Analisi del Software

Alessio Gjergji

Indice

Capitolo 1

Analisi

L'analisi può essere costruito dal codice o può essere basata dul modello Il `CFG` è fedele per l'analisi statica, ma può essere distante dal sistema, dipende dalla costruzione del modello.

- Analisi Statica (*senza esecuzione del codice*): dicidibile, il prezzo da pagare è la precisione.
- Analisi Dinamica (*basata sull'esecuzione del codice*): visione ortogonale ricadendo nelle problematiche di non terminazione, non posso testare tutti gli input, nel dinamico abbiamo certezze se abbiamo risposte, ma non siamo esaustivi sull'insieme. Se pensiamo al testing fissiamo un range di risposte e siamo sicuri solamente sui test effettuati.

Il limite è la non decidibilità (*teorema di Rice*), per superarla si paga il prezzo della precisione.

Abbiamo bisogno di un linguaggio Turing completo. I programmi analizzabili sono tutti programmi nel linguaggio $\mathcal{L}$ Lanciando un programma p, l'esecuzione termina se raggiungo la fine di p in un tempo finito.

Ideale: Analizzare che $\forall$programma il $\mathcal{L}$ analizza in modo automatico la proprietà di interesse (*proprietà semantiche, quindi del comportamento, e quindi non decidibili per il teorema di Rice*) π dando risposta certa per ogni input in un tempo finito.

Per ottenere un'analisi possiamo rinunciare a qualcosa eliminando la completa automazione, utilizzare un analizzatore su una classe limitata di programmi e accettare un'imprecisione (*controllata, astraendo il risultato, ma con una risposta vera*).

Togliendo l'automazione tolgo la garanzia della non presenza di errori. L'accettare imprecisione invece è completamente automatizzabile, ed è presente nell'analisi statica.

1.1 Approssimazione

Per approssimazione intendiamo che se la risposta non è accurata comunque è accettabile, quindi:

$$\text{inaccuratezza} \neq \text{sbagliato}$$

Quindi è fondamentale che l'errore sia riconosciuto e quindi caratterizzabile.

π proprietà di analizzare, p programma, quindi Analisi$_\pi(p)$.

$$\forall p \in \mathcal{L} \qquad Analisi_\pi(p) = true \Leftrightarrow p\, soddisfa\, \pi$$

Ogni programma quindi soddisfa o non verifica la proprietà.

purtroppo ciò non è sempre possibile, rinunciando alla bi-implicazione.

Soundness (*Correttezza*)

$$Analisi_\pi(p) = true \Rightarrow p\, soddisfa\, \pi$$

Se $Analisi_\pi(p) = false$ l'analizzatore sovrastima i programmi che non soddisfano la proprietà, includendo programmi che in realtà potrebbero soddisfarla.

Completezza

$$Analisi_\pi(p) = true \Leftarrow p\, soddisfa\, \pi$$

quindi

$$p\, soddisfa\, \pi \Rightarrow Analisi_\pi(p) = true$$

Se l'analisi di p sulla proprietà π è vera potrei avere dei falsi positivi.

Supponiamo di avere un programma $p \in \mathcal{L}$, dobbiamo capire se pi vale su p. In generale non è possibile un'analisi diretta su p. Trasformiamo il nostro codice in un modello su cui possiamo utilizzare degli strumenti automatici su cui ricavare informazione ($|CFG|$, *automi, ...*). Sui modelli abbiamo tecniche algoritmiche decidibili per determinare se π' vale su p, dove π' è la π riportata sul modello.

La perdita di precisione avviene quando la risposta certa ottenuta dal modello avviene durante il passaggio dalla risposta ottenuta dal modello alla risposta ottenuta sul programma originale. La precisione si perde dall'implicazione da:

$$\pi' \text{ vale sul modello di } p \Longrightarrow \pi \text{ vale su } p$$

1.1.1 Statico dinamico

Supponiamo di avere un programma e la nostra specifica π. nell'analisi statica analizziamo la proprietà π sul codice aperto. L'analisi dinamica prende in input e analizza la relazione tra input e output, in questo codice può non conoscere il codice (*testing*).

1.1.2 Caratteristiche di rinuncia dell'analisi

Le caratteristiche a cui rinunciare per superare Rice sono:

- Automatico:
- $\forall p \in \mathcal{L}$:
- Corretto:
- Completo:

L'analisi può essere quindi dinamica e statica. In funzione a ciò che rinunciamo abbiamo varie classi di analisi:

- Verifica (*Model checking*): garantire che un programma si comporti rispettando le specifiche.
- Analisi Conservative (*Statiche*): cercano i estrarre informazione in modo statico sul programma, danno quindi una semantica approssimata ma conservativa del programma, la semantica approssimata implica la semantica concreta.
- Bug finding (*Debugging*): ricerca di cause di errore.
- Testing: opera su un sottoinsieme di input.

Model checking

Il model checking rinuncia l'appartenenza di ogni programma, prendendo in considerazione insiemi finiti, rimane una tecnica automatica, ed è corretto e completo sul modello.

Analisi conservative statiche

Le analisi conservative si basano su concetto di approssimazione, rinunciando quindi alla completezza. Riesco a dare proprietà approssimate attraverso la sovrastima del dubbio della risposta. Si tratta di una tecnica automatica, in generale lavora su programmi rappresentabili finitamente ed è corretta, ma non completa (*lavorando solo su specifiche proprietà*). L'analizzatore è costruito su proprietà fissate e per ogni programma da delle risposte. Le proprietà sono infinite, perciò non posso analizzare tutte le proprietà.

Debugging

Con debugging abbiamo una tecnica di supporto dello sviluppatore, le risposte date possono avere sia perdita di correttezza che di completezza.

Testing

Il testing è una tecnica dinamica di esecuzione del programma su una selezione di input. Non ha limitazioni di programmi, esso rinuncia alla correttezza, intesa come insieme finito di input. Quando un test viola la proprietà allora possiamo dire che la proprietà non la soddisfa.