Dataflow Testing

Chapter 9

Dataflow Testing

- Testing All-Nodes and All-Edges in a control flow graph may miss significant test cases
- Testing All-Paths in a control flow graph is often too time-consuming
- Can we select a subset of these paths that will reveal the most faults?
- Dataflow Testing focuses on the points at which variables receive values and the points at which these values are used

Dataflow Analysis

- Can reveal interesting bugs
  - **A variable that is defined but never used**
  - **A variable that is used but never defined**
  - **A variable that is defined twice before it is used**
  - **Sending a modifier message to an object more than once between accesses**
  - **Deallocating a variable before it is used**
    - **Container problem**
      - Deallocating container loses references to items in the container, memory leak

Definitions

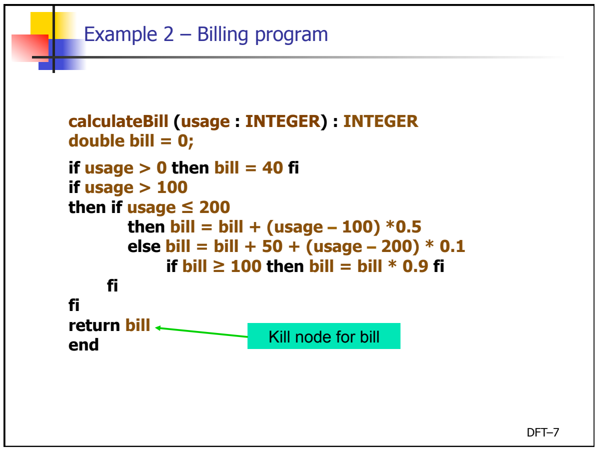
- A node **n** in the program graph is a **defining** node for variable **v** – **DEF(v, n)** – if the value of **v** is defined at the statement fragment in that node
  - **Input, assignment, procedure calls**
- A node in the program graph is a **usage** node for variable **v** – **USE(v, n)** – if the value of **v** is used at the statement fragment in that node
  - **Output, assignment, conditionals**

Definitions – 2

- A usage node is a predicate use, **P-use**, if variable **v** appears in a predicate expression
  - **Always in nodes with outdegree ≥ 2**
- A usage node is a computation use, **C-use**, if variable **v** appears in a computation
  - **Always in nodes with outdegree ≤ 1**

Definitions – 3

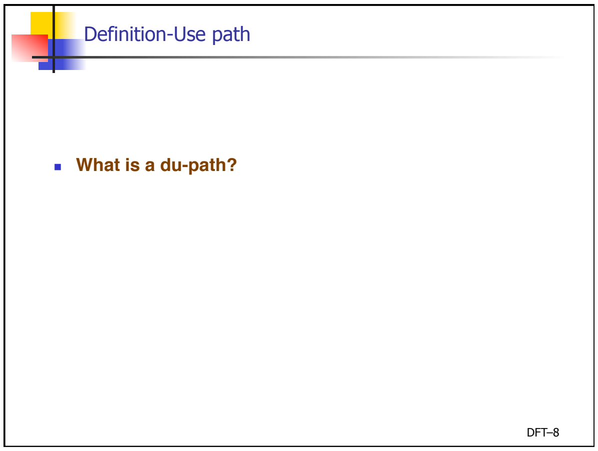
- A node in the program is a **kill** node for a variable **v** – **KILL(v, n)** – if the variable is deallocated at the statement fragment in that node



Example 2 – Billing program

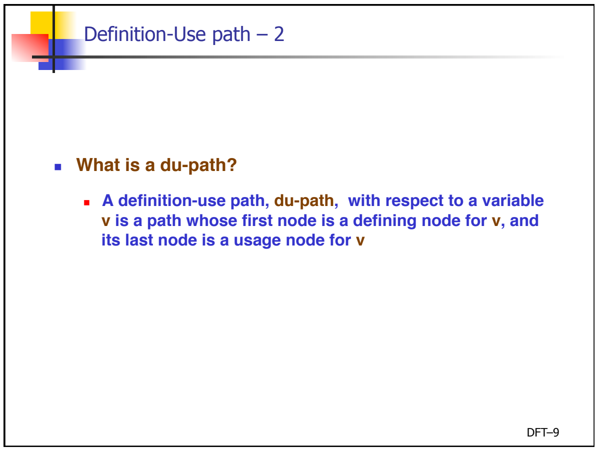
```
calculateBill (usage : INTEGER) : INTEGER
double bill = 0;
if usage > 0 then bill = 40 fi
if usage > 100
then if usage ≤ 200
          then bill = bill + (usage – 100) *0.5
          else bill = bill + 50 + (usage – 200) * 0.1
                if bill ≥ 100 then bill = bill * 0.9 fi
      fi
fi
return bill
end
```

Kill node for bill



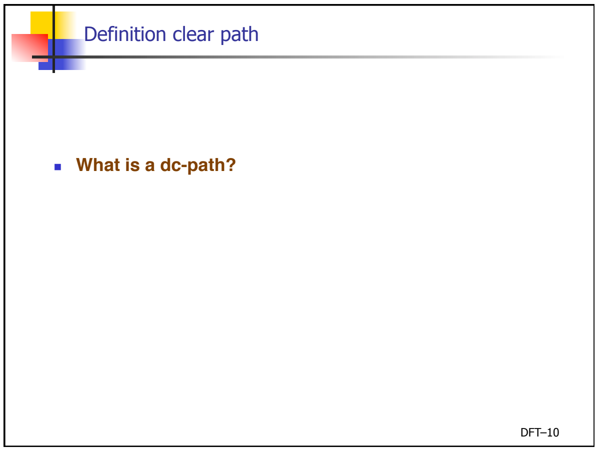
Definition-Use path

- What is a du-path?



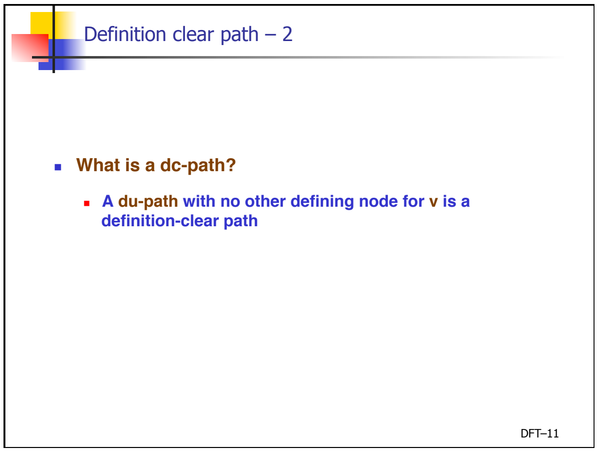
Definition-Use path – 2

- What is a du-path?
    - A definition-use path, du-path, with respect to a variable v is a path whose first node is a defining node for v, and its last node is a usage node for v



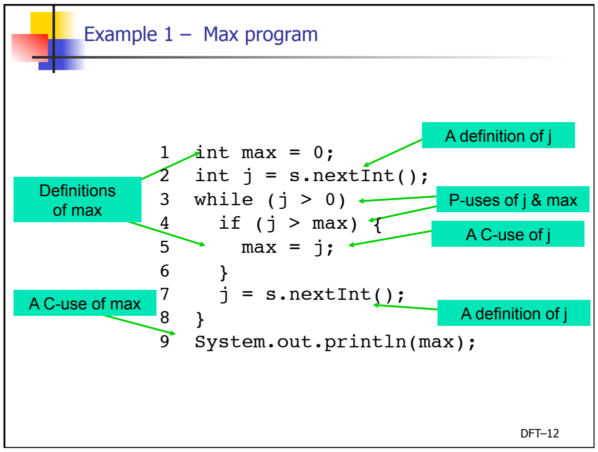
Definition clear path

- What is a dc-path?



Definition clear path – 2

- What is a dc-path?
    - A du-path with no other defining node for v is a definition-clear path



Example 1 – Max program

```
int max = 0;
int j = s.nextInt();
while (j > 0)
  if (j > max) {
    max = j;
  }
  j = s.nextInt();
}
System.out.println(max);
```

A definition of j

Definitions of max

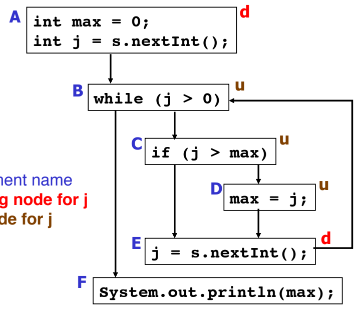
P-uses of j & max

A C-use of j

A C-use of max

A definition of j

Max program – analysis



Legend
A..F Segment name
d defining node for j
u use node for j

dc-paths j
A B
A B C
A B C D
E B
E B C
E B C D

dc-paths max
A B F
A B C
D E B C
D E B F

Dataflow Coverage Metrics

- Based on these definitions we can define a set of coverage metrics for a set of test cases
- We have already seen
  - **All-Nodes**
  - **All-Edges**
  - **All-Paths**
- Data flow has additional test metrics for a set T of paths in a program graph
  - **All assume that all paths in T are feasible**

All-Defs Criterion

- The set T satisfies the All-Def criterion
  - **For every variable v, T contains a dc-path from every defining node for v to at least one usage node for v**
    - **Not all use nodes need to be reached**

$$\forall v \in V(P), nd \in prog\_graph(P) \mid DEF(v,nd)$$
$$\bullet \exists nu \in prog\_graph(P) \mid USE(v,nu)$$
$$\bullet dc\_path(nd,nu) \in T$$

All-Uses Criterion

- The set T satisfies the All-Uses criterion iff
  - **For every variable v, T contains dc-paths that start at every defining node for v, and terminate at every usage node for v**
    - **Not DEF(v, n) × USE(v, n) – not possible to have a dc-path from every defining node to every usage node**

$$(\forall v \in V(P), nu \in prog\_graph(P) \mid USE(v,nu)$$
$$\bullet \exists nd \in prog\_graph(P) \mid DEF(v,nd) \bullet dc\_path(nd,nu) \in T)$$
$$\wedge$$
$$all\_defs\_criterion$$

All-P-uses / Some-C-uses

- The set T satisfies the All-P-uses/Some-C-uses criterion iff
  - **For every variable v in the program P, T contains a dc-path from every defining node of v to every P-use node for v**
    - **If a definition of v has no P-uses, a dc-path leads to at least one C-use node for v**

$$(\forall v \in V(P), nu \in prog\_graph(P) \mid P\_use(v,nu)$$
$$\bullet \exists nd \in prog\_graph(P) \mid DEF(v,nd) \bullet dc\_path(nd,nu) \in T)$$
$$\wedge$$
$$all\_defs\_criterion$$

All-C-uses / Some-P-uses

- The test set T satisfies the All-C-uses/Some-P-uses criterion iff
  - **For every variable v in the program P, T contains a dc-path from every defining node of v to every C-use of v**
    - **If a definition of v has no C-uses, a dc-path leads to at least one P-use**

$$(\forall v \in V(P), nu \in prog\_graph(P) \mid C\_use(v,nu)$$
$$\bullet \exists nd \in prog\_graph(P) \mid DEF(v,nd) \bullet dc\_path(nd,nu) \in T)$$
$$\wedge$$
$$all\_defs\_criterion$$

Miles-per-gallon Program

```
miles_per_gallon ( miles, gallons, price : INTEGER )
if gallons = 0 then
   // Watch for division by zero!!
   Print("You have " + gallons + "gallons of gas")
else if miles/gallons > 25
   then print( "Excellent car.  Your mpg is "
               + miles/gallon)
   else print( "You must be going broke.  Your mpg is "
               + miles/gallon + " cost " + gallons * price)
fi
end
```

Miles-per-gallon Program – 2

- We want du- and dc-paths
- What do you do next?

Mile-per-gallon Program – Segmented

gasguzzler (miles, gallons, price : INTEGER)	A
if gallons = 0 then	B

// Watch for division by zero!!
Print("You have " + gallons + "gallons of gas")
	C

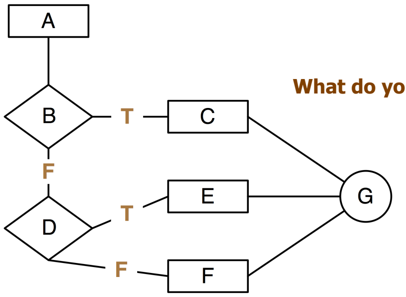
else if miles/gallons > 25	D
then print("Excellent car. Your mpg is " + miles/gallon)	E
else print("You must be going broke. Your mpg is " + miles/gallon + " cost " + gallons * price)	F

fi
end
	G

Miles-per-gallon Program – 3

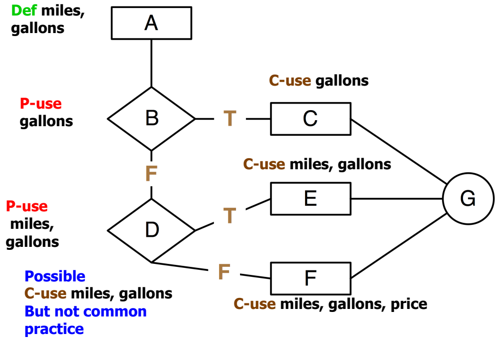
- We want du- and dc-paths
- What do you do next?

MPG program graph



What do you do now?

MPG program graph



Def miles, gallons

P-use gallons

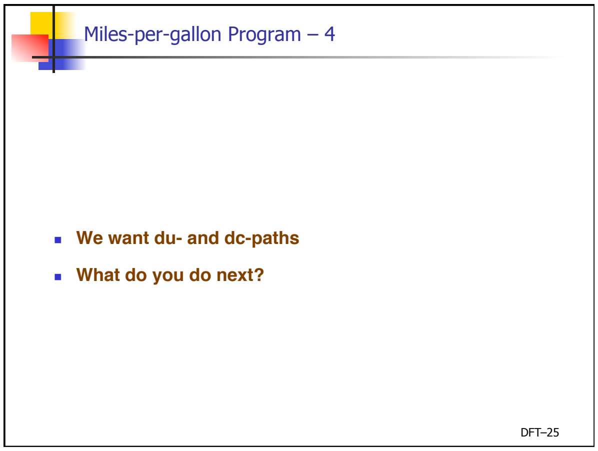
C-use gallons

C-use miles, gallons

P-use miles, gallons

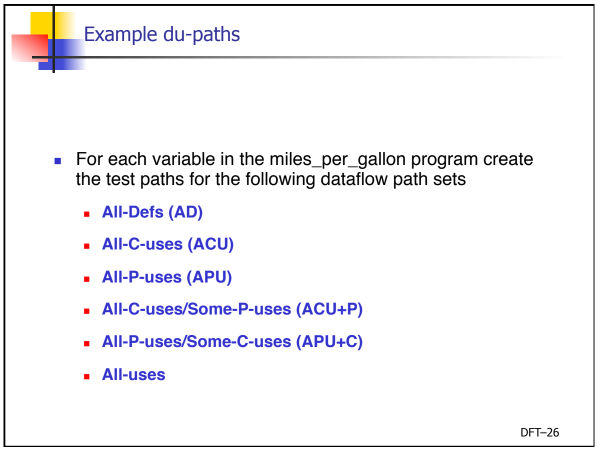
Possible C-use miles, gallons But not common practice

C-use miles, gallons, price



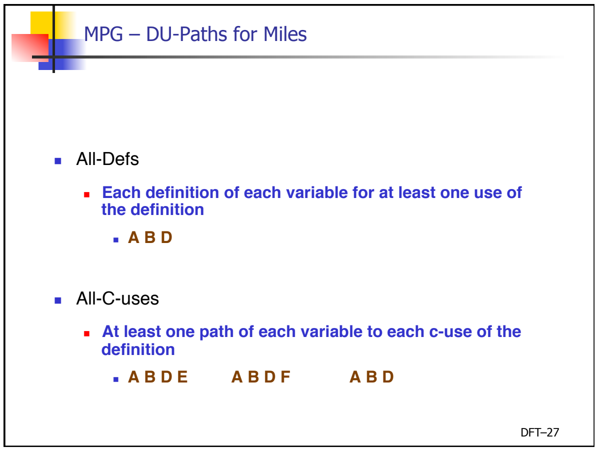
Miles-per-gallon Program – 4

- **We want du- and dc-paths**
- **What do you do next?**



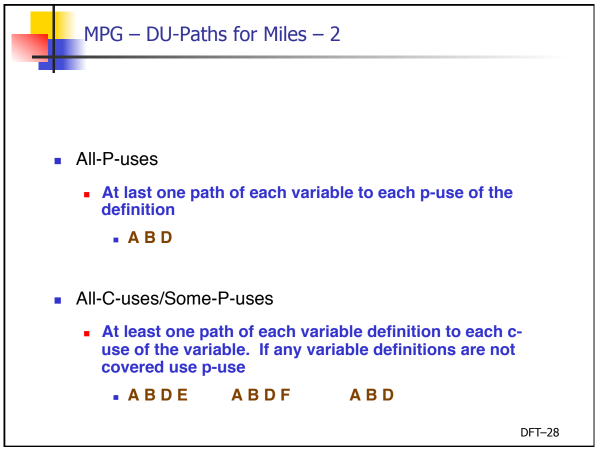
Example du-paths

- For each variable in the miles_per_gallon program create the test paths for the following dataflow path sets
  - **All-Defs (AD)**
  - **All-C-uses (ACU)**
  - **All-P-uses (APU)**
  - **All-C-uses/Some-P-uses (ACU+P)**
  - **All-P-uses/Some-C-uses (APU+C)**
  - **All-uses**



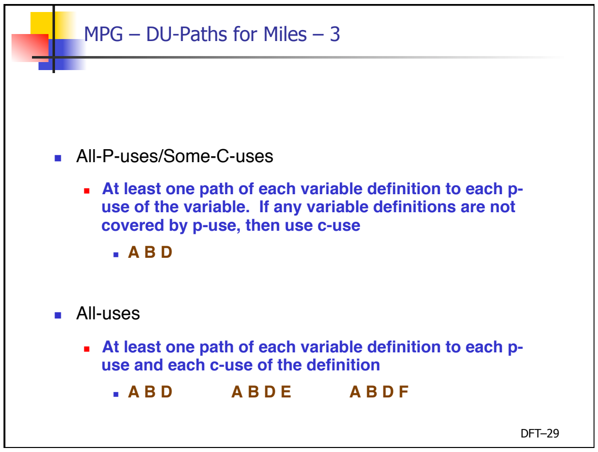
MPG – DU-Paths for Miles

- All-Defs
  - **Each definition of each variable for at least one use of the definition**
    - **A B D**
- All-C-uses
  - **At least one path of each variable to each c-use of the definition**
    - **A B D E   A B D F   A B D**



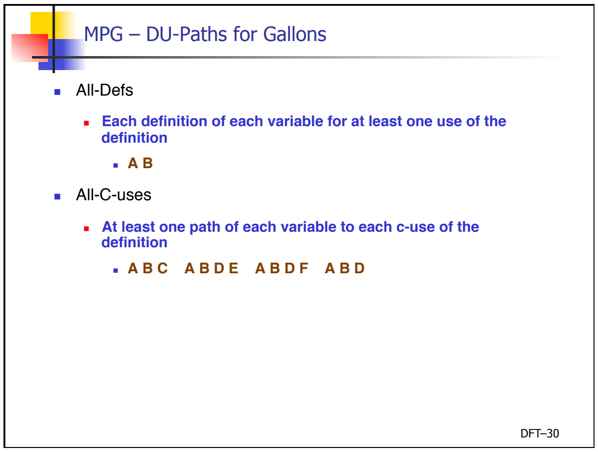
MPG – DU-Paths for Miles – 2

- All-P-uses
  - **At last one path of each variable to each p-use of the definition**
    - **A B D**
- All-C-uses/Some-P-uses
  - **At least one path of each variable definition to each c-use of the variable. If any variable definitions are not covered use p-use**
    - **A B D E   A B D F   A B D**



MPG – DU-Paths for Miles – 3

- All-P-uses/Some-C-uses
  - **At least one path of each variable definition to each p-use of the variable. If any variable definitions are not covered by p-use, then use c-use**
    - **A B D**
- All-uses
  - **At least one path of each variable definition to each p-use and each c-use of the definition**
    - **A B D   A B D E   A B D F**



MPG – DU-Paths for Gallons

- All-Defs
  - **Each definition of each variable for at least one use of the definition**
    - **A B**
- All-C-uses
  - **At least one path of each variable to each c-use of the definition**
    - **A B C   A B D E   A B D F   A B D**

MPG – DU-Paths for Gallons – 2

- All-P-uses
  - At least one path of each variable definition to each p-use of the definition
    - A B A B D
- All-C-uses/Some-P-uses
  - At least one path of each variable definition to each c-use of the variable. If any variable definitions are not covered by c-use, then use p-use
    - A B C A B D E A B D F A B D

MPG – DU-Paths for Gallons – 3

- All-P-uses/Some-C-uses
  - At least one path of each variable definition to each p-use of the variable. If any variable definitions are not covered use c-use
    - A B A B D
- All-uses
  - At least one path of each variable definition to each p-use and each c-use of the definition
    - A B A B C A B D A B D E A B D F

MPG – DU-Paths for Price

- All-Defs
  - Each definition of each variable for at least one use of the definition
    - A B D F
- All-C-uses
  - At least one path of each variable to each c-use of the definition
    - A B D F

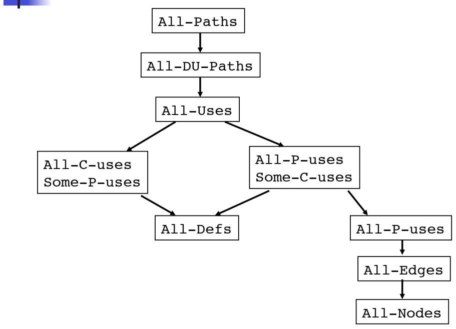
MPG – DU-Paths for Price – 2

- All-P-uses
  - At least one path of each variable definition to each p-use of the definition
    - None
- All-C-uses/Some-P-uses
  - At least one path of each variable definition to each c-use of the variable. If any variable definitions are not covered use p-use
    - A B D F

MPG – DU-Paths for Price – 2

- All-P-uses/Some-C-uses
  - At least one path of each variable definition to each p-use of the variable. If any variable definitions are not covered use c-use
    - A B D F
- All-uses
  - At least one path of each variable definition to each p-use and each c-use of the definition
    - A B D F

Rapps-Weyuker data flow hierarchy



All-Paths

All-DU-Paths

All-Uses

All-C-uses Some-P-uses

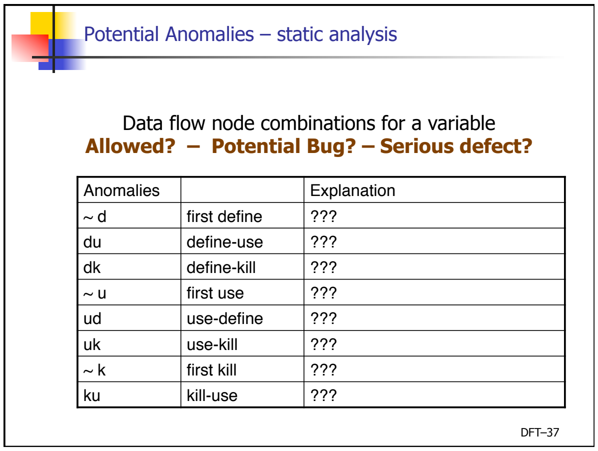
All-P-uses Some-C-uses

All-Defs

All-P-uses

All-Edges

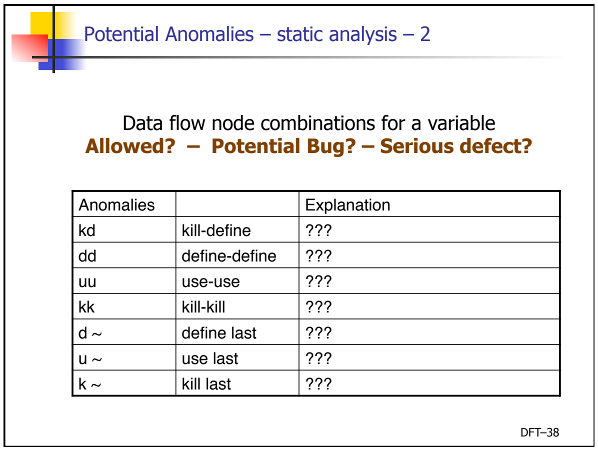
All-Nodes



Potential Anomalies – static analysis

Data flow node combinations for a variable
Allowed? – Potential Bug? – Serious defect?

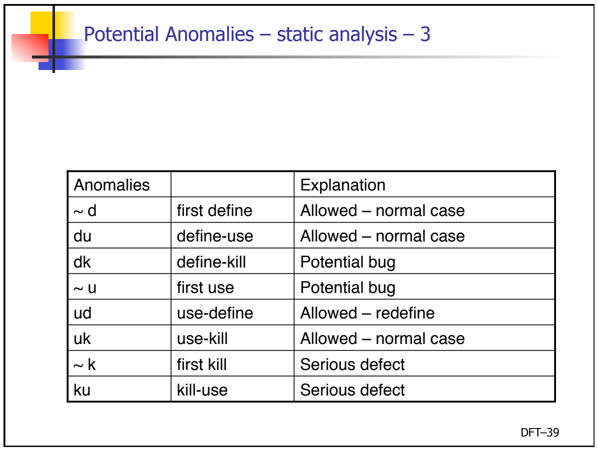
Anomalies		Explanation
~ d	first define	???
du	define-use	???
dk	define-kill	???
~ u	first use	???
ud	use-define	???
uk	use-kill	???
~ k	first kill	???
ku	kill-use	???



Potential Anomalies – static analysis – 2

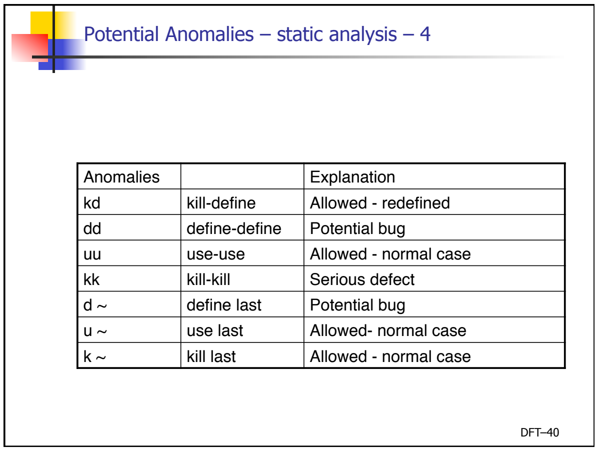
Data flow node combinations for a variable
Allowed? – Potential Bug? – Serious defect?

Anomalies		Explanation
kd	kill-define	???
dd	define-define	???
uu	use-use	???
kk	kill-kill	???
d ~	define last	???
u ~	use last	???
k ~	kill last	???



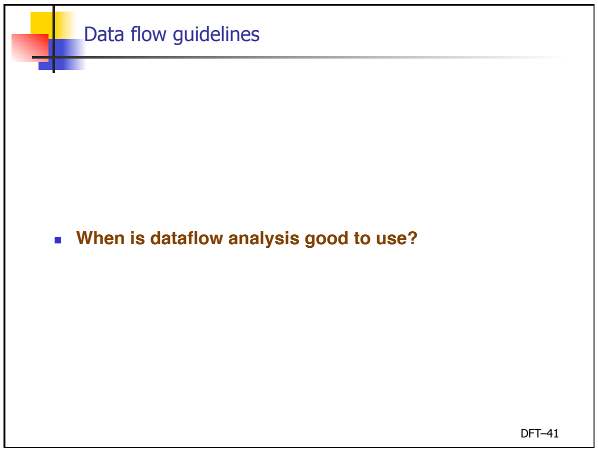
Potential Anomalies – static analysis – 3

Anomalies		Explanation
~ d	first define	Allowed – normal case
du	define-use	Allowed – normal case
dk	define-kill	Potential bug
~ u	first use	Potential bug
ud	use-define	Allowed – redefine
uk	use-kill	Allowed – normal case
~ k	first kill	Serious defect
ku	kill-use	Serious defect



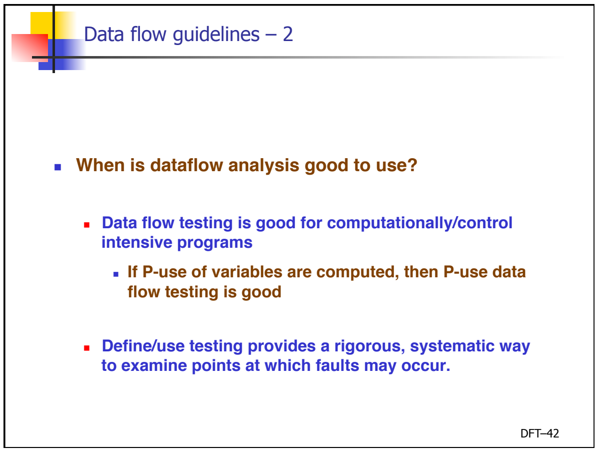
Potential Anomalies – static analysis – 4

Anomalies		Explanation
kd	kill-define	Allowed - redefined
dd	define-define	Potential bug
uu	use-use	Allowed - normal case
kk	kill-kill	Serious defect
d ~	define last	Potential bug
u ~	use last	Allowed- normal case
k ~	kill last	Allowed - normal case



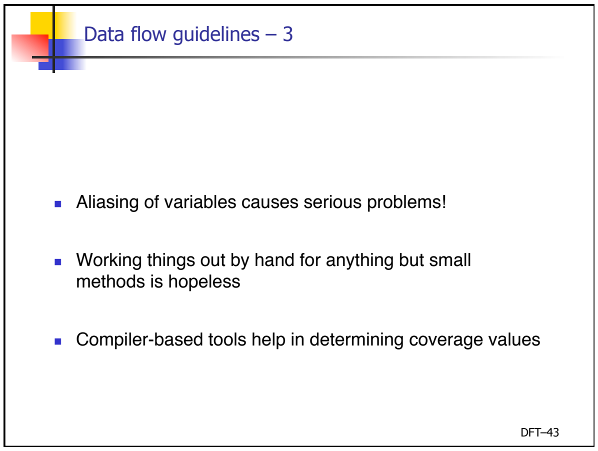
Data flow guidelines

- **When is dataflow analysis good to use?**



Data flow guidelines – 2

- **When is dataflow analysis good to use?**
  - **Data flow testing is good for computationally/control intensive programs**
    - **If P-use of variables are computed, then P-use data flow testing is good**
  - **Define/use testing provides a rigorous, systematic way to examine points at which faults may occur.**



Data flow guidelines – 3

- Aliasing of variables causes serious problems!
- Working things out by hand for anything but small methods is hopeless
- Compiler-based tools help in determining coverage values