Data Flow Testing

Dr. Durga Prasad Mohapatra
Professor
CSE Department
NIT Rourkela

Introduction

- In path coverage, the emphasis was to cover a path using statement or branch coverage.

- However, data and data integrity are as important as code and code integrity of a module.

- We have checked every possibility of the control flow of a module. But what about the data flow in the module?

- These questions can be answered,, if we consider data objects in the control flow of a module.

Introduction cont…

- Data flow testing is a white-box technique that can be used to detect improper use of data values due to coding errors.
- Errors may be unintentionally introduced in a program by programmers,
    - e.g. a programmer might use a variable without defining it.
- Data flow testing gives a chance to look out for
    - inappropriate data definition, their use in predicate, computations, and termination.

Introduction cont…

- It identifies the potential bugs
  - by examining the patterns in which that piece of data is used.

- Example: If an out-of-scope data is being used in a computation, then it is a bug. There may be several patterns like this which indicate data anomalies.

Introduction cont…

- To examine the patterns, the control flow graph of a program is used.

- This test strategy selects the paths in the module's control flow such that various sequences of data objects can be chosen.

- The major focus is on the points at which the data receives values and the places at which the data initialized has been referenced.

- Thus, we have to choose enough paths in the control flow to ensure that every data is initialized before use and all the defined data have been used somewhere.

Data Flow-Based Testing

- Selects test paths of a program:
  - According to the locations of
    - Definitions and Uses of different variables in a program.

Example

```
 1   X(){
2     int a=5; /* Defines variable a */
        ….
 3    While(c>5) {
 4     if (d<50)
5              b=a*a;   /*Uses variable a */
6               a=a-1; /* Defines variable a */
…
 7     }
 8     print(a); } /*Uses variable a */
```

Data Flow-Based Testing cont …

- For a statement numbered S,
  - DEF(S) = {X / statement S contains a definition of X}
  - USES(S)= {X / statement S contains a use of X}
  - Example: **1: a=b;** DEF(1)={a}, USES(1)={b}.
  - Example: 2: **a=a+b;** DEF(1)={a}, USES(1)={a,b}.

Data Flow-Based Testing cont …

- A variable X is said to be live at statement S1, if
  - X is defined at a statement S, and
  - there exists a path from S to S1 not containing any definition of X.

DU Chain Example

```
X(){
 int a=5; /* Defines variable a */
 While(c>5) {
   if (d<50)
       b=a*a;   /*Uses variable a */
       a=a-1; /* Defines variable a */
   }
  print(a); } /*Uses variable a */
```

Definition-use chain (DU chain)

- [X,S,S1],
  - S and S1 are statement numbers,
  - $X \in$ DEF(S),
  - $X \in$ USES(S1), and
  - the definition of X in the statement S is live at statement S1.

Data Flow-Based Testing Strategy

- One simple data flow testing strategy:
  - **Every DU chain in a program be covered at least once.**
- Data flow testing strategies:
  - Useful for selecting test paths of a program containing nested if and loop statements.

Example

```
X(){
 B1;      /* Defines variable a */
 While(C1) {
    if (C2)
          if(C4) B4; /*Uses variable a */
          else B5;
         else  if (C3) B2;
          else B3;     }
 B6 }
```

Example cont …

- [a,1,5]: a DU chain.
- Assume:
    - DEF(X) = {B1, B2, B3, B4, B5}
    - USES(X) = {B2, B3, B4, B5, B6}
    - There are 25 DU chains.
- However only 5 paths are needed to cover these chains.

Data Flow Testing cont…

- It also closely examines the state of the data in the CFG resulting in a richer test suite
  - than the one obtained from CFG based path testing strategies like statement coverage, branch coverage, etc.

States of a Data Object

- Defined (d):
- Killed / Undefined / Released (k):
- Usage (u):
- Computational use (c-use) or
- Predicate use (p-use).

State of a Data Object cont …

A data object can be in the following states:

- ***Defined (d)*** A data object is called defined when it is initialized, i.e., when it is on the left side of an assignment statement. Defined state can also be used to mean that a file has been opened, a dynamically allocated object has been allocated, something is pushed onto the stack, a record written, and so on.

State of a Data Object cont…

- ***Kill/Undefined/Released (k)***

- When the data has been reinitialized or the scope of a loop control variable finishes, i.e., exiting the loop or memory is released dynamically or a file has been closed.

State of a Data Object cont…

- ***Usage (u)*** When the data object is on the right side of assignment or the control variable in a loop, or in an expression used to evaluate the control flow of a case statement, or as a pointer to an object, etc.
- In general, we say that the usage is either computational use (c-use) or predicate use (p-use).

Data-Flow Anomalies

- Data-flow anomalies represent the patterns of data usage which may lead to an incorrect execution of the code.
- An anomaly is denoted by a two-character sequence of actions.

Data-Flow Anomalies cont…

- Example: ‘dk’ means a variable is defined and killed without any use, which is a potential bug.

- There are nine possible two-character combinations out of which only four are data anomalies, as shown in next Table.

Table 1: Two-character data-flow anomalies

Anomaly	Explanation	Effect of Anomaly
du	Define-use	Allowed, Normal case.
dk	Define-Kill	Potential bug. Data is killed without use after definition.
ud	Use-define	Data is used and then redefine. Allowed, Usually not a bug because the language permits reassignment at almost any time.
uk	Use-Kill	Allowed, Normal situation.
ku	Kill-use	Serious bug because the data is used after being killed.
kd	Kill-define	Data is Killed and then redefined, Allowed
dd	Define-define	Redefining a variable without using it . Harmless bug, but not allowed.
uu	Use-use	Allowed Normal case.
kk	Kill-kill	Harmless bug, but not allowed.

- Not all data-flow anomalies are harmful, but most of them are suspicious and indicate that an error can occur.
- There may be single-character data anomalies also.
- To represent these types of anomalies, we take the following conventions:

  - ~x: indicates all prior actions are not of interest to x.
  - x~: indicates all post actions are not listed of interest to x.

Table 2: Single-character data-flow anomalies

Anomaly	explanation	Effect of Anomaly
~d	First definition	Normal situation, Allowed.
~u	First Use	Data is used without defining it. Potential bug.
~k	First Kill	Data is killed before defining it, Potential bug.
D~	Define last	Potential bug.
U~	Use last	Normal case, Allowed.
K~	Kill last	Normal case, Allowed.

Some Terminologies

Suppose P is a program that has a graph G (P) and a set of variables V. The graph has a single entry and exit node.

- ***Definition node*** Defining a variable means assigning value to a variable for the very first time in a program. For example, input statements, assignment statements, loop control statements, procedure calls, etc.

Some Terminologies contd…

- ***Usage node*** It means the variable has been used in some statement of the program. Node n that belongs to G(p) is usage node of variable *v*, if the value of variable v is used at the statements corresponding to node n.

Some Terminologies contd…

- A usage node can be of the following two types:

1) Predicate usage Node: If usage node n is a predicate node, then **n** is a predicate usage node.
2) Computation Usage Node: If usage node **n** corresponds to a computation statement in a program other than predicate, then it is called a computation usage node.

Some Terminologies contd…

- ***Loop-free path segment*** It is a path segment for which every node is visited once at most.

- ***Simple path segment*** It is a path segment in which at most one node is visited twice. A simple path segment is either loop-free or if there is a loop, only one node is involved.

- ***Definition-use path (du-path)*** A du-path with respect to a variable v is a path between the definition node and usage node of that variable, Usage node can either be a p-usage or a c-usage node.

Some Terminologies contd…

- ***Definition-clear path (dc-path)*** A dc-path with respect to a variable v is a path between the definition node and the usage node such that no other node in the path is a defining node of variable *v*.

- The du paths which are not dc paths are important, as these are potential spots for testing persons.

- Those du-paths which are definition-clear are easy to test in comparison to du-paths which are not dc-paths.

- The du-paths which are not dc-paths need more attention.

Static Data Flow Testing

With static analysis, the source code is analysed without executing it.

EXAMPLE:

Consider a program for calculating the gross salary of an employee in an organization. If his basic salary < 1500, then HRA=10% of the Basic and DA=90% of basic. If his salary >= 1500, then HRA=500 and DA=98% of basic. Calculate the gross salary.

```
main()
{
  float bs, gs, da, hra=0;
  printf(“Enter basic salary”);
  scanf(“%f”,&bs);
  if(bs < 1500)
  {
      hra=bs * 10/100;
      da= bs * 90/100;
  }
  else
  {
       hra = 500;
       da= bs * 98/100;
  }
   gs= bs+ hra+ da;
   printf(“Gross Salary =  Rs. %f”, gs);
  }
```

Find out the define-use-kill patterns for all the variables in the program

Solution

Pattern	Line Number	Explanation
~d	3	Normal case. Allowed
du	3-4	Normal case. Allowed
uu	4-6,6-7,7-12,12-14	Normal case. Allowed
uk	14-16	Normal case. Allowed
K~	16	Normal case. Allowed

Define-use-kill patterns for variable ‘bs’

Solution cont…

Pattern	Line Number	Explanation
~d	14	Normal case.Allowed
du	14-15	Normal case.Allowed
uk	15-16	Normal case.Allowed
K~	16	Normal case.Allowed

Define-use-kill patterns for variable ‘gs’

Solution cont…

Pattern	Line Number	Explanation
~d	7	Normal case. Allowed
du	7-14	Normal case. Allowed
uk	14-16	Normal case. Allowed
K~	16	Normal case. Allowed

Define-use-kill patterns for variable ‘da’

Solution cont…

Pattern	Line Number	Explanation
~d	1	Normal case. Allowed
dd	1-6 or 1-11	Double definition. Not allowed. Harmless bug.
du	6-14 or 11-14	Normal case. Allowed
uk	14-16	Normal case. Allowed
K~	16	Normal case. Allowed

Define-use-kill patterns for variable ‘hra’

From the above static data flow testing, only one bug is found, i.e in variable HRA of double definition.

Summary

- Discussed the basic concepts of data flow testing.
- Explained DU Chain.
- Presented the different states of a data object.
- Explained the different data-flow anomalies.
- Explained static data flow testing with an example.

References

1. Rajib Mall, Fundamentals of Software Engineering, (Chapter – 10), Fifth Edition, PHI Learning Pvt. Ltd., 2018.
2. Naresh Chauhan, Software Testing: Principles and Practices, (Chapter – 5), Second Edition, Oxford University Press, 2016.

Thank you