Linguaggi di Programmazione

Concetti di teoria dei linguaggi formali,
paradigmi linguistici, Sintassi e Semantica,
Macchina astratta

Testo di riferimento

GABBRIELLI, M., MARTINI, S., LINGUAGGI DI PROGRAMMAZIONE: PRINCIPI E PARADIGMI. MCGRAW-HILL, Seconda Edizione

Ispirate (ringraziamo) dalle Slide del prof. Giorgio Levi (Unipi)

Struttura del corso 1

macchine astratte, interpreti, compilatori, implementazioni miste

cenni di semantica denotazionale ed operazionale

tipi di dato, tipi di dato astratti, tipi

espressioni e comandi

ambiente, dichiarazioni, blocchi

sottoprogrammi, regole di scoping, passaggio di parametri

Struttura del corso 2

classi e oggetti

gestione dell'ambiente: implementazione

gestione della memoria: implementazione

ambiente globale, moduli, compilazione separata

analisi statica, interpretazione astratta, esempi di analizzatori (inclusa inferenza di tipi)

specializzazione di interpreti e generazione del codice via valutazione parziale

struttura della macchina intermedia: esempi

Spirito del corso 1

definizioni “formali” e realizzazioni implementate

trattando vari linguaggi anche orientati ad oggetti,
con frammenti di linguaggio funzionale
E di linguaggio imperativo

dalla semantica denotazionale al codice compilato attraverso
una serie di trasformazioni “sistematiche” di semantiche
l’utilizzazione di tecniche formali di analisi e trasformazione

Spirito del corso 2

accanto al filone principale, ci saranno delle “digressioni” (indicate esplicitamente) per descrivere

costrutti e meccanismi diversi da quelli del linguaggio didattico

implementazioni alternative

non sempre le digressioni saranno descritte in modo “eseguibile”

le digressioni riguarderanno spesso i linguaggi “veri”

- sia quelli ‘vivi”
- che alcuni reperti archeologici importanti

dei paradigmi linguistici importanti

non ignoreremo i linguaggi (e i costrutti) per la programmazione concorrente

tratteremo in modo superficiale la programmazione logica

- perché non “omogenea” agli altri paradigmi

Spirito del corso 3

come vedremo, esistono un insieme di concetti semantici e di strutture di implementazione in termini dei quali si descrivono in modo naturale linguaggi diversi e loro implementazioni

ed esiste una chiara relazione tra concetti semantici e strutture di implementazione

tali concetti e strutture mettono in evidenza ciò che è comune ai vari linguaggi e possono essere trattati a prescindere dal particolare linguaggio

Spirito del corso 4

il livello di descrizione sarà raramente quello di un "manuale d'uso", che quasi sempre non contiene

nè una descrizione formale della semantica indipendente dall'implementazione

- necessaria per poter ragionare sul significato dei programmi che scriviamo

nè una descrizione delle strutture a tempo di esecuzione della particolare implementazione

- necessaria per ragionare sulla "performance" dei nostri programmi

Spirito del corso 4

conoscere questi principi di semantica e di tecniche di implementazione consente di

migliorare la conoscenza del linguaggio che usate comunemente

- perché quel meccanismo non è fornito o è particolarmente costoso?

migliorare il vostro “vocabolario di costrutti”

- un costrutto che ci sarebbe utile, ma non viene fornito dal linguaggio, può spesso essere simulato

imparare agevolmente un nuovo linguaggio

scegliere il linguaggio più adatto alle vostre esigenze

- o almeno avere argomenti tecnici per discutere con il capo che vi vuol fare programmare in COBOL!

progettare un nuovo linguaggio o estenderne uno esistente

- capita più spesso di quanto possiate immaginare!

Linguaggi di programmazione e astrazione

i linguaggi di programmazione ad alto livello moderni sono il più potente strumento di astrazione messo a disposizione dei programmatori

che possono, con un solo costrutto del linguaggio, “rappresentare” un numero (anche infinito) di interminabili sequenze di istruzioni macchina corrispondenti

i linguaggi si sono evoluti trasformando in costrutti linguistici (e realizzando una volta per tutte nell’implementazione del linguaggio)

tecniche e metodologie sviluppate nell’ambito della programmazione, degli algoritmi, dell’ingegneria del software e dei sistemi operativi

- in certi casi perfino in settori di applicazioni (basi di dati, intelligenza artificiale, simulazione, etc.)

di fondamentale importanza è stata l’introduzione nei linguaggi di vari meccanismi di astrazione, che permettono di

estendere il linguaggio (con nuove operazioni, nuovi tipi di dato, etc.) semplicemente scrivendo dei programmi nel linguaggio stesso

Un po’ di storia

i linguaggi di programmazione nascono con la macchina di Von Neumann (macchina a programma memorizzato)

i programmi sono un particolare tipo di dato rappresentato nella memoria della macchina

la macchina possiede un interprete capace di fare eseguire il programma memorizzato, e quindi di implementare un qualunque algoritmo descrivibile nel “linguaggio macchina”

un qualunque linguaggio macchina dotato di semplici operazioni primitive per effettuare la scelta e per iterare (o simili) è Turing-equivalente (o Turing Completo), cioè può descrivere tutti gli algoritmi

i linguaggi hanno tutti lo stesso potere espressivo, ma la caratteristica distintiva importante è il “quanto costa esprimere”

direttamente legato al “livello di astrazione” fornito dal linguaggio

I linguaggi macchina ad alto livello

dai linguaggi macchina ai linguaggi Assembler

nomi simbolici per operazioni e dati

(anni 50) FORTRAN e COBOL (sempreverdi)

notazioni ad alto livello orientate rispettivamente al calcolo scientifico (numerico) ed alla gestione dati (anche su memoria secondaria)

astrazione procedurale (sottoprogrammi, ma con caratteristiche molto simili ai costrutti forniti dai linguaggi macchina)

nuove operazioni e strutture dati

(per esempio, gli array in FORTRAN, e i records in COBOL)

nulla di significativamente diverso dai linguaggi macchina

I favolosi anni ’60: LISP e ALGOL’60

risultati teorici a monte

formalizzazione degli aspetti sintattici

primi risultati semantici basati sul λ-calcolo

caratteristiche comuni

introduzione dell’ambiente

vera astrazione procedurale con ricorsione

argomenti procedurali e per nome

ALGOL’60

primo linguaggio imperativo veramente ad alto livello

scoping statico

gestione dinamica della memoria a stack

LISP (sempreverde, ancora oggi il linguaggio dell’A.I.)

primo linguaggio funzionale, direttamente ispirato al λ-calcolo

scoping dinamico

strutture dati dinamiche, gestione dinamica della memoria a heap con garbage collector

Estendere la macchina fisica o implementare una logica

ALGOL'60, prototipo dei linguaggi imperativi

parte dalla struttura della macchina fisica

la estende con nuovi potenti meccanismi

LISP, prototipo dei linguaggi logici e funzionali

parte da un calcolo logico (λ-calcolo)

ne definisce una implementazione sulla macchina fisica

ne nascono concetti simili

non a caso basati sulla teoria

gli approcci restano diversi e danno origine a due filoni

il filone imperativo

il filone logico

che sono tuttora vitali

La fine degli anni ’60

PL/I: il primo tentativo di linguaggio “totalitario” (targato IBM)

tentativo di sintesi fra LISP, ALGOL’60 e COBOL

- fallito per mancanza di una visione semantica unitaria

SIMULA’67: nasce la classe

estensione di ALGOL’60 orientato alla simulazione discreta

quasi sconosciuto, riscoperto 15 anni dopo

Evoluzione del filone imperativo

risultati anni ’70

metodologie di programmazione, tipi di dati astratti, modularità, classi e oggetti

programmazione di sistema in linguaggi ad alto livello: eccezioni e concorrenza

PASCAL

estensione di ALGOL’60 con la definizione di tipi (non astratti), l’uso esplicito di puntatori e la gestione dinamica della memoria a heap (senza garbage collector)

semplice implementazione mista (vedi dopo) facilmente portabile

Il dopo PASCAL

C

PASCAL + moduli + tipi astratti + eccezioni + semplice interfaccia per interagire con il sistema operativo

ADA: il secondo tentativo di linguaggio “totalitario” (targato Dipartimento della Difesa U.S.A.)

come sopra + concorrenza + costrutti per la programmazione in tempo reale

progetto ambizioso, anche dal punto di vista semantico, con una grande enfasi sulla semantica statica (proprietà verificabili dal compilatore)

C++

C + classi e oggetti (allocati sulla heap, ancora senza garbage collector)

L’evoluzione del filone logico: programmazione logica

PROLOG

implementazione di un frammento del calcolo dei predicati del primo ordine

strutture dati molto flessibili (termini) con calcolo effettuato dall’algoritmo di unificazione

computazioni non-deterministiche

gestione della memoria a heap con garbage collector

CLP (Constraint Logic Programming)

PROLOG + calcolo su domini diversi (anche numerici) con opportuni algoritmi di soluzione di vincoli

L’evoluzione del filone logico: programmazione funzionale

ML

implementazione del λ-calcolo tipato

definizione di nuovi tipi ricorsivi

i valori dei nuovi tipi sono termini, che possono essere visitati con un meccanismo di pattern matching (versione semplificata dell’unificazione)

scoping statico (a differenza di LISP)

semantica statica molto potente (inferenza e controllo dei tipi)

- un programma “corretto” per la semantica statica quasi sempre va bene

gestione della memoria a heap con garbage collector

HASKELL

ML con regola di valutazione “lazy”

JAVA

molte caratteristche dal filone imperativo

essenzialmente tutte quelle del linguaggio più avanzato del filone, cioè C++

alcune caratteristiche dei linguaggi del filone logico

gestione della memoria con garbage collector

utilizza il meccanismo delle classi e dell'ereditarietà per ridurre il numero di meccanismi primitivi

quasi tutto viene realizzato con classi predefinite nelle librerie

ha una implementazione mista (anch'essa tipica del filone logico)

che ne facilita la portabilità e lo rende particolarmente adatto ad essere integrato nelle applicazioni di rete

Rust

eredita caratteristiche sia dal filone funzionale che OO

Cerca di prevenire problemi di gestione della memoria

Aggiungendo controlli a compile time

Sulla condivisione di variabili

Permette di realizzare software più affidabile

sia sequenziale che parallelo

Perchè limita errori di programmazione

dovuti a riferimenti errati ad aree di memoria condivisa

Materiale didattico, esame, istruzioni per l'uso del corso

il materiale didattico delle lezioni sarà disponibile via Dropbox

(previo announce sul gruppo Google

In490_1920

Inviate mail a

lombardi@mat.uniroma3.it

per essere invitati)

così come tutti i testi delle esercitazioni

esame = prova scritta + discussione progetto

consigli

seguire il corso

partecipare (attivamente) alle lezioni

sostenere le prove intermedie