Strategije testiranja toka podataka

Zašto nam treba analiza toka podataka?

Analiza toka kontrole

- Koji iskazi su izvršeni?
- Koje grane odluke su pređene?
- Kojom putanjom se stiglo do izlaza?

Analiza toka podataka

- Gde je promenljiva dobila vrednost?
- Gde se ta vrednost koristi?
- Može li druga dodela da je zameni pre upotrebe?

DEF(S) i USE(S): dva skupa za svaki iskaz

Analiziramo promenljive čije se vrednosti dodeljuju ili čitaju u datom iskazu.

DEF(S)

Skup promenljivih kojima iskaz S dodeljuje vrednost.

USE(S)

Skup promenljivih čije vrednosti iskaz S čita ili koristi.

Primeri

```
a = b;
```

DEF = {a} USE = {b}

```
a = a + b;
```

DEF = {a} USE = {a, b}

Napomena: u `a = a + b` stara vrednost a se čita pre nego što se dodeli nova vrednost promenljivoj a.

Dve vrste upotrebe: c-use i p-use

Razlika je u ulozi koju vrednost promenljive ima u iskazu.

c-use — računska upotreba

Vrednost se koristi u računanju ili u izrazu koji proizvodi vrednost.

```
z = x + 2;
Math.sqrt(d)
```

p-use — predikatska upotreba

Vrednost se koristi u uslovu koji kontroliše tok izvršavanja.

```
if (d < 0.0)
if (!isComplex)
```

Pitanje za iskaz **„Da li vrednost učestvuje u računanju — ili odlučuje kojom granom program nastavlja?“**

Ista promenljiva može imati više upotreba, i to različitog tipa, na različitim mestima programa.

Primer

Metod vraća sumu svih parnih brojeva u listi, ali prekida I vraća -1 ako naiđe na negativan broj.

Line	Code	Variable `total` Status	Variable `index` Status
	`def sum_evens(numbers):`		
1	`  total = 0`	**Def** (Initial)	-
2	`  index = 0`	-	**Def** (Initial)
3	`  while index < len(numbers):`	-	**P-Use** (Loop condition)
4	`    val = numbers[index]`	-	**C-Use** (Array index)
5	`    if val < 0:`	-	-
6	`      total = -1`	**Def** (Negative flag)	-
7	`      break`	-	-
8	`    if val % 2 == 0:`	-	-
9	`      total = total + val`	**C-Use** (Right side) & **Def** (Left side)	-
10	`     index = index + 1`	-	**C-Use** (Right side) & **Def** (Left side)
11	`  return total`	**C-Use** (Return value)	-

DU lanac: od definicije do upotrebe

Notacija [X, S, S1], gde su S i S1 iskazi, a X promenljiva.

[X, S, S1] ⇔ X ∈ DEF(S), X ∈ USE(S1), i vrednost iz S stiže do S1 bez redefinicije X

1 Definicija S

Iskaz S dodeljuje vrednost promenljivoj X.

2 Definicija ostaje živa

Postoji putanja od S do S1 bez nove dodele X.

3 Upotreba S1

Iskaz S1 čita vrednost promenljive X.

Putanja između S i S1 mora biti bez redefinicije promenljive X. Ne zahteva se da ne sadrži druge iskaze ili odluke.

DU parovi (lanci)

DU lanac čini putanja od definicije promenljive do linije gde se upotrebljava, bez redefinicije iste promenljive duž putanje

```
def sum_evens(numbers):
    1: total = 0
    2: index = 0
    3: while index < len(numbers):
    4:     val = numbers[index]
    5:     if val < 0:
    6:         total = -1
    7:         break
    8:     if val % 2 == 0:
    9:         total = total + val
   10:     index = index + 1
   11: return total
```

For total:

Pair 1: Def at Line 1 → C-Use at Line 9 (First loop iteration, even number)

Pair 2: Def at Line 1 → C-Use at Line 11 (Empty list loop skip, or list has only odd numbers)

Pair 3: Def at Line 6 → C-Use at Line 11 (Broke out due to negative number)

Pair 4: Def at Line 9 → C-Use at Line 9 (Subsequent even numbers adding up)

Pair 5: Def at Line 9 → C-Use at Line 11 (Loop ends normally after accumulating evens)

For index:

Pair 6: Def at Line 2 → P-Use at Line 3 (First loop condition check)

Pair 7: Def at Line 2 → C-Use at Line 4 (First array access)

Pair 8: Def at Line 10 → P-Use at Line 3 (Subsequent loop checks)

Pair 9: Def at Line 10 → C-Use at Line 4 (Subsequent array accesses)

Koji testovi aktiviraju ove definicije?

```
def sum_evens(numbers):
    1: total = 0
    2: index = 0
    3: while index < len(numbers):
    4:     val = numbers[index]
    5:     if val < 0:
    6:         total = -1
    7:         break
    8:     if val % 2 == 0:
    9:         total = total + val
   10:     index = index + 1
   11: return total
```

Test Case A: numbers = [] (Empty list)

Test Case B: numbers = [3, -5] (Odd, then negative)

Test Case C: numbers = [2, 4] (All evens)

Šta mora biti pokriveno testiranjem? Sedam strategija Rapps–Weyuker

1 Sve definicije
Od svake definicije do najmanje jedne upotrebe.

2 Sve c-upotrebe
Od svake definicije do svih njenih c-use.

3 Sve c-use, neke p-use
Svi c-use; ako ih nema, bar jedan p-use.

4 Sve p-upotrebe
Od svake definicije do svih njenih p-use.

5 Sve p-use, neke c-use
Svi p-use; ako ih nema, bar jedan c-use.

6 Sve upotrebe
Sve dostupne c-use i p-use svake definicije.

7 Svi DU lanci / putanje
Sve jednostavne DU-putanje do svih upotreba.

Kriterijumi data flow testiranja, I deo

```
def sum_evens(numbers):
    1: total = 0
    2: index = 0
    3: while index < len(numbers):
    4:     val = numbers[index]
    5:     if val < 0:
    6:         total = -1
    7:         break
    8:     if val % 2 == 0:
    9:         total = total + val
   10:     index = index + 1
   11: return total
```

Kriterijum	Cilj / Zahtev	Minimalni skup testova	Komentar
All-Defs	Od svake definicije do **bar jedne** upotrebe.	**Test B i Test C**	**Test B** pokriva Def na Liniji 6 (`total=-1`). **Test C** pokriva Def na Liniji 1 (`total=0`), Liniji 9 (`total+val`), Liniji 2 (`index=0`), i Liniji 10 (`index+1`). Sve definicije su pokrivene.
All-P-Uses	Od svake definicije do **svih** predikatskih upotreba	**Test B** i **Test C**	Fokus na promenljive odlučivanja. Definicije na linijama 2 i 10 moraju stići do uslova petlje na Liniji 3. Oba Testa B i C zadovoljavaju ovo.
All-C-Uses	Od svake definicije do **svih** računskih upotreba	**Test A**, **Test B** i **Test C**	Svaka računaska upotreba mora biti pokrivena. **Test A** obezbeđuje da Def na Liniji 1 dolazi do return na Liniji 11 ne izvršavajući Liniju 9. **Test C** obeybeđuje da Def na Liniji 1 dolazi do Linije 9.

Test A: numbers = [] (Empty list) **Test B**: numbers = [3, -5] (Odd, then negative) **Test C**: numbers = [2, 4] (All evens)

Sve DU putanje (All-DU-paths)

Ne proverava se samo da je upotreba dostignuta; zahtev se odnosi na svaku jednostavnu DU-putanju od definicije do upotrebe.

All-uses

Pokriva svaku zahtevanu vezu definicija → upotreba. Dve različite putanje koje ostvaruju istu vezu ne moraju obe biti pokrivene.

All-DU-paths

Pokriva svaku zahtevanu jednostavnu putanju između definicije i upotrebe. Više putanja može nastati zbog grananja ili petlji.

Posledica

100% pokrivenost svih DU-putanja može biti nedostižna ako postoje neizvodljive putanje.

Praktičan pristup

Analizirati neizvodljivost i navesti izuzete zahteve, umesto da se prikriva nedostižna pokrivenost.

Kriterijumi data flow testiranja, II deo

```
def sum_evens(numbers):
    1: total = 0
    2: index = 0
    3: while index < len(numbers):
    4:     val = numbers[index]
    5:     if val < 0:
    6:         total = -1
    7:         break
    8:     if val % 2 == 0:
    9:         total = total + val
   10:     index = index + 1
   11: return total
```

Kriterijum	Cilj / Zahtev	Minimalni skup testova	Komentar
All-Uses	Od svih definicija do svih njenih C-upotreba i P-upotreba.	**Test A**, **Test B** i **Test C**	Ovo je kombinacija prethodna dva. Svih 9 DU parova sa slajda 7 mora biti izvršeno. Ovo je postignuto testovima A, B, i C.
All-DU-Paths	Svaka **jednostavna putanja** između svake Def i Use mora se testirati.	**Test A**, **Test B**, **Test C**, i `numbers = [1, 3]`	Ovaj kriterijum je najiscpniji. Razmotrimo **Pair 2** (Def na Lin 1 → Use na Lin 11). Postoje dve jednostavne putanje: potpuno preskočiti petlju (**Test A**) i prolaz kroz petlju samo za neparne brojeve (`[1, 3]`). All-DU-Paths zahteva pokrivanje obe ove putanje.

Test A: numbers = [] (Empty list) **Test B**: numbers = [3, -5] (Odd, then negative) **Test C**: numbers = [2, 4] (All evens)

Šta nam analiza toka podataka govori — a šta ne?

Može da otkrije / organizuje

- Definiciju koja nikad ne stiže do upotrebe
- Upotrebu koja zavisi od određene grane
- Redefiniciju koja prekida DU vezu
- Koje veze definicija–upotreba treba testirati

Ne garantuje samo po sebi

- Da je očekivani rezultat ispravan
- Da su asercije adekvatne
- Da ne postoje greške u algoritmu
- Da alat za običnu pokrivenost meri DU lance