Verificarea programelor prin model-checking

7 noiembrie 2013

Scopul verificării

demonstrarea că programul e *corect*

(dacă metoda și programul permite)

găsirea de erori

metode dedicate doar *găsirii de erori*

sau ca efect lateral, la eșecul demonstrării corectitudinii

Metode de verificare

statică = fără execuția codului

- detecție de tipare
- *analiză de flux de date*
- *verificare formală*

dinamică = cu execuția codului

- instrumentare / rulare pe mașină virtuală
- execuție simbolică

Ce încredere am în rezultat ?

Metoda e:
consistentă ? (engl. *sound*) = orice răspuns e bun ?
completă ? = găsește toate răspunsurile ?

Verificare:
consistent: un sistem raportat corect e corect
complet: poate determina corectitudinea oricărui sistem

Detecție de erori:
consistent: o eroare raportată e reală
complet: găsește toate erorile

Verificarea formală

Sistemul e modelat matematic
⇒sunt posibile rezultate *garantate* (certificate)
în limitele posibilităților//presupunerilor de modelare

Demonstrare de teoreme
condiții de verificare (din reguli Hoare)
demonstrații (sau algoritmi de realizabilitate a formulelor) (SAT-solvers)
probleme: necesar de anotații pentru formule complexe
interacțiune intensă cu expertul uman

Model checking
sistem = automat cu stări finite
algoritm = explorarea spațiului stărilor (traversare de grafuri)
automat; dă contraexemplu în caz de eroare
problema: explozia spațiului stărilor

Model checking software in practică

Proiectul SLAM [Microsoft Research] (cu începere din 2000)
(Software (Specifications), Languages, Analysis and Model checking)
ulterior, multe altele: BLAST (UC Berkeley), CBMC (Oxford), ...

Scopul: verificarea unor proprietăți de siguranță (invarianți)
concret: un program respectă regulile de utilizare API
(ex: apelurile la `lock()` și `unlock()` alternează

– concentrat mai ales pe descoperirea erorilor de *interfață*
– utilizat practic pentru device drivers în Windows NT/XP

Caracteristici:
– nu necesită anotarea programului de către utilizator
(doar specificarea unor reguli sub formă de automat - monitor)
– verificarea e făcută automat, pentru *toate* execuțiile posibile;
– se generează un contraexemplu (execuție concretă) în caz de eroare

Exemplu de program

```
do {      // Fragment de device driver [Ball & Rajamani '01]
  KeAcquireSpinLock(&devExt->writeListLock);
  nPacketsOld = nPackets;
  request = devExt->WriteListHeadVa;
  if(request && request->status) {
    devExt->WriteListHeadVa = request->Next;
    KeReleaseSpinLock(&devExt->writeListLock);
    irp = request->irp;
    if (request->status > 0) {
      irp->IoStatus.Status = STATUS_SUCCESS;
      irp->IoStatus.Information = request->Status;
    } else {
      irp->IoStatus.Status = STATUS_UNSUCCESSFUL;
      irp->IoStatus.Information = request->Status;
    }
    SmartDevFreeBlock(request);
    IoCompleteRequest(irp, IO_NO_INCREMENT);
    nPackets++;
  }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock(&devExt->writeListLock);
```

Specificarea proprietăților

```
state {
  enum { Unlocked=0, Locked=1 }
    state = Unlocked;
}
KeAcquireSpinLock.return {
  if (state == Locked) abort;
  else state = Locked;
}
KeReleaseSpinLock.return {
  if (state == Unlocked) abort;
  else state = Unlocked;
}
```

Specificarea este tradusă în C și programul original este instrumentat

(programul original corect $\Leftrightarrow$ programul instrumentat nu atinge eroare)

Abstracția în verificare

– programul în general poate fi foarte complex
– potențial, multe instrucțiuni din program nu sunt relevante pentru proprietatea dorită
– am dori să ne concentrăm asupra porțiunii relevante din program
Tehnică: *program slicing* – determinarea fragmentului (slice) de program care afectează o anumită proprietate (slicing criterion) a programului (ex. valoarea unei variabile într-un punct)

Noțiune mai generală: *abstracția* = generarea unui model (program) simplificat, prin a cărui analiză deducem proprietăți ale programului original
predicat = condiție booleană (expresie din variabilele programului)

Generarea programului boolean

Pornește de la predicatele din specificație

nondeterminism în control; `skip` pentru instrucțiuni irelevante;

```
do {
A: KeAcquireSpinLock_return();
  skip;
  if(*) {
B:   KeReleaseSpinLock_return();
    if (*) {
      skip;
    } else {
      skip;
    }
  }
} while (*);
C: KeReleaseSpinLock_return();
```

Analiza programului boolean (model checking)

Se calculează mulțimea stărilor atinse:

stare = atribuire pentru variabilele din domeniul de vizibilitate
mulțime de stări = funcție booleană, reprezentată eficient
(diagrame de decizie binare, BDD)
calculul cu mulțimi de stări: capturează corelările între variabile
relație de tranziție: tot o formulă booleană

$$state = 0 \wedge state' = 1$$

Pentru exemplul dat: se semnalează eroare, se poate parcurge
`A: KeAcquireSpinLock()` de două ori succesiv

Contraexemplul și generarea noilor predicate

Se stabilește dacă contraexemplul în programul abstract reprezintă un contraexemplu în programul concret.

(execuție simbolică înainte pe calea contraexemplului).

Dacă contraexemplul e fezabil, reprezintă o eroare reală.

Dacă contraexemplul nu e fezabil:

se pleacă înapoi din starea de eroare în programul abstract

cu reguli Hoare / wp Dijkstra se parcurge calea de eroare înapoi (în programul abstract),
acumulând constrângerile deduse până găsește o inconsistență
$\Rightarrow$ se caută una minimală, se generează predicate corespunzătoare.

In exemplu: `nPacketsOld = nPackets` și `nPacketsOld != nPackets`
procedurile de decizie sunt incomplete $\Rightarrow$ poate returna "nu știu"

Se regenerează programul boolean cu noile predicate și se reia verificarea.

Tehnica: *counterexample-guided abstraction refinement*

Al doilea program boolean

```
do {
A: KeAcquireSpinLock_return();
  b = T;      /* b == (nPackets == nPacketsOld) */
  if(*) {
B:   KeReleaseSpinLock_return();
    if (*) {
      skip;
    } else {
      skip;
    }
    b := choose(F, b);      /* choose(p1, p2) == p1 ? T : p2 ? F :
nondet */
  }
} while (!b);
C: KeReleaseSpinLock_return();
```

Abstracția obținută este suficientă pentru a demonstra corectitudinea.