Cuprins

Mutation testing (mutation analysis)

Tehnica de evaluare a unui set de teste pentru un program (având un set de teste generat, putem evalua cât de eficient este, pe baza rezultatelor obținute de acest test asupra mutanților programului)

Mutation = modificare foarte mică (din punct de vedere sintactic) a unui program

Pentru un program P, un mutant M al lui P este un program obținut modificând foarte ușor P; M trebuie să fie corect din punct de vedere sintactic.

Program P

```
begin
      int x, y;
      read(x, y);
      if (x > 0)
            write(x+y)
      else
            write(x*y)
end
```

Mutant M1

```
begin
      int x, y;
      read(x, y);
      if (x >= 0)
            write(x+y)
      else
            write(x*y)
end
```

Mutant M2

```
begin
      int x, y;
      read(x, y);
      if (x > 0)
            write(x-y)
      else
            write(x*y)
end
```

Mutant M3

```
begin
      int x, y;
      read(x, y);
      if (x > 0)
            write(x+y+1)
      else
            write(x*y)
```

```
end
```

Tehnica Mutation testing:

- Generarea mutanților pentru programul P (folosind o mulțime de operatori de mutație)
- Rularea setului de teste asupra programului P și asupra setului de mutanți; dacă un test distinge între P și un mutant M spunem că P omoară mutantul M.

Mutanți de primul ordin / mutanți de ordin mai mare (first-order/higher-order mutants)

First-order mutants = mutanți obținuți făcând o singură modificare în program

n-order mutants = mutanți obținuți făcând n modificari în program

n-order mutant = first-order mutant of a (n-1)-order mutant, n > 1

n-order mutant, n > 1, sunt numiți higher-order mutants

Mutant de ordin 2

```
begin
      int x, y;
      read(x, y);
      if (x >= 0)
            write(x-y)
      else
            write(x*y)
end
```

În general, în practică sunt folosiți doar mutanții de ordin 1. Motive:

- Numarul mare de mutanți de ordin 2 sau mai mare
- Coupling-effect

Principiile de bază ale mutation testing

- Competent programmer hypothesis (CPH):

Pentru o problemă dată, programatorul va scrie un program care se află în vecinatatea unui program care rezolvă în mod corect problema (și deci, erorile vor fi detectate folosind mutanți de ordinul 1)

- Coupling effect

Datele de test care disting orice program care diferă cu puțin de programul corect sunt suficient de puternice pentru a distinge erori mai complexe.

Rezultate experimentale arată că un set de teste care distinge un program de mutanții săi de ordin 1 este foarte aproape de a distinge programul de mutanții de ordin 2

Explicație intuitivă: în general erorile simple sunt mai greu de detectat. Erorile complexe pot fi detectate de aproape orice test

Strong mutation/ weak mutation

Un test t omoară mutantul M (distinge M față de P) dacă cele două se comportă diferit pentru testul t. Întrebare: când observam comportamentul celor două programe ?

- Testul t aduce pe P și M în stari diferite - se observă starea programului (valorile variabilelor afectate) dupa execuția instrucțiunii mutate.
- Schimbarea stării se propagă la sfarșitul programului - se observă valorile variabilelor returnate și alte efecte (schimbarea variabilelor globale, fișiere, baza de date), imediat dupa terminarea programului

Weak mutation: prima condiție este satisfăcută

Strong mutation: ambele condiții sunt satisfăcute

Program P

```
begin
      int x, y;
      read(x, y);
      y := y+1;
      if (x > 0)
```

```
            write(x)
      else
            write(y)
end
```

Mutant M

```
begin
      int x, y;
      read(x, y);
      y := y-1;
      if (x > 0)
            write(x)
      else
            write(y)
end
```

Testul (1, 1) distinge între P și M d.p.d.v. weak mutation, dar nu distinge între P si M d.p.d.v. strong mutation

Testul (0, 1) distinge între P și M d.p.d.v. strong mutation

Strong mutation: mai puternică. Se asigură că testul t detectează cu adevarat problema

Weak mutation: necesită mai puțina putere de calcul; strâns legată de ideea de acoperire

Mutanți echivalenți

Un mutant M a lui P se numește echivalent dacă el se comportă identic cu programul P pentru *orice* date de intrare. Altfel, se spune că M poate fi distins de P.

Din punct de vedere teoretic: în general, problema determinării dacă un mutant este echivalent cu programul părinte este nedecidabilă (este echivalentă cu halting problem)

În practică: determinarea echivalenței se face prin analiza codului

Determinarea mutanților echivalenți poate fi un proces foarte complex – principala problemă practică a tehnicii mutation testing (avem nevoie să decidem dacă mutanții sunt sau nu echivalenți pentru a putea evalua eficiența testelor)

Utilitatea mutation testing

- Evaluarea unui set de date existent (și construirea de noi teste, dacă testele existente nu omoară toți mutanții)
- Detectarea unor erori în cod

Evaluarea unui set de date existent (exemplu)

Program P

```
begin
      int x, y;
      read(x, y);
      if (x > 0)
            write(x+y)
      else
            write(x*y)
end
```

Considerăm următorii operatori de mutație:

+ inlocuit de –

* inlocuit de /

o variabilă sau o constantă x este înlocuită de x+1

```
begin
      int x, y;
      read(x, y);
      if (x > 0)              M1 if (x+1 > 0)
                              M2 if (x> 0+1)
            write(x+y)        M3 write(x+1+y)
```

```
                              M4 write(x+y+1)

                              M5 write(x-y)

      else

            write(x*y)        M6 write((x+1)*y)

                              M7 write(x*(y+1))

                              M8 write(x/y)

end
```

Set de teste T = {t1, t2, t3, t4} t1 = (0, 0), t2 = (0, 1), t3 = (1, 0), t4 = (-1, -1)

P	**t1**	**t2**	**t3**	**t4**	**Mutant distins**
P(t)	0	0	1	1	
M1(t)	0	**1**	NE	NE	Y
M2(t)	0	0	**0**	NE	Y
M3(t)	0	0	**2**	NE	Y
M4(t)	0	0	**2**	NE	Y
M5(t)	0	0	1	1	**N**
M6(t)	0	1	NE	NE	Y
M7(t)	0	0	1	**0**	Y
M8(t)	ND	NE	NE	NE	Y

Mutanți nedistinși (alive) = {M5}

Întrebare: Este M5 mutant echivalent ?

Răspuns: Nu. (1, 1) distinge între P și M5

Mutation score: MS(T) = D/(L+D), unde

D – numărul de mutanți distinși

L – numărul de mutanți nedistinși (live mutants) neechivalenți

Pentru exemplu: MS(T) = 7/8

Detectarea erorilor folosind mutația (exemplu)

Program P

```
begin
      int x, y
      read(x)
      y = 1
      y = 2            Eroare: instructiune lipsa
      if (x < 0)
            y = 3
      if (x > 2)
            y = 4
      write(y)
end
```

Mutant M

```
...
      if (x < 1)
            y = 3
...
```

Arătăm că mutantul M generează teste care detectează eroarea.

Pentru ca un test t să distingă între P și M trebuie ca:

- Reachability: Instrucțiunea mutată să fie executată la aplicarea lui t
- State infection: Instrucțiunea mutată să afecteze starea programului
- State propagation: Schimbarea de stare să se propage în exterior

Pentru exemplul dat, pentru ca un test t să distingă între P si M:

- Reachability: TRUE
- State infection: $(x < 1 \wedge \neg (x < 0))$
- State propagation: $\neg (x > 2)$

Condiția rezultată: $(x = 0) \wedge (x <= 2) \Leftrightarrow x = 0$

Pentru x = 0 programul corect întoarce 2 în timp ce programul greșit returnează 3

Operator de mutație

Operator de mutație = Regula care se aplică unui program pentru a crea mutanți (e.g. înlocuirea/adăugarea/ștergerea unor operanzi, ștergerea unor instrucțiuni, etc.)

Programul nou obținut trebuie să fie valid din punct de vedere sintactic

Operator de mutație din Java (MuJava //deprecated)

- Traditional mutation operators (method-level operators) – operatori aplicabili oricărui limbaj procedural
- Class mutation operator – operatori specifici paradigmei orientate pe obiect și sintaxei Java
    - Încapsulare
    - Moștenire
    - Polimorfism și dynamic binding
    - Suprascrierea metodelor
    - Java specific

Operatori traditionali

http://cs.gmu.edu/~offutt/mujava/mutopsMethod.pdf

Operatori de clasă

http://cs.gmu.edu/~offutt/mujava/mutopsClass.pdf