Fondamenti - modulo Linguaggi A.A. 2017/2018[1]

Massimo Merro

Università degli Studi di Verona

3 Ottobre 2017

[1] Parte del materiale proviene dai corsi tenuti, dai proff. Matthew Hennessy, Trinity College Dublin, e Peter Sewell, University of Cambridge.

"Quelli che s'innamoran di pratica sanza scientia

son come 'l nochiere ch'entra in navilio sanza timone o bussola

che mai a' certezza dove si vada;

sempre la pratica debbe essere edificata sopra la bona teorica."

– Leonardo da Vinci.

Progettazione di un sistema informatico

- Un ingegnere civile costruisce ponti, palazzi, etc
- Un ingegnere nucleare costruisce centrali nucleari...
- Entrambi iniziano la fase di costruzione solo dopo aver fatto approfonditi studi attraverso modelli matematici appropriati
- E un ingegnere informatico? Che progetta sistemi software?
- Nel cosiddetto "software engineering" la verifica viene spesso fatta a *posteriori*, attraverso il **testing**, mandando in esecuzione il sistema in condizioni ambientali diverse (tale processo può durare mesi)
- Il testing però non è affidabile per la verifica di sistemi concorrenti e/o distribuiti in cui le computazioni possibili sono un mumero altissimo
- L'*informatica teorica* si occupa dello sviluppo di **modelli formali** per la progettazione e la verifica di sistemi software complessi; un po' come un ingegnere civile usa le equazioni differenziali per valutare il carico sostenibile di un ponte. *A priori* non *a posteriori*!

Metodi Formali

Modelli matematici e/o *logici* sono ormai ampiamente applicati in campo informatico:

- *Progettazione* di nuovi Linguaggi di Programmazione
  - questo presuppone l'essere in grado di descrivere l'esatto comportamento di un qualsiasi programma del linguaggio
- *Modelli matematici* per rappresentare il comportamento di sistemi
  - Obiettivo: predire il comportamento dei sistemi informatici avvalendosi di tecniche di analisi statica come il *model checking*
- *Verifica* che programmi/sistemi informatici si comportino come previsto:
  - In passato: solo testing intensivo
  - Attualmente: testing integrato con l'uso di metodi formali di verifica.

Interessi industriali

Intel, IBM, Microsoft, Google, Oracle, NASA ...

Fondamenti: modulo di Linguaggi

Obiettivi del corso:

Introdurre lo studente (i) all'uso dei metodo formali per specificare la semantica dei linguaggi di programmazione (ii) a tecniche formali per la verifica statica di proprietà di buon comportamento dei programmi.

Cosa si impara in questo corso:

Alla fine del corso lo studente sarà in grado di:

- fornire la semantica operazionale di semplici linguaggi imperativi, funzionali, concorrenti, e orientati a oggetti.
- definire un sistema di tipi per questi linguaggi, in grado di individuare staticamente programmi scorretti
- provare formalmente il buon comportamento di piccoli programmi
- capire i principi fondamentali su cui si basa la nozione di equivalenza comportamentale tra programmi.

Definizione di un Linguaggio di Programmazione

- **Sintassi**: quali stringhe sono ammissibili in un programma? Cioè, quali sono i vocaboli del nostro linguaggio? Per rispondere a questa domanda usiamo strumenti formali come: grammatiche, analizzatori lessicali e sintattici, teoria degli automi.....
- **Pragmatica**: esempi che ci facciano capire quale sono le caratteristiche del linguaggio, gli obiettivi progettuali, implementativi, etc. In pratica, le ragioni per introdurre un nuovo linguaggio di programmazione invece di usarne uno già esistente.
- **Semantica**: Il *significato* dei programmi scritti nel linguaggio, cioè il loro comportamento a tempo di esecuzione. Per capire quando due programmi apparentemente diversi si comportano in realtà nello stesso modo. Oppure, per poter dire quando un programma soddisfa la sua specifica.

Benefici di una semantica formale

- **Implementazione**: Consente di fornire la *specifica* (del comportamento) dei programmi indipendentemente dalla macchina o dal compilatore utilizzato.
  Correttezza di un compilatore, ottimizzazioni, analisi statiche, etc
- **Verifica**: una semantica formale consente di ragionare sui programmi e sulle loro proprietà di correttezza. Sia con carta e penna (per casi piccoli) che attraverso strumenti automatici (ad es. model checker) in grado di verificare sistemi di grosse dimensioni.
- **Progettazione del Linguaggio**: spesso una semantica formale consente di scoprire ambiguità all'interno di linguaggi già esistenti. Questo aiuta a progettare nuovi linguaggi in maniera più chiara e concisa.

Descrizioni informali

Vediamo la *definizione ufficiale* in C, del costrutto:

```
while B do C
```

Tratto da: B.W. Kerrigan and D.M. Ritchie, *The C Programming Language*, Prentice-Hall, 1976, pp 202:

"*The command C is executed repeatedly so long as the value on the expression B remains true. The test takes place before each execution of the command.*"

Descrizioni informali

Un estratto dal manuale dell'Algol 60:

"*Finally the procedure body, modified as above, is inserted in place of the procedure statement and executed. If a procedure is called from a place outside the scope of any non-local quantity of the procedure body, the conflicts between the identifiers inserted through this process of body replacement and the identifiers whose declarations are valid at the place of the procedure statement or function designator will be avoided through suitable systematic changes of the latter identifiers.*"

Qualcuno di voi saprebbe spiegare di cosa si parla nel testo?

Descrizioni formali

Statement formale:

$$\langle P, s_{\mathrm{i}} \rangle \Downarrow s_{\mathrm{f}}$$

Significato:

Se il programma P viene eseguito in uno stato iniziale s_{i} allora terminerà in uno stato finale s_{f}.

Logica:

- Quali regole logiche sono soddisfatte da statement del genere?
- Come viene determinato da queste regole logiche il comportamento a run time di un programma?

Descrizioni formali

Statement formale:

$$\langle C, s_{\mathrm{i}} \rangle \Downarrow s_{\mathrm{f}}$$

Significato:

Se il programma C viene eseguito in uno stato iniziale s_{i} allora terminerà in uno stato finale s_{f}.

Regole logiche:

$$\text{(IFF)}\ \frac{\langle B, s\rangle \Downarrow \mathtt{ff} \quad \langle C_2, s\rangle \Downarrow s'}{\langle \text{if } B \text{ then } C_1 \text{ else } C_2,\ s\rangle \Downarrow s'} \qquad \text{(IFT)}\ \frac{\langle B, s\rangle \Downarrow \mathtt{tt} \quad \langle C_1, s\rangle \Downarrow s'}{\langle \text{if } B \text{ do } C_1 \text{ else } C_2,\ s\rangle \Downarrow s'}$$

Stili differenti per fornire semantiche formali

Esistono vari stili per fornire una semantica formale di un linguaggio di programmazione.

- **Operazionale**: Il significato di un programma è dato fornendo i passi della computazione che ha luogo quando il programma viene eseguito.
- **Denotazionale**: Il significato di un programma è dato da un elemento di una qualche struttura matematica (ad esempio, un insieme).
- **Assiomatica**: Il significato di un programma è dato indirettamente, attraverso assiomi e regole di una qualche logica che esprimono proprietà del linguaggio.

Stato dell'arte

- **Operazionale**: la maggior parte dei linguaggi imperativi e sequenziali e alcuni linguaggi concorrenti sono provvisti di una semantica operazionale.
- **Denotazionale**: usata in semplici linguaggi imperativi e su alcuni linguaggi funzionali. La teoria matematica è in generale molto elegante.
- **Assiomatica**: Poco è stato fatto al di là dello studio di semplici linguaggi imperativi.

La ricerca prosegue su tutti e tre i fronti.

Fornire una semantica formale serve davvero?

La storia moderna è piena di malfunzionamenti, anche molto gravi, di sistemi informatici complessi che non sono stati *adeguatamente verificati* usando metodi formali:

- 1962: la sonda spaziale USA Mariner 1 viene distrutta per via di un trattino mancante nel programma di guida, scritto in Fortran. Costo: 80 milioni di dollari.
- 1985-1987: il software di radioterapia Therac-25 sottopone pazienti malati di cancro a sovradosaggi letali di radiazioni, mietendo quattro vittime.
- 1990: la rete interurbana della AT&T collassa per nove ore; la causa fu attribuita a una singola riga di codice errata.

- 1991: Guerra del Golfo. I missili Patriot intercettano numerosi missili scud iracheni, ma falliscono frequentemente. Il fallimento più grave consente a uno Scud di uccidere 28 soldati americani a Dhahran, in Arabia Saudita. Il sistema antimissile era concepito per funzionare ininterrottamente per un massimo di 14 ore. Durante l'attacco a Dhahran, la batteria difensiva di Patriot rimase in funzione per 100 ore consecutive...
- 1994: Il processore Pentium sbaglia a calcolare la divisione! La Intel si vede costretta a offrire la sostituzione gratuita di tutti i processori difettosi. Costo stimato di 450 milioni di dollari.
- 1995: il software MacInTax della Intuit mette in piazza i segreti fiscali dei contribuenti americani. Il codice presente nei file di debug di MacInTax consente agli utenti Unix di accedere al computer centrale della Intuit, dove vengono conservate tutte le dichiarazioni redatte con MacInTax, e modificarle o cancellarle a piacimento.

- 1996: il vettore Ariane 5 esplode al decollo. Il disastro avviene perchè il programma del sistema di navigazione contiene un banale baco. Il sistema di navigazione, progettato per l'Ariane 4, viene riciclato perchè dimostratosi affidabile. Il sistema di navigazione di Ariane 5 tenta di convertire la velocità laterale del missile dal formato a 64 bit al formato a 16 bit. Però l'Ariane 5 vola molto più velocemente dell'Ariane 4, per cui il valore della velocità laterale è più elevato di quanto possa essere gestito dalla routine di conversione, che a differenza di molte altre routine di conversione a bordo è priva dei normali controlli che evitano problemi di gestione.
  Risultato: overflow, spegnimento del sistema di guida, trasferimento del controllo al secondo sistema di guida, che però, essendo progettato allo stesso modo, va anch'esso in tilt. Privo di guida, il vettore si autodistrugge. Costo: 1 miliardo di euro.
- 1997: l'incrociatore lanciamissili USS Yorktown va in crash grazie a Windows NT per un *buffer overflow*...
- 2000: Millennium Bug!

- 2003: North America Blackout due to a *data race* in monitoring software.
- 2004: NASA's Spirit rover. Too many files in the flash memory.
- 2004: Sovraccarico di Mars Rover. Mentre si trova ad operare sulla superficie di Marte, il robot della NASA Mars Rover congela a causa dei troppi file aperti nella sua memoria flash!
- 2005: Toyota richiama 1600000 Prius. Spie sul cruscotto che si accendono senza ragione e sistemi di iniezione del carburante bloccati si rivelano essere la manifestazione non di un problema hardware ma di un baco software!
- 2007: Six F-22 Raptors. Multiple crashes at 180^o meridian.
- 2013: Toyota ETCS. Bugs could cause unintended acceleration.

Semantica formale

- Tutti questi disastri hanno un minimo comune denominatore
- I sistemi in questione sono molto complessi (ovviamente, relativamente al periodo in cui sono stati sviluppati: il computer di bordo dell'Apollo 11 aveva meno di 64Kb di RAM... Come il Commodore 64 degli anni 80!)
- La semantica dei linguaggi usati non era definita formalmente
- Senza una semantica formale non è possibile fare verifica formale dei codici
- Un banale errore, non opportunamente gestito, in una sola riga, può ripercuotersi disastrosamente su tutto il sistema.

Questo corso:

- Semantica operazionale per alcuni semplici linguaggi imperativi, funzionali e concorrenti
- metodi formali per provare proprietà di correttezza delle semantiche operazionali fornite
- sistemi di tipaggio e sottotipaggio per i suddetti linguaggi
- nozioni di equivalenze semantiche tra programmi.

Requisiti matematici

- Questo corso richiede alcuni concetti matematici di base
- La nozione di induzione matematica