Ch2 Code Unit Test

Automatic Unit Tests Design(2)

Instructor: Haiying SUN

E-mail: hysun@sei.ecnu.edu.cn

Office: ECNU Science Build B1104

Available Time: Wednesday 8:00 -12:00 a.m.

Overview

- Control Based Tests Generation
  - Prime Path Testing
- Data Flow Based Testing
  - Data Flow Graph (DFG)
  - Data Flow Testing Coverage Criteria
- Mutation Testing

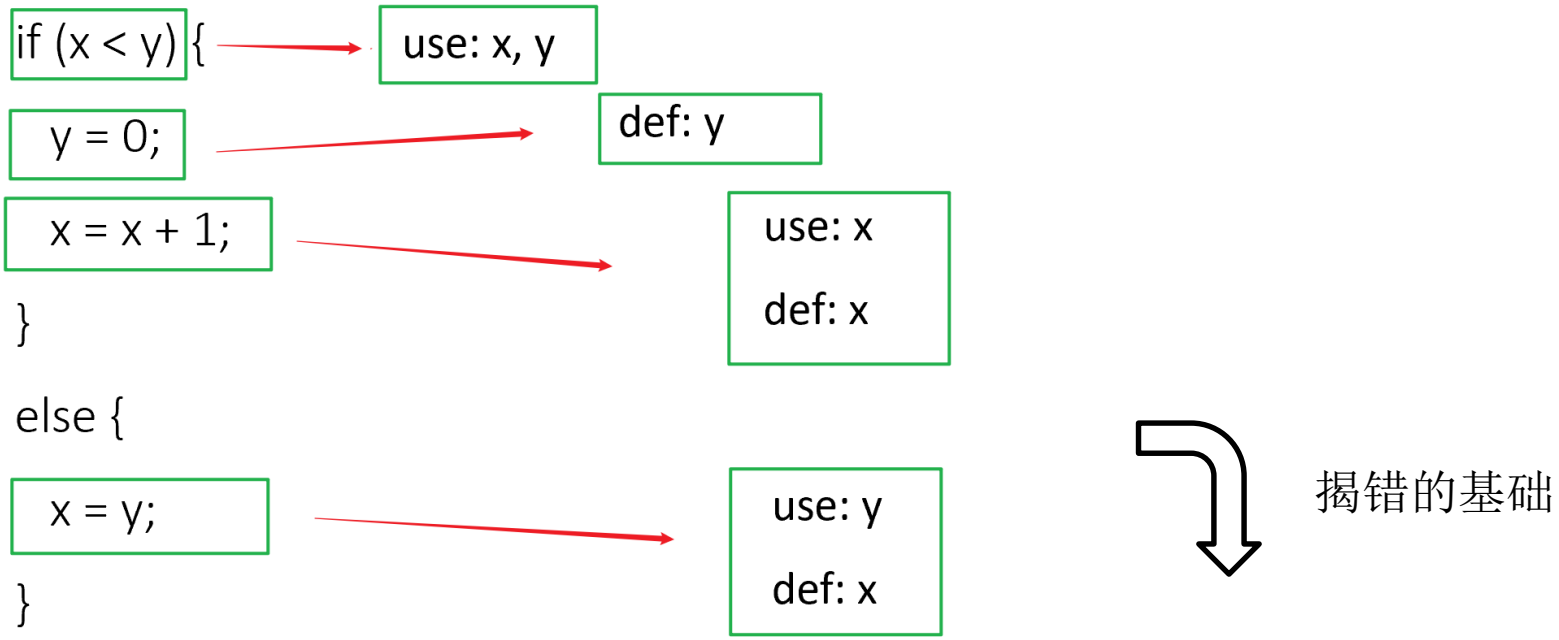
Def and Use

- def : a location where a value **is stored into memory**
  1. x appears on the left side of an assignment (x = 44;)
  2. x is an input to a program
  3. It should be considered carefully when
     - x is an actual parameter in a call and the method changes its value
     - x is a formal parameter of a method (implicit def when method starts)
- use : a location where variable's value **is accessed**
  1. x appears on the right side of an assignment
  2. x appears in a conditional test
  3. x is an actual parameter to a method
  4. x is an output of the program
  5. x is an output of a method in a return statement
- If a def and a use appear on the same node, then it is not only a Def node but also a use node

Def and Use

- c-use (computation-use)
  - 使用节点USE(v,n)是一个计算使用(记做C-use)，当且仅当语句n是计算语句 （对于计算使用的节点永远有外度=1）
- p-use (predicate-use)
  - 使用节点是一个谓词使用（记做p－use），当且仅当语句n是谓词语句（对于谓词使用的节点永远有外度≥2）

Def and Use



The values given in Defs should reach at least one, some, or all possible Uses

Exercise

1、哪些语句是**变量str**的定义节点，哪些是使用节点？

2、哪些语句是**变量p**的定义节点，哪些是使用节点？

1. str是指针变量，其定义节点: 12，13（赋值左侧str）；使用节点: 13（赋值右侧str），14，15。
2. p是指针变量，其定义节点： 6， 7；使用节点：8。

```
1   #include <iostream>
2   #include <cstring>
3   #include<malloc.h>
4   using  namespace std;
5
6   char*  get_mry(char  *p){
7     p=(char*)malloc(100);
8     return p;
9   }
10
11  int main () {
12      char * str =NULL;
13      str = get_mry(str);
14      strcpy(str, "hello,world");
15      cout<<"str:"<< str << endl;
16      return 0;
17  }
```

代码有缺陷!!

Exercise

1、哪些语句是**变量str**的定义节点，哪些是使用节点？

2、哪些语句是**变量p**的定义节点，哪些是使用节点？

1. str是指针变量，其定义节点：11，12；使用节点：12，13，14。
2. p是指针的指针，其定义节点：6；使用节点: 7

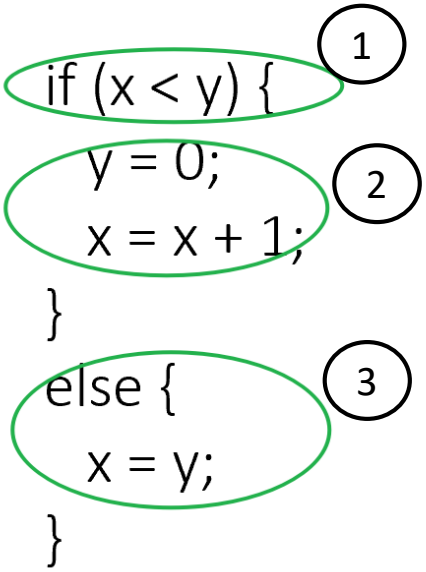
```
1   #include <iostream>
2   #include <cstring>
3   #include<malloc.h>
4   using  namespace std;
5
6   void  get_mry(char  **p){
7     *p=(char*)malloc(100);
8   }
9
10  int main () {
11      char * str =NULL;
12      get_mry(&str);
13      strcpy(str, "hello,world");
14      cout<<"str:"<< str << endl;
15      return 0;
16  }
```

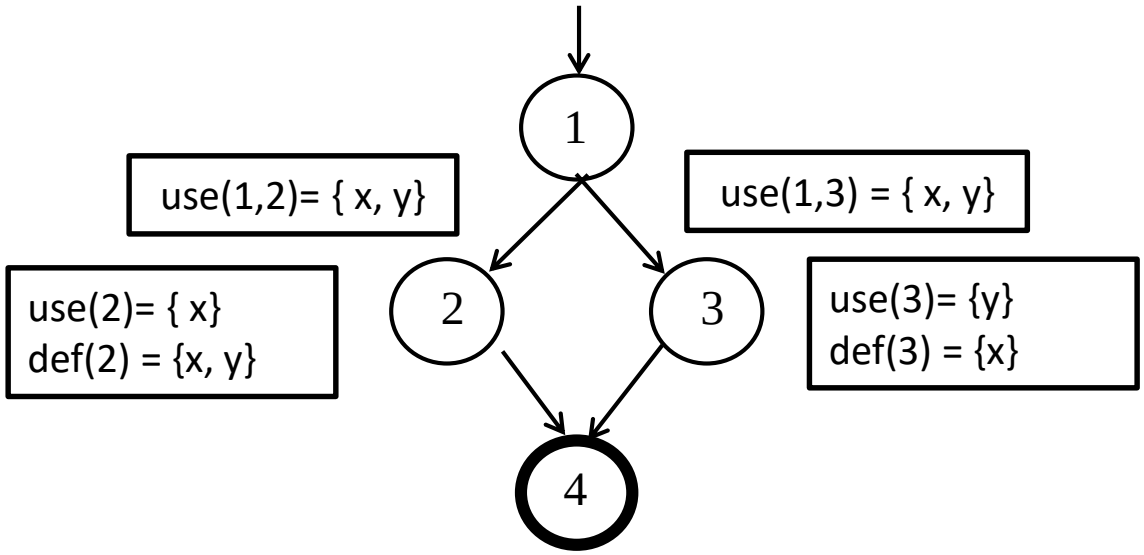
代码有缺陷!!

Data Flow Graph

- Given a CFG: (N, N_0, N_f, E), if def (n) or def (e) denotes the set of variables that are defined by node n or edge e, use (n) or use (e) denotes the set of variables that are used by node n or edge e, then the Data Flow Graph (DFG) can be defined as a tuple: (N_D, N_0, N_f, E_D), where
  1. $N_D = \{ (n, def(n) \cup use(n)) \mid n \in N \}$
  2. $E_D = \{ (e, def(e) \cup use(e)) \mid e \in E \}$

Data Flow Graph





N_D = { (1, ϕ), (2, def(2) U use(2)), (3, def(3) U use(3)), (4, ϕ) }
N_0 = { 1 }
N_f = { 4 }
E_D = { ((1,2), use(1,2)), ((1,3), use(1,3)), ((2,4), ϕ), ((3,4), ϕ) }

Data Flow Base Test Generation

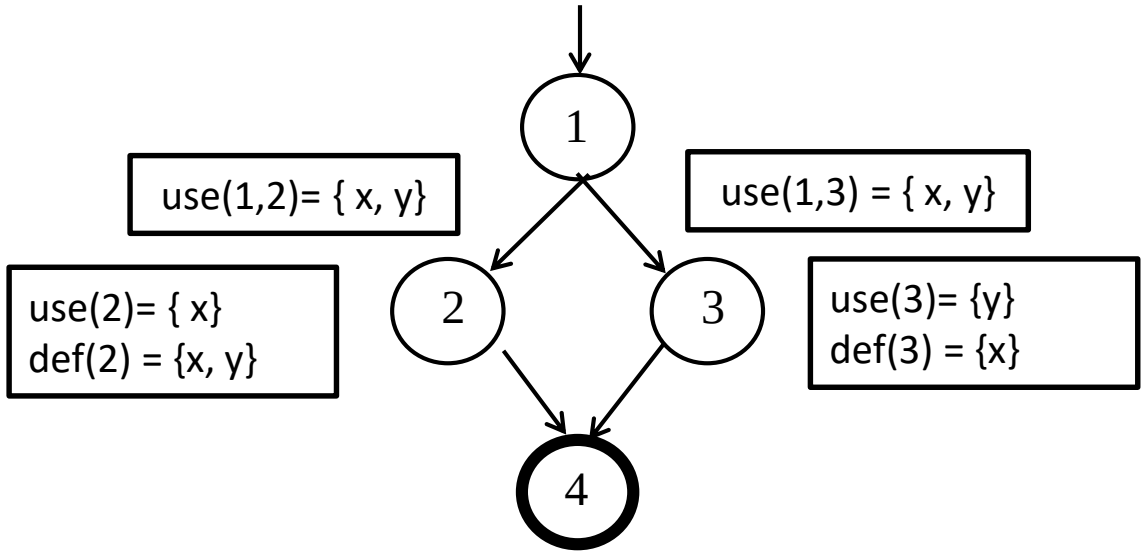
- 控制流测试设计的思想
  - 执行程序路径到达期望的**控制覆盖准则**
- 数据流测试设计的思想
  - 执行程序路径到达期望的**数据流覆盖准则**
- 回顾概念
  1. Path：路径
  2. Simple Path：简单路径
  3. Prime Path：基路径
  4. Complete Path：完整路径

DU Pairs and DU Paths

- du pair : A pair of locations (l_i, l_j) such that a variable v is defined at l_i and used at l_j
- def-clear : A path from l_i to l_j is *def-clear* with respect to variable *v* if *v* is not given another value on any of the nodes or edges in the path
- du-path : A **simple** subpath that is def-clear with respect to *v* from a def of *v* to a use of *v*
  1. du (n_i, n_j, *v*) : the set of du-paths that start at n_i and end at node n_j
  2. du (n_i, v) : the set of du-paths that start at n_i

DU Pairs and DU Paths

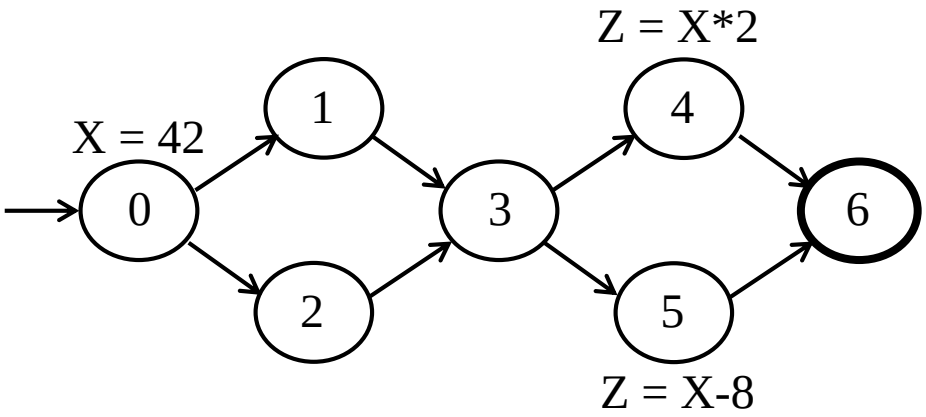
```
1  if (x < y) {
2    y = 0;
     x = x + 1;
   }
3  else {
     x = y;
   }
```



du Pairs for Variable y: ϕ

du Pairs for Variable x: ϕ

DU Pairs and DU Paths



du paths for *X*
[0, 1, 3, 4]
[0, 2, 3, 4]
[0, 1, 3, 5]
[0, 2, 3, 5]

du(0,4,x)

du(0,5,x)

du pairs for *X*
(0, 4)
(0, 5)

du(0, x)

du paths for *X*
[0, 1, 3, 4]
[0, 2, 3, 4]
[0, 1, 3, 5]
[0, 2, 3, 5]

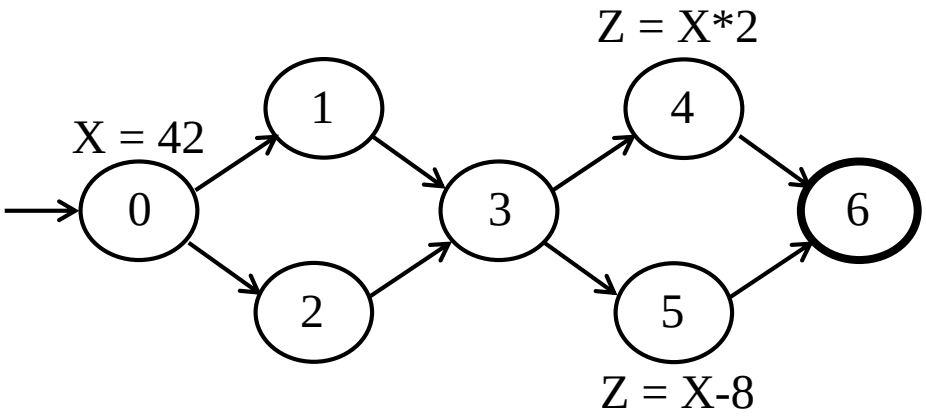
Fundamental Data Flow Test Criteria

- All-defs coverage (ADC) : For each variable v and its set of du-paths S = du (n_i, v), TR contains at least one path d in S.
  - every def reaches a use
- All-uses coverage (AUC) : For each variable v and its each set of du-paths to uses S = du (n_i, n_j, v), TR contains at least one path d in S.
  - every def reaches all possible uses

Fundamental Data Flow Test Criteria

- All-du-paths coverage (ADUPC) : For each varialbe v and its each set S = du (ni, nj, v), TR contains every path d in S.
  - cover all the paths between defs and uses

Example



All-defs for *X*
[0, 1, 3, 4]

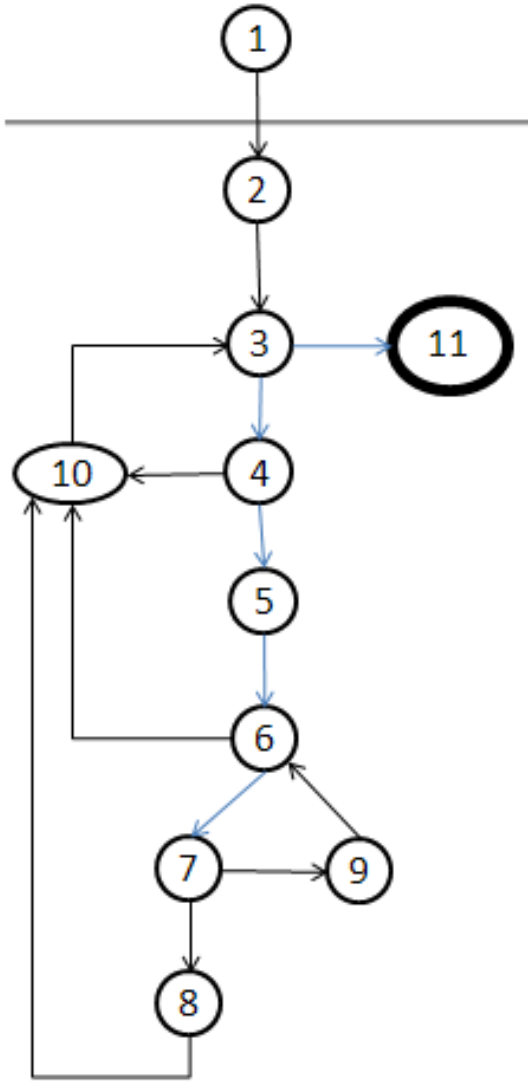
用于测试生成路径：

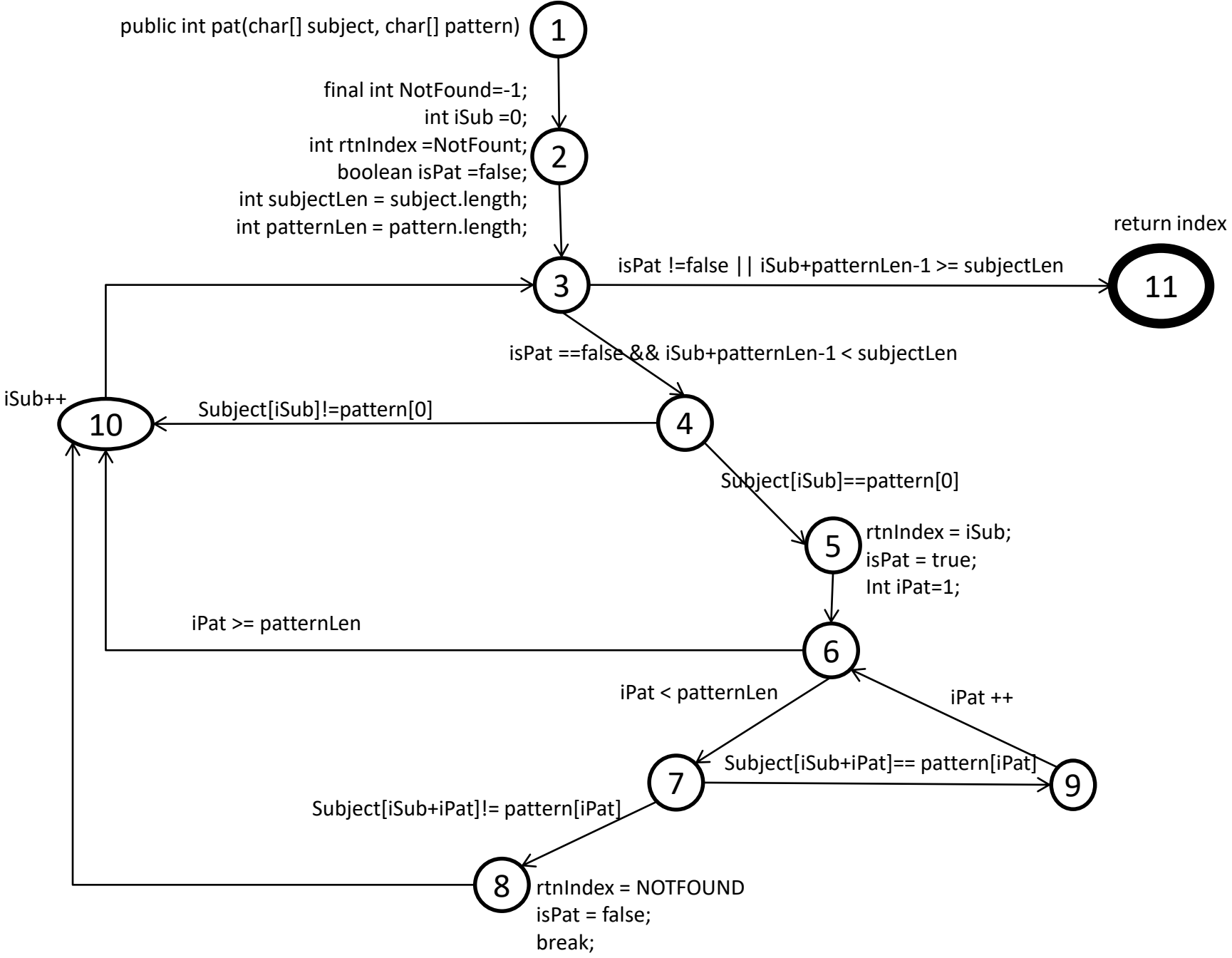
[0, 1, 3, 4, 6]

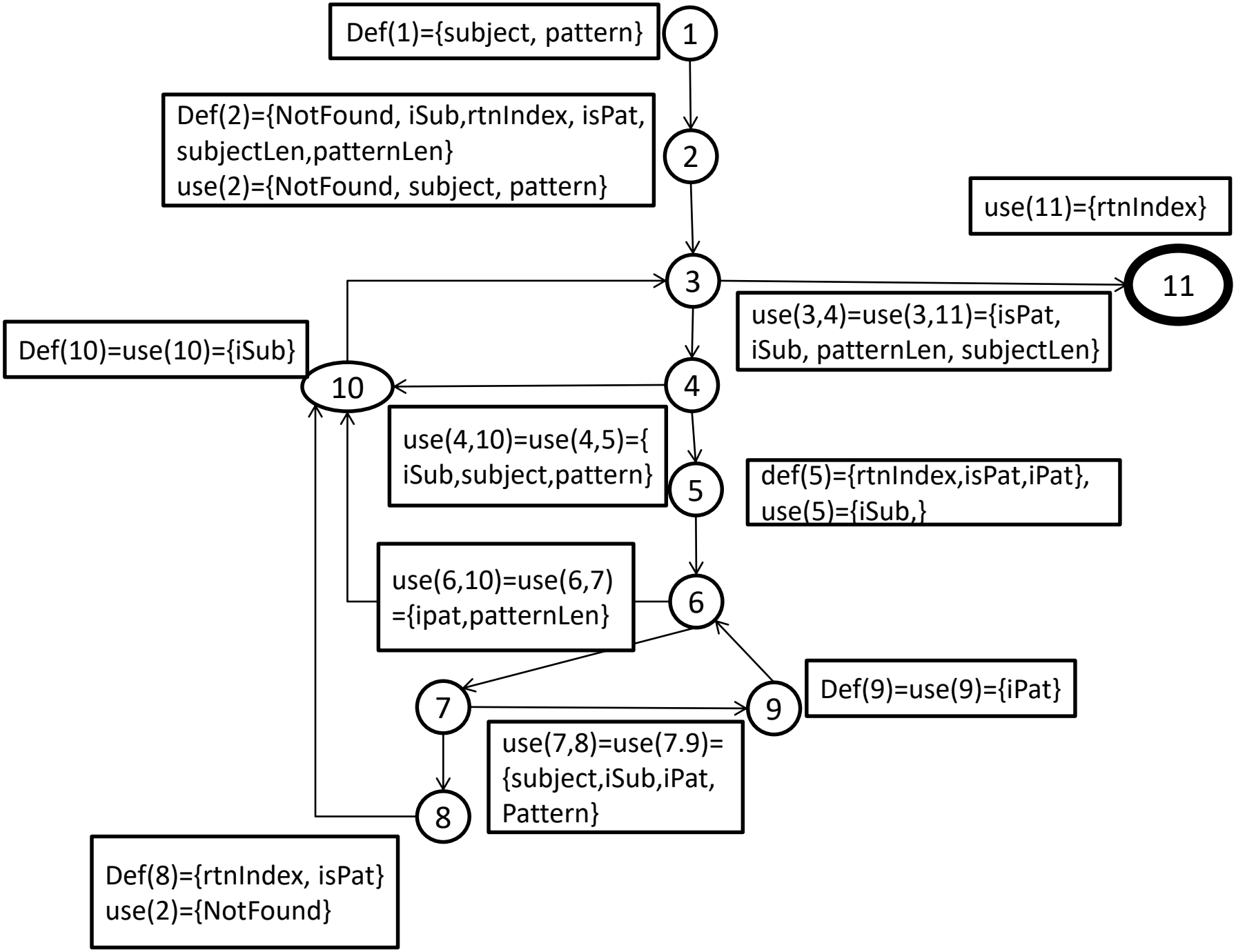
All-uses for *X*
[0, 1, 3, 4]
[0, 1, 3, 5]

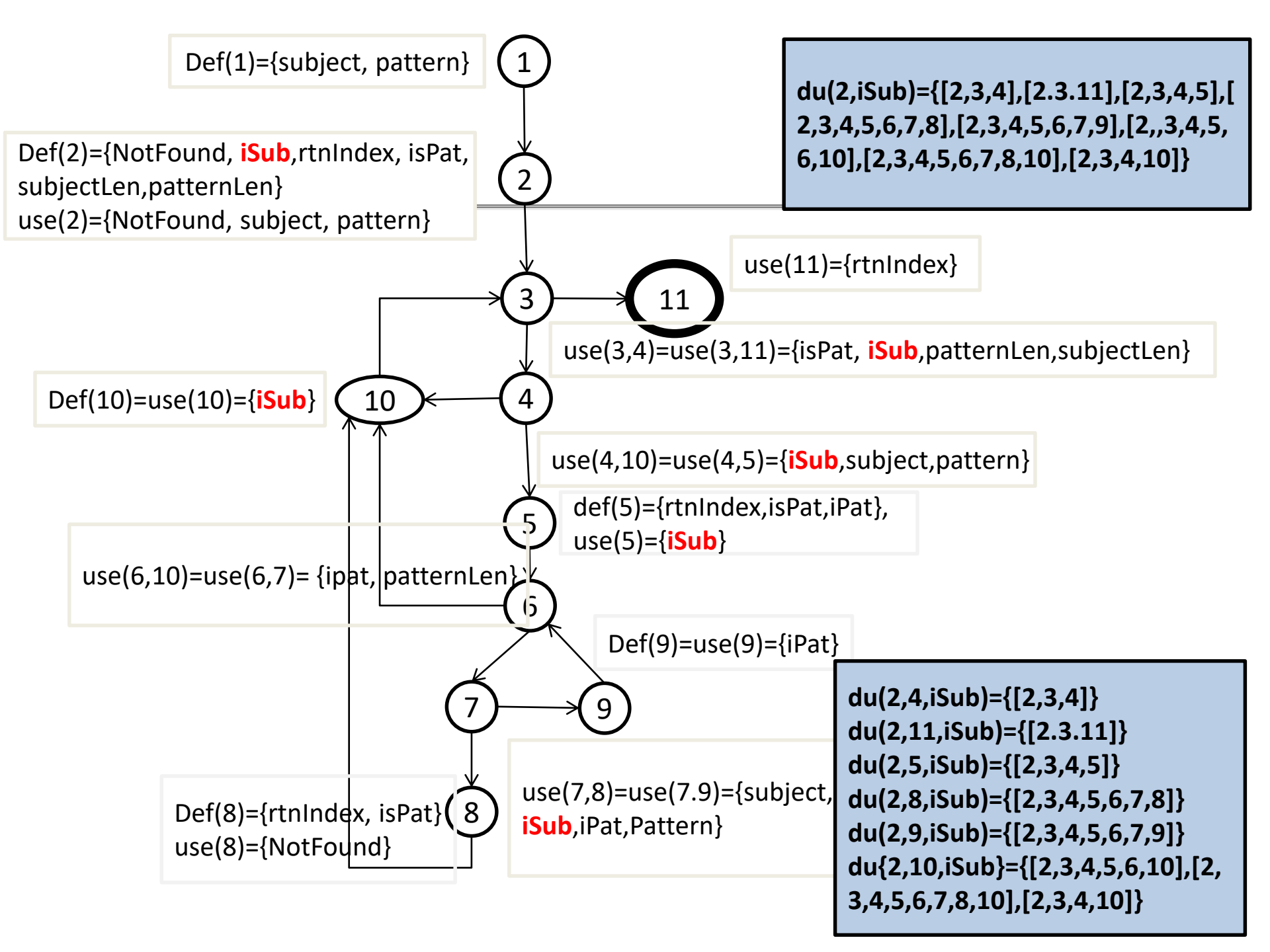
All-du-paths for *X*
[0, 1, 3, 4]
[0, 2, 3, 4]
[0, 1, 3, 5]
[0, 2, 3, 5]

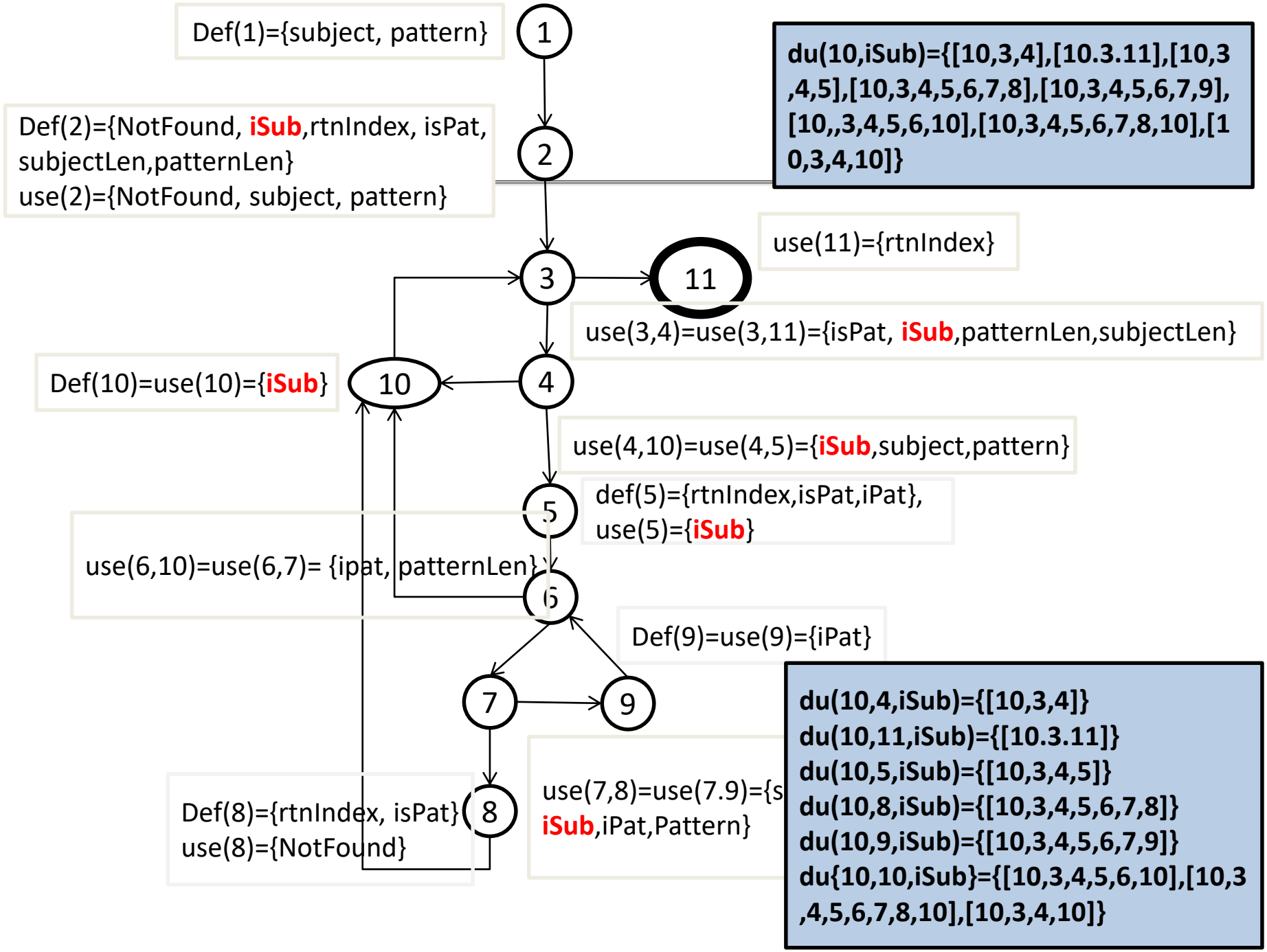
```
1  public int pat(char[] subject, char[] pattern){
       final int NotFound = -1;
2      int iSub = 0;
       int rtnIndex = NotFound;
       boolean isPat = false;
       int subjectLen = subject.length;
       int patternLen = pattern.length;

3      while(isPat == false && iSub + patternLen-1 < subjectLen ){
4          if(subject[iSub]==pattern[0]){
               rtnIndex=iSub;
5              isPat=true;
6              for(int iPat = 1; iPat<patternLen; iPat++){
7                  if(subject[iSub+iPat]!=pattern[iPat]){
8                      rtnIndex = NotFound;
                       isPat = false;
                       break;
                   }
               }
           }
10         iSub++;
       }
11     return rtnIndex;
   }
}
```




public int pat(char[] subject, char[] pattern)
1
final int NotFound=-1;
int iSub =0;
int rtnIndex =NotFount;
boolean isPat =false;
int subjectLen = subject.length;
int patternLen = pattern.length;
2
return index
3
isPat !=false || iSub+patternLen-1 >= subjectLen
11
isPat ==false && iSub+patternLen-1 < subjectLen
iSub++
10
Subject[iSub]!=pattern[0]
4
Subject[iSub]==pattern[0]
5
rtnIndex = iSub;
isPat = true;
Int iPat=1;
iPat >= patternLen
6
iPat < patternLen
iPat ++
7
Subject[iSub+iPat]== pattern[iPat]
9
Subject[iSub+iPat]!= pattern[iPat]
8
rtnIndex = NOTFOUND
isPat = false;
break;


Def(1)={subject, pattern}
1
Def(2)={NotFound, iSub,rtnIndex, isPat, subjectLen,patternLen}
use(2)={NotFound, subject, pattern}
2
use(11)={rtnIndex}
3
11
use(3,4)=use(3,11)={isPat, iSub, patternLen, subjectLen}
Def(10)=use(10)={iSub}
10
4
use(4,10)=use(4,5)={ iSub,subject,pattern}
5
def(5)={rtnIndex,isPat,iPat}, use(5)={iSub,}
use(6,10)=use(6,7) ={ipat,patternLen}
6
7
9
Def(9)=use(9)={iPat}
use(7,8)=use(7.9)= {subject,iSub,iPat, Pattern}
8
Def(8)={rtnIndex, isPat}
use(2)={NotFound}


Def(1)={subject, pattern}
1
du(2,iSub)={[2,3,4],[2.3.11],[2,3,4,5],[2,3,4,5,6,7,8],[2,3,4,5,6,7,9],[2,,3,4,5,6,10],[2,3,4,5,6,7,8,10],[2,3,4,10]}
Def(2)={NotFound, iSub,rtnIndex, isPat, subjectLen,patternLen}
use(2)={NotFound, subject, pattern}
2
use(11)={rtnIndex}
3
11
use(3,4)=use(3,11)={isPat, iSub,patternLen,subjectLen}
Def(10)=use(10)={iSub}
10
4
use(4,10)=use(4,5)={iSub,subject,pattern}
5
def(5)={rtnIndex,isPat,iPat},
use(5)={iSub}
use(6,10)=use(6,7)= {ipat, patternLen}
6
Def(9)=use(9)={iPat}
7
9
du(2,4,iSub)={[2,3,4]}
du(2,11,iSub)={[2.3.11]}
du(2,5,iSub)={[2,3,4,5]}
du(2,8,iSub)={[2,3,4,5,6,7,8]}
du(2,9,iSub)={[2,3,4,5,6,7,9]}
du{2,10,iSub}={[2,3,4,5,6,10],[2,3,4,5,6,7,8,10],[2,3,4,10]}
Def(8)={rtnIndex, isPat}
use(8)={NotFound}
8
use(7,8)=use(7.9)={subject, iSub,iPat,Pattern}


Def(1)={subject, pattern}
1
Def(2)={NotFound, iSub,rtnIndex, isPat, subjectLen,patternLen}
use(2)={NotFound, subject, pattern}
2
du(10,iSub)={[10,3,4],[10.3.11],[10,3,4,5],[10,3,4,5,6,7,8],[10,3,4,5,6,7,9],[10,,3,4,5,6,10],[10,3,4,5,6,7,8,10],[10,3,4,10]}
use(11)={rtnIndex}
3
11
use(3,4)=use(3,11)={isPat, iSub,patternLen,subjectLen}
Def(10)=use(10)={iSub}
10
4
use(4,10)=use(4,5)={iSub,subject,pattern}
5
def(5)={rtnIndex,isPat,iPat},
use(5)={iSub}
use(6,10)=use(6,7)= {ipat, patternLen}
6
Def(9)=use(9)={iPat}
7
9
du(10,4,iSub)={[10,3,4]}
du(10,11,iSub)={[10.3.11]}
du(10,5,iSub)={[10,3,4,5]}
du(10,8,iSub)={[10,3,4,5,6,7,8]}
du(10,9,iSub)={[10,3,4,5,6,7,9]}
du{10,10,iSub}={[10,3,4,5,6,10],[10,3,4,5,6,7,8,10],[10,3,4,10]}
Def(8)={rtnIndex, isPat}
use(8)={NotFound}
8
use(7,8)=use(7.9)={s
iSub,iPat,Pattern}

Example

du(2,iSub)={[2,3,4],[2,3,11],[2,3,4,5],[2,3,4,5,6,7,8],[2,3,4,5,6,7,9],[2,,3,4,5,6,10],[2,3,4,5,6,7,8, 10], [2,3,4,10]}

du(10,iSub)={[10,3,4],[10,3,4,5],[10,3,4,5,6,7,8],[10,3,4,5,6,7,9],[10,,3,4,5,6,10],[10,3,4,5,6,7,8, 10],[10,3,4,10]}

All-defs for *iSub*
[2,3,4,5,6,7,8], [10,3,4]

Example

du(2,4,iSub)={[2,3,4]}
du(2,11,iSub)={[2.3.11]}
du(2,5,iSub)={[2,3,4,5]}
du(2,8,iSub)={[2,3,4,5,6,7,8]}
du(2,9,iSub)={[2,3,4,5,6,7,9]}
du{2,10,iSub}={[2,3,4,5,6,10],[2,3,4,5,6,7,8,10],[2,3,4,10]}

du(10,4,iSub)={[10,3,4]}
du(10,11,iSub)={[10.3.11]}
du(10,5,iSub)={[10,3,4,5]}
du(10,8,iSub)={[10,3,4,5,6,7,8]}
du(10,9,iSub)={[10,3,4,5,6,7,9]}
du{10,10,iSub}={[10,3,4,5,6,10],[10,3,4,5,6,7,8,10],[10,3,4,10]}

All-uses for *iSub*

[2,3,4], [2,3,11], [2,3,4,5], [2,3,4,5,6,7,8], [2,3,4,5,6,7,9], [2,3,4,5,6,10], [10,3,4], [10,3,11],[10,3,4,5],[10,3,4,5,6,7,8],[10,3,4,5,6,7,9], [10,3,4,5,6,10]

Example

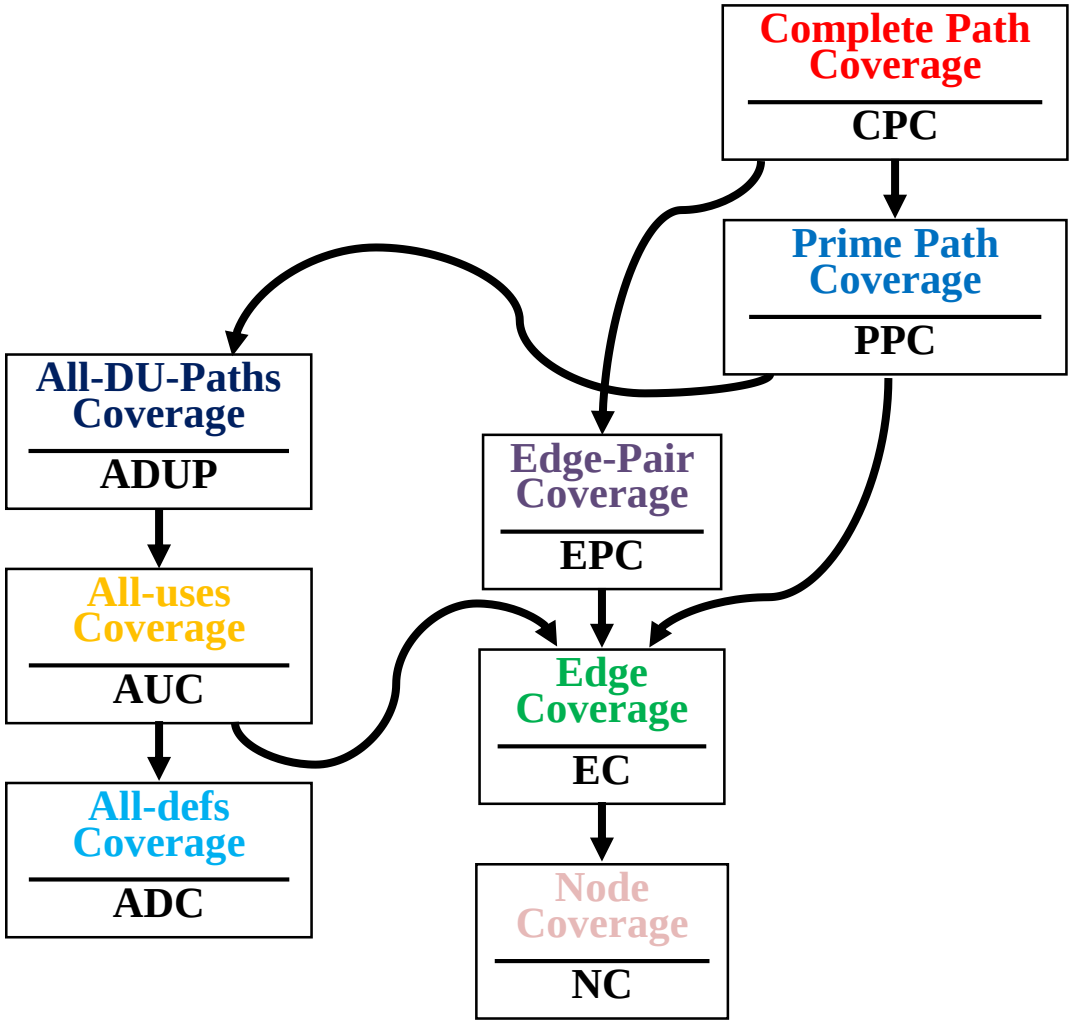
du(2,4,iSub)={[2,3,4]}
du(2,11,iSub)={[2.3.11]}
du(2,5,iSub)={[2,3,4,5]}
du(2,8,iSub)={[2,3,4,5,6,7,8]}
du(2,9,iSub)={[2,3,4,5,6,7,9]}
du{2,10,iSub}={[2,3,4,5,6,10],[2,3,4,5,6,7,8,10],[2,3,4,10]}

du(10,4,iSub)={[10,3,4]}
du(10,11,iSub)={[10.3.11]}
du(10,5,iSub)={[10,3,4,5]}
du(10,8,iSub)={[10,3,4,5,6,7,8]}
du(10,9,iSub)={[10,3,4,5,6,7,9]}
du{10,10,iSub}={[10,3,4,5,6,10],[10,3,4,5,6,7,8,10],[10,3,4,10]}

All-du-path for *iSub*

[2,3,4], [2,3,11], [2,3,4,5], [2,3,4,5,6,7,8], [2,3,4,5,6,7,9], [2,3,4,5,6,10], [2,3,4,5,6,7,8,10], [2,3,4,10], [10,3,4], [10,3,11], [10,3,4,5], [10,3,4,5,6,7,8], [10,3,4,5,6,7,9], [10,3,4,5,6,10], [10,3,4,5,6,7,8,10], [10,3,4,10]

Defect Detection Ability Evaluation



Summary

- Data Flow Testing focuses on the correctness of variables definition and corresponding uses
- Test cases may be derived based on various data flow coverage criteria
- The defect detection Ability of data flow criteria are weaker than PPC
- Data flow testing can be more useful on integration testing than control flow testing

The End