Graph: Data Flow Coverage Criteria

CS 4501 / 6501
Software Testing

[Ammann and Offutt, “Introduction to Software Testing,” Ch. 7]
Structures for Criteria-Based Testing

Four structures for modeling software

Input space

Graph
- Source
- Design
- Specs
- Use cases

Applied to

Logic
- Source
- Specs
- FSMs
- DNF

Applied to

Syntax
- Source
- Models
- Integration
- Inputs

Applied to

---R
R--R
RI--R
RIPR
Graph Coverage Criteria

Satisfaction

- Given a set $TR$ of test requirements for a criterion $C$, a set of tests $T$ satisfies $C$ on a graph if and only if for every test requirement in $TR$, there is a test path in $\text{path}(T)$ that meets the test requirement $tr$

Two types

1. **Structural coverage criteria**
   - Define a graph just in terms of nodes and edges

2. **Data flow coverage criteria**
   - Requires a graph to be annotated with references to variables
Today’s Objectives

- Analyze data flow of software artifacts
- Understand how to integrate data flow into a graph model of the program under test
- Focusing on the flow of data, understand how to define criteria and design tests
  - All-Defs Coverage (ADC)
  - All-Uses Coverage (AUC)
  - All-DU-Paths Coverage (ADUPC)
Data Flow Criteria

• Goal: Ensure that the values are created and used correctly
• How: Focus on definitions and uses of values
• Definition (def): A location where a value for a variable is stored in a memory
• Use: A location where a variable’s value is accessed

Value are carried from defs to uses, refer to as “du-pairs”

• Also known definition-use, def-use, du associations
Data Flow Criteria

Data flow coverage criteria define test requirements $TR$ in terms of the flows of data values in a graph $G$

Steps:

1. Develop a model of the software as a graph
2. Integrate data flow into the graph
3. A test requirement is met by visiting a particular node or edge or by touring a particular path
Def, Use, and DU Pairs

- **def(n) or def(e):** The set of variables defined by node $n$ or edge $e$

- **use(n) or use(e):** The set of variables used by node $n$ or edge $e$

- **DU-pair:** A pair of locations $(l_i, l_j)$ such that a variable $v$ is defined at $l_i$ and used at $l_j$
Example: Defs, Uses, DU-Pairs

\[ X = 42 \]

\[ Z = X \times 2 \]

\[ Z = X - 8 \]

**defs:**
- \( \text{def}(1) = \{ X \} \)
- \( \text{def}(5) = \{ Z \} \)
- \( \text{def}(6) = \{ Z \} \)

**uses:**
- \( \text{use}(5) = \{ X \} \)
- \( \text{use}(6) = \{ X \} \)

**du-pairs:** for variable \( X \)
- \( (1, 5) \)
- \( (1, 6) \)
Example (2)

- All variables involved in a decision are assumed to be used on the associated edges

\[
def(1) = \{a, b\} \\
use(1,2) = \{a, b\} \\
def(1) = \{a, b\} \\
use(1,3) = \{a, b\} \\
use(1,4) = \{a, b\} \\
use(1,4) = \{a, b\} \\
\]
Def-clear and Reach

- **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.

- The values given in *defs* should reach at least one, some, or all possible *uses*. However, a def of a variable may or may not reach a particular use.

- **Why?**
  - No path goes from a location where a variable is defined to a location where the variable is used.
  - A variable’s value may be changed by another def before it reaches the use.

- **Reach**: If there is a def-clear path from $l_i$ to $l_j$ with respect to $v$, the def of $v$ at $l_i$ reaches the use at $l_j$. 
DU-Paths

- **du-path**: A simple subpath that is def-clear with respect to a variable $v$ from a def of $v$ to a use of $v$

- **$du(n_i, n_j, v)$**: the set of du-paths from $n_i$ to $n_j$

- **$du(n_i, v)$**: the set of du-paths that start at $n_i$

- Keep the path simple to ensure a reasonably small number of paths
Example: DU-Paths

defs:
- def(1) = \{ X \}
- def(5) = \{ Z \}
- def(6) = \{ Z \}

uses:
- use(5) = \{ X \}
- use(6) = \{ X \}

du-paths: for variable X
- du(1, 5, X) = \{ [1,2,4,5], [1,3,4,5] \}
- du(1, 6, X) = \{ [1,2,4,6], [1,3,4,6] \}

du-pairs: for variable X
- (1, 5)
- (1, 6)
Categorizing DU-Paths

• The core of data flow testing – allowing definitions to flow to uses

• The test criteria for data flow will be defined as sets of du-paths. Thus, we first categorize the du-paths according to:

  • def-path set
    • du($n_i, \nu$): All simple paths w.r.t. a given variable $\nu$ defined in a given node

  • def-pair set
    • du($n_i, n_j, \nu$): All simple paths w.r.t. a given variable $\nu$ from a given definition ($n_i$) to a given use ($n_j$)
Example: Def-Path and Def-Pair

\[ X = 42 \]

\[ Z = X - 8 \]

\[ Z = X \times 2 \]

- **du-path sets**
  \[ du(1, X) = \{[1,2,4,5], [1,3,4,5], [1,2,4,6], [1,3,4,6]\} \]

- **du-pair sets**
  \[ du(1, 5, X) = \{[1,2,4,5], [1,3,4,5]\} \]
  \[ du(1, 6, X) = \{[1,2,4,6], [1,3,4,6]\} \]
Touring DU-Paths

A test path $p$ du-tours subpath $d$ with respect to $v$ if $p$ tours $d$ and the subpath taken is def-clear with respect to $v$.

Sidetrips can be used, just as with previous touring.

Test path $[1,2,4,5,7]$ du-tours du-path $[1,2,4,5]$

$X = 42$

$Z = X \times 2$

$Z = X - 8$

du-path sets
- $du(1, X) = \{[1,2,4,5], [1,3,4,5], [1,2,4,6], [1,3,4,6]\}$

du-pair sets
- $du(1, 5, X) = \{[1,2,4,5], [1,3,4,5]\}$
- $du(1, 6, X) = \{[1,2,4,6], [1,3,4,6]\}$
Data Flow Coverage Criteria

- **All-Defs Coverage (ADC)**
  - Use every def

- **All-Uses Coverage (AUC)**
  - Get to every use

- **All-du-Paths Coverage (ADUPC)**
  - Follow all du-paths
All-Defs Coverage (ADC)

- For each set of du-paths \( S = du(n,v) \), \( TR \) contains at least one path \( d \) in \( S \)
  - For each def, at least one use must be reached

du-path sets

- \( du(1, X) = \{ [1,2,4,5], [1,3,4,5], [1,2,4,6], [1,3,4,6] \} \)

\[ Z = X \times 2 \]

\[ Z = X - 8 \]

\[ X = 42 \]

\[ X = 42 \]

\[ 2 \rightarrow 4 \rightarrow 7 \]

\[ 3 \rightarrow 6 \]

\[ 1 \rightarrow 4 \rightarrow 5 \]

TR for \( X = \{ [1,2,4,5] \} \)

Test paths = \{[1,2,4,5,7]\}
All-Uses Coverage (AUC)

- For each set of du-paths $S = \text{du}(n_i, n_j, v)$, TR contains at least one path $d$ in $S$
  - For each def, all uses must be reached

```
X = 42
Z = X*2
Z = X-8
```

**du-pair sets**
- $\text{du}(1, 5, X) = \{[1,2,4,5], [1,3,4,5]\}$
- $\text{du}(1, 6, X) = \{[1,2,4,6], [1,3,4,6]\}$

TR for $X = \{[1,2,4,5], [1,2,4,6]\}$

Test paths = $\{[1,2,4,5,7], [1,2,4,6,7]\}$
For each set of du-paths $S = du(n_i, n_j, v)$, TR contains every path $d$ in $S$.

For each def-use pair, all paths between defs and uses must be covered.

\[ X = 42 \]
\[ Z = X - 8 \]
\[ Z = X \times 2 \]

**du-pair sets**

- $du(1, 5, X) = \{[1, 2, 4, 5], [1, 3, 4, 5]\}$
- $du(1, 6, X) = \{[1, 2, 4, 6], [1, 3, 4, 6]\}$

TR for $X = \{[1, 2, 4, 5], [1, 3, 4, 5], [1, 2, 4, 6], [1, 3, 4, 6]\}$

Test paths = \{[1, 2, 4, 5, 7], [1, 3, 4, 5, 7], [1, 2, 4, 6, 7], [1, 3, 4, 6, 7]\}
subject, pattern are forwarded parameters

subject.charAt(iSub) == pattern.charAt(0)

subject.charAt(iSub + patternLen - 1) < subjectLen && isPat == false

(iSub + patternLen - 1) >= subjectLen || isPat != False

subject.charAt(iSub) != pattern.charAt(0)

break

iSub++

iPat++

rtnIndex = NOTFOUND isPat = false;

rtnIndex = iSub isPat = true iPat = 1

iPat >= patternLen

subject.charAt(iSub + iPat) != pattern.charAt(iPat)

subject.charAt(iSub + iPat) <= pattern.charAt(iPat)
def(1) = {subject, pattern}

def(2) = {NOTFOUND, iSub, rtnIndex, isPat, subjectLen, patternLen}
use(2) = {subject, pattern, NOTFOUND}

use(3, 11) = use(3, 4) = {iSub, patternLen, subjectLen, isPat}

use(4, 10) = use(4, 5) = {subject, iSub, pattern}

def(5) = {rtnIndex, isPat, iPat}
use(5) = {iSub}

def(8) = {rtnIndex, isPat}
use(8) = {NOTFOUND}

use(6, 10) = use(6, 7) = {iPat, patternLen}

use(7, 8) = use(7, 9) = {subject, pattern, iSub, iPat}

def(9) = {iPat}
use(9) = {iPat}

def(10) = {iSub}
use(10) = {iSub}
break

use(11) = {rtnIndex}
Example: DU-Paths

Consider iSub, categorize def-path sets and def-pair sets

**def-path sets**

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

**def-pair sets**

\[
\text{du}(10, 4, \text{iSub}) = \{[10,3,4]\}
\]
\[
\text{du}(10, 5, \text{iSub}) = \{[10,3,4,5]\}
\]
\[
\text{du}(10, 8, \text{iSub}) = \{[10,3,4,5,6,7,8]\}
\]
\[
\text{du}(10, 9, \text{iSub}) = \{[10,3,4,5,6,7,9]\}
\]
\[
\text{du}(10, 10, \text{iSub}) = \{[10,3,4,5,6,10], [10,3,4,5,6,7,8,10], [10,3,4,10]\}
\]
\[
\text{du}(10, 11, \text{iSub}) = \{[10,3,11]\}
\]
Def for subject

Node 1

Use for subject

Node 2

Edges (4,10), (4,5), (7,8), (7,9)

ADC for subject

TR = {[1,2]}
Test path = {[1,2,3,11]}

Def for pattern

Node 1

Use for pattern

Node 2

Edges (4,10), (4,5), (7,8), (7,9)

ADC for pattern

TR = {[1,2]}
Test path = {[1,2,3,11]}
<table>
<thead>
<tr>
<th>Variable</th>
<th>Test paths (All-Defs Coverage)</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject</td>
<td>[1,2,3,11]</td>
</tr>
<tr>
<td>pattern</td>
<td>[1,2,3,11]</td>
</tr>
<tr>
<td>NOTFOUND</td>
<td>[1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td>iSub</td>
<td>[1,2,3,4,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,10,3,4,10,3,11]</td>
</tr>
<tr>
<td>rtnIndex</td>
<td>[1,2,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td>ispat</td>
<td>No path or No path needed</td>
</tr>
<tr>
<td>subjectLen</td>
<td>[1,2,3,4,10,3,11]</td>
</tr>
<tr>
<td>patternLen</td>
<td>[1,2,3,4,10,3,11]</td>
</tr>
<tr>
<td>iPat</td>
<td>[1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,7,9,6,7,8,10,3,11]</td>
</tr>
<tr>
<td>isPat</td>
<td>No path or No path needed</td>
</tr>
<tr>
<td>Variable</td>
<td>Test paths (All-Uses Coverage)</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>subject</td>
<td>[1,2,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,7,9,6,10,3,11]</td>
</tr>
<tr>
<td>pattern</td>
<td>[1,2,3,4,5,6,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,7,9,6,10,3,11]</td>
</tr>
<tr>
<td>NOTFOUND</td>
<td>[1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td>iSub</td>
<td>[1,2,3,4,5,6,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,10,3,4,5,6,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,10,3,4,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,7,9,6,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,10,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,10,3,4,5,6,7,9,6,10,3,11]</td>
</tr>
<tr>
<td>rtnIndex</td>
<td>[1,2,3,4,5,6,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td>isPat</td>
<td>No path or No path needed</td>
</tr>
<tr>
<td>subjectLen</td>
<td>[1,2,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,10,3,11]</td>
</tr>
<tr>
<td>patternLen</td>
<td>[1,2,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,10,3,11]</td>
</tr>
<tr>
<td></td>
<td>[1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td>isPat</td>
<td>No path or No path needed</td>
</tr>
<tr>
<td>isPat</td>
<td>No path or No path needed</td>
</tr>
<tr>
<td>Variable</td>
<td>Test paths (All-DU-Paths Coverage)</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>subject</td>
<td>[1,2,3,11], [1,2,3,4,10,3,11], [1,2,3,4,5,6,10,3,11], [1,2,3,4,5,6,7,9,6,10,3,11], [1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td>pattern</td>
<td>[1,2,3,11], [1,2,3,4,10,3,11], [1,2,3,4,5,6,7,8,10,3,11], [1,2,3,4,5,6,7,9,6,10,3,11]</td>
</tr>
<tr>
<td>NOTFOUND</td>
<td>[1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td>iSub</td>
<td>[1,2,3,11], [1,2,3,4,10,3,11], [1,2,3,4,5,6,10,3,11], [1,2,3,4,5,6,7,8,10,3,11], [1,2,3,4,10,3,4,10,3,11], [1,2,3,4,10,3,4,5,6,10,3,11], [1,2,3,4,5,6,7,8,10,3,11], [1,2,3,4,5,6,7,9,6,10,3,11]</td>
</tr>
<tr>
<td>rtnIndex</td>
<td>[1,2,3,11], [1,2,3,4,5,6,10,3,11], [1,2,3,4,5,6,7,8,10,3,11]</td>
</tr>
<tr>
<td>isPat</td>
<td>No path or No path needed</td>
</tr>
<tr>
<td>subjectLen</td>
<td>[1,2,3,4,10,3,11], [1,2,3,11]</td>
</tr>
<tr>
<td>patternLen</td>
<td>[1,2,3,11], [1,2,3,4,5,6,7,8,10,3,11], [1,2,3,4,5,6,10,3,11]</td>
</tr>
<tr>
<td>iPat</td>
<td>[1,2,3,4,5,6,7,8,10,3,11], [1,2,3,4,5,6,10,3,11]</td>
</tr>
<tr>
<td>isPat</td>
<td>No path or No path needed</td>
</tr>
</tbody>
</table>
Summary

- Graphs are very powerful abstraction for designing tests
- Graphs appear in many situations in software
- Each criterion has its own cost/benefit tradeoffs
  - No silver bullet
  - When possible, choose the criterion that yields the smallest number of test requirements while maintaining fault detection capability
Graph Coverage Criteria Subsumption

- Complete Path Coverage (CPC)
- Prime Path Coverage (PPC)
- Complete Round Trip Coverage (CRTC)
- Simple Round Trip Coverage (SRTC)
- Edge-Pair Coverage (EPC)
- Edge Coverage (EC)
- Node Coverage (NC)
- All-DU-Paths Coverage (ADUP)
- All-uses Coverage (AUC)
- All-defs Coverage (ADC)