

Symbolisk Felinjerering

Daniel Larsson

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola

CEDES-seminarium
4 maj, 2006

Symbolisk felinjecering (SFI)

Inte en etablerad metod.

Pågående forskning inom ramen för CEDES

Syfte: Utvärdera tillförlitligheten hos säkerhetskritiska datorsystem

Innehåll

- Feltolerans
- Felinjicering
- Brister med konventionell felinjicering
- Symbolisk exekvering & Formell verifiering
- Symbolisk felinjicering
- Fallstudie: CRC
- Begränsningar

Feltolerans

Omöjligt garantera att datorsystem är felfritt

Även vid användning av bästa tillgängliga tekniker för

- design av hårdvara/mjukvara
- tillverkning av hårdvarukomponenter
- testning
- ...

kan systemet fortfarande innehålla defekter.

Dessutom: Kan inte garantera frånvaron av tillfälliga felorsaker:

- elektromagnetisk interferens
- joniserande strålning
- ...

Feltolerans

Hur åstadkomma feltolerans?

- Olika former av redundans, d v s extra hård- och/eller mjukvara som används för att bygga felhanteringsmekanismer:
 - feldetektering
 - felåterhämtning
 - felmaskering
- Detta seminarium handlar om:
Ny metod för att utvärdera felhanteringsmekanismer

Feltolerans

- Feltolerans kan åstadkommas med hårdvara eller mjukvara
- Om möjligt är mjukvarulösningar ofta att föredra:
 - Slipper dyr hårdvaruredundans
 - Standardkomponenter kan användas
 - Mer flexibelt
- Specialfall—mjukvara för att tolerera fel i hårdvara:
Mjukvaruimplementerad hårdvarufeltolerans
- *Eng.:* SIHFT (Software-Impl. Hardware Fault Tolerance)
- **Symbolisk felinjicering applicerbar på mjukvaruimplementerad hårdvarufeltolerans**

Felinjicering

En samling tekniker för att

- avsiktligt introducera fel i ett datorsystem
- undersöka systemets beteende i närvaron av dessa fel

Hårdvaruimplementerad felinjicering. Systemhårdvaran utsätts för

- strålning av tunga joner
- elektromagnetisk interferens
- ...

Mjukvaruimplementerad felinjicering

- T ex GOOFI, utvecklad vid Chalmers. GOOFI använder sig av debug-hårdvara i processorn för att simulera hårdvarufel

Felinjicering

Felinjiceringstekniker baseras på en *felmodell* som specificerar vilka typer av fel som injiceras. Exempel:

- Hårdvarufel
 - enkla bit-flippar
 - multipla bit-flippar
 - “stuck-at” -fel
 - ...
- Mjukvarufel
 - mutationer: syntaktiskt korrekta ändringar i originalmjukvaran
- Felmodell för symbolisk felinjicering:
enkla och multipla bit-flippar

Brister med konventionell felinjicering

Experimentell utvärdering

- Omöjligt åstadkomma 100% täckning
 - Exempel: Undersök konsekvenser för alla tillstånd som resulterar från 3 bit-flippar i 32-bitars minnescell
 - \Rightarrow 4960 testfall
 - Dessutom: De resultat man får gäller bara *en* viss indata
- Stor andel av injicerade fel aktiveras inte. Injiceras i
 - delar av minnet som inte används
 - register som skrivs över innan det läses
- Tidskrävande analys av stora mängder “övervaknings”-data

Hur kan symbolisk felinjecering hjälpa?

Kan *komplettera*, inte ersätta konventionella metoder

- Analytisk metod (inte experimentell)
- Bättre täckning: Givet väldefinierat problem möjligt med 100% täckning
 - symboliska fel
 - symbolisk indata
- Fler fel aktiveras
- Kan kombineras med formell *verifiering*

Symbolisk felinjecering baseras på *symbolisk exekvering*

Symbolisk exekvering & Formell verifiering

Symbolisk exekvering

- Exekveringen simuleras
- Indata: symboler/variabelnamn istället för konkreta värden
- Utdata: Logiska uttryck över dessa symboler
- Varför: Undersöka programmets beteende oberoende av indata
- Symbolisk exekvering används i vissa tekniker av *formell verifiering*. KeY är ett exempel på ett verifieringsverktyg som använder symbolisk exekvering

Symbolisk exekvering & Formell verifiering

- Formell verifiering =
Skapa formellt bevis för att ett program har vissa egenskaper
- Egenskaperna uttrycks i *formell specifikation*,
t ex i form av ett postcondition
- Program & specifikation ger en s.k. *proof obligation*:
 $\langle p \rangle \phi$

Symbolisk exekvering & Formell verifiering

Exempelprogram:

```
int m(int x) {  
    int r;  
    if (x >= 0) {r = x;}  
    else {r = 0;}  
    return r;  
}
```

Egenskap vi vill bevisa:

m returnerar alltid ett icke-negativt värde

Symbolisk exekvering & Formell verifiering

Regler för symbolisk exekvering:

$$\textit{if} \frac{b = \textit{true} \vdash \langle s1 \ \omega \rangle \phi \quad b = \textit{false} \vdash \langle s2 \ \omega \rangle \phi}{\vdash \langle \textit{if}(b) \ \{s1\} \ \textit{else} \ \{s2\} \ \omega \rangle \phi}$$

$$\textit{assignment} \frac{\vdash \{v:=e\} \langle \omega \rangle \phi}{\vdash \langle v = e; \ \omega \rangle \phi}$$

Symbolisk exekvering & Formell verifisering

$$\frac{}{\vdash \langle \text{if}(x \geq 0) \{ r = x; \} \text{else} \{ r = 0; \} \text{return } r; \rangle \text{res} \geq 0}$$

Symbolisk exekvering & Formell verifisering

$$\frac{x \geq 0 \vdash \langle r=x; \text{return } r; \rangle \text{res} \geq 0 \quad x < 0 \vdash \langle r=0; \text{return } r; \rangle \text{res} \geq 0}{\vdash \langle \text{if}(x \geq 0) \{ r=x; \} \text{else} \{ r=0; \} \text{return } r; \rangle \text{res} \geq 0}$$

Symbolisk exekvering & Formell verifisering

$$\frac{x \geq 0 \vdash \{r:=x\} \langle \text{return } r; \rangle \text{res} \geq 0 \quad x < 0 \vdash \{r:=0\} \langle \text{return } r; \rangle \text{res} \geq 0}{x \geq 0 \vdash \langle r:=x; \text{return } r; \rangle \text{res} \geq 0 \quad x < 0 \vdash \langle r:=0; \text{return } r; \rangle \text{res} \geq 0} \vdash \langle \text{if}(x \geq 0) \{r:=x; \} \text{else} \{r:=0; \} \text{return } r; \rangle \text{res} \geq 0$$

Symbolisk exekvering & Formell verifisering

$$\frac{x \geq 0 \vdash \{r:=x\}\{res:=r\}\langle \rangle res \geq 0}{x \geq 0 \vdash \{r:=x\}\langle return r; \rangle res \geq 0} \quad \frac{x < 0 \vdash \{r:=0\}\{res:=r\}\langle \rangle res \geq 0}{x < 0 \vdash \{r:=0\}\langle return r; \rangle res \geq 0}$$
$$\frac{x \geq 0 \vdash \langle r:=x; return r; \rangle res \geq 0 \quad x < 0 \vdash \langle r:=0; return r; \rangle res \geq 0}{\vdash \langle if(x \geq 0)\{r:=x;\}else\{r:=0;\} return r; \rangle res \geq 0}$$

Symbolisk exekvering & Formell verifisering

$$\frac{x \geq 0 \vdash \{res:=x\} \langle \rangle res \geq 0}{x \geq 0 \vdash \{r:=x\} \{res:=r\} \langle \rangle res \geq 0} \quad \frac{x < 0 \vdash \{res:=0\} \langle \rangle res \geq 0}{x < 0 \vdash \{r:=0\} \{res:=r\} \langle \rangle res \geq 0}$$
$$\frac{x \geq 0 \vdash \{r:=x\} \langle return r; \rangle res \geq 0}{x \geq 0 \vdash \langle r=x; return r; \rangle res \geq 0} \quad \frac{x < 0 \vdash \{r:=0\} \langle return r; \rangle res \geq 0}{x < 0 \vdash \langle r=0; return r; \rangle res \geq 0}$$
$$\frac{}{\vdash \langle if(x \geq 0) \{r=x; \} else \{r=0; \} return r; \rangle res \geq 0}$$

Symbolisk exekvering & Formell verifierring

$$\frac{\frac{\frac{x \geq 0 \vdash x \geq 0}{x \geq 0 \vdash \{res:=x\}\langle\rangle res \geq 0}}{x \geq 0 \vdash \{r:=x\}\{res:=r\}\langle\rangle res \geq 0}}{x \geq 0 \vdash \{r:=x\}\langle return r; \rangle res \geq 0}}{x \geq 0 \vdash \langle r:=x; return r; \rangle res \geq 0} \quad \frac{\frac{\frac{x < 0 \vdash 0 \geq 0}{x < 0 \vdash \{res:=0\}\langle\rangle res \geq 0}}{x < 0 \vdash \{r:=0\}\{res:=r\}\langle\rangle res \geq 0}}{x < 0 \vdash \{r:=0\}\langle return r; \rangle res \geq 0}}{x < 0 \vdash \langle r:=0; return r; \rangle res \geq 0}$$
$$\vdash \langle \text{if}(x \geq 0) \{r:=x;\} \text{else} \{r:=0;\} \text{return } r; \rangle res \geq 0$$

Symbolisk exekvering & Formell verifierring

$$\frac{\frac{\frac{[\vdash \text{true}]}{x \geq 0 \vdash x \geq 0}}{x \geq 0 \vdash \{res:=x\}\langle \rangle res \geq 0}}{x \geq 0 \vdash \{r:=x\}\{res:=r\}\langle \rangle res \geq 0}}{x \geq 0 \vdash \{r:=x\}\langle \text{return } r; \rangle res \geq 0}}{x \geq 0 \vdash \langle r:=x; \text{return } r; \rangle res \geq 0} \quad \frac{\frac{\frac{[\vdash \text{true}]}{x < 0 \vdash 0 \geq 0}}{x < 0 \vdash \{res:=0\}\langle \rangle res \geq 0}}{x < 0 \vdash \{r:=0\}\{res:=r\}\langle \rangle res \geq 0}}{x < 0 \vdash \{r:=0\}\langle \text{return } r; \rangle res \geq 0}}{x < 0 \vdash \langle r:=0; \text{return } r; \rangle res \geq 0}$$
$$\frac{}{\vdash \langle \text{if}(x \geq 0) \{r:=x; \} \text{else} \{r:=0; \} \text{return } r; \rangle res \geq 0}$$

Symbolisk felinjicering

Ide:

- Injicera *symboliska* fel (repr. hela klasser av “verkliga” fel) under symbolisk exekvering
- Fortsatt exekvering visar konsekvenser av dessa fel

Ett steg längre:

- Kombinera denna teknik med formell verifiering
- Möjligt bevisa att program har vissa egenskaper även i närvaro av fel
- Möjligt bevisa att ett program är tolerant mot vissa typer av fel

Symbolisk felinjecering

Vad karakteriserar symbolisk felinjecering?

- Simuleringsbaserad
- Appliceras på källkods nivå
- Felmodell: Enkla och multipla bit-flippar
- Fellokalisering: Dataarean av minnet

Symbolisk felinjicering

Hur fungerar symbolisk felinjicering?

- Källkoden instrumenteras med pseudo-instruktioner `inject(location);`
- Mekanismen för symbolisk exekvering utökas med lämpliga regler för dessa `inject` -instruktioner

Regler för `inject`:

$$\text{boolean} \frac{\vdash \{b:=\text{true}\}\langle\omega\rangle\phi \quad \vdash \{b:=\text{false}\}\langle\omega\rangle\phi}{\vdash \langle\text{inject}(b); \omega\rangle\phi}$$

$$\text{int} \frac{\vdash \forall j : \text{int}. \{i := j\}\langle\omega\rangle\phi}{\vdash \langle\text{inject}(i); \omega\rangle\phi}$$

Symbolisk felinjecering

Exempelprogram:

```
int m(int x) {  
    int r;  
    if (x >= 0) {r = x;}  
    else {r = 0;}  
    inject(r);  
    return r;  
}
```

Egenskap vi vill bevisa:

m returnerar alltid ett icke-negativt värde

Symbolisk felinjicering

$$\vdash \langle \text{if}(x \geq 0) \{ r = x; \} \text{else} \{ r = 0; \} \text{ inject}(r); \text{return } r; \rangle \text{result} \geq 0$$

Symbolisk felinjicering

$$\begin{aligned} & x \geq 0 \vdash \langle r=x; \text{inject}(r); \text{return } r; \rangle \text{result} \geq 0 \\ & \vdash \langle \text{if}(x \geq 0) \{ r=x; \} \text{else} \{ r=0; \} \text{inject}(r); \text{return } r; \rangle \text{result} \geq 0 \end{aligned}$$

Symbolisk felinjicering

$$\frac{x \geq 0 \vdash \{r:=x\} \langle \text{inject}(r); \text{return } r; \rangle \text{result} \geq 0}{x \geq 0 \vdash \langle r:=x; \text{inject}(r); \text{return } r; \rangle \text{result} \geq 0}$$
$$\vdash \langle \text{if}(x \geq 0) \{r:=x; \} \text{else} \{r=0; \} \text{inject}(r); \text{return } r; \rangle \text{result} \geq 0$$

Symbolisk felinjicering

$$\frac{x \geq 0 \vdash \forall j : int. \{r:=x\} \{r:=j\} \langle \text{return } r; \rangle \text{result} \geq 0}{x \geq 0 \vdash \{r:=x\} \langle \text{inject}(r); \text{return } r; \rangle \text{result} \geq 0}$$
$$\frac{x \geq 0 \vdash \langle r:=x; \text{inject}(r); \text{return } r; \rangle \text{result} \geq 0}{\vdash \langle \text{if}(x \geq 0) \{r:=x; \} \text{else} \{r:=0; \} \text{inject}(r); \text{return } r; \rangle \text{result} \geq 0}$$

Symbolisk felinjicering

$$\frac{x \geq 0 \vdash \forall j : int. j \geq 0}{x \geq 0 \vdash \forall j : int. \{result:=j\} \langle \rangle result \geq 0}$$
$$\frac{x \geq 0 \vdash \forall j : int. \{r:=j\} \{result:=r\} \langle \rangle result \geq 0}{x \geq 0 \vdash \forall j : int. \{r:=j\} \langle return r; \rangle result \geq 0}$$
$$\frac{x \geq 0 \vdash \forall j : int. \{r:=x\} \{r:=j\} \langle return r; \rangle result \geq 0}{x \geq 0 \vdash \{r:=x\} \langle inject(r); return r; \rangle result \geq 0}$$
$$\frac{x \geq 0 \vdash \{r:=x\} \langle inject(r); return r; \rangle result \geq 0}{x \geq 0 \vdash \langle r=x; inject(r); return r; \rangle result \geq 0}$$
$$\vdots$$
$$\vdash \langle \text{if}(x \geq 0) \{r=x; \} \text{else} \{r=0; \} inject(r); return r; \rangle result \geq 0$$

Fallstudie: CRC

- Fel detekteringsmekanism CRC (Cyclic Redundancy Check)
- Checksumme-algoritm
 - Sändning: Beräkna checksumma på data, skicka data+checksumma
 - Mottagning: Beräkna checksumma på data. Jmf med medskickad checksumma
 - Överensstämmer *inte*: Datan har modifierats
 - Överensstämmer: Sannolikt ingen modifiering. Dock ingen garanti för detta.
- En bra checksumme-algoritm minimerar risken att flera fel “tar ut varandra” m a p checksumman

Fallstudie: CRC

Kort beskrivning av CRC-algoritmen

- Behandlar datablock som binär repr. av ett heltal
- Heltalet delas med en förutbestämd nämnare
- Resten (remainder) av divisionen utgör checksumman
- Polynomdivision istället för “vanlig” division

Implementering av CRC

- Checksumman kan inte beräknas i ett steg
- Datablocket matas successivt genom register medan diverse aritmetiska operationer appliceras på dess innehåll

Fallstudie: CRC

```
public static byte compute(byte[] buffer) {
    int count = buffer.length;
    byte register = (byte)0x0;
    while (count > 0) {
        byte element = buffer[buffer.length - count];
        int t = ((int)(register ^ element) & 0xff);
        register <<= 8;
        register ^= table[t];
        count--;
    }
    return register;
}
```

Fallstudie: CRC

- Teori bakom CRC välkänd—varför intressant bevisa något om den?
- Vet inte om just denna *implementering* av CRC är buggfri

Fallstudie: CRC

- Bevisförsök: CRC-implementeringen detekterar alla enkla bit-flippar
- Ännu inte bevisat detta för godtycklig storlek på datablocket. Kräver icke-trivialt induktionsbevis
- Har bevisat detta för arraystorlek på 5 element
- Metod:
 - Skapade program där checksumma beräknas 2 ggr för godt. datablock (av nämnd storlek), en med och en utan symbolisk felinjicering. Checksummorna jämförs och resultatet av jämförelsen returneras.
 - Skapade formell specifikation (pre- & postcondition) som uttrycker att programmet alltid ska returnera *true*

Fallstudie: CRC

Program:

```
static boolean crcTest(byte[] msg) {  
    byte crc1 = Crc.compute(msg);  
    injectSingle(msg[0]);  
    byte crc2 = Crc.compute(msg);  
    return (crc1 != crc2);  
}
```

Proof obligation:

$$\forall \text{byte}[] \text{msg}; (\text{msg.length} \doteq 5$$
$$\rightarrow \{ \text{msg} := \text{msg} \} \langle \text{Crc.crcTest}(\text{msg}); \rangle (\text{result} \doteq \text{true}))$$

Begränsningar

Vår teknik kan inte ...

- simulera fel i specifika delar av hårdvaran
- utvärdera realtidsegenskaper
- hantera flyttal
- hantera program med multipla trådar/processer

Sammanfattning

Symbolisk felinjicering

- Metod för att utvärdera tillförlitligheten hos säkerhetskritiska datorsystem
- Baseras på symbolisk exekvering av källkod
- Kan komplettera konventionella metoder
 - Ger bättre täckning
 - Fler injicerade fel aktiveras
 - Kan kombineras med formell verifiering
- Fallstudie: implementering av CRC-algoritmen bevisades detektera alla möjliga enkla bit-flippar i datablock av fix storlek

Frågor?