Ch2 Code Unit Test

Write Code to Test Code(5)

Instructor: Haiying SUN

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

Office: ECNU Science Build B1104

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

Agenda

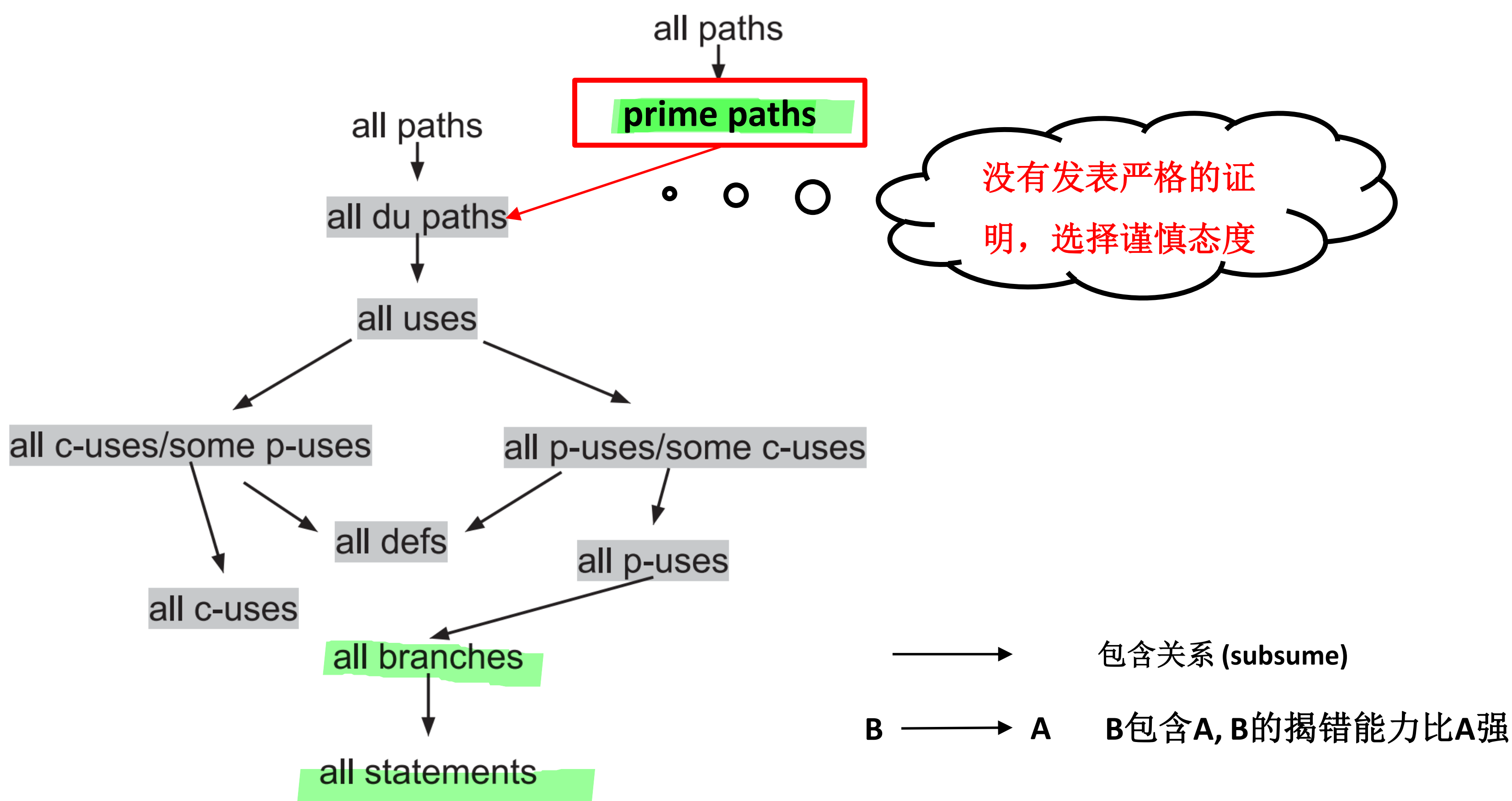
- Introduction to Unit Testing
- Common Code Defect Categories
- Unit Tests Design Heuristic Rules
- Unit Tests Implementation
  - Junit & Mockito & Qualified test scripts
- Code Test Adequacy Criteria
  - Control flow based & Jacoco
  - Data flow based
  - Mutation Based
- Code Test Generation

Data Flow Coverage Criteria Background

- Origin from Data flow analysis
  - Data flow analysis focuses on how variables are bound to values, and how these variables are to be used, widely used in code optimization
- Data Flow Coverage Criteria are proposed as a solution to reduce the complexity of achieving all path coverage [1]
- Data Flow Coverage Criteria may include
  - 全定义覆盖 (all defs coverage)
  - 全计算使用覆盖 (all c-uses coverage)
  - 全谓词使用覆盖 (all p-uses coverage)
  - 全计算使用/部分谓词使用覆盖 (all c-uses/some p-uses coverage)
  - 全谓词使用/部分计算使用覆盖 (all p-uses/some c-uses coverage)
  - 全使用覆盖 (all uses coverage)
  - 全计算使用路径覆盖 (all du paths coverage)
  - ……

1. S. Rapps and E.J. Weyuker, "Data Flow Analysis Techniques for Test Data Selection", Proc. Sixth Int. Conf. Software Engineering, Tokyo, 1982..

Data Flow Coverage Criteria Background



Subsume Relationship Among different structural coverage criteria

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

Def and Use

```
if (x < y) {      →  use: x, y
  y = 0;          →  def: y
  x = x + 1;      →  use: x
                     def: x
}
else {
  x = y;          →  use: y
                     def: x
}
```

揭错的基础

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

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

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

use: x, y

p-use: x, y

def: y

use: x

def: x

c-use: x

use: y

def: x

c-use: y

Def and Use

```java
    /**
     * return the maximum of arr
     * */
    public int DefsAndUses(int[] arr){
        int max = Integer.MIN_VALUE ;
        for(int i = 0; i < arr.length-1; i++){
            max =  arr[i] > max ? arr[i]:max;
        }
        return max;
```

- i++定义了什么？使用了什么？
- max = arr[i] > max ? arr[i]:max 定义了什么？使用了什么？
- 数组arr的定义处和使用处有哪些？ arr作为一个变量整体考虑，还是arr[1]作为一个变量考虑？
- 复杂数据类型的定义处和使用处如何考虑？比如动态数组ArrayList
- 指针的定义处和使用处如何考虑？

Def and Use

```
    proc(condition1,condition2)
        int condition1,condition2;
    {
        int x, y, z, *p;
24:             z = 17;
25:             x = 13;
26:             if (condition1) {
28:                     p = &y;
29:                     *p = z;
                }
                else {
35:                     if (condition2)
36:                             p = &x;
37:                     else
38:                             p = &z;
40:                     *p = 7 + z;
42:                     y = 53;
43:                     p = &x;
                }
49:             x = x + y + z;
50:             *p = *p + 5;
51:             y = x + y;
```

Figure 1. Example C Program

Thomas J. Ostrand and E.J. Weyuker, Data Flow Based Test Adquency Analysis for languages with Pointers, Proceedings of the symposium on Testing, analysis, and verification, 74-86, 1991 .

Elaine J. Weyuker

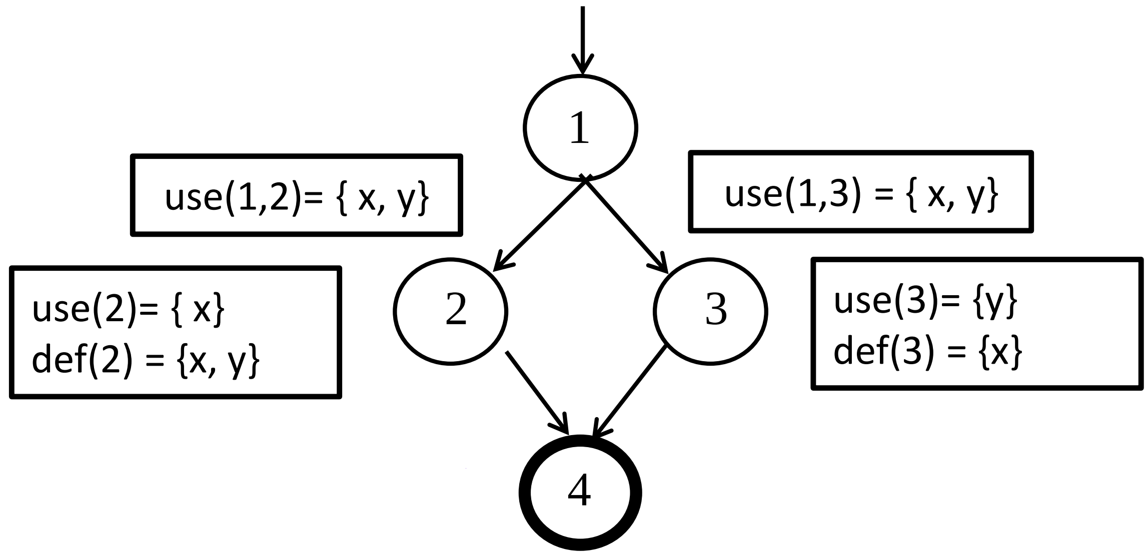
Challenge: The calculation of defs and uses is complicated by the use of indirect references, aliases, and procedure calls, since the actual storage referenced by a given identifier may not be determinable at compile time.

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 Example

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



$N_D = \{\, (1, \phi),\ (2, def(2) \cup use(2)),\ (3, def(3) \cup use(3)),\ (4, \phi) \,\}$

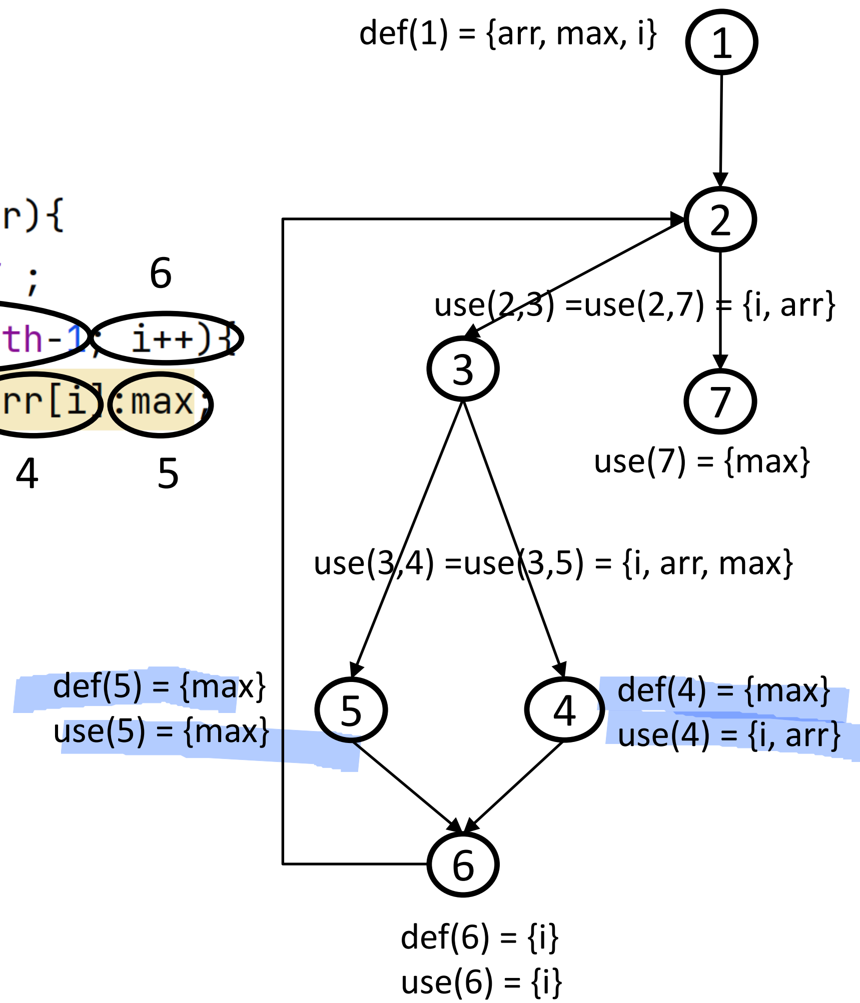
$N_0 = \{ 1 \}$

$N_f = \{ 4 \}$

$E_D = \{\, ((1,2), use(1,2)),\ ((1,3), use(1,3)),\ ((2,4), \phi),\ ((3,4), \phi) \,\}$

Data Flow Graph Example

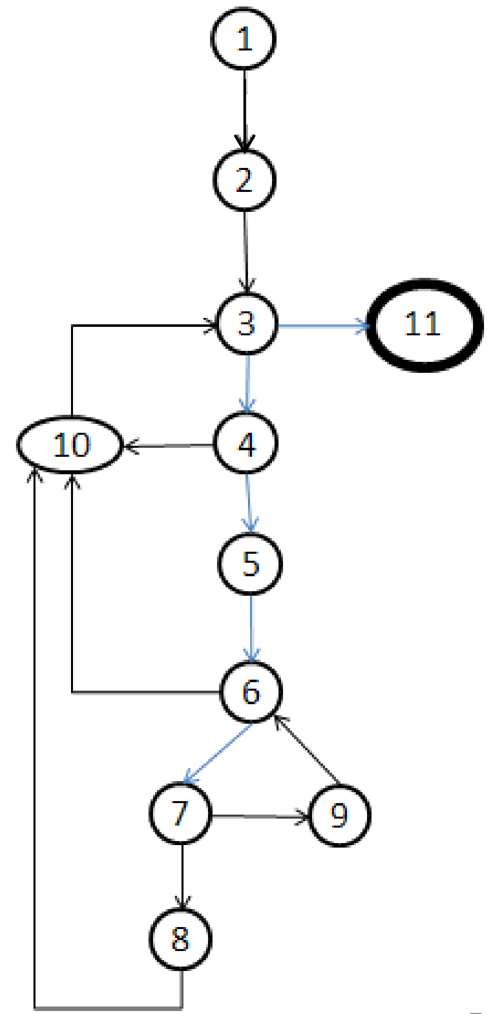
```java
/**
 * return the maximum of arr
 * */
public int DefsAndUses(int[] arr){
    int max = Integer.MIN_VALUE ;
    for(int i = 0; i < arr.length-1; i++){
        max = arr[i] > max ? arr[i]:max;
    }
    return max;
```

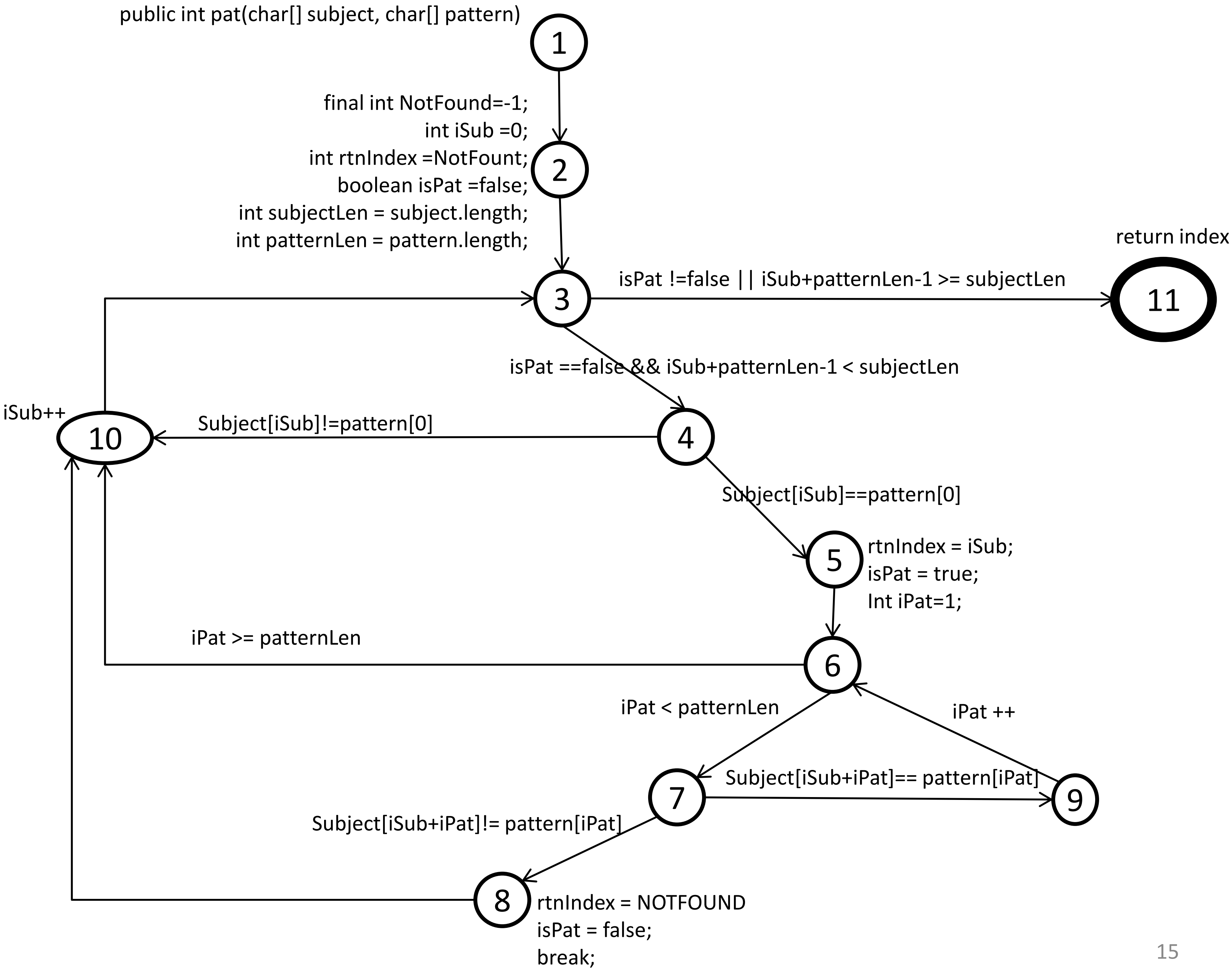


Data Flow Graph Example

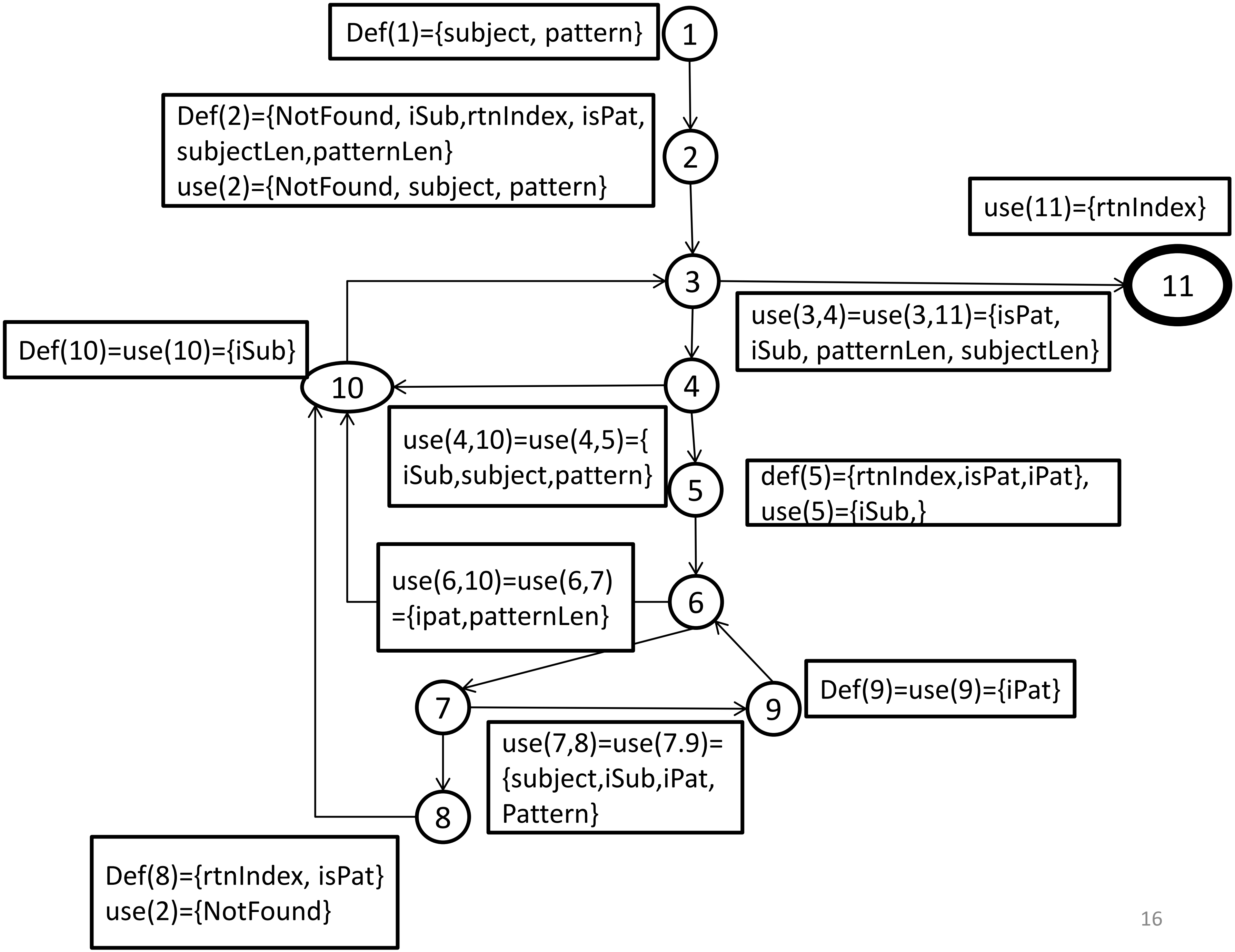
```java
public int pat(char[] subject, char[] pattern){
    final int NotFound = -1;
    int iSub = 0;
    int rtnIndex = NotFound;
    boolean isPat = false;
    int subjectLen = subject.length;
    int patternLen = pattern.length;

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





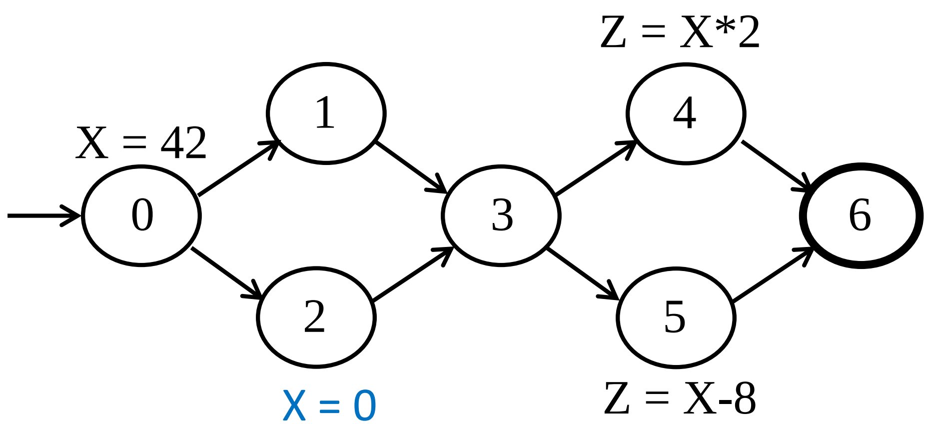
public int pat(char[] subject, char[] pattern)
1
final int NotFound=-1;
int iSub =0;
int rtnIndex =NotFount;
boolean isPat =false;
int subjectLen = subject.length;
int patternLen = pattern.length;
2
return index
3
isPat !=false || iSub+patternLen-1 >= subjectLen
11
isPat ==false && iSub+patternLen-1 < 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}

DU Pairs and DU Paths

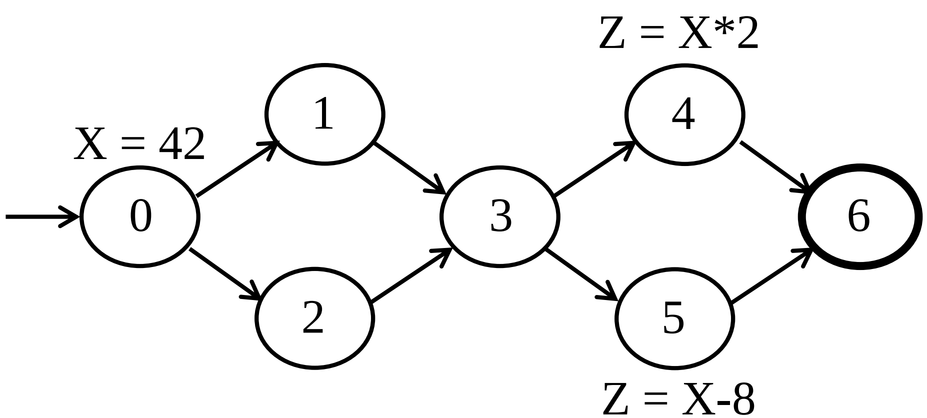
- [du pair] A pair of locations (l_i, l_j) such that a variable v is defined at l_i and used at l_j
- [def clear] A path from l_i to l_j is *def-clear* with respect to variable *v* if *v* is not given another value on any of the nodes or edges in the path
- [du path] A **simple** subpath that is def-clear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

Def Clear



- ✓ 路径(0,2,3,5) 不是def clear的路径，因为节点2对x进行了重新定义
- ✓ 同理，路径(0,2,3,4)也不是def clear的
- ✓ 路径(0,1,3,4)和(0,1,3,5)是def clear的

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)

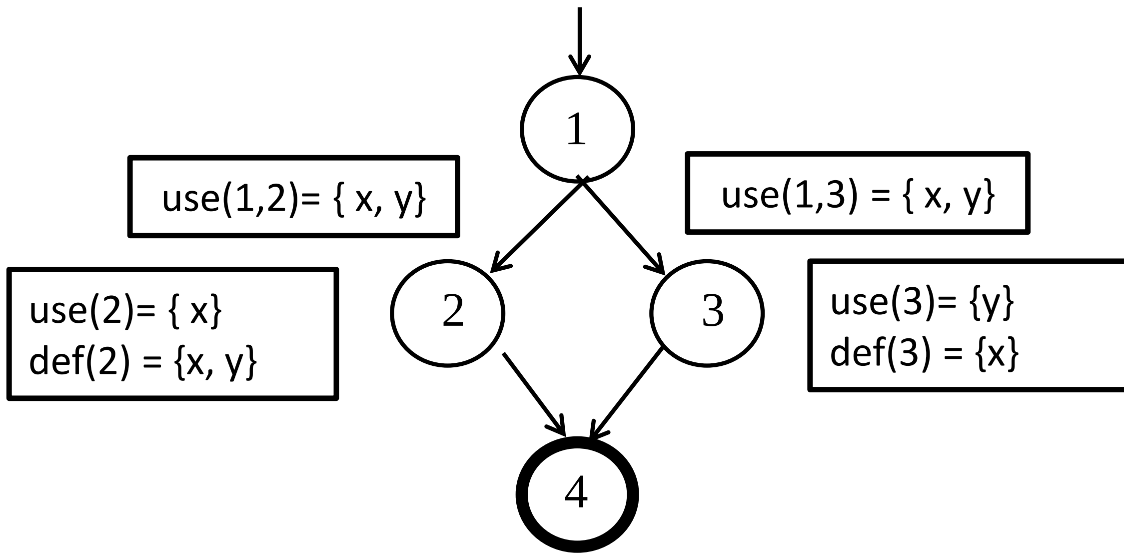
du pairs for *X*
(0, 4)
(0, 5)

du(0, x)

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

DU Pairs and DU Paths

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



du Pairs for Variable y: ϕ

du Pairs for Variable x: ϕ

DU Paths Example

变量max的定义处: 1,4,5, 使用处: (3,4), (3,5),5,7

du(1,(3,4),max) = {(1,2,3,4)}
du(1,(3,5),max) = {(1,2,3,5)}
du(1,5,max) = {(1,2,3,5)}
du(1,7,max) = {(1,2,7)}

du(1,max)

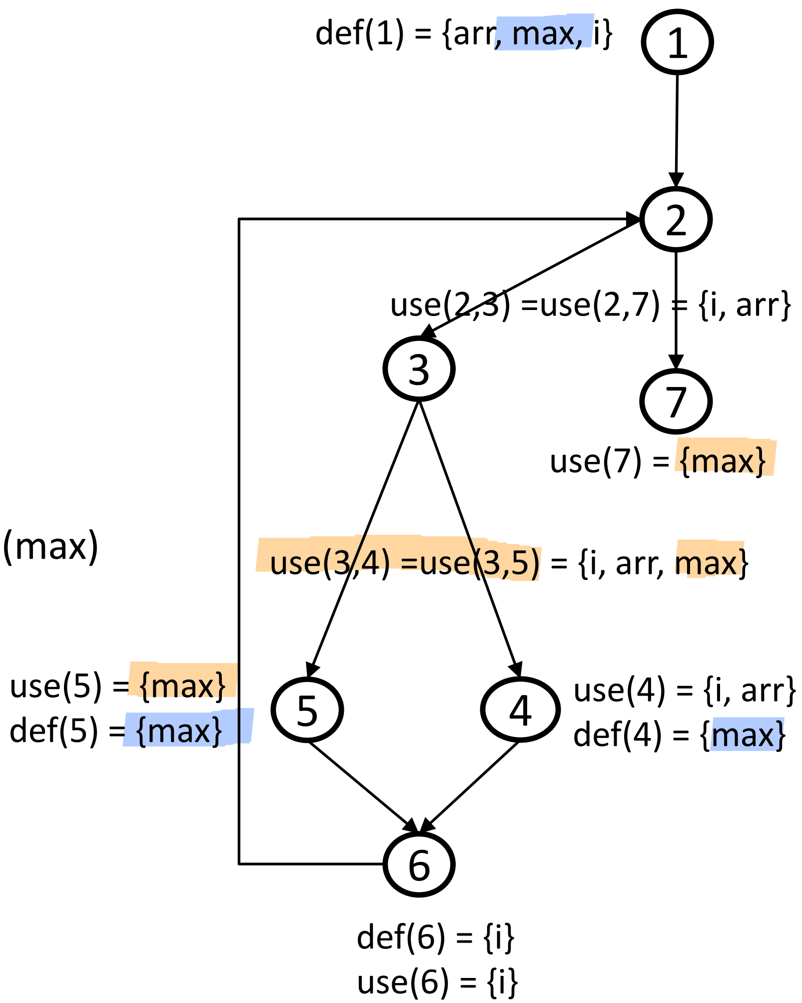
du(4,(3,4),max) = {(4,6,2,3,4)}
du(4,(3,5),max) = {(4,6,2,3,5)}
du(4,5,max) = {(4,6,2,3,5)}
du(4,7,max) = {(4,6,2,7)}

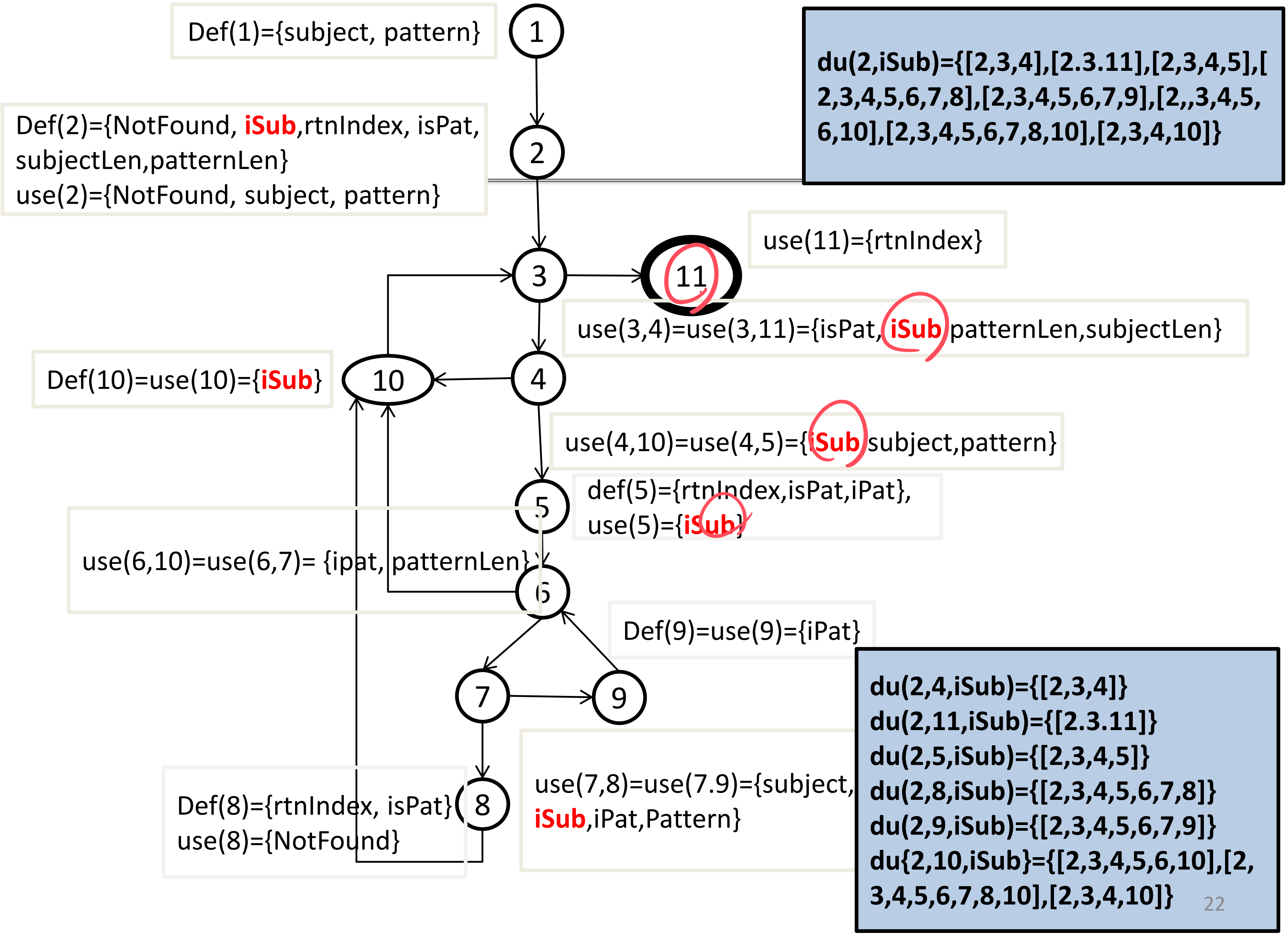
du(4,max)

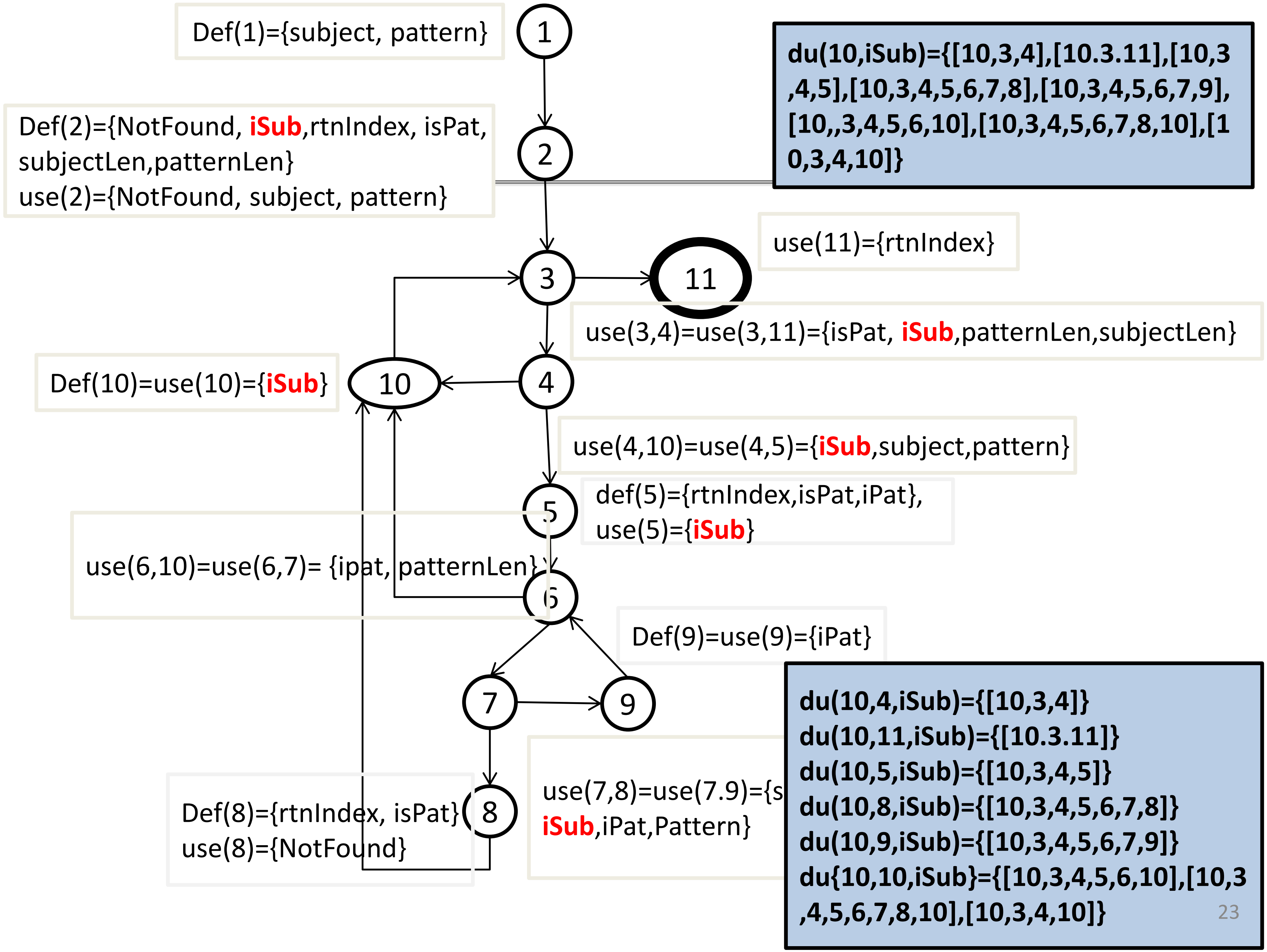
du(5,(3,4),max) = {(5,6,2,3,4)}
du(5,(3,5),max) = {(5,6,2,3,5)}
du(5,5,max) = {(5,6,2,3,5)}
du(5,7,max) = {(5,6,2,7)}

du(5,max)

du(max)




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}
def(5)={rtnIndex,isPat,iPat},
use(5)={iSub}
5
use(6,10)=use(6,7)= {ipat, patternLen}
6
Def(9)=use(9)={iPat}
7
9
Def(8)={rtnIndex, isPat}
use(8)={NotFound}
8
use(7,8)=use(7.9)={subject,
iSub,iPat,Pattern}
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]}

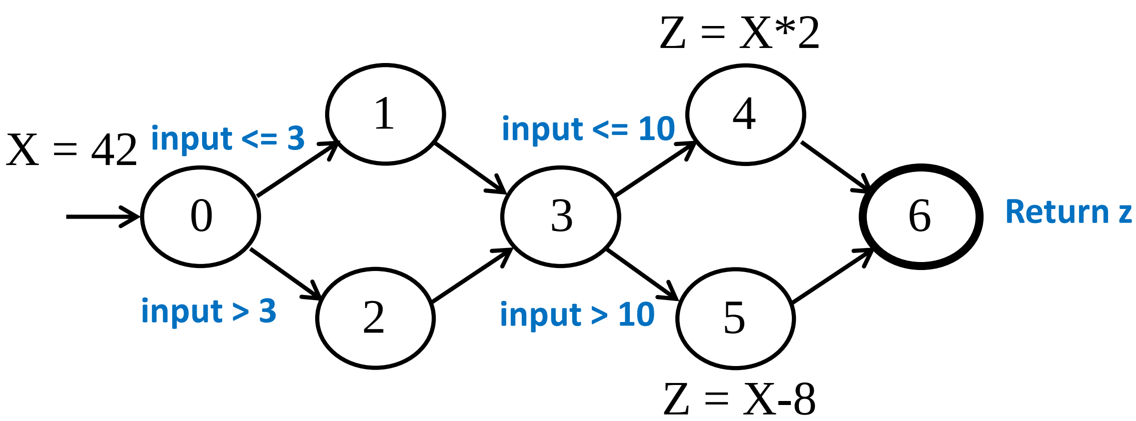

Def(1)={subject, pattern}
1
du(10,iSub)={[10,3,4],[10.3.11],[10,3,4,5],[10,3,4,5,6,7,8],[10,3,4,5,6,7,9],[10,,3,4,5,6,10],[10,3,4,5,6,7,8,10],[10,3,4,10]}
Def(2)={NotFound, iSub,rtnIndex, isPat, subjectLen,patternLen}
use(2)={NotFound, subject, pattern}
2
use(11)={rtnIndex}
3
11
use(3,4)=use(3,11)={isPat,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

- 全定义覆盖（All Defs Coverage）
  - 衡量被测代码中变量的每个定义得到使用的程度。
  - 全定义覆盖的正式定义为：测试集合T满足全定义覆盖，当且仅当对于数据流图中每个变量v_i的每个定义处d_j，执行T产生的完整路径集合L中存在一条路径 $l \in L$的某个子路径π，满足$\pi \in du(d_j, v_i)$ 。若def(v_i)是变量v_i在数据流图中定义处的集合，$\prod_{\mathrm{L}}^{\mathrm{D}}(v_i) = \{\pi \mid \pi \in du(d_j, v_i), \pi$是$l \in L$的子路径$\}$，且满足$\prod_{\mathrm{L}}^{\mathrm{D}}(v_i)$ 的任意两个元素是v_i的不同定义处开始的定义使用路径，则测试集合T的全定义覆盖率为：

$$\frac{\sum_{i=1}^{m} \left\| \prod_{\mathrm{L}}^{\mathrm{D}}(v_i) \right\|}{\sum_{i=1}^{m} \| def(v_i) \|} * 100\%, \text{ m为程序中变量的个数}$$

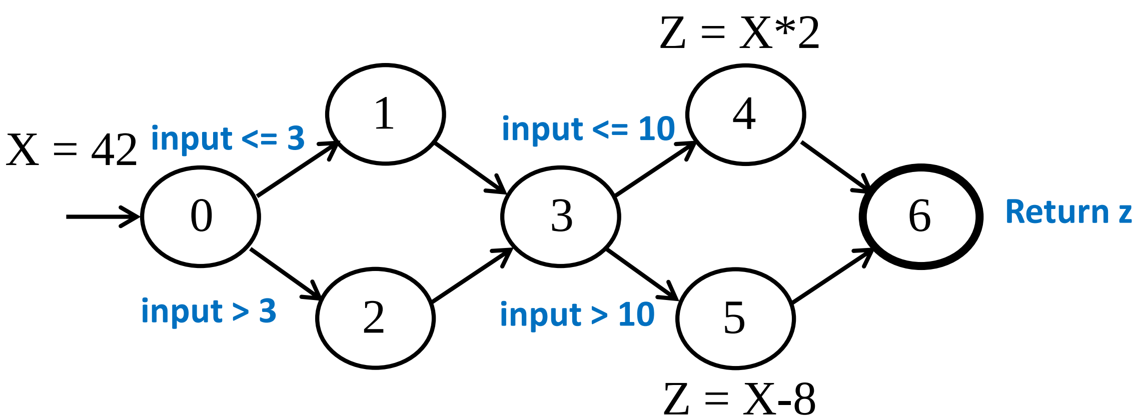
Example



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

- 对于变量x而言，只有1个定义处0，满足x的全定义覆盖，测试路径需覆盖(0, 1, 3, 4)，(0, 2, 3, 4)， (0, 1, 3, 5) ， (0, 2, 3, 5)的任意一条即可
- 测试输入集合：input = 4
- 测试执行路径：(0,2,3,4,6) 覆盖(0, 2, 3, 4)，满足x的全定义覆盖
- 对于整个程序是否满足全定义覆盖，还要考察变量z, input的情况

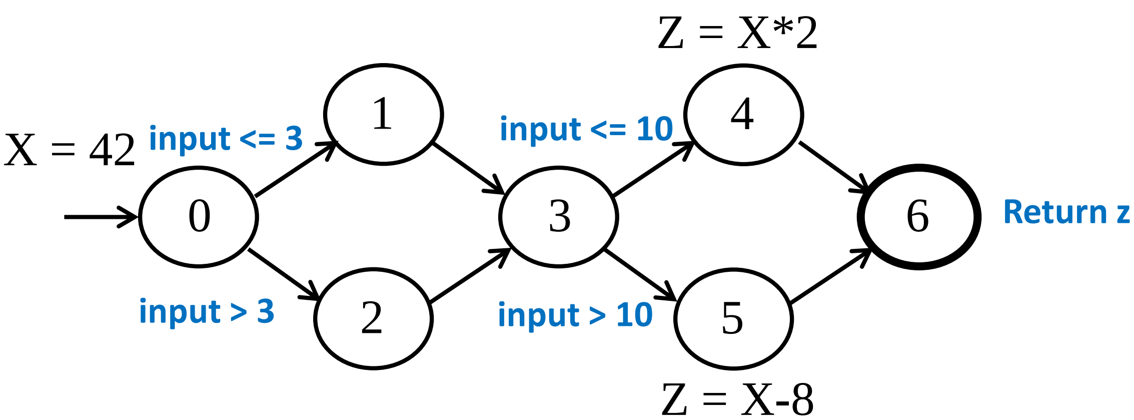
Example



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

- 对于变量input而言，只有1个定义处0，满足input的全定义覆盖，测试路径需覆盖(0, 1), (0, 2), (0, 1, 3, 4) , (0,1,3,5), (0, 2, 3, 4) , (0,2,3,5)的任意一条即可，测试输入集合input = 4 的执行路径(0,2,3,4,6) 覆盖 (0, 2), (0,2,3,4)满足input的全定义覆盖
- 对于变量z而言，有2个定义处4,5，满足z的全定义覆盖，测试路径需覆盖(4, 6)和(5,6) 。测试输入集合input = 4的执行路径(0,2,3,4,6) 覆盖 了(4,6), 没有覆盖(5,6)，因此不满足z 的全定义覆盖

Example



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

- 根据前面的分析计算全定义覆盖率：

1. Def(x) = {0}， $\prod_{\mathrm{L}}^{\mathrm{D}}(\mathrm{x})$ = {(0,2,3,4)}
2. Def(input) = {0}, $\prod_{\mathrm{L}}^{\mathrm{D}}(input)$ = {(0,2)}/ $\prod_{\mathrm{L}}^{\mathrm{D}}(input)$ = {(0,2,3)} （两个集合哪个都可以）
3. Def(z) = {4,5}, $\prod_{\mathrm{L}}^{\mathrm{D}}(z)$ = {(4,6)}

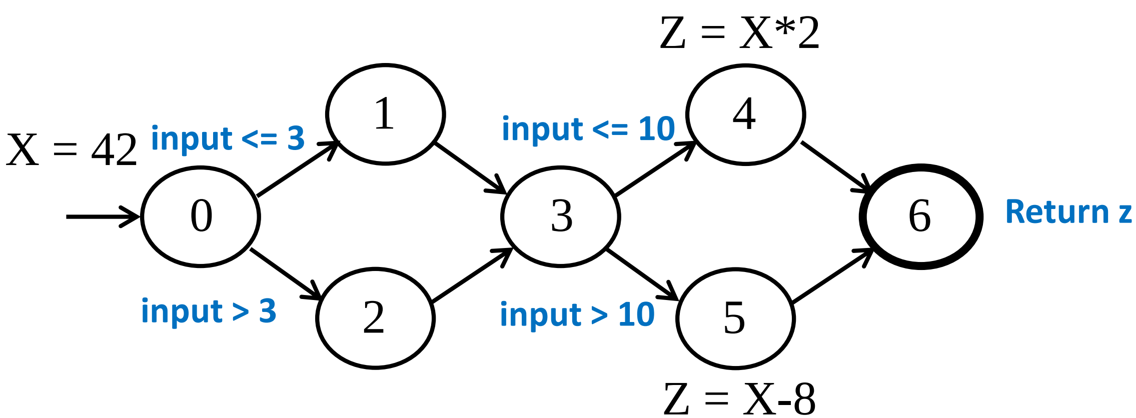
$$\frac{\sum_{i=1}^{m}\left\|\Pi_{\mathrm{L}}^{\mathrm{D}}(vi)\right\|}{\sum_{i=1}^{m}\|def(v_i)\|} * 100\% = \frac{\left\|\Pi_{\mathrm{L}}^{\mathrm{D}}(x)\right\|+\left\|\Pi_{\mathrm{L}}^{\mathrm{D}}(input)\right\|+\left\|\Pi_{\mathrm{L}}^{\mathrm{D}}(z)\right\|}{\|def(x)\|+\|def(input)\|+\|def(z)\|} * 100\% = \frac{1+1+1}{1+1+2} * 100\% = 75\%$$

All Uses Coverage

- 全使用覆盖（All Uses Coverage）
  - 衡量被测代码中变量的每个使用得到执行的程度。
  - 全使用覆盖的正式定义为：测试集合T满足全使用覆盖，当且仅当对于数据流图中每个变量v_i的每个定义d_j的每个使用u_k，执行T产生的完整路径集合L中存在一条路径 $l \in L$的某个子路径π，满足$\pi \in du(d_j, u_k, v_i)$。若use($v_i$)是变量$v_i$在数据流图中定义使用对的集合，$\prod_L^U(v_i) = \{\pi | \pi \in du(d_j, u_k, v_i), \pi是l \in L的子路径\}$，且满足$\prod_L^U(v_i)$的任意两个元素是$v_i$的不同定义处开始和使用处结束的定义使用路径，则测试集合T的全使用覆盖率为：

$$\frac{\sum_{i=1}^{m}\left\|\prod_{\mathrm{L}}^{U}(vi)\right\|}{\sum_{i=1}^{m}\|\mathrm{use}(v_i)\|}$$ *100%, m为程序中变量的个数

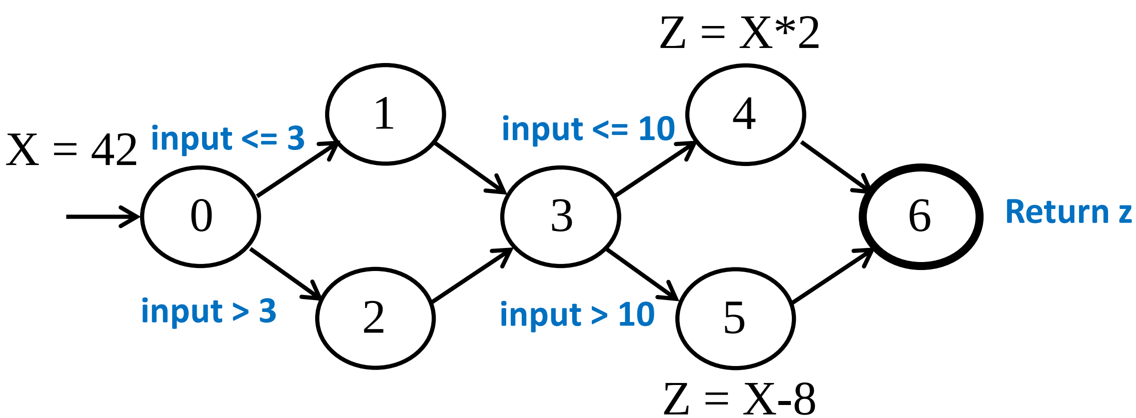
Example



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

- 对于变量x而言，有2个使用处，满足x的全使用覆盖，测试路径需覆盖
  - 针对使用4：[0, 1, 3, 4]和 [0, 2, 3, 4]中的任意一条
  - 针对使用5：[0, 1, 3, 5]和[0, 2, 3, 5]的任意一条
- 测试输入集合：input = 4，input = 11
- 测试执行路径：(0,2,3,4,6) 覆盖[0, 2, 3, 4]，(0,2,3,5,6)覆盖[0,2,3,5], 满足x的全使用覆盖
- 对于整个程序是否满足全使用覆盖，还要考察变量z, input的情况，显然在该测试输入集合下，变量input的全使用没有覆盖到，变量z的全使用覆盖到了

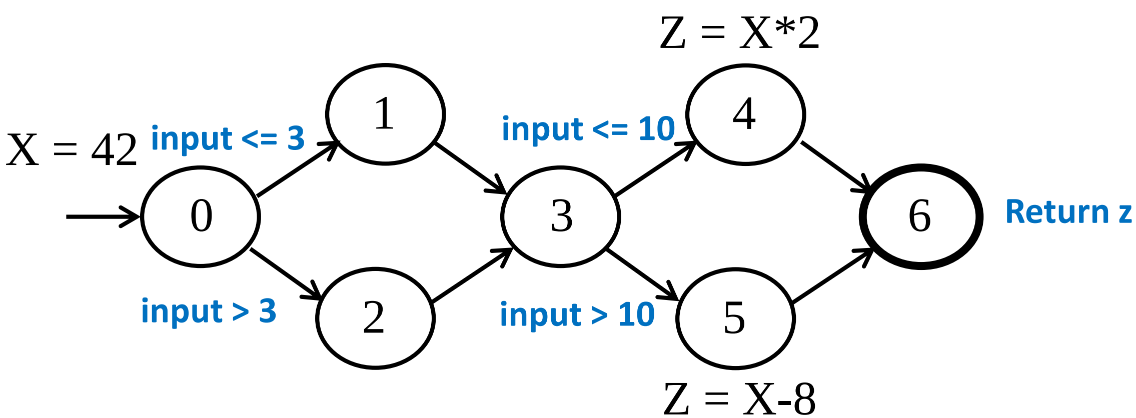
Example



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

- 对于变量x而言，有2个使用处，满足x的全使用覆盖，测试路径需覆盖
  - 针对使用4：(0, 1, 3, 4)和 (0, 2, 3, 4)中的任意一条
  - 针对使用5：(0, 1, 3, 5)和(0, 2, 3, 5)的任意一条
- 测试输入集合：input = 4， input = 11
- 测试执行路径：(0,2,3,4,6) 覆盖(0, 2, 3, 4)，(0,2,3,5,6)覆盖(0,2,3,5), 满足x的全使用覆盖
- 对于整个程序是否满足全使用覆盖，还要考察变量z, input的情况

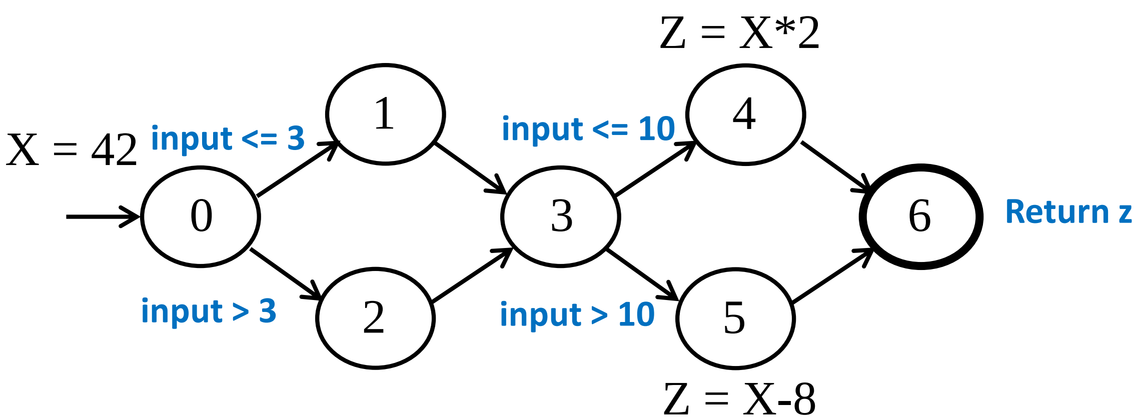
Example



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

- 对于变量input而言，有4个使用处，满足input的全使用覆盖，测试路径需覆盖
  - 针对使用(0,1)：(0, 1)
  - 针对使用(0,2)：(0, 2)
  - 针对使用(3,4)：(0, 1,3,4)和(0,2,3,4)任意一条
  - 针对使用(3,5)：(0, 1,3,5)和(0,2,3,5)任意一条
- 测试输入集合：input = 4，input = 11
- input = 4的测试执行路径(0,2,3,4,6) 覆盖(0, 2),(0,2,3,4) ;
- input = 11的测试执行路径(0,2,3,5,6)覆盖(0, 2), (0,2,3,5)
- 使用(0,1)未覆盖到，不满足input的全使用覆盖

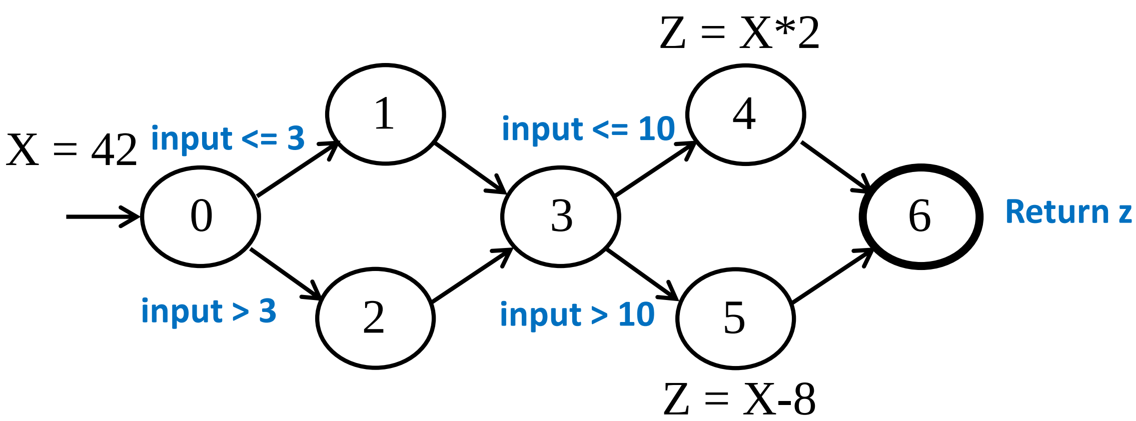
Example



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

- 对于变量z而言，有1个使用处6，满足z的全使用覆盖，测试路径需覆盖
  - 针对使用6：(4, 6)和(5,6)都需要覆盖
- 测试输入集合：input = 4， input = 11
- input = 4的测试执行路径(0,2,3,4,6) 覆盖 (4,6) ;
- input = 11的测试执行路径(0,2,3,5,6)覆盖(5,6)
- 满足z的全使用覆盖

Example



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

- 根据前面的分析计算全使用覆盖率：

1. use(x) = {(0,4),(0,5)}， $\prod_{\mathrm{L}}^{U}(\mathrm{x})$ = {(0,2,3,4),(0,2,3,5)}
2. use(input) = {(0,(0,1)),(0,(0,2)),(0,(3,4)),(0,(3,5))}, $\prod_{\mathrm{L}}^{U}(input)$ = {(0,2), (0,2,3,4),(0,2,3,5)}
3. use(z) = {(4,6), (5,6)}, $\prod_{\mathrm{L}}^{U}(z)$ = {(4,6),(5,6)}

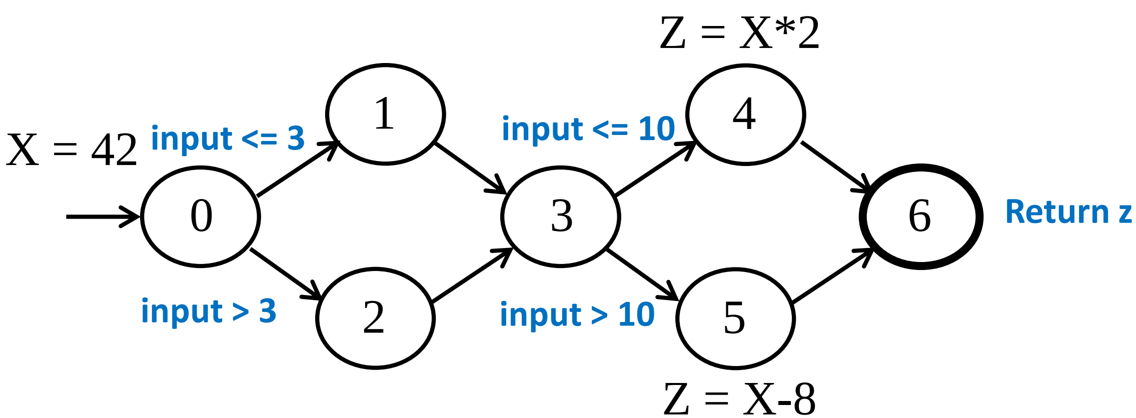
$$\frac{\sum_{i=1}^{m}\left\|\prod_{\mathrm{L}}^{U}(vi)\right\|}{\sum_{i=1}^{m}\|use(v_i)\|} *100\% = \frac{\left\|\prod_{\mathrm{L}}^{U}(x)\right\|+\left\|\prod_{\mathrm{L}}^{U}(input)\right\|+\left\|\prod_{\mathrm{L}}^{U}(z)\right\|}{\|use(x)\|+\|use(input)\|+\|use(z)\|} *100\% = \frac{2+3+2}{2+4+2} * 100\% = \boxed{87.5\%}$$

All Def Use Paths Coverage

- 全定义使用覆盖（All du path Coverage）
  - 衡量被测代码中变量的每条定义使用路径得到执行的程度。
  - 全定义使用路径覆盖的正式定义为：测试集合T满足全定义使用路径覆盖，当且仅当对于数据流图中每个变量v_i的每条定义使用路径，执行T产生的完整路径集合L中存在一条路径 $l \in L$的某个子路径π，满足$\pi \in du(v_i)$。若du(v_i)是变量v_i在数据流图中定义使用路径集合，$\Pi_L(v_i) = \{\pi | \pi \in du(v_i), \pi$是$l \in L$的子路径$\}$，则测试集合T的全定义使用路径覆盖率为：

$$\frac{\sum_{i=1}^{m} \|\Pi_L(v_i)\|}{\sum_{i=1}^{m} \|\mathrm{du}(v_i)\|}$$ *100%, m为程序中变量的个数

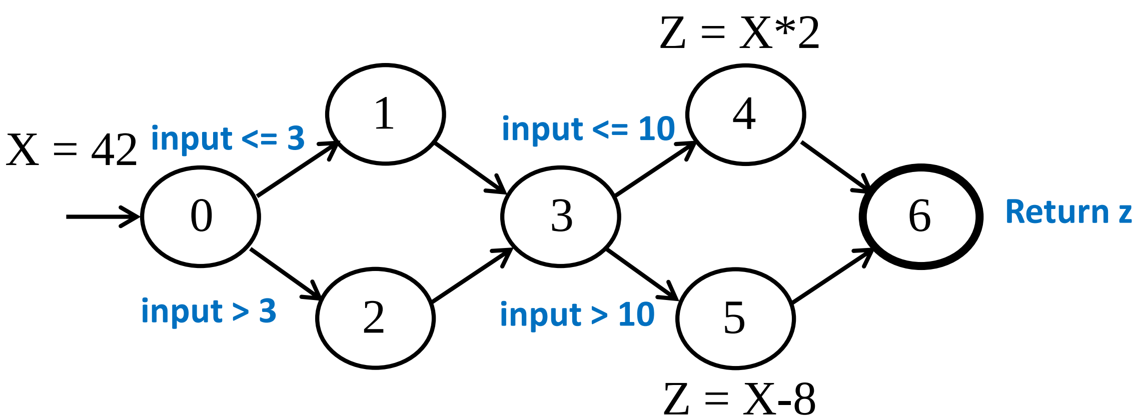
Example



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

- 对于变量x而言，满足x的全定义使用覆盖，测试路径需覆盖[0, 1, 3, 4], [0, 2, 3, 4], [0, 1, 3, 5], [0, 2, 3, 5], 注意到[0,1,3,5]是不可执行路径无法覆盖, 因此, 不可能满足全定义使用覆盖。不可执行路径在计算覆盖度是不建议删除，报低不报高，避免遗漏缺陷。
- 测试输入集合：input = 4，input = 11, input = 2
- 测试执行路径：(0,2,3,4,6) 覆盖[0, 2, 3, 4]，(0,2,3,5,6)覆盖[0,2,3,5], (0, 1, 3, 4,6)覆盖[0, 1, 3, 4]
- 对于整个程序是否满足全定义使用覆盖，还要考察变量z, input的情况，在该测试输入集合下，变量input, z的全定义使用覆盖满足

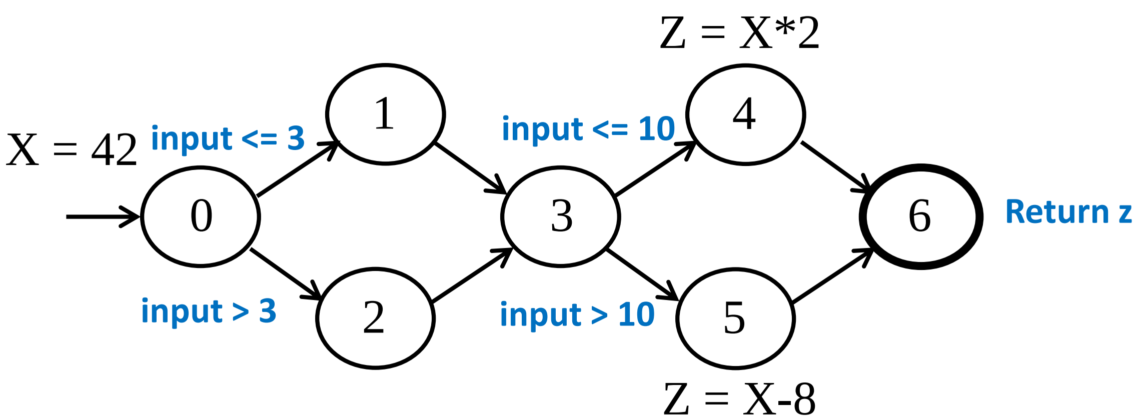
Example



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

- 对于变量input而言，满足input的全定义使用覆盖，测试路径需覆盖(0,1),(0, 1, 3, 4), (0,1,3,5) (0, 2),(0,2, 3, 4), (0, 2, 3, 5), 注意到(0,1,3,5)是不可执行路径无法覆盖
- 测试输入集合：input = 4， input = 11, input = 2
- 测试执行路径：(0,2,3,4,6) 覆盖(0, 2),(0,2, 3, 4)； (0,2,3,5,6)覆盖(0, 2),(0,2, 3, 5)； (0, 1, 3, 4,6)覆盖(0,1),(0, 1, 3, 4)
- 不满足input的全定义使用路径覆盖，因为(0,1,3,5)没有覆盖

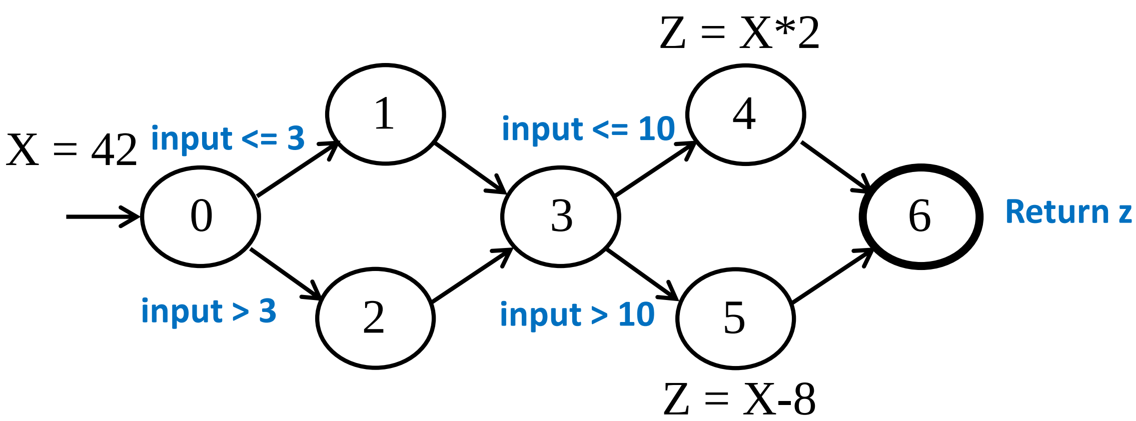
Example



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

- 对于变量z而言，满足z的全定义使用覆盖，测试路径需覆盖(4,6), (5,6)
- 测试输入集合：input = 4， input = 11, input = 2
- 测试执行路径：(0,2,3,4,6) 覆盖(4,6)； (0,2,3,5,6)覆盖(5,6)；(0, 1, 3, 4,6)覆盖(4,6); 满足z的全定义使用路径覆盖

Example



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

- 根据前面的分析计算全定义使用路径覆盖率：

1. du(x) = {(0,1,3,4),(0,2,3,4),(0,1,3,5),(0,2,3,5)}， $\prod_{\mathrm{L}}(\mathrm{x})$ = {(0,1,3,4),(0,2,3,4),(0,2,3,5)}
2. du(input) = {(0,1),(0,2), (0,1,3,4),(0,2,3,4),(0,1,3,5),(0,2,3,5)}, $\prod_{\mathrm{L}}(input)$ = {(0,1),(0,2), (0,1,3,4),(0,2,3,4),(0,2,3,5)}
3. du(z) = {(4,6), (5,6)}, $\prod_{\mathrm{L}}(z)$ = {(4,6),(5,6)}

$$\frac{\sum_{i=1}^{m}\|\prod_{\mathrm{L}}(vi)\|}{\sum_{i=1}^{m}\|du(v_i)\|} * 100\% = \frac{\|\prod_{\mathrm{L}}(x)\|+\|\prod_{\mathrm{L}}(input)\|+\|\prod_{\mathrm{L}}(z)\|}{\|du(x)\|+\|du(input)\|+\|du(z)\|} * 100\% = \frac{3+5+2}{4+6+2} * 100\% = 83.3\%$$

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]

Summary

- Data Flow Coverage Criteria are proposed as a solution to reduce the complexity of achieving all path coverage by examining the def and use path for variables
- For a variable, a def is a location which writes value to its memory
- For a variable, an use is a location which reads value from its memory.
- All defs, all uses, all def uses are well known Data Flow Coverage Criteria
- Complex data type, indirect reference, alias, procedure calls make data flow analysis much harder

The End