Szoftverellenőrzési technikák

# Kód alapú tesztelés
## Tesztgenerálás struktúra alapján

Majzik István, Micskei Zoltán

http://www.inf.mit.bme.hu/

# Tartalom

- Tesztelési alapok
- Modell alapú tesztelés
  - Tesztesetek származtatása modellből
- **Kód alapú tesztelés**
  - **Tesztesetek származtatása (forrás)kódból**

# Ötlet

- Adott egy program (bináris vagy forráskód)
- Generáljunk hozzá teszteket!
- Valamilyen lefedettség alapján

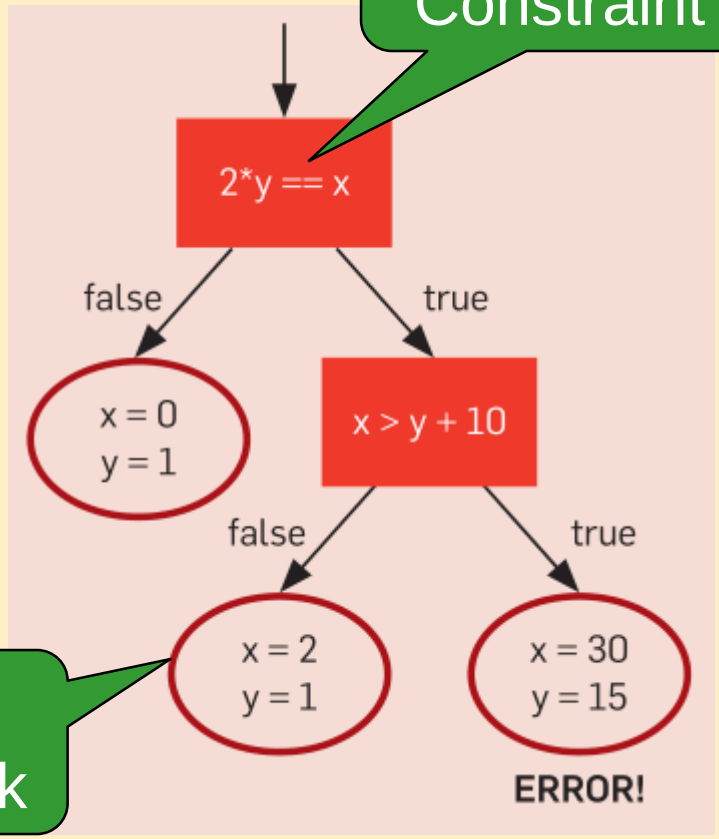
# MÓDSZEREK

# Módszerek

- **Szimbolikus végrehajtás (symbolic execution)**

# Példa

```
void testme (int x, int y) {
  z = 2 * y;
  if (z == x) {
    if (x > y+10) {
      ERROR;
    }
  }
}
```




Forrás: C. Cadar, K. Sen. Symbolic execution for software testing: three decades later, 2013

# Mit hagytunk ki?

- Honnan lesznek elvárt kimenetek a bemenethez?

- Mit lehet tenni:
  - Alap, általános hibák (kivételt dob, segfault…)
  - Kódban lévő assert() hívások megsértése
  - Kézzel határozzuk meg az elvárt eredményt
  - Meglévő kimenet felhasználása
    - Regressziós teszt: későbbi változatokkal összehasonlítás
    - Más implementációk összehasonlítása

# Szimbolikus végrehajtás (SE): alapötlet

- Programanalízis technika a ’70-es évekből
- Felhasználás tesztgenerálásra:
  - Szimbolikus változók
  - Utakhoz tartozó kényszerek összegyűjtése
  - Kényszerek megoldása (pl. SMT-megoldó)
  - Megoldások lesznek a tesztek
- 2000-es évek:
  - Elég erős gépek, jó SMT-megoldók
  - Sok ötlet és új eszköz

# „Sima” szimbolikus végrehajtás javítása

- SE sok helyen elakadhat
- Ilyenkor váltsunk át konkrét végrehajtásra
  - Véletlenszerűen választott bemenetekkel futtatás
  - Bejárt utak és feltételek rögzítése
  - Ez segít bizonyos kényszereket megoldani
- Elnevezés:
  - Dynamic Symbolic Execution
  - Concolic testing: *conc*erete + symb*olic*

# Eszközök

| Név | Platform | Bemenet | Egyéb |
|---|---|---|---|
| KLEE | Linux | C (LLVM bitcode) | Nyílt forrású |
| PEX | Windows | .NET assembly | VS Power Tool |
| SAGE | Windows | x86 bináris | MS internal, security hibák |
| Jalangi | - | JavaScript | |
| Symbolic PathFinder | - | Java | |

További (jobbára nem fejlesztett) eszközök:

- CATG, CREST, CUTE, Euclide, EXE, jCUTE, jFuzz, LCT, Palus, PET…

# Példa az SPF megvalósításából

## IFGE bytekód művelet:

```
  public class IFGE extends Instruction{
   public Instruction
          execute (... ThreadInfo th) {
[1]     condition = (th.pop() >= 0);
[2]     if (condition)
[3]       next=getTarget();
[4]     else
[5]       next=getNext(th);
[6]     return next;
    }
  }
```

konkrét megvalósítás (JPF)

```
   public class IFGE extends ....bytecode.IFGE{
    public Instruction
           execute (... ThreadInfo th) {
     IntegerExpression sym_v;
     PCChoiceGenerator cg;
[1]  sym_v = ... .getOperandAttr();
[2]  if(sym_v == null)
[3]    // the condition is concrete
[4]    return super.execute(... th);
[5]  else {
       // the condition is symbolic
[6]    cg = new PCChoiceGenerator(2);
       ...
[7]    condition=cg.getNextChoice()==0?false:true;
[8]    th.pop();
[9]   if (condition) {
[10]     pc._add_GE(sym_v, 0);
[11]     next = getTarget();
       }
[12]   else {
[13]     pc._add_LT(sym_v, 0);
[14]     next = getNext(th);
       }
[15]   if(!pc.isSatisfiable())
[16]     ... // instruct JPF to backtrack
[17]   else
[18]     cg.setCurrentPC(pc);
[19]   return next;
  } } }
```

szimbolikus megvalósítás (SPF)

C. S. Pasareanu et al. (2013) "Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis," Automated Software Engineering 20:3, pp 391-425. DOI: 10.1007/s10515-013-0122-2

# Kihívások

- Futási utak exponenciális növekedése
- Komplex aritmetikai kifejezések
- Lebegőpontos számítások
- Összetett struktúrák és objektumok
- Mutató műveletek
- Interakció a környezettel
- Többszálú alkalmazások
- …

T. Chen et al. „State of the art: Dynamic symbolic execution for automated test generation”. Future Generation Computer Systems, 29(7), 2013

# Kihívások (1)

- Futási utak exponenciális növekedése

```
int hardToTest(int x){
	for (int i=0; i<100; i++){
		int j = complexMathCalc(i,x);
		if (j > 0) break;
	}

	return i;
}
```

- Ötlet:
  - DFS helyett más bejárási algoritmusok
  - *Summary*: bonyolult függvényt helyettesít

# Kihívások (2)

- Komplex aritmetikai kifejezések

```
int hardToTest2(int x){
        if (log(x) > 10)
                return x
        else
                return -x;
}
```

- Legtöbb SMT-megoldó nem boldogul ilyennel
  - De: pl. CORAL-t pont ilyenre fejlesztik
  - Többféle megoldó használata a feltételtől függően

# Kihívások (3)

- Összetett struktúrák és objektumok
- Struktúrák, rekurzív adatszerkezetek
- Ötlet: **Lazy initialization**
  - Kezdetben minden mező (field) nem inicializált
  - Csak akkor adunk értéket, ha hozzányúlunk
    - Érték: null, referencia egy új objektumra/meglévőre

# Kihívások (4)

- Interakció a környezettel

```
int hardToTest3(string s){
  FileStream fs = File.Open(s, FileMode.Open);

  if (fs.Lenth > 1024){
    return 1;
  } else
    return 0;
  }
}
```

- Platform- vagy könyvtárhívás gyakori
- Ötlet:
  - „Environment models” (KLEE): egyszerű C programok

# Kitekintés: tesztgeneráló eszközök vizsálata

- Programrészekből álló bemenet az eszközöknek
  - Fontos nyelvi elemek lefedése
  - Problémás esetek (ld. kihívások)
- 300 darab Java/.NET kódrészlet
  - 3+2 eszközön végrehajtva
- Tapasztalat:
  - Egyszerű dolgokon is elakadnak

Cseppentő Lajos: Szimbolikus végrehajtást használó tesztgeneráló eszközök egységes összehasonlítása, TDK, BME VIK, 2013

# Módszerek

- Szimbolikus végrehajtás (symbolic execution)
- **Véletlen generálás (random generation)**

# Véletlen tesztgenerálás

- Bemeneti tartományból véletlen választás
- Előny:
  - Gyors, olcsó
- Ötletek:
  - Ha egy bemenet nem talál hibát, más részt próbálni
- Eszköz:
  - RANDOOP: visszacsatolás vezérelt generálás
    - Java és .NET könyvtárak különböző verziók összehasonlítása
- (Lásd még a robusztusság tesztelésénél)

# Módszerek

- Szimbolikus végrehajtás (symbolic execution)
- Véletlen generálás (random generation)
- **Annotációk alapján**

# Annotációk felhasználása

Ha vannak a kódban:

- elő- és utófeltételek (pl.: design by contract)
- egyéb annotációk

ezek irányíthatják a tesztgenerálást.

```
/*@ requires amt > 0 && amt <= acc.bal;
    @ assignable bal, acc.bal;
    @ ensures bal == \old(bal) + amt
    @   && acc.bal == \old(acc.bal - amt); @*/
  public void transfer(int amt, Account acc) {
    acc.withdraw(amt);
    deposit(amt);
  }
```

# Eszközök

- AutoTest:
  - Eiffel kód, Design by Contract
  - Bemenet: „object pool”, véletlen generálás
    - Ötlet: előfeltételeket kielégítő bemenetek hozzávétele
  - Elvárt kimenet: szerződés

AutoTest: Bertrand Meyer et al., "Program that Test Themselves", IEEE Computer 42:9, 2009.

- JET:
  - Java kód, JML (Java Modeling Language) annotáció
  - Utófeltételek megsértését nézi (véletlen tesztek)

# Módszerek

- Szimbolikus végrehajtás (symbolic execution)
- Véletlen generálás (random generation)
- Annotációk alapján
- **Keresés alapú (search-based)**

# Keresés alapú technikák

- Search based Software Engineering (SBSE)
- Metaheurisztikus algoritmusok
  - genetikus alg., szimulált lehűlés, hegymászó…
  - Fitnesz függvény: tesztelési célt ír le
- Jó eredmények nagy projektekre is

- Eszköz: EvoSuite
  - Többféle cél alapján keres
    - Pl. nagy lefedettség + kis tesztkészlet
  - „Sandbox” használata

# ÉRTÉKELÉS

# Mennyire használhatók ezek a módszerek?

- SF100 benchmark
  - G. Fraser and A. Arcuri, “Sound Empirical Evidence in Software Testing,” ICSE 2013
  - 100 projekt SourceForge-ról (Java nyelvűek)
  - EvouSuite 48%-os elágazás lefedettséget ér el
    - De nagy a szórás!
- X. Qu, B. Robinson: A Case Study of Concolic Testing Tools and Their Limitations, 2011
  - CREST és KLEE futtatása az ABB egy projektjén
  - Kb. 60%-os elágazás lefedettséget érnek el
    - De sok esettel nem boldogulnak!

# Tényleg jók ezek  a technikák?

- A. Pretschner et al., „A Generic Fault Model for Quality Assurance”, MODELS 2013
  - Nem kód lefedettséget, hanem hibatípusokat kéne megcélozni a kiválasztásnál

- G. Fraser et al., “Does Automated White-Box Test Generation Really Help Software Testers?,” ISSTA 2013
  - Nagy kód lefedettséget elérő generált tesztek nem találtak több hibát, mint a kézzel előállított tesztek.

# Olvasnivaló


1. S. Anand et al. (2013) "*An orchestrated survey of methodologies for automated software test case generation*," Journal of Systems and Software, 86:8, pp. 1978–2001. DOI: j.jss.2013.02.061

2. C. S. Pasareanu, W. Visser (2009) "*A survey of new trends in symbolic execution for software testing and analysis*". STTT, 11(4): 339-353. DOI: 10.1007/s10009-009-0118-1

# Összefoglalás

- Tesztbemenetek generálása struktúra alapján
- Többféle módszer
- Kihívások:
  - Skálázódás
  - Orákulum előállítása
- Aktív kutatási terület!