




Verifica e validazione: analisi statica



Anno accademico 2017/18
Ingegneria del Software

Tullio Vardanega, tullio.vardanega@math.unipd.it

Laurea in Informatica, Università di Padova 1/28



Verifica e validazione: analisi statica

Premessa – 2

- ❑ La programmazione **non** deve ostacolare la verifica
 - Pochi linguaggi la facilitano attivamente
 - La disciplina del programmatore è fattore critico
- ❑ Occorre bilanciare l'aumento del potere espressivo con il costo di verifica
 - Funzionalità vs. integrità
- ❑ Fissato il linguaggio di programmazione, occorre scegliere quali costrutti usarne in funzione del loro impatto su quel bilancio

Laurea in Informatica, Università di Padova 3/28



Verifica e validazione: analisi statica

Premessa – 1

- ❑ Un SW di buona qualità deve possedere
 - Tutte le capacità **funzionali** specificate nei requisiti, che determinano cosa il sistema debba fare
 - Tutte le caratteristiche **non funzionali** necessarie per garantire che il sistema lavori sempre come previsto
- ❑ Ciò comporta il possesso **verificato** di determinate proprietà
 - Di costruzione (architettura, codifica, integrazione)
 - D'uso
 - Di funzionamento

Laurea in Informatica, Università di Padova 2/28



Verifica e validazione: analisi statica

Scrivere programmi verificabili – 1

- ❑ Serve dotarsi di uno standard di codifica coerente con le esigenze di verifica
 - Buone prassi attive e restrizioni sui costrutti inappropriati
- ❑ La verifica solo retrospettiva (a valle dello sviluppo) è **insufficiente** 
- ❑ Il costo di rilevazione e correzione di errori cresce con l'avanzare dello sviluppo

Laurea in Informatica, Università di Padova 4/28

Verifica e validazione: analisi statica

Costo di correzione di errori

Stage	Incremento di costo
Requirements	1
Design	5
Code	10
Unit Test	20
Acceptance Test	50
Maintenance	200

Laurea in Informatica, Università di Padova 5/28

Verifica e validazione: analisi statica



Scrivere programmi verificabili – 3

- ❑ **Regolamentare l'uso del linguaggio di programmazione tramite principi da riflettere nelle Norme di Progetto**
 - Per assicurare comportamento predicibile
 - Per usare criteri di programmazione ben fondati
 - Per ragioni pragmatiche
- ❑ **Vediamo ciascuna di queste tre dimensioni**

Laurea in Informatica, Università di Padova 7/28

Verifica e validazione: analisi statica

Scrivere programmi verificabili – 2

- ❑ **Posticipare la verifica in modalità *as-late-as-possible* segue l'approccio retrospettivo**
 - *Seeking correctness by correction* 
- ❑ **Accompagnare la produzione con la verifica costituisce un approccio costruttivo**
 - *Pursuing correctness by construction* 

The Great Divide
Technically Speaking

Laurea in Informatica, Università di Padova 6/28

Verifica e validazione: analisi statica

Comportamento predicibile

- ❑ **Codice sorgente senza ambiguità**
 - **Effetti laterali (p.es. di funzioni)**
 - Invocazioni della stessa funzione possono produrre risultati diversi
 - **Ordine di elaborazione e inizializzazione**
 - L'effetto del programma può dipendere dall'**ordine di elaborazione** delle sue parti
 - P.es., l'attivazione di *thread* in Java è fonte di imprevedibilità
 - **Modalità di passaggio dei parametri**
 - La scelta di una modalità di passaggio (per valore, per riferimento) può influenzare l'esito dell'esecuzione

Laurea in Informatica, Università di Padova 8/28

Verifica e validazione: analisi statica

Criteri di programmazione

- Riflettere l'architettura nel codice**
 - Usare programmazione strutturata per esprimere componenti, moduli, unità come da progettazione, e facilitare il riuso
- Separare le interfacce dall'implementazione**
 - Fissare bene le interfacce già a partire dall'architettura logica
 - Esporre le prime, nascondere la seconda
- Massimizzare l'incapsulazione (*information hiding*)**
 - Usare membri privati e metodi pubblici per l'accesso
- Usare tipi specializzati per specificare dati**
 - La composizione e la specializzazione aumentano il potere espressivo del sistema di tipi del programma

Laurea in Informatica, Università di Padova 9/28

Verifica e validazione: analisi statica

Tracciamento – 1

- Dimostrare completezza ed economicità della soluzione**
 - Soddisfacimento di tutti i requisiti
 - Nessuna funzionalità superflua
 - Nessun componente ingiustificato
- Ha luogo**
 - Su ogni passaggio dello sviluppo (ramo discendente)
 - Su ogni passaggio della verifica (ramo ascendente)
- Può essere altamente automatizzato**



Laurea in Informatica, Università di Padova 11/28

Verifica e validazione: analisi statica

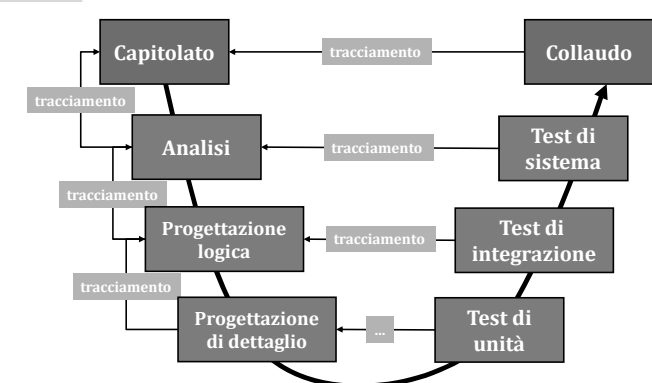
Considerazioni pragmatiche

- L'efficacia dei metodi di analisi è funzione della qualità di strutturazione del codice**
 - P.es: un modulo con un solo punto di ingresso (pre-) e un solo punto di uscita (post-) è più facilmente analizzabile per il suo effetto sullo stato
- La verifica di un programma relaciona frammenti di codice con frammenti di specifica**
 - La verificabilità è funzione inversa della dimensione del contesto
 - Conviene confinare gli ambiti (*scope*) e la visibilità
 - Una buona architettura facilita la verifica
 - Esempio: incapsulazione dello stato e controllo di accesso


Laurea in Informatica, Università di Padova 10/28

Verifica e validazione: analisi statica

Tracciamento – 2



Laurea in Informatica, Università di Padova 12/28

 Verifica e validazione: analisi statica

Tracciamento – 3

- ❑ Particolari stili di programmazione facilitano il tracciamento
 - Assegnare singoli requisiti elementari a singoli moduli del programma richiede una sola procedura di prova e facilita il tracciamento
- ❑ Maggiore l'astrazione (potenza espressiva) di un costruito maggiore la quantità di codice oggetto eseguito per esso e maggiore l'onere di dimostrazione di corrispondenza
 - Bassa astrazione: addizione tra interi
 - Alta astrazione: attivazione di *thread*

Laurea in Informatica, Università di Padova 13/28

 Verifica e validazione: analisi statica

Analisi di flusso di controllo

- ❑ Accertare che il codice esegua nella sequenza specificata
- ❑ Accertare che il codice sia ben strutturato
- ❑ Localizzare codice non raggiungibile
- ❑ Identificare segmenti d'esecuzione che possano non terminare
 - L'analisi dell'albero delle chiamate (*call-tree analysis*) mostra se l'ordine di chiamata corrisponda alla specifica e rileva la presenza di ricorsione diretta o indiretta
 - Divieto di modifica di variabili di controllo delle iterazioni


Laurea in Informatica, Università di Padova 15/28

 Verifica e validazione: analisi statica

Tipi di analisi statica

- A. Flusso di controllo
- B. Flusso dei dati
- C. Flusso dell'informazione
- D. Esecuzione simbolica
- E. Verifica formale del codice
- F. Verifica di limite
- G. Uso dello *stack*
- H. Comportamento temporale
- I. Interferenza
- J. Codice oggetto

Prima di e in aggiunta all'analisi dinamica



Laurea in Informatica, Università di Padova 14/28

 Verifica e validazione: analisi statica

Analisi di flusso dei dati

- ❑ Accertare che nessun cammino d'esecuzione del programma acceda a variabili prive di valore
 - Concentrando l'analisi di flusso di controllo sulla sequenza e le modalità di accesso alle variabili (lettura, scrittura)
- ❑ Rilevare possibili anomalie
 - Esempio: più scritture successive senza letture intermedie
- ❑ Attività complicata dalla presenza e dall'uso di dati globali raggiungibili da ogni parte del programma

Laurea in Informatica, Università di Padova 16/28




Verifica e validazione: analisi statica

Analisi di flusso d'informazione

- ❑ **Determinare quali dipendenze tra ingressi e uscite risultino dall'esecuzione di una unità di codice**
- ❑ **Le sole dipendenze consentite sono quelle previste dalla specifica**
 - Consente l'identificazione di effetti laterali inattesi o indesiderati
- ❑ **Può limitarsi a un singolo modulo oppure estendere a più moduli correlati oppure anche all'intero sistema**

Laurea in Informatica, Università di Padova17/28



Verifica e validazione: analisi statica

Esempio

Assumendo assenza di *aliasing* e di effetti laterali di funzioni

```
X = A+B; -- X dipende da A e B
Y = D-C; -- Y dipende da C e D
if (X>0)
  Z = Y+1; -- Z dipende da A, B, C e D
```

```
A + B ≤ 0 →
  X = A + B, Y = D - C, Z = Z
A + B > 0 →
  X = A + B, Y = D - C, Z = D - C + 1
```

Laurea in Informatica, Università di Padova19/28




Verifica e validazione: analisi statica

Esecuzione simbolica

- ❑ **Verificare proprietà del programma mediante manipolazione algebrica del codice sorgente**
 - Combinando tecniche di analisi di flusso di controllo, di flusso di dati e di flusso di informazione
- ❑ **Si esegue effettuando "sostituzioni inverse"**
 - Sostituendo a ogni LHS di un assegnamento il suo RHS
- ❑ **Trasformando così il flusso sequenziale del programma in un insieme di assegnamenti paralleli le cui uscite sono funzione degli ingressi**

Laurea in Informatica, Università di Padova18/28



Verifica e validazione: analisi statica

Verifica formale del codice

- ❑ **Provare la correttezza del codice sorgente rispetto alla specifica algebraica dei requisiti**
 - Esplorando tutte le esecuzioni possibili
 - Non fattibile tramite analisi dinamica
- ❑ **Correttezza parziale**
 - **Le condizioni di verifica sono espresse come teoremi la cui verità implica certe pre-condizioni in ingresso e certe post-condizioni in uscita**
 - La prova di correttezza vale sotto l'ipotesi di terminazione del programma
 - La prova di correttezza totale richiede prova di terminazione

Laurea in Informatica, Università di Padova20/28

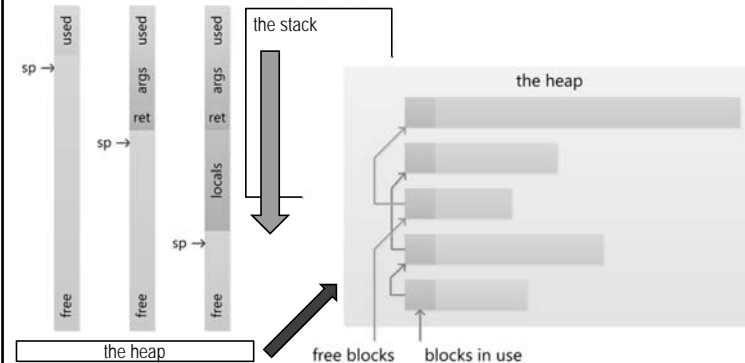


Analisi di limite

- ❑ **Verificare che i dati del programma restino entro i limiti del loro tipo e della precisione desiderata**
 - Analisi di *overflow* e *underflow*
 - Analisi di errori di arrotondamento
 - Rispetto dei limiti (*range checking*)
 - Analisi di limite di strutture
- ❑ **Linguaggi evoluti assegnano limiti statici a tipi discreti consentendo verifiche automatiche sulle corrispondenti variabili**
 - Più problematico con tipi enumerati e reali



Stack & heap



Analisi d'uso di *stack* - 1

- ❑ **Lo *stack* è l'area di memoria il programma usa per ospitare dati locali, temporanei, e indirizzi di ritorno generati dal compilatore alla chiamata di sottoprogrammi**
 - Ogni flusso di controllo (*thread*) ha il suo *stack*
 - La sua dimensione cresce con l'annidamento di chiamate di procedura
- ❑ **L'*heap* invece è l'area della memoria globale del programma**
 - La sua dimensione è fissata a configurazione
 - Il suo contenuto usato alla creazione dinamica di oggetti



Analisi d'uso di *stack* - 2

- ❑ **Determinare la massima domanda di *stack* richiesta da un'esecuzione in relazione con la dimensione dell'area di memoria assegnata al processo**
- ❑ **Verificare che non vi sia rischio di collisione tra *stack* e *heap* per qualche esecuzione**




Verifica e validazione: analisi statica

Analisi temporale

- ❑ **Studiare le proprietà temporali richieste ed esibite dalle dipendenze delle uscite dagli ingressi del programma**
 - Sapere di produrre il valore giusto quando atteso
- ❑ **Limiti espressivi dei linguaggi e delle tecniche di programmazione complicano questa analisi**
 - Iterazioni prive di limite statico (*while*), ricorso sistematico a strutture dati dinamiche (*new*), ...

Laurea in Informatica, Università di Padova25/28




Verifica e validazione: analisi statica

Analisi di codice oggetto

- ❑ **Assicurare che il codice oggetto da eseguire sia una traduzione corretta del codice sorgente corrispondente e che nessun errore od omissione siano stati introdotti dal compilatore**
 - Viene ancora effettuata manualmente
 - Viene facilitata dalle informazioni di corrispondenza prodotte dal compilatore

Laurea in Informatica, Università di Padova27/28



Verifica e validazione: analisi statica

Analisi d'interferenza


- ❑ **Mostrare l'assenza di effetti di interferenza tra parti isolate ("partizioni") del sistema**
 - Non necessariamente limitate a componenti SW
- ❑ **Veicoli tipici di interferenza**
 - Memoria (virtualmente) condivisa, dove parti separate di programma lasciano traccia di dati abbandonati ma non distrutti
 - Fenomeno noto come *memory leak*
 - Azzeramento delle pagine di memoria prima del riuso (p.es. NT v5.x)
 - I/O e altri dispositivi programmabili con effetti a livello sistema (p.es., DMA)

Laurea in Informatica, Università di Padova26/28



Verifica e validazione: analisi statica

Analizzabilità del sistema

- ❑ **L'analisi statica costruisce modelli astratti del programma**
 - Rappresentandolo come un grafo diretto per studiare i cammini possibili in esso
 - Le transizioni tra stati (gli archi) hanno etichette che descrivono proprietà sintattiche o semantiche dell'istruzione corrispondente
 - La presenza di flussi di eccezione e di risoluzione dinamica di chiamata (*dynamic dispatching*) complica la struttura del grafo
- ❑ **Ciascun flusso di controllo (*thread*) viene analizzato individualmente**
 - MA questo richiede assenza di interferenza tra loro! 

Laurea in Informatica, Università di Padova28/28