Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Igor Rožanc

Testiranje vhodnih podatkov

Testiranje sintakse

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

Ljubljana, 2017/18

Kazalo

Pokritja

Štirje abstraktni modeli pokritij zajemajo vse možne testne tehnike:

1. **pokritje z grafi (ang. Graph Coverage)**
2. **pokritje z logični izrazi (ang. Logic Coverage)**
3. **delitev prostora vhodnih podatkov (ang. Input Space Partitioning)**
4. **sintaksno testiranje (ang. Syntax-Based Testing)**

Pokritje prostora vhodnih podatkov

- Vhodna domena: vsi možni vhodi v program
- Običajno že za majhne programe neskončna množica vhodov
- Za testiranje izbiramo končne množice vrednosti
- Vhodni parametri določajo namen vhodne domene:
  - Parametri metode
  - Podatki datoteke
  - Globalne spremenljivke
  - Uporabnikov vnos
- Domeno delimo za vsak vhodni parameter na področja
- Izbrati moramo vsaj eno vrednost s področja

Pokritje prostora vhodnih podatkov

Prednost pristopa:

- uporaben za različne ravni testiranja
- enostaven za uporabo tudi brez avtomatizacije
- enostavno prilagajanje števila potrebnih testov
- ni potrebnih specifičnih znanj

Domena D:

- shema particije q določa množico blokov $B_q=b_1,b_2, \ldots b_n$
- Dve zahtevi:
  - Bloki se paroma ne smejo prekrivati
  - Vsi skupaj v celoti pokrivajo domeno

Pokritje prostora vhodnih podatkov

Predpostavke pri uporabi particij:

- Uporabimo eno vrednost iz vsake particije
- Vsaka vrednost iz particije je enako primerna za testiranje
- Za uporabo pri testiranju:
  - določi značilnosti vhodov,
  - razdeli značilnosti in
  - izberi ustrezne teste
- Primeri značilnosti:
  - vhod je null
  - urejenost vhodnih podatkov (pozor na prekrivanje),
  - zunanja vhodna enota, ...

Pokritje prostora vhodnih podatkov 6

Dva pristopa pri modeliranju vhodne domene (IDM – Input Domain Model):

1) Pristop na podlagi vmesnika:

- Določi značilnosti neposredno iz vhodnih parametrov
- Enostavna uporaba, lahko je deloma avtomatiziran
- Obravnava vsak parameter posebej
- Uporablja predvsem sintakso
- Ne uporablja vseh semantičnih informacij
- Ignorira medsebojni vpliv med parametri
- Primer: branje vhodnih parametrov za tri stranice trikotnika
  - Vsi so int, opis značilnosti: stranica trikotnika

Pokritje prostora vhodnih podatkov 7

2) Pristop na podlagi funkcionalnosti:

- Določi značilnosti na podlagi obnašanja testiranega programa
- Težji za uporabo, zahteva več izkušenj
- Lahko doseže boljše rezultate - manj testov
- Upošteva semantične informacije
- Medsebojni vpliv med parametri je pomemben
- Primer: branje vhodnih parametrov za stranice trikotnika
  - Podatki so vnešeni kot celoten trikotnik oz. vrsta trikotnika

Pokritje prostora vhodnih podatkov

5 korakov modeliranja vhodne domene:

1) Določi funkcije za testiranje

- Metoda ima običajno eno
- Razred – običajno več metod eno
- Programi – zapleteno, več značilnosti, več funkcij
- Sistem – še bolj, različni deli ...

2) Poišči vse parametre

- Običajno neposredno
- Pomembno, da je popolno
- Metode, komponente: parametri in nelokalne spremeljivke
- Sistem: vsi vhodi, tudi datoteke, PB

Pokritje prostora vhodnih podatkov

3) Modeliraj vhodno domeno

- Najbolj kreativni del
- Domena je deljena glede na parametre
- Strukturo določajo značilnosti
- Vsaka značilnost določa bloke, blok predstavlja množica vrednosti

4) Uporabi testni kriterij za izbiro kombinacij vrednosti

- Test ima vrednost za vsak parameter
- En blok za vsako značilnost
- Izberemo podmnožico, ker je izbira vseh kombinacij neizvedljiva

5) Natančno posreduj kombinacije blokov v testne vhode

Pokritje prostora vhodnih podatkov

Nasveti:

- Modeliranje je kreativna inženirska aktivnost
- Več karakteristik pomeni več testov
- Običajno parametre prevedemo v značilnosti, ki imajo malo blokov
- Kandidati: predpogoji, razmerja spremenljivk, posebne vrednosti spremenljivk ..
- Ne uporabljamo kode – le vhodne podatke
- Strategije: uporaba T in F, postopno dejenje blokov, upoštevanje mej, usklajeno število blokov o značilnostih, preverjanje popolnosti in prekrivanja ,...
- Če preveč parametrov – postopoma z deli in vladaj ...
- Pri določanju testov uporabimo kriterije za določanje učinkovitih podmnožic

Pokritje prostora vhodnih podatkov

1.Kriterij pokritja vseh kombinacij – All Combinations (AC): Vse kombinacije blokov za vse značilnosti je treba preveriti.

- Veliko testov – produkt števil različnih blokov za vse značilnosti

2. Kriterij pokritja vseh izbir – Each Choice Criteria (EC): Po ena vrednost iz vsakega bloka mora biti uporabljena vsaj pri enem testu.

- Število testov je enako največjemu številu blokov značilnosti
- Bistveno manj

3. Kriterij pokritja parov – Pair-Wise Criteria (PW): Po ena vrednost iz vsakega bloka za vsako značilnost mora biti kombinirana z vrednostjo iz vsakega bloka za vsako drugo značilnost.

- Število testov je vsaj enako produktu števila blokov dveh največjih značilnosti

Pokritje prostora vhodnih podatkov

4.Kriterij pokritja t-kombinacij – t-Wise (TW): Po ena vrednost iz vsakega bloka za vsako skupino t značilnosti moramo kombinirati med seboj.

- Posplošitev – število testov je produkt števil blokov t-največjih značilnosti
- Podoben AC – drag, niso jasni učinki

5. Kriterij osnovnega pokritja – Base Choice (BC): Izberemo osnovni blok za vsako značilnost in tvorimo osnovni test za vsako značilnost. Nadaljnje teste tvorimo tako, da nastavimo vse vrednosti razen ene konstantne in uporabimo vse ostale neosnovne izbire za vse značilnosti.

- Uporabimo znanje o domeni – prepoznamo pomembne vrednosti
- Število: en osnovni test + en test za vsak ostali blok

Pokritje prostora vhodnih podatkov

6. Kriterij večkratnega osnovnega pokritja – Multiple Base Choice (MBC): En ali več osnovnih blokov izberemo za vsako značilnost in tvorimo več osnovnih testov za izbrane osnovne bloke vsake značilnosti. Nadaljnje teste tvorimo tako, da nastavimo vse vrednosti razen ene konstantne in uporabimo vse ostale neosnovne izbire za vse značilnosti.

- Posplošitev prejšnjega
- Število testov je enako številu osnovnih testov + število osnovnih testov za vsako značilnost

Pokritje prostora vhodnih podatkov

Neizvedljive kombinacije blokov:

- Predstavljajo omejitve med bloki
  - 2 vrsti: prepovedane in obvezne povezave blokov
- Različna pravila za različne kriterije:
  - AC, PW, TW: ignoriramo
  - BC, MBC: poskusimo z drugo neosnovno izbiro bloka

Razmerje med pokritji:

AC

TW **MBC**
PW **BC**

EC

Enostavno, neposredno, učinkovito, se veliko uporablja v praksi!

Pokritja

Štirje abstraktni modeli pokritij zajemajo vse možne testne tehnike**:**

1. **pokritje z grafi (ang. Graph Coverage)**
2. **pokritje z logični izrazi (ang. Logic Coverage)**
3. **delitev prostora vhodnih podatkov (ang. Input Space Partitioning)**
4. **sintaksno testiranje (ang. Syntax-Based Testing)**

Ogledali si bomo samo osnovne principe!

Pokritje sintakse

16

Veliko programske opreme upošteva striktna sintaksna pravila

Sintakso običajno opisujemo za gramatikami (recimo BNF)

Viri:

- Programska koda
- Specifikacije
- Načrti
- Opisi vhodnih podatkov

Dva osnovna cilja:

- Nekako pokriti sintakso
- Kršiti sintaksna pravila

Pokritje sintakse

17

GRAMATIKE:

TPO uporablja teorijo avtomatov za:

- Opis programskih jezikov z BNF
- Opis obnašanja programov z FSM
- Definiranje vhodov z gramatikami

Primer: regularni izrazi: (G s n | B t n)*

- G, B – ukaz, metoda
- s, t – število
- n – datum

Če niz ustreza regularnemu izrazu – pripada gramatiki

Test je zaporedje nizov iz gramatike

Pokritje sintakse

BNF notacija gramatike:

```
Stream ::=  action*
action ::=  actG  |  actB
actG   ::=  "G" s  n
actB   ::=  "B"  t  n
s      ::=  digit[1-3]
t      ::=  digit[1-3]
n      ::=  digit[2]  "."  digit[2]  "."  digit[2]
digit  ::=  "0" | "1" | "2" | "3" | "4" | "5" | "6" |
            "7" | "8" | "9"
```


Pokritje sintakse

Gramatike rabimo za

- **prepoznavanje** (ali je niz v gramatiki) in
- **generiranje** nizov iz gramatike

- Na ta način dobimo pravilne nize iz gramatike
- (Bolj) zanimivi za testiranje so nepravilni nizi ...

Pokritje sintakse

NA PODLAGI GRAMATIK:

1. Kriterij pokritja končnih simbolov – Terminal Symbol Coverage (TSC): TR vsebuje vsak terminalni simbol gramatike G.

2. Kriterij pokritja produkcij – Production Coverage (PDC): TR vsebuje vsako produkcijo gramatike G.

- Testiramo pripadnost niza gramatiki
- PDC vsebuje TSC
- Obstaja preslikava med gramatikami in grafi – lahko uporabimo pokritja grafov

3. Kriterij pokritja izpeljav – Derivation Coverage (DC): TR vsebuje vsak izpeljani niz gramatike G.

- V nasprotju z prvima dvema ogromno testov

Pokritje sintakse

Mutacija je sprememba pravil, opisov gramatike, sintakse, objektov, ki ima za posledico napačne (ali slučajno pravilne) nize

Testiranje gramatik

Izpeljava brez mutacij **Izpeljava z mutacijami**

Mutacija gramatike **Mutacija nizov**

Napačni nizi **Pravilni nizi**

Mutant: spremenjeni niz zaradi spremembe gramatike ali niza samega

Ideja mutacijskega testiranja: iz izvornega niza z mutacijami tvorimo večje število spremenjenih nizov - mutantov, s katerimi preverjamo učinkovitost delovanja (testiranja)

Pokritje sintakse

Vsaka mutacija je določena s:

- Pravili mutacije – kateri mutacijski operatorji

  (recimo „zamenjava aritmetičnih operatorjev“, ...)
- Sintakso besedila, ki ga testiramo

  (recimo za javanske programe opis jave z gramatiko)
- Sintakso spremembe

  (vsebina mutacijskih operatorjev, recimo zamenjava + v -, + v * ...
- Nad čim se izvaja

  (nad sintakso (gramatiko) ali objektom (nizom))

Pokritje sintakse

- **Mutacijski operator:** pravilo, ki določa, na kakšen način spremenimo izvorni niz v mutanta (recimo „Zamenjava spremenljivk istega tipa“)
- **Ubijanje mutanta:** mutanta ubijemo, če ga pri primerjavi z izvornim nizom lahko prepoznamo (ima drugačen odziv)
- Mutanti so lahko:
  - **Primerni** (pravilni, z ustreznimi testi jih lahko ubijemo)
  - **Napačni** (neuporabni),
  - **Trivialni** (pravilni, vsak test jih ubije)
  - **Ekvivalentni** (pravilni, noben test jih ne ubije)
- **Mutacijski rezultat:** delež ubitih mutantov (v %) izmed vseh

Pokritje sintakse

Konkreten primer mutacijskega testiranja:

- program v javi je izvorni niz
- z uporabo nabora mutacijskih operatorjev (recimo „zamenjava aritmetičnih operatorjev“, „zamenjava logičnih operatorjev“, „zamenjava spremenljivk istega tipa“) tvorimo večje število mutantov (spremenjenih programov, ki pa se prevedejo)
- Preverjamo obstoječi nabor testov – če je ta dober mora prepoznati (ubiti) **vse** mutante
- v praksi je to (zaradi ekvivalentnih mutantov) zelo težko, zato določamo delež ubitih mutantov (mutacijski rezultat) in na podlagi tega sodimo o kakovosti nabora testov

Pokritje sintakse

NA PODLAGI IZVEDBE MUTACIJ:

4. Kriterij pokritja mutacij – Mutation Coverage (MC): za vsakega mutanta velja, da TR vsebuje natanko eno zahtevo, ki ga ubije.

- Primerjamo izvedbo izvornega in mutiranega niza
- Mutirani niz mora biti ubit (prepoznan)
- RIP – doseg, sprememba stanja in drugačen odziv

5. Kriterij šibkega pokritja mutacij – Weak Mutation Coverage (WMC): za vsakega mutanta velja, da TR vsebuje natanko eno zahtevo, ki ga šibko ubije.

- Šibko ubije – pod določenimi predpostavkami lažja prepoznava
- RI - ni treba drugačnega odziva, zadostuje spremenjeno stanje

Pokritje sintakse

NA PODLAGI OBSEGA MUTACIJ

6. Kriterij pokritja operatorjev mutacij – Mutation Operator Coverage (MOC): za vsak operator mutacij, TR vsebuje natanko eno zahtevo; da tvori mutirani niz, ki je izpeljan z uporabo tega operatorja.

- Operator mutacij: pravilo, kako iz veljavnega niza tvorimo mutanta
- Nezahtevno – malo mutacij, za vsak operator mutacij ena

7. Kriterij pokritja produkcij mutacij – Mutation Production Coverage (MPC): za vsak operator mutacij, TR vsebuje več zahtev; da tvorijo mutirani niz, ki vsebuje vsako produkcijo za izpeljavo z uporabo tega operatorja.

- Veliko zahtevneje - vsak operator mutacij na vse možne načine

Pokritje sintakse

Mutacije izvajamo nad različnimi deli programske opreme:

- Nad programsko kodo:
  - Najpogosteje je niz program, mutacija pa spremenjen program
  - Preverjamo učinkovitost testov
- Nad načrti (ob integracijskem testiranju)
  - Mutiramo parametre pri klicih metod
  - OO – mutiramo določila dostopa, (dedovane) tipe.
- Nad različnimi modeli
  - Končni avtomati – mutiramo prehode, predpogoje
- Nad vhodnimi podatki
  - XML – mutiramo format podatkov

Pokritje sintakse

- Nad različnimi deli se uporabljajo različni mutacijski operatorji
- Ključno: dober nabor mutacijskih operatorjev
- Veliko raziskav, predlagano število mutacijskih operatorjev
  - javnska koda (11),
  - integracija metod (5),
  - integracija OO (20),
  - vhodni podatki (5)
- Na izvedbo testiranja (število in zahtevnost ubijanja mutantov) vpliva tudi kriterij pokritosti

Pokritje sintakse

Primer učinkovitih mutacijskih operatorjev za Javo:

1. ABS — Absolute Value Insertion
2. AOR — Arithmetic Operator Replacement
3. ROR — Relational Operator Replacement
4. COR — Conditional Operator Replacement
5. SOR — Shift Operator Replacement
6. LOR — Logical Operator Replacement
7. ASR — Assignment Operator Replacement
8. UOI — Unary Operator Insertion
9. UOD — Unary Operator Deletion
10. SVR — Scalar Variable Replacement
11. BSR — Bomb Statement Replacement

Pokritje sintakse

Obljuba mutacijskega testiranja: z uporabo učinkovitega nabora mutacijskih operatorjev odkrijemo večino napak

Ugotovitve raziskav:

- Primerno je mutante tvoriti tako, da uporabimo **le eno mutacijo**
- Mutacije so **zelo učinkovite**, če je izbran dober nabor mutacijskih operatorjev
- Dobro zastavljeno mutacijsko testiranje **vsebuje številna druga pokritja** (NC, EC, CC, ACC, AllDefsC)
- Mutacije so **drage** (veliko mutantov, testiranj), težko jih izvajamo ročno
- Obvezno **avtomatiziramo** njihovo uporabo (orodja, recimo MUjava)
- Z mutacijami **odkrijemo tudi napake**, ki jih drugi pristopi **ne znajo**
- Uporabljamo jih **za presojo**, če drugi kriteriji zares delujejo

Literatura

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