CSCE 747 Software Testing and Quality Assurance

Lecture 07 – Dataflow Testing

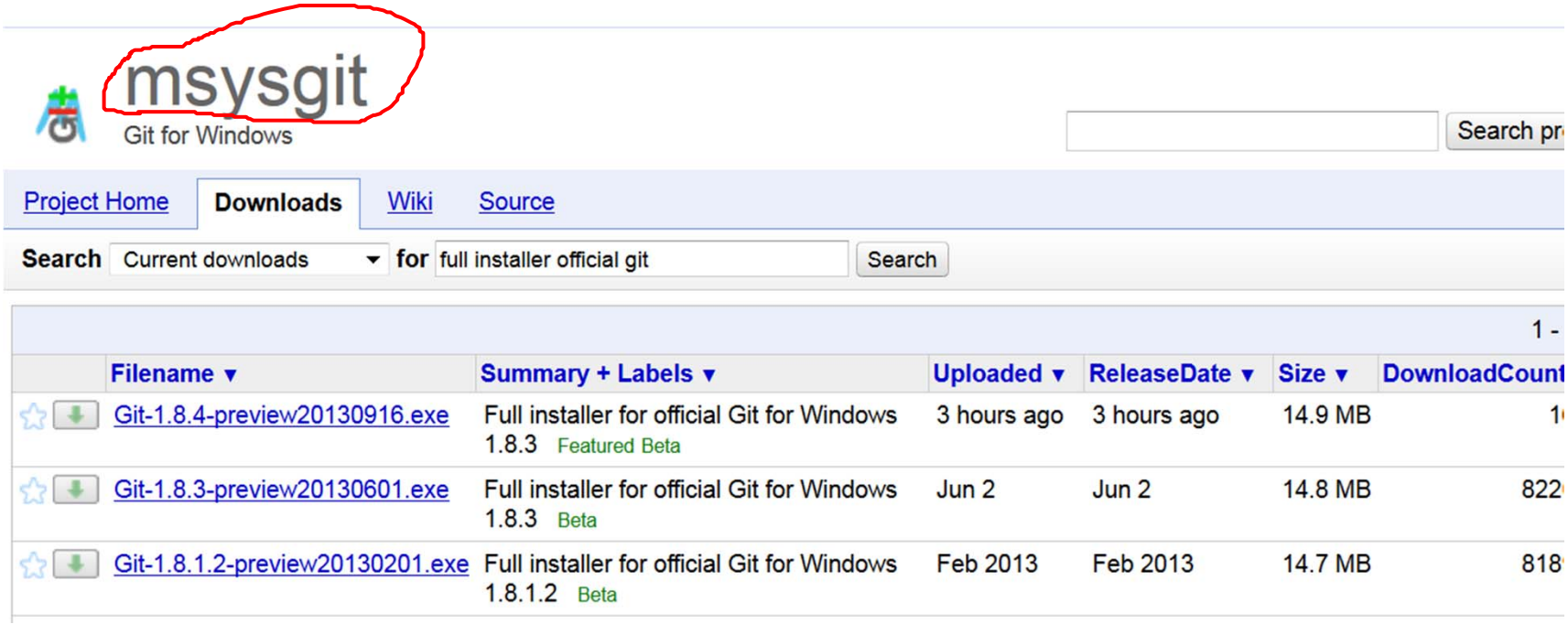
Last Time

- Lecture 06 Slides 1-19 covered last time
- Case Study Question after class
- Path Testing continued
- Ch 9 pp 131-149

Today

- Dataflow Testing
- Ch 10, pp 151-167

MSYSGIT


msysgit
Git for Windows
Search pr
Project Home
Downloads
Wiki
Source
Search
Current downloads
for
full installer official git
Search
1 -
Filename
Summary + Labels
Uploaded
ReleaseDate
Size
DownloadCount
Git-1.8.4-preview20130916.exe
Full installer for official Git for Windows 1.8.3 Featured Beta
3 hours ago
3 hours ago
14.9 MB
Git-1.8.3-preview20130601.exe
Full installer for official Git for Windows 1.8.3 Beta
Jun 2
Jun 2
14.8 MB
822
Git-1.8.1.2-preview20130201.exe
Full installer for official Git for Windows 1.8.1.2 Beta
Feb 2013
Feb 2013
14.7 MB
818

Dataflow Testing

- Dataflow testing refers to forms of structural testing that focus on:
  - the points at which variables receive values and
  - the points at which these values are used
- "Dataflow testing serves as a reality check on path testing;"

Forms of dataflow testing

- Two main forms of dataflow testing:
  - One provides a set of basic definitions and a unifying structure of test coverage metrics
  - The other based on a concept called a program slice.
- Start with program graph but move back towards functional testing

Define/Use of Variables

- Define/Use information
  - x = y +3*z
    - Define a new x; use variables y and z
- Concordances that list statement numbers in which variable names occur
- Define/reference anomalies:
  - A variable that is defined but never used
  - A variable that is used before it is defined
  - A variable that is defined twice before it is used

Static Analysis

- **Static analysis: finding faults in source code without executing it.**

Define/Use Testing

- define/use testing was done by
  - Rapps and Weyuker, IEEE Transactions on Software Engineering, Vol. SE-11, 1985
- Definitions: Given a program P
  - G(P) – the program graph; single entry; single exit
  - PATHS(P) – the set of all paths in P

"Definition" and "Usage" nodes for variable

- Definition: Node n ∈ G(P) is a defining node of the variable v ∈ V, written as DEF(v, n)

- Definition: Node n ∈ G(P) is a usage node of the variable v ∈ V, written as USE(v, n)
  - Definitions(n) – variables v that are defined in statement n
  - Usage(n) – variables that are used in statement n
  - Definitions(v) – statements that define v
  - Usage(v) – statements that use v
  - Next-Use(n, v) – list of statements following n that use v

- Node n: statement fragment x = y + z

Example (Commission)

```
10.  totalLocks = 0
11.  totalStocks = 0
12.  totalBarrels = 0
13.  Input(locks)
14.  While NOT(locks = -1) 'loop condition uses -1
15.              Input(stocks, barrels)
16.              totalLocks = totalLocks + locks
17.              totalStocks = totalStocks + stocks
18.              totalBarrels = totalBarrels + barrels
19.              Input(locks)
20.      EndWhile
21.      Output("Locks sold: ", totalLocks)
```

Defs (t, l) = {10, 16}

Predicate/Computation Use

- USE(v, n) can be classified as
  - Predicate use (P-use)
  - Computation use(C-use)
- USE(a, 7) ?
- USE(a,9)?

```
'Step 2: Is A Triangle? Modified
6. t1 = b + c
7. t2 = a + c
8. t3 = a + b
9. If (a < t1) AND (b < t2) AND(c < t3)
10.       Then IsATriangle = True
11.       Else IsATriangle = False
12. EndIf
...
```

definition-use path

- Definition: A definition-use path with respect to a variable v (denoted du-path) is a path in PATHS(P) such that
- for some $v \in V$, there are define and usage nodes DEF(v, m) and USE(v, n) such that
- m and n are the initial and final nodes of the path.

Definition-Clear Path

- Definition: A definition-clear path with respect to a variable v (denoted dc-path) is a definition-use path in PATHS(P)
- with initial and final nodes DEF (v, m) and USE (v, n) such that no other node in the path is a defining node of v
- Du-paths and dc-paths describe the flow of data across source statements from points at which the values are defined to points at which the values are used.
- Du-paths that are not definition-clear are potential trouble spots.

Compilers Again: Register Allocation

```
1. Program Commission (INPUT,OUTPUT)
2. Dim …
7. lockPrice = 45.0
8. stockPrice = 30.0
9. barrelPrice = 25.0
10. totalLocks = 0
11. totalStocks = 0
12. totalBarrels = 0
13. Input(locks)
14. While NOT(locks = -1)
15.         Input(stocks, brrls)
16.         totalLocks = totalLocks + locks
17.         totalStocks = totalStocks + stocks
18.         totalBarrels = totalBarrels + brrls
19.         Input(locks)
20.EndWhile
21. Output("Locks sold: ", totalLocks)
22. Output("Stocks sold: ", totalStocks)
23. Output("Barrels sold: ", totalBarrels)
24. lockSales = lockPrice * totalLocks
25. stockSales = stockPrice * totalStocks
26. barrelSales = barrelPrice * totalBarrels
27. sales = lockSales + stockSales + barrelSales
28. Output("Total sales: ", sales)
29. If (sales > 1800.0)
30.    Then
31.         commission = 0.10 * 1000.0
32.          commission = comm. + 0.15 * 800.0
33.          comm. = comm. + .20 *(sales-1800.0)
34.      Else If (sales > 1000.0)
35.         Then
36.              commission = 0.10 * 1000.0
37.              comm=comm + .15 *(sales-1000)
38.         Else
39.              commission = 0.10 * sales
40.          EndIf
41. EndIf
42. Output("Commission is $", commission)
43. End Commission
```

DD-Paths in Figure 10.1 (previous slide)

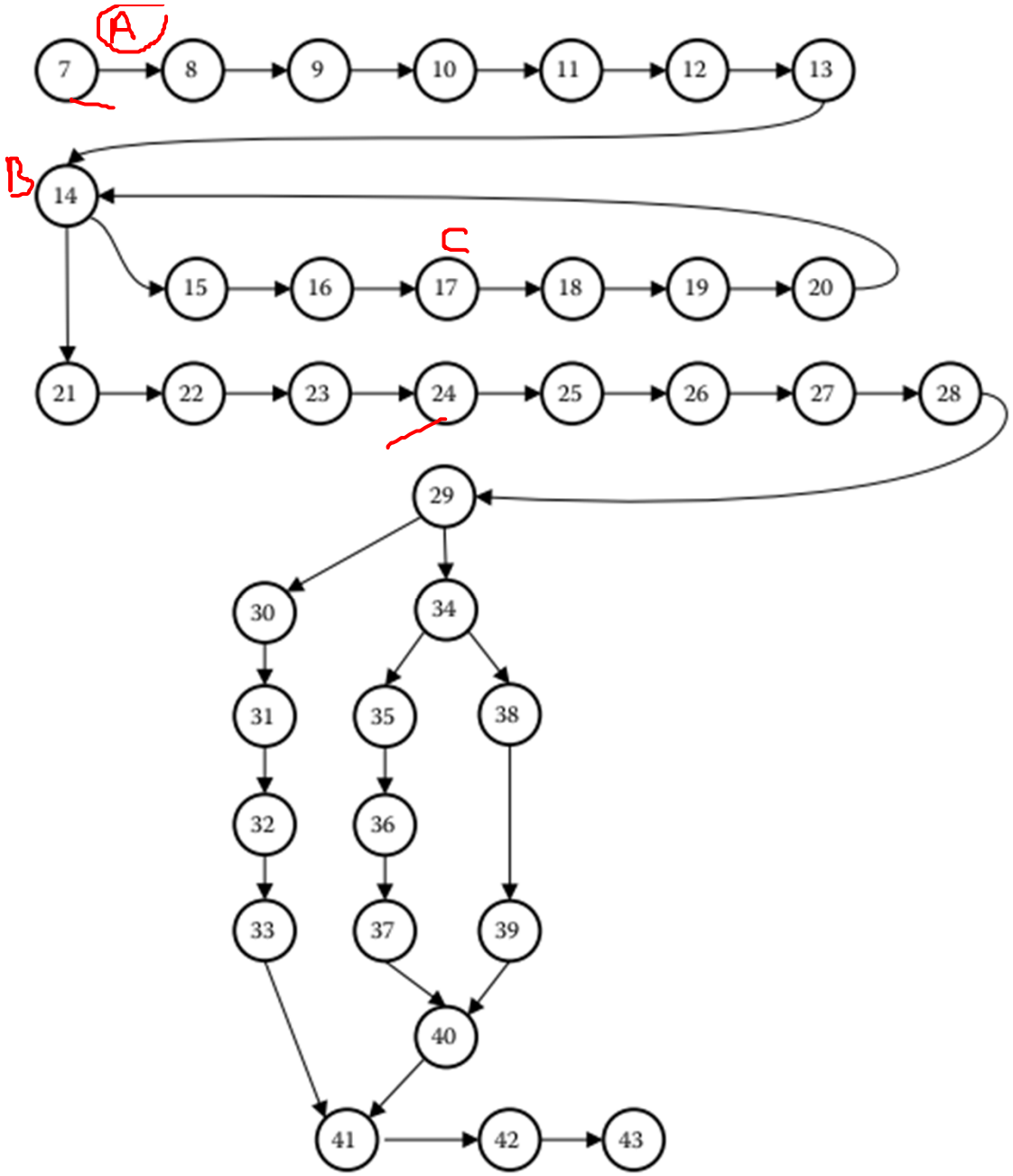
- Table 10.1 DD-Paths in Figure 10.1

DD-Path	Nodes
A	7, 8, 9, 10, 11, 12, 13
B	14
C	15, 16, 17, 18, 19,20
D	21, 22, 23, 24, 25, 26, 27, 28
E	29
F	30, 31, 32, 33
G	34
H	35, 36, 37
I	38, 39
J	40
K	41, 42, 43

Not du-path

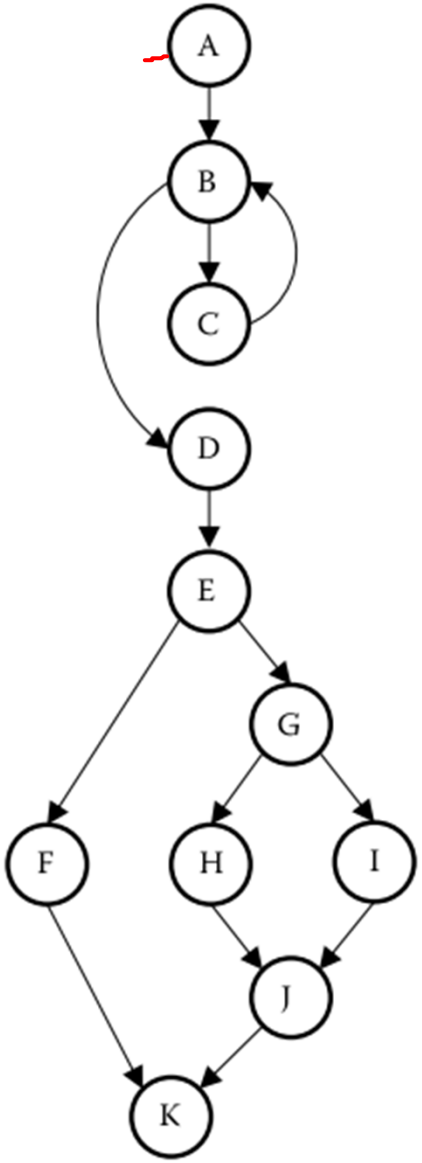
loop

Fig 10.1 program graph for commission program



Jorgensen, Paul C. Software Testing
A Craftsman Approach

Figure 10.2 DD-Path graph of the commission program.



10.1.2 du-Paths for Stocks

- Def(stocks, 15)
- Use(stocks, 17)

10.1.3 du-Paths for Locks

- p1 = <13, 14> exit
- p2 = <13, 14, 15, 16>
- p3 = <19, 20, 14>
- p4 = <19, 20, 14, 15, 16>

10.1.4 du-Paths for Total-Locks

- p6 = <10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21>
- p7 = <10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 21, 22, 23, 24>
- p7 = < p6, 22, 23, 24>
- p8 = <16, 17, 18, 19, 20, 14, 21>
- p9 = <16, 17, 18, 19, 20, 14, 21, 22, 23, 24>

Table 10.2 Define/Use Nodes for Variables in the Commission Problem

Variable	*Defined at Node*	*Used at Node*
lockPrice	7	24
stockPrice	8	25
barrelPrice	9	26
totalLocks	10, 16	16, 21, 24
totalStocks	11, 17	17, 22, 25
totalBarrels	12, 18	18, 23, 26
locks	13, 19	14, 16
stocks	15	17
barrels	15	18
lockSales	24	27
stockSales	25	27
barrelSales	26	27
sales	27	28, 29, 33, 34, 37, 39
commission	31, 32, 33, 36, 37, 39	32, 33, 37, 42

10.1.5 du-Paths for Sales

- p10 = <27, 28>
- p11 = <27, 28, 29>
- p12 = <27, 28, 29, 30, 31, 32, 33>

- p13 = <27, 28, 29, 34>
- p14 = <27, 28, 29, 34, 35, 36, 37>
- p15 = <27, 28, 29, 34, 38,39>

Du-Paths for commission

Table 10.3 Selected Define/Use Paths

Variable	*Path (Beginning, End) Nodes*	*Definition-Clear?*
lockPrice	7, 24	Yes
stockPrice	8, 25	Yes
barrelPrice	9, 26	Yes
totalStocks	11, 17	Yes
totalStocks	11, 22	No
totalStocks	11, 25	No
totalStocks	17, 17	Yes
totalStocks	17, 22	No
totalStocks	17, 25	No
locks	13, 14	Yes
locks	13, 16	Yes
locks	19, 14	Yes
locks	19, 16	Yes
sales	27, 28	Yes
sales	27, 29	Yes
sales	27, 33	Yes
sales	27, 34	Yes
sales	27, 37	Yes
sales	27, 39	Yes

Jorgens
A

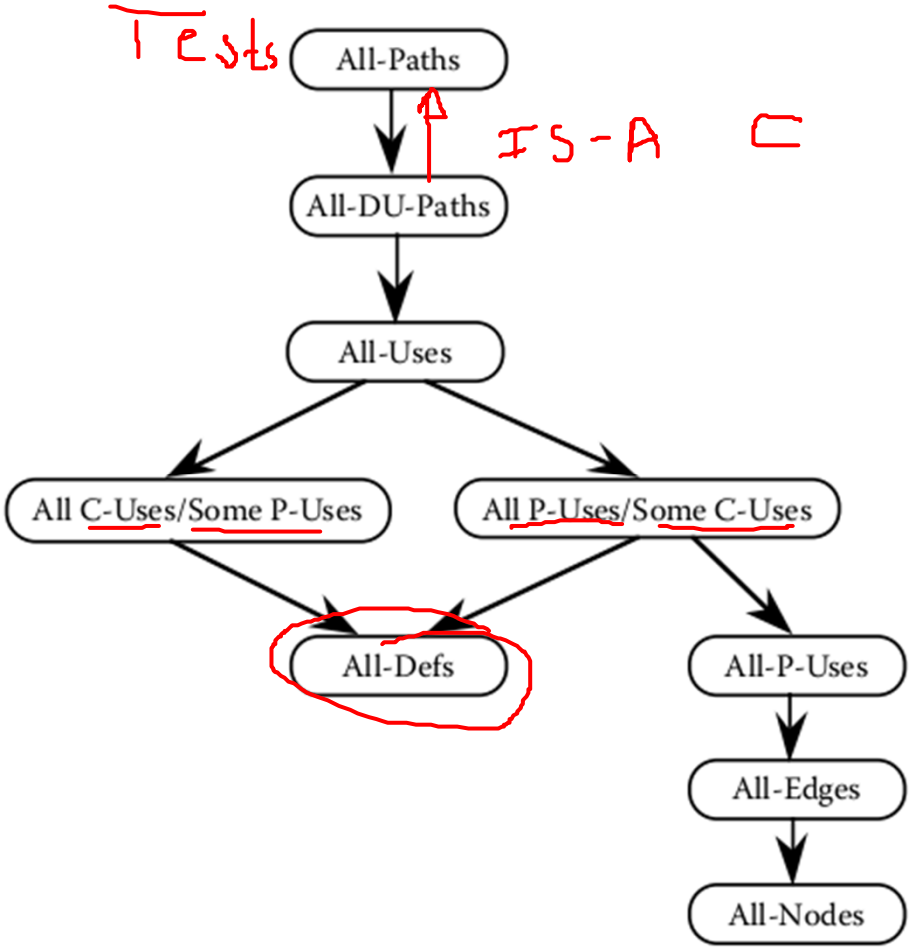
Table 10.4 Define/Use Paths for Commission

Variable	Path (Beginning, End) Nodes	Feasible?	Definition-Clear?
commission	31, 32	Yes	Yes
commission	31, 33	Yes	No
commission	31, 37	No	n/a
commission	31, 42	Yes	No
commission	32, 32	Yes	Yes
commission	32, 33	Yes	Yes
commission	32, 37	No	n/a
commission	32, 42	Yes	No
commission	33, 32	No	n/a
commission	33, 33	Yes	Yes
commission	33, 37	No	n/a
commission	33, 42	Yes	Yes
commission	36, 32	No	n/a
commission	36, 33	No	n/a
commission	36, 37	Yes	Yes
commission	36, 42	Yes	No
commission	37, 32	No	n/a
commission	37, 33	No	n/a
commission	37, 37	Yes	Yes
commission	37, 42	Yes	Yes
commission	38, 32	No	n/a
commission	38, 33	No	n/a
commission	38, 37	No	n/a
commission	38, 42	Yes	Yes

10.1.7 du-Path Test Coverage Metrics

- **Definition: The set T satisfies the All-Defs criterion for the program P iff for every variable v ∈ V, T contains definition-clear paths from every defining node of v to a use of v.**
- **Definition: The set T satisfies the All-Uses criterion for the program P iff for every variable v ∈ V, T contains definition-clear paths from every defining node of v to every use of v, and to the successor node of each USE(v, n).**
- **Definition: The set T satisfies the All-P-Uses/Some C-Uses criterion for the program P iff for every variable v ∈ V, T contains definition-clear paths from every defining node of v to every predicate use of v;**
- **if a definition of v has no P-uses, a definition-clear path leads to at least one computation use.**

Rapps–Weyuker hierarchy of dataflow coverage metrics.



Jorgensen, Paul C. Software Testing
A Craftsman Approach

- **Definition: The set T satisfies the All-C-Uses/Some P-Uses criterion for the program P iff for every variable v ∈ V, T contains definition-clear paths from every defining node of v to every computation use of v;**
- **if a definition of v has no C-uses, a definition-clear path leads to at least one predicate use.**
- **Definition: The set T satisfies the All-du-paths criterion for the program P iff for every variable v ∈ V, T contains definition-clear paths from every defining node of v to every use of v and to the successor node of each USE(v, n), and that these paths are either single-loop traversals or cycle-free.**

Slices

- Slices of History:
  - American History
  - Western Civ.
  - Chinese History
  - Russian History
  - … history focusing on one group

HIST 201, 202
HIST 101, 102

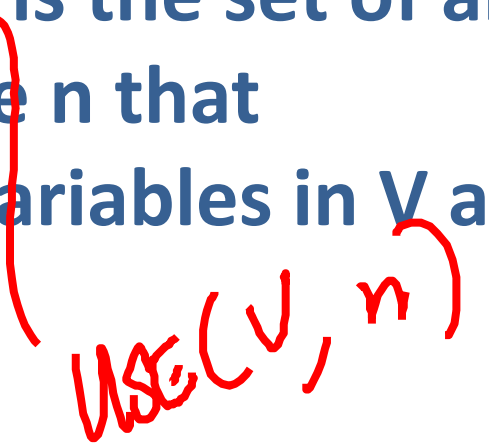
- Slices of program
  - "set of program statements that contributes to or
  - affects a value for a variable at some point in the program"

10.2 Slice-Based Testing

- Program slices have surfaced and submerged in software engineering literature since the early 1980s.
- They were originally proposed in Weiser (1988), used as an approach to software maintenance in Gallagher and Lyle (1991), and more recently
- used to quantify functional cohesion in Bieman and Ott (1994)
- Part of this versatility is due to the natural, intuitively clear intent of the program slice concept.
- Informally, a program slice is a set of program statements that contributes to or affects a value for a variable at some point in the program.

slice on the variable set V

- **Definition: Given a program P and a set V of variables in P, a slice on the variable set V at statement n, written S(V, n), is the set of all statements in P prior to node n that contribute to the values of variables in V at node n.**



Use nodes

- USE relationship pertains to five forms of usage:
- P-use Used in a predicate (decision)
- C-use Used in computation
- O-use Used for output
- L-use Used for location (pointers, subscripts)
- I-use Iteration (internal counters, loop indices)

Definition Nodes

- two forms of definition nodes:
- I-def Defined by input
- A-def Defined by assignment

- Slices on the locks
- S1: S(locks, 13) = {13}
- S2: S(locks, 14) = {13, 14, 19, 20}
- S3: S(locks, 16) = {13, 14, 19, 20}

- Jorgensen, Paul C. (2011-07-16). Software Testing (Page 162). Auerbach Publications. Kindle Edition.

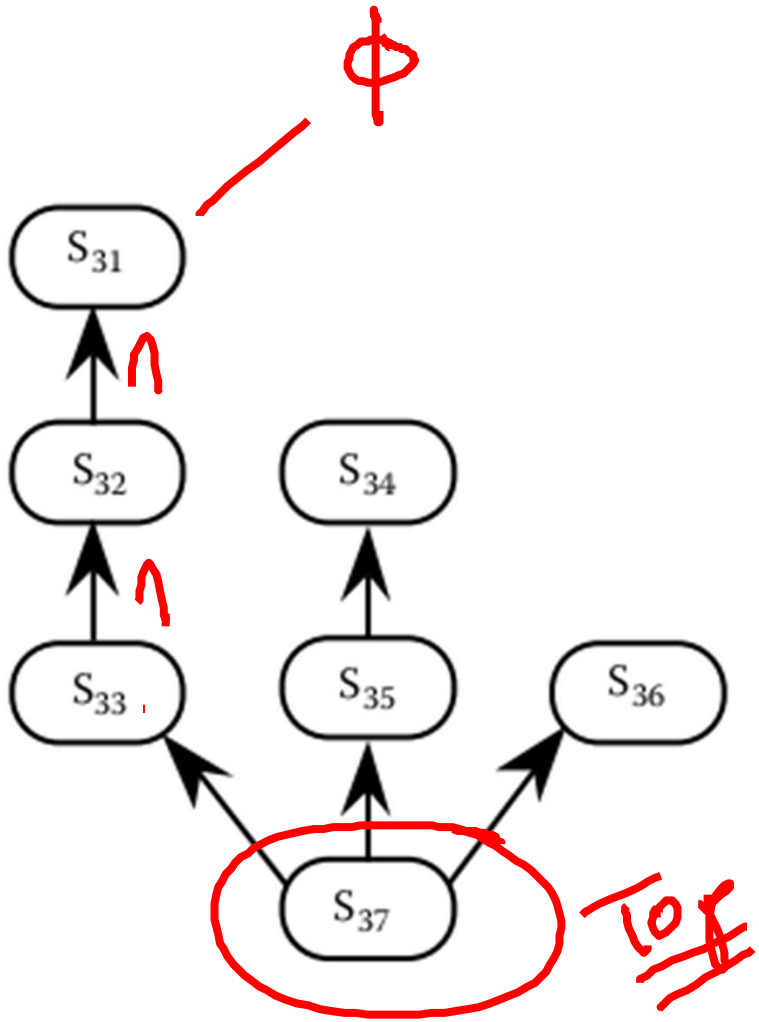
Slices and Loops

- S9: S(totalLocks, 10) = {10}
- S10: S(totalLocks, 16) = {10, 13, 14, 16, 19, 20}
- S11: S(totalLocks, 21) = {10, 13, 14, 16, 19, 20}

- **S12: S(totalStocks, 11) = {11}**
- **S13: S(totalStocks, 17) = {11, 13, 14, 15, 17, 19, 20}**
- **S14: S(totalStocks, 22) = {11, 13, 14, 15, 17, 19, 20}**
- **S15: S(totalBarrels, 12) = {12}**
- **S16: S(totalBarrels, 18) = {12, 13, 14, 15, 18, 19, 20}**
- **S17: S(totalBarrels, 23) = {12, 13, 14, 15, 18, 19, 20}**

Lattice of Slices for Commission Problem

- S31: S(commission, 31) = {31}
- S32: S(commission, 32) = {31, 32}
- S33: S(commission, 33) = {7, 8, 9 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 29, 30, 31, 32, 33}
- S34: S(commission, 36) = {36}
- S35: S(commission, 37) = {7, 8, 9 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 36, 37}
- S36: S(commission, 39) = {7, 8, 9 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 29, 34, 38, 39}
- S37: S(commission, 41) = {7, 8, 9 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39}



Jorgensen, Paul C. Software Testing
A Craftsman Approach