzione di codice deve essere eseguita, la macchina virtuale ne richiede la *compilazione*
- viene eseguito il codice compilato
- il controllo torna alla macchina virtuale
- N.B. ogni porzione di codice è compilata *una sola volta* (indipendentemente dal numero di attivazioni)

Es. C#

Linguaggi interpretati e compilati

N.B. in linea di principio, *ogni* linguaggio può essere *sia* compilato *che* interpretato

- nulla vieta di scrivere interpreti per C.
- nulla vieta di compilare ed ottimizzare codice SCHEME, ottenendo degli eseguibili.
- ... oppure operare scelte intermedie