Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Igor Rožanc

Testiranje z uporabo grafov

Študijsko gradivo za interno uporabo pri predmetu Testiranje in kakovost (TiK)

Ljubljana, 2017/18

Kazalo

Uvod

Pokritje grafov - glavni kriterij pokritosti testiranja

- Najprej splošen pogled, nato praktičen pogled za specifične programske izdelke
- Osnovna ideja: programski izdelek predstavimo kot usmerjen graf in preverjamo pokritje na njem
- Graf je abstrakten prikaz programskega izdelka – testni primer predstavlja pot čez del grafa
- Graf tipično izhaja iz:
  - strukture programskega izdelka ali
  - podatkovnih tokov skozi njega

Definicija grafa

Graf G sestavlja:

- množica vozlišč N (nodes)
- množica začetnih vozlišč N_0 (initial nodes)
- množica končnih vozlišč N_f (final nodes)
- množica povezav E (edges)

Grafi pri testiranju so usmerjeni povezani grafi ter vsebujejo vsaj eno začetno in končno vozlišče

Tipična predstava: vozlišče je stavek v kodi, povezava je veja pri IF stavku

Podgraf vsebuje del množice vozlišč s pripadajočim delom začetnih, končnih vozlišč in povezav

Risanje grafov – primeri

Definicija grafa

***Pot* je zaporedje vozlišč [n_1,n_2,n_3,n_4, ...,n_M], kjer so vsa sosednja vozlišča n_i, n_{i+1} v množici povezav E**

- Dolžina poti je enaka številu povezav
- Pot poteka OD prvega vozlišča (povezave) DO zadnjega vozlišča (povezave)

Vozlišče n (povezava e) je *sintaktično dosegljivo* iz vozlišča m (povezave f), če v grafu obstaja pot med obema. To vozlišče (povezava) je tudi *semantično dosegljivo*, če obstaja vsaj en vhod (vrednost testnega primera), ki to pot dejansko izvede.

- Za preverjanje dosegljivosti - standardni algoritmi nad grafi

Funkcija *reach$_G$()* obsega množico vseh vozlišč ali povezav, ki so v grafu G dosegljive iz vozlišča, povezave, množice vozlišč ali množice povezav

- Običajno se ukvarjamo z grafi, ki obsegajo reach$_G$(N_0)

Testne poti

Testna pot predstavlja izvedbo testnega primera

Ena testna pot lahko predstavlja množico testnih primerov (ali nobenega)

31. Testna pot (test path) je pot (lahko tudi dolžine 0), ki se začne v nekem začetnem vozlišču in konča v nekem končnem vozlišču.

Poseben primer: SESE grafi

Terminologija:

- testna pot obiskuje (visit) vozlišča (povezave)
- del testne poti (vsebovana "podpot") imenujemo sprehod (tour)
- ***path$_G$(t)*** je preslikava testnega primera t v testno pot
- ***path$_G$(T)*** je množica testnih poti za vse testne primere iz množice testnih primerov T.

Deterministične strukture: vsakemu testnemu primeru ustreza ena testna pot.

Kriterij pokritja grafov

Dva tipa pokritja grafov:

a) Pokritje grafa strukture (kontrolnega toka)

b) Pokritje grafa podatkovnih tokov

Ideja: definiramo testne zahteve v povezavi s strukturo grafa

Konkretno: tipična testna zahteva je dosežena z obiskom vozlišča, povezave ali dela tesne poti v grafu G.

32. Pokritje grafa (*ang. graph coverage*): za množico testnih zahtev *TR* in kriterij pokritja grafa *C* velja, da testna množica *T* zadošča *C* na grafu G takrat in samo takrat, ko za vsako testno zahtevo *tr* iz *TR* obstaja vsaj ena testna pot *p* v *path(T)* tako, da *p* zadošča *tr*.

Pokritje grafa strukture

Obstaja več kriterijev pokritja grafa s stališča strukture:

33. Pokritje vozlišč - formalno (*ang. node coverage - NC*) za vsako vozlišče n, ki je dosegljivo iz množice začetnih vozlišč, množica testnih zahtev *TR* vsebuje predikat "obišči n"

1.Kriterij pokritja vozlišč (NC): TR vsebuje vsa vozlišča grafa G.

34. Pokritje vozlišč - standardno (*ang. node coverage - NC*) testna množica T zadošča kriteriju pokritja vozlišč na grafu G če in samo če za vsako sintaktično dosegljivo vozlišče iz N obstaja (vsaj ena) pot v *path(T)*, ki obišče vozlišče n

Pokritje grafa strukture

2.Kriterij pokritja povezav (EC): TR vsebuje vsako dosegljivo pot (dolžin 0 in 1) v grafu G

- Zajamemo tudi posamezna vozlišča
- Dejansko zapišemo samo najdaljše poti
- Reši problem IF-THEN

3. Kriterij pokritja parov povezav (EPC): TR vsebuje vsako dosegljivo pot (dolžine 2 ali manj) v grafu G

- Bistveno bolje za vgnezdene IF
- Težava: zanke

Pokritje grafa strukture

4. Kriterij pokritja vseh poti (complete path coverage - CPC): TR vsebuje vse poti grafa G.

- Klasična definicija
- Če obstaja le ena zanka, je neskončno poti !!!

5. Kriterij pokritja izbranih poti (specified path coverage - SPC): TR vsebuje množico S poti grafa G, kjer je S posredovan kot parameter.

- Pogosta rešitev
- Subjektivno, problem primerjave

Pokritje grafa strukture

Krožna pot **(round trip path) je pot dolžine najmanj ena, ki se začne in konča v istem vozlišču.**

6. Kriterij pokritja enostavnih krožnih poti (simple round trip coverage - SRTC): TR vsebuje vsaj eno krožno pot za vsako dosegljivo vozlišče grafa G, ki začenja in končuje krožno pot.

- Obravnava enostavnih zank
- Problem vgnezdene zanke

7. Kriterij pokritja vseh krožnih poti (complete round trip coverage - CRTC): TR vsebuje vse krožne poti za vsako dosegljivo vozlišče grafa G.

- Primerno za vgnezdene zanke
- Problem ostale strukture (IF THEN)
- Ne zahteva niti NC

Pokritje grafa strukture

Pot med vozliščema n in m je *enostavna* (simple), če se v njej nobeno vozlišče ne pojavi več kot enkrat (z izjemo začetnega vozlišča).

- Enostavne poti nimajo zank
- Enostavne poti so sestavljene iz enostavnih delov poti
- Tudi grafi enostavnih programov imajo veliko enostavnih poti

35. Primitivna pot (ang. prime path): pot med n in m je primitivna (prime), če in samo če je enostavna in ne nastopa kot del katerekoli druge enostavne poti grafa.

Pokritje grafa strukture

8. Kriterij pokritja primitivnih poti (prime path coverage - PPC): TR vsebuje vse primitivne poti grafa G.

- ugodno: obravnava vse, majhno število testnih zahtev
- problem: neizvedljive primitivne poti, ki so sestavljene iz izvedljivih delov

Implementacija PPC: čeprav enostavne poti (ki določajo testne zahteve) nimajo ciklov, testne poti (ki ustrezajo tem zahtevam) te lahko imajo.

Pokritje grafa strukture

Določanje primitivnih testnih poti

- postopno določanje primitivnih poti dolžin 0 (vozlišča), 1 (povezave), 2, ...
- če izhodiščna pot ni označena z *,+ ali !:
  - če nadaljevanje ni možno, izhodiščno pot označimo z +
  - sicer podaljšamo pot na vse možne načine
- če smo pri tem:
  - prišli do začetnega vozlišča, novo pot označimo z *
  - prišli do končnega vozlišča, novo pot označimo z !
- zaključimo, ko ni več možnosti – dolžina poti kvečjemu (število vozlišč -1)
- naredimo unijo
- izločimo vse delne/vsebovane poti (brez *,+ in !) – od spodaj navzgor
- končna množica testnih poti: na najkrajši način dopolnimo od začetnega vozlišča do začetka poti ter od konca poti do končnega vozlišča ...

Zgled ...

Pokritje grafa strukture

Natančno zaporedje obiskov vozlišč je včasih preveč strogo:

- **Izlet** (side trip): če zahtevana pot [a,b,d], dopušča [a,b,c,b,d]
- **Obvoz** (detour): če zahtevana pot [a,b,d], dopušča [a,b,c,d]

36. Sprehod (ang. tour): Testna pot p se sprehodi po delu poti q če in samo če je q del poti p.

37. Sprehod z izleti (ang. tour with side trips): Testna pot p se sprehodi po delu poti q z izleti če in samo če je vsaka skupna povezava v q in p v istem vrstnem redu.

38. Sprehod z obvozi (ang. tour with detours): Testna pot p se sprehodi po delu poti q z obvozi če in samo če je vsako skupno vozlišče v q in p v istem vrstnem redu.

Pokritje grafa strukture

Uporaba sprehodov, sprehodov z izleti ter sprehodov z obvozi predstavlja različno stroge stopnje pokritij

Izleti lahko rešujejo problem neizvršljivih zahtev: problem do-while zank

Pristop: najprej brez izletov, kar se ne da z izleti

39. Najučinkovitejše sprehajanje (ang. Best effort touring): naj TR1 predstavlja podmnožico zahtev, ki jih je mogoče izvesti in TR2 množica zahtev, ki jih je mogoče izvesti z izleti (TR1 je podmnožica TR2). Množica testnih poti T obsega najučinkovitejše sprehajanje, če za vsako pot v TR1obstaja pot v T, ki se sprehodi brez izletov, ter za vsako pot v TR2 obstaja pot v T, ki se sprehodi z ali brez izletov.

Pokritje grafa podatkovnih tokov

16

Ideja: ustrezno testiranje zahteva spremljanje tokov podatkov

Podatki so na določeni točki programa definirani in nato (enkrat ali večkrat) uporabljeni

Definicija (definition - DEF) je mesto, kjer je pripisana vrednost določeni spremenljivki (prirejanje, inicializacija, vnos podatka, ...)

Uporaba (ang. Use - USE) je lokacija, kjer do določene spremenljivke dostopamo (odločitve, izpis, ...).

DU pari (definition-use, def-use pairs)

Spremljamo program z vidika DU parov na več načinov

Pokritje grafa podatkovnih tokov

17

Graf

• Množica spremenljivk programskega izdelka V

• vsako vozlišče in povezava definira podmnožico V: def(n) in def(e) - ne za programe

• vsako vozlišče in povezava uporablja podmnožico V: use(n) in use(e) – pri programih tipično odločitve

Pomemben koncept: def neke spremenljivke lahko doseže določen use ali pa ne, ker lahko vmes pride do spremembe vrednosti

Def-clear pot za spremenljivko v: pot med vozliščem (povezavo) i in j, ko za vsa vozlišča in povezave na tej poti noben def(n) in def(e) ne vsebuje vozlišča v

V tem primeru i doseže (reaches) j.

Pokritje grafa podatkovnih tokov

18

DU pot (ang. du-path) za spremenljivko v je enostavna in def-clear pot od vozlišča n (kjer je ta spremenljivka v množici def()) do m (kjer je ta spremenljivka v množici use(m)).

Zgled...

Kriterij testiranja je definiran nad množicami du-poti, ki jih je treba natančneje opredeliti:

Def-pot množica du(n,v) je množica tistih du-poti z vidika spremenljivke v, ki se začnejo v vozlišču n.

All-Defs kriterij: vsaj ena DU pot iz vsake def-poti mora biti izvršljiva.

- običajno veliko vozlišč in spremenljivk, slab delež pokritosti
- use-path ni uporaben

Pokritje grafa podatkovnih tokov

19

Def-par množica du(n,m,v) je množica DU poti z vidika spremenljivke v, ki se začnejo v n in končajo v m.

Konkretno: množica enostavnih poti od definicije do uporabe.

All-uses kriterij: vsaj ena DU-pot iz vsake def-par množice mora biti izvršljiva.

Velja: du(n,v) = Unija (du(n,m,v))

Testna pot p je du-sprehod d z vidika spremenljivke v če se p sprehodi po d in je ta del p def-clear z vidika spremenljivke v.

Pokritje grafa podatkovnih tokov

Trije kriteriji pokritja za grafe podatkovnih tokov:

- Vsak def vsaj eno uporabo
- Vsak def vse uporabe ter
- Vsake def vse dosegljive uporabe preko du-poti.

9. Kriterij All-Defs pokritja (ADC): za vsako def-pot množico S=du(n,v), TR vsebuje vsaj eno pot d iz S.

10. Kriterij All-Uses pokritja (AUC): za vsako def-par množico S=du(n,m,v), TR vsebuje vsaj eno pot d iz S.

11. Kriterij pokritja vseh DU-poti (ADUPC): za vsako def-par množico S=du(n,m,v), TR vsebuje vse poti d iz S.

Pokritje grafa podatkovnih tokov

Zaključne pripombe:

• omejili smo se na def v vozliščih, če def tudi v povezavah se zaplete

• možnosti sprehodov – ni omejitev

• predvidevamo najučinkovitejše sprehajanje

Pokritje grafov in vsebovanost

22

Osnova: Pokritje povezav (EC) vsebuje pokritje vozlišč (NC) – pozor dostopnost

Ob upoštevanju najučinkovitejšega sprehajanja (tudi ob neizvedljivih testnih zahtevah) velja:

CPC vsebuje PPC

PPC vsebuje ADUPC, EPC(razen posebnega primera) in CRTC

ADUPC vsebuje AUC

EPC vsebuje EC

CRTC vsebuje SRTC

AUC vsebuje EC in ADC

EC vsebuje NC

Pokritje grafov programske kode

23

Grafi programske kode – najpogosteje

STRUKTURNI GRAFI

Graf kontrolnega toka (control flow graph - CFG)

- Vozlišče = osnovni blok zaporedno izvedljivih stavkov
- Povezava = veje pri odločitvah ali ponavljanjih
- Pogosto vozlišča zaradi strukture:
  - odločitvena (decision) in združitvena (junction) vozlišča
  - slepa (dummy) vozlišča: zanke
- CFG vsebujejo dodatne informacije: predikate, defs, uses
- Tipične strukture grafov za različne stavke

Pokritje grafov programske kode

PRIMERI:

- if-else stavek
- if stavek
- if stavek z izhodom
- while zanka
- for zanka
- do-while zanka
- zanka z break in continue
- switch stavek

Pokritje vozlišč NC = pokritje stavkov ali pokritje osnovnih blokov

Pokritje povezav EC = pokritje vej

Pokritje grafov programske kode

GRAFI PODATKOVNEGA TOKA

def za spremenljivko x:

- x je na levi strani prirejanja
- x se prebere z vhoda
- x je parameter klican po referenci, ki se spremeni v klicani metodi
- x je formalni parameter metode

use za spremenljivko x:

- x je na desni strani prirejanja
- x se izpiše
- x je del pogoja
- x je klicni parameter metode
- x je vrnjena vrednost metode

Pokritje grafov programske kode

Upoštevamo le globalno uporabo spremenljivk, ne lokalne

PRIMER: Program za statistično obdelavo tabele števil

```
import java.util.Scanner;
public class Statistika {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Velikost tabele: ");
    int[] tabela = new int[sc.nextInt()];
    for (int i=0; i<tabela.length; i++)
      tabela[i]=(int)(Math.random()*99+1);
    statistika(tabela);
  }
```


Pokritje grafov programske kode


```
public static void statistika(int[] t) {
     // 1
      int len = t.length, med;
      double var, sd, mean, sum=0, varsum=0;
      // 2,3,4
      for(int i=0; i<len; i++)
              sum +=t[i];
      // 5
      med = t[len/2];
      mean = sum/(double)len;
      // (5),6,7
      for(int i=0; i<len; i++)
              varsum += (t[i]-mean)*(t[i]-mean);
      // 8
      var = varsum/(len-1.0);
      sd = Math.sqrt(var);
      System.out.printf(" St: %d %n Med: %d %n Pov: %.3f %n
                        Var: %.3f %n Std.odk: %.3f %n",
                        len, med, mean, var, sd);

}}
```

Pokritje grafov programske kode

28

ANALIZA PRIMERA:

Graf strukture:

- EC, EPC, PPC – vhodi ...

Graf pod.tokov:

- tabele def in use za vozlišča in povezave …
- du-pari in poti za vsako spremenljivko …
- 38 DU poti, le 12 enoličnih
- 0, 1 ali 2 iteraciji zank – vhodi ...

Pokritje grafov načrtovanja

29

Načrtovanje: modularnost in ponovna uporaba vnaša zahtevnost

Testiranje načrtovanja je zato vse pomembnejše - i**ntegracijsko testiranje**

Sestavljanje različnih modulov – važne so povezave in odvisnosti

STRUKTURNI GRAF

- Najpogosteje
- Povezave med elementi programskega sistema (metodami v Javi)
- Strukturni grafi relativno enostavni
- Običajno – graf klicev metod (call graph)
- NC = pokritje metod (method coverige)– klic vsake metode vsaj enkrat
- EC = pokritje klicev (call coverige)– izvedi vsak klic vsaj enkrat

Pokritje grafov načrtovanja

Načrtovanje objektnih programov

- Metode v razredih – ne deluje dobro
  - veliko nepovezanih grafov
  - pri testiranju potrebna povezava
  - potrebujemo druge načine testiranja – ne grafi
- Dedovanje in polimorfizem
  - ne da se testirati brez tvorbe objektov
  - pripnemo objekte posameznih razredov, izvedemo pokritje klicev
  - še vedno ni dobro ...

OO pokritje klicev (OO call coverage): TR vsebuje vsa dosegljiva vozlišča v grafu klicev za tvorjen objekt iz vsakega razreda v hierarhiji

OO pokritje klicev objektov (OO object call coverage): TR vsebuje vsa dosegljiva vozlišča v grafu klicev za vsak tvorjen objekt iz vsakega razreda v hierarhiji

Pokritje grafov načrtovanja

GRAF PODATKOVNEGA TOKA

- Bolj uporabno kot skrukturni graf
- Omogočajo učinkovito odkrivanje težkih napak
- Bolj zahtevno kot v metodi: parametri spreminjajo imena, različni načini posredovanja podatkov, težko določanje def in use v različnih delih...
- POZOR - ravno tu so napake ...
- Ugotavljanje podatkovnega toka je predrago za vse DU pare, ker imamo preveč možnosti
- VENDAR: pri integracijskem testiranju nas zanima le vmesnik, ne podrobnosti ...

Pokritje grafov načrtovanja

GRAF PODATKOVNEGA TOKA

- Ključno: def in use so v različnih modulih
- Klicatelj (dejanski parameter) in klicani (formalni parameter)
  - Klicatelj:
    - Samo zadnja definicija pred klicem
  - Klicani:
    - Samo prva uporaba v klicani metodi
    - Samo zadnja definicija pred vračanjem vrednosti
  - Klicatelj:
    - Samo prva uporaba po klicu
- Dodatno - deljeno posredovanje podatkov (shared data coupling) – globalne spremenljivke ...

Pokritje grafov načrtovanja

Zadnja def (last def): množica vozlišč, ki definirajo spremenljivko x in imajo def-clear pot do uporabe v drugem (običajno klicanem) modulu (enoti, metodi)

- Lahko posredovana kot parameter ali drugeče klicanemu
- Lahko posredovana kot vrnjena vrednost klicatelju

Prvi use (first use): množica vozlišč, ki uporabljajo spremenljivko x in imajo def-clear od definicije v drugem (običajno klicajočem) modulu (enoti, metodi)

Pokritje grafov načrtovanja

Povezovanje Podatkovnih tokov

- le za spremenljivke, ki jih definira ali uporablja klicatelj
- upoštevamo tudi razredne in globalne spremenljivk
- omejimo se na enostavne klice, brez tranzitivnosti:
  A() kliče B(), B() kliče C(), spremenljivka iz A() uporabljena v C()
- tabele: klic enega elementa je klic vseh elementov tabele ...

ZGLED: Quadric – reševanje kvadratnih enačb

Pokritje grafov načrtovanja


```
public class Quadric {
  private static float r1, r2;
  public static void main (String[] argv) {  // 0
        int x, y, z;
        boolean ok;
        int br =Integer.parseInt(argv[0]);
        if (br == 1) {    // 1
                x = Integer.parseInt(argv[1]);  // 2
                y = Integer.parseInt(argv[2]);  // 2
                z = Integer.parseInt(argv[3]);  // 2
        }
        else { x = 1; y = 0; z = 0; } // 3
     ok = korena(x, y, z); // 4
     if (ok) // 5
        System.out.println(„Nicli: " + r1 +"  "+ r2); // 6
    else
        System.out.println („Ni resitve."); // 7
  } // 8
```

Pokritje grafov načrtovanja


```
   private static boolean korena(int a, int b, int c) {  // 9
         float det;
         boolean r;
         det = (float) (Math.pow((double)b,(double)2)-4.0*a*c); // 9
         if (det < 0.0){  // 10
                r = false; // 11
                return r;  // 11
         }
         r1=(float)((-b + Math.sqrt(det))/(2.0*a)); // 12
         r2=(float)((-b - Math.sqrt(det))/(2.0*a)); // 12
         r = true; // 12
         return r; // 12
   }

}
```


Pokritje grafov načrtovanja

Dedovanje, polimorfizem, dinamično povezovanje

- Dodatne nadzorne in podatkovne povezave naredijo analizo še zahtevnejšo
- Upoštevati je treba različne razredne hierarhije, def iz ene enote lahko dosega use v vsakem razredu dedovalne hierarhije: če je B podrazred razreda A, potem so vse spremenljivke in metode iz A tudi del B (poleg njegovih lastnih)
- Podobno je z dinamičnim povezovanjem: spremenljivka tipa A je lahko objekt tipa A ali B (ali kateregakoli iz njega izpeljanega tipa)
- Uporablja se le neposredno OO povezovanje, še za to ni orodij
- Uporaba posrednega v teoriji in praksi je vprašljiva

Podobno je s testiranjem porazdeljenih in spletnih aplikacij

Pokritje grafov specifikacij

Specifikacije opisujejo, kako naj bi se programski izdelek obnašal

Dejanska izvedba je lahko taka ali pa tudi ne

Pogosto jih imenujemo modeli programskega izdelka

Vrste specifikacij:

- **Zaporedje omejitev na metodah razredov (sequencing constrains)**
- **Opis obnašanja PO s stanji (state behavior)**
- **Opis specifikacij z UML diagrami**

Pokritje grafov specifikacij

Zaporedje omejitev: pravila, ki določajo omejitve pri zaporedju klicev metod

- Pogosto so zapisane kot predpogoj za izvedbo metode
- Tudi ko nimamo povezanih metod (razred `Stack`) lahko za teste določimo ustrezna zaporedja
- Omejitve so lahko eksplicitne, implicitne ali pa jih ni
- Če jih ni, uporabimo: načrt, seznam zahtev, uporabnika, (v sili) kodo
- POZOR - če jih ni, je veliko potencialnih napak na tem področju
- Ne zajemajo celotnega obnašanja
- Tipičen primer: metoda pop pričakuje, da je vsaj en element na skladu – pred tem je treba klicati push – implicitni predpogoj ali posledica
- Primer: odpirane, pisanje in zapiranje datoteke

Pokritje grafov specifikacij

DVA NAČINA UPORABE:

Statično preverjanje

- ali obstaja pot v grafu, ki krši omejitve

Generiranje testnih zahtev

- skušamo tvoriti testne zahteve, ki kršijo omejitve
- tak test mora biti popolnoma neizvedljiv
- če lahko naredimo tak test, bo ta skoraj gotovo našel napako

Pokritje grafov specifikacij

Opis obnašanja z grafom stanj

Graf s končnim številom stanj (finite state machine - FSM) opisuje spremembe programskih spremenljivk med izvajanjem programa

- Vozlišča: stanja, predstavljajo množico vrednosti za ključne spremenljivke
- Povezave: možne spremembe stanj (prehodi med stanji)

Primer z tropskimi viharji

Pokritje grafov specifikacij

• **Primeren za veliko vrst programske opreme** (vgrajena ali nadzorna PO, abstraktni podatkovni tipi, prevajalniki in operacijski sistemi, spletne aplikacije)

• **Omogoča učinkovito odkrivanje napak**

• **Več jezikov za opis FSM** (UML, avtomati, tabele stanj, petrijeve mreže...)

• **Omejitev**: ni primeren za PO z veliko stanji (recimo GUI)

• **FSM lahko povezan z različnimi tipi akcij**: akcija ob prehodu, ob dostopu v vozlišče ali ob izhodu iz vozlišča

• **Akcije izražajo spremembe spremenljivk ali pogojev za spremenljivke**

• **Osnoven primer za naše zglede**: poznamo predpogoj za izvedbo prehoda ter sprožilec za izvedbo prehoda (podobno, a ne enako kot UML)

Pokritje grafov specifikacij

POKRITJA FSM

Struktura FSM:

- NC: izvedi vsako stanje - pokritje stanj
- EC: izvedi vsak prehod - pokritje prehodov
- EPC: pokritje parov prehodov

Podatkovni tokovi FSM:

- vozlišča nimajo def ali use za spremenljivke
- sprožilci: spremenljivke so uporabljene takoj (v naslednjem stanju)
- predpogoji: za te določamo def in use
- običajno smo omejeni le na del spremenljivk

Posledica: Tvorba FSM je običajno težji del kot določanje ustreznih pokritij

Pokritje grafov specifikacij

TVORBA FSM

- običajno obstaja – razvit med načrtovanjem (preveri, če še velja)
- Če ga ni, ga naredi, ker je zelo uporaben za testirenje:
  1. kombiniraj grafe kontrolnega toka
  2. uporabi strukturo PO
  3. modeliraj spremenljivke stanj
  4. uporabi posredne ali neposredne specifikacije

Primer: digitalna ura

Primer


```
public class Watch
{
   private static final int NEXT = 0;
   private static final int UP = 1;
   private static final int DOWN = 2;
   private static final int TIME = 5;
   private static final int STOPWATCH = 6;
   private static final int ALARM = 7;

   private int mode = TIME;

   private Time watch, stopwatch, alarm;

   public Watch() {
     watch= new Time();
     stopwatch= new Time();
     alarm= new Time();
   }
   ...
 }
```

Primer


```
public void doTransition(int button) {
   switch(mode) {
     case TIME:
       if (button == NEXT)
         mode = STOPWATCH;
       else
         watch.changeTime(button);
       break;
     case STOPWATCH:
       if (button == NEXT)
         mode = ALARM;
       else
         stopwatch.changeTime(button);
       break;
     case ALARM:
       if (button == NEXT)
          mode = TIME;
       else
          alarm.changeTime(button);
     }
  }
```


Primer


```
private class Time {
   private int hour = 1, minute = 0;

   public void changeTime(int button) {
     if (button == UP) {
        minute += 1;
        if (minute >= 60) {
           minute = 0; hour += 1;
           if (hour > 12)
              hour = 1;
        }
     }
     else if (button == DOWN) {
         minute -= 1;
         if (minute < 0) {
            minute = 59; hour -= 1;
            if (hour <= 0)
               hour = 12;
         }
     }
  }
```

Pokritje grafov specifikacij

1. **Kombiniranje grafov kontrolnega toka:**

- Prvi prijem – ne naredi zares FSM
- Težave:
  - vračanje po klicih metod – nedeterminizem
  - obstajati mora izvedba
  - ni mogoče razširjati

2. **Uporaba strukture PO**

- Bolje – povezava metod in stanj
- Vendar metode niso stanja
- Težave:
  - različni razvijalci bodo tvorili različne grafe - subjektivno
  - zahteva poglobljeno razumevanje izvedbe
  - že obstaja podroben načrt rešitve

Pokritje grafov specifikacij

3. **Modeliraj spremenljivke stanj**

- Avtomatsko – spremenljivke stanj poznamo zgodaj
- Definiraj vse spremenljivke, izberi relevantne:
  - konstate niso stanja
  - pomembno različno število vrednosti spremenljivk
  - zahteva podrobno razumevanje problema
- Teorija: vsaka kombinacija spremenljivk svoje stanje
  - lahko zelo veliko stanj, zahteven graf
  - popolnoma deterministični graf
- Praksa: združi množice tovrstnih stanj v eno stanje
  - podrobno poznavanje problema
  - lahko pride do nedeterminizma, ki ga v implementaciji zares ni
- Nekatera stanja ne bodo dosegljiva

Pokritje grafov specifikacij

4. Uporabi posredne ali neposredne specifikacije

- Uporabimo eksplicitne specifikacije iz formalnih specifikacij
- Lahko jih izpeljemo, če jih ni
- Včasih podobne 1., včasih pa podobne 3. varianti
- Težave:
  - včasih ne pokriva vseh pomembnih delov implementacije

Pokritje grafov specifikacij

PREDNOSTI POKRITIJ FSM

- Teste lahko pripravimo pred implementacijo
- Analiza FSM je lažja od analize programske kode

SLABOSTI POKRITIJ FSM

- Nekateri deli implementacije niso opisani v FSM
- Subjektivnost pri izdelavi – različni rezultati
- Spremenljivke v implementaciji so lahko drugačne kot v FSM – potrebno je pretvarjanje vrednosti

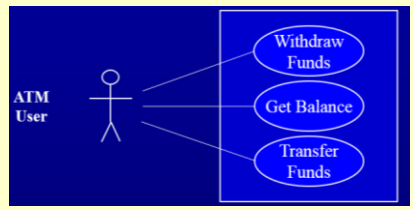
Pokritje grafov specifikacij

UML diagrami

Jedrnat opis uporabnikovih zahtev - prikaz podatkov, delovnega toka ...

USE CASE diagrami:

- dokumentirani, sistematično in natančno opisani
- opis predpogojev in posledic, vsebujejo opis alternativ
- žal pregrobi, dokaj neuporabni

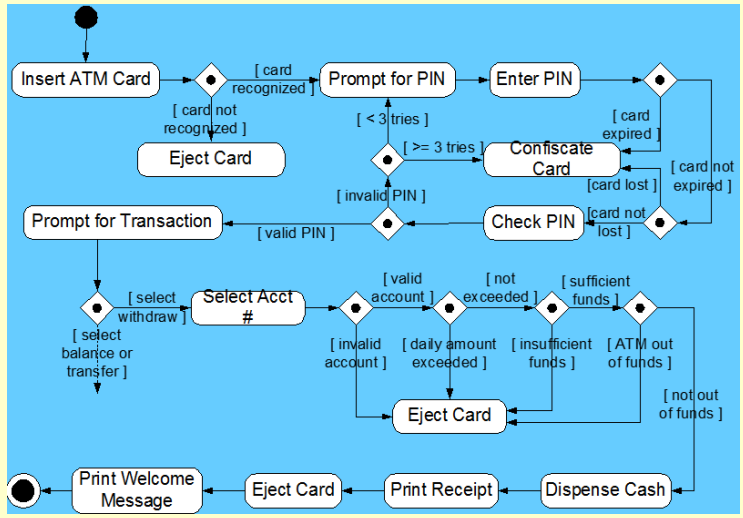


Pokritje grafov specifikacij

Akcijski diagrami:

- Najbolj podoben del UML opisa, transakcijski graf
- Opisujejo tok med aktivnostmi
- Vozlišča dveh vrst: akcijska stanja (opisi) in zaporedne veje vozlišč (alternative)
- Primerni za uporabo: malo zank, enostavni predikati, ni očitnih DU poti
- Uporabljamo: NC, EC, za testiranje scenarijev smiselno določena pokritja poti
- Pokritje toka podatkov ni smiselno

Pokritje grafov specifikacij



Algebraični zapis grafov

Ljudje: grafičen opis grafov, računalniki: strnjen algebraični opis

PRAVILA za prepis grafa v algebraični opis (regularni izraz):

- Povezava med določenimi vozlišči = enolična oznaka: a
- Zaporedje povezav – operator stik : ab
- Izbira – operator +: a+b
- Zanke – kot eksponenti – običajno z *: $(ab)^* \cdot (ab)^3$, $(ab)^+$

Produkt poti (path product): združeno zaporedje povezav

Izraz poti (path expression): produkt poti z operatorji + in eksponenti

Podobnost z algebro: produkt poti NI komutativen in je asociativen, vsota poti je tako komutativna kot asociativna

Algebraični zapis grafov

PRAZNE POTI – IDENTITETE

Produkt: lambda - λ (prazna pot – empty path)

Vsota: fi - Φ (null path - množica poti, ki nima poti)

Pravila:

$\lambda + \lambda = \lambda$

$A + \Phi = \Phi + A = A$

$\lambda A = A \lambda = A$

$A \Phi = \Phi A = \Phi$

$\lambda^n = \lambda^+ = \lambda^* = \lambda$

$\Phi^* = \lambda + \Phi + \Phi^2 + .. = \lambda$

$\lambda^+ + \lambda = \lambda^* = \lambda$

Algebraični zapis grafov

POSTOPEK DOLOČANJA IZRAZA POTI GRAFA

PONAVLJAJ 4 KORAKE:

1. Sestavi zaporedni povezavi (produkt)
2. Seštej paralelni povezavi (vsota)
3. Odpravi zanke na istem vozlišču (dodaj novo vozlišče - produkt)
4. Odstrani izbrano nekončno vozlišče (dodaj produkte od vseh predhodnikov do naslednikov, odstrani vozlišče)

Algebraični zapis grafov

UPORABA TAKO DOLOČENIH IZRAZOV POTI

1. Določanje testnih primerov
2. Štetje poti v grafu tokov
3. Najmanjše število poti za zagotavljanje pokritja vseh povezav
4. Analiza dopolnitevnih operacij

Algebraični zapis grafov

1. DOLOČANJE TESTNIH PRIMEROV

- Poseben primer kriterija pokritja specifičnih poti
- Pokriti je treba vse produkte poti izraza
- Problem neskončnih eksponentov – določi končne eksponente

2. ŠTETJE POTI V GRAFU

- Včasih smiselno – možen aritmetičen izračun
- Zanke – neskončnost = primerna končna vrednost (max, ave)
- Cene poti – običajno 1 – nadomeščajo povezave
- Rezultat: ocena vrednosti izvedbe vseh poti grafa

Algebraični zapis grafov

3. NAJMANJŠE ŠTEVILO POTI ZA EC

- Za določanje (ALL) EDGES CRITERIA
- Podobno kot prej, le drugače AB = max(A,B) in A^n=1 ali A (presoja)

4. ANALIZA DOPOLNITVENIH OPERACIJ

- Način iskanja potencialnih anomalij (DU,...)
- Komplementarne operacije: izvede se ena ali druga, druga naredi obratno od prve (push-pop, open-close, ...)
- Na povezave namesto cene napišemo: C (tvori), D (uniči), I (nobeno)
- Za vsoto in produkt posebne tabele, malo drugačna pravila
- Vprašanja: ali imamo lahko več D kot C oz. več C kot D?
- Vsak pritrdilni odgovor zahteva test, ker nakazuje anomalije...

Algebraični zapis grafov

ZAKLJUČEK

- Zapis z regularnimi izrazi zelo uporaben
- Presoja risanih grafov pogosto subjektivna, za izraze poti obstajajo jasni postopki
- Opisani postopki niso vedno uporabni – omejeni primeri
- Ni učinkovitih orodij

Literatura

1. Paul Ammann, Jeff Offutt: Introduction to software testing, Cambridge University Press, 2008.
2. Wikipedia