Lecture 3 – Coverage Criteria

AAA705: Software Testing and Quality Assurance

Jihyeok Park

PLRG

2024 Spring

Recall

- Random Testing (RT)
  - Probabilistic Analysis
  - Weaknesses of Random Testing
  - Examples
- Adaptive Random Testing (ART)
  - Levenshtein (Edit) Distance
  - Distance Comparison Target
  - Complexity of ART
  - Quasi-Random Strategy for ART
- Fuzz Testing
  - Pre-process
  - Input Generation – Mutation-Based Fuzzing
  - Input Generation – Generation-Based Fuzzing
  - Test Oracles (Sanitizers)
  - De-duplication

Contents

Contents

Graph Coverage

- **Graphs** are the most **commonly** used structure in software.
  - Control Flow Graphs (CFGs)
  - Call Graphs
  - Design Structure
  - Finite State Machines (FSMs)
  - etc.

Graph Coverage

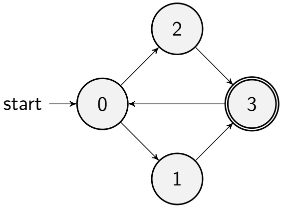
- **Graphs** are the most **commonly** used structure in software.
  - Control Flow Graphs (CFGs)
  - Call Graphs
  - Design Structure
  - Finite State Machines (FSMs)
  - etc.
- We want to ensure that our **tests** properly **cover** the graph.

Graphs

Definition (Graph)

A **graph** $G = (N, E, N_s, N_f)$ is a quadruple consisting of

1. a set of **nodes** N,
2. a set of **edges** $E \subseteq N \times N$,
3. a set of **start nodes** $N_s \subseteq N$, and
4. a set of **final nodes** $N_f \subseteq N$.



$$G = \begin{cases} N = \{0, 1, 2, 3\} \\ E = \{(0, 1), (0, 2), (1, 3), (2, 3), (3, 0)\} \\ N_s = \{0\} \\ N_f = \{3\} \end{cases}$$

Paths

- A **path** $p = (n_0, n_1, \ldots, n_k) \in N^*$ in a graph $G = (N, E, N_s, N_F)$ is a sequence of nodes such that $(n_i, n_{i+1}) \in E$ for $0 \leq i < k$.

$$P_G = \{p \in N^* \mid p \text{ is a path in } G\}$$

Paths

- A **path** $p = (n_0, n_1, \ldots, n_k) \in N^*$ in a graph $G = (N, E, N_s, N_F)$ is a sequence of nodes such that $(n_i, n_{i+1}) \in E$ for $0 \leq i < k$.

$$P_G = \{p \in N^* \mid p \text{ is a path in } G\}$$

- The **length** of a path is the **number of edges** in the path.

$$|p| = k$$

Paths

- A **path** $p = (n_0, n_1, \ldots, n_k) \in N^*$ in a graph $G = (N, E, N_s, N_F)$ is a sequence of nodes such that $(n_i, n_{i+1}) \in E$ for $0 \leq i < k$.

$$P_G = \{p \in N^* \mid p \text{ is a path in } G\}$$

- The **length** of a path is the **number of edges** in the path.

$$|p| = k$$

- A **subpath** q of a path p is a **subsequence** of p (i.e., $q \preceq p$).

$$q \preceq p \quad \iff \quad \exists 0 \leq i \leq j \leq k.\ q = (n_i, n_{i+1}, \ldots, n_j)$$

Paths

- A **path** $p = (n_0, n_1, \ldots, n_k) \in N^*$ in a graph $G = (N, E, N_s, N_F)$ is a sequence of nodes such that $(n_i, n_{i+1}) \in E$ for $0 \leq i < k$.

$$P_G = \{p \in N^* \mid p \text{ is a path in } G\}$$

- The **length** of a path is the **number of edges** in the path.

$$|p| = k$$

- A **subpath** q of a path p is a **subsequence** of p (i.e., $q \preceq p$).

$$q \preceq p \quad \iff \quad \exists 0 \leq i \leq j \leq k.\ q = (n_i, n_{i+1}, \ldots, n_j)$$

- A path p is a **test path** is if it starts from the **start node** and ends at a **final node**, and it represents an **execution** of a **test case**.

$$n_0 \in S \land n_k \in F$$

Paths

- A test path p **visits** a node n if it is in the path.

$$\exists 0 \leq i \leq k.\ n_i = n$$

Paths

- A test path p **visits** a node n if it is in the path.

$$\exists 0 \leq i \leq k.\ n_i = n$$

- A test path p **visits** an edge (n, m) if it is in the path.

$$\exists 0 \leq i \leq k.\ n_i = n \wedge n_{i+1} = m$$

Paths

- A test path p **visits** a node n if it is in the path.

$$\exists 0 \leq i \leq k.\ n_i = n$$

- A test path p **visits** an edge (n, m) if it is in the path.

$$\exists 0 \leq i \leq k.\ n_i = n \land n_{i+1} = m$$

- The $\mathbf{path}(t)$ is the test path executed by a test case t.

Paths

- A test path p **visits** a node n if it is in the path.

$$\exists 0 \leq i \leq k.\ n_i = n$$

- A test path p **visits** an edge (n, m) if it is in the path.

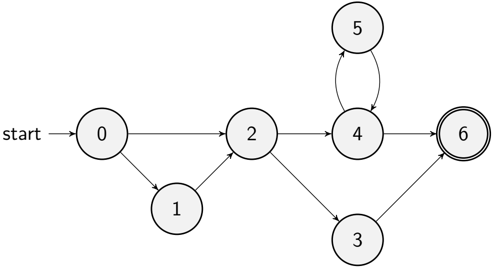
$$\exists 0 \leq i \leq k.\ n_i = n \land n_{i+1} = m$$

- The **path(t)** is the test path executed by a test case t.
- The **path(T)** is the set of test paths executed by a test suite T.

Paths – Example

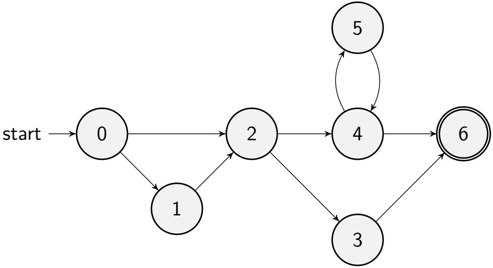
Consider a path $p = [0, 2, 4, 6]$ in the graph G.

$G \quad =$



Paths – Example

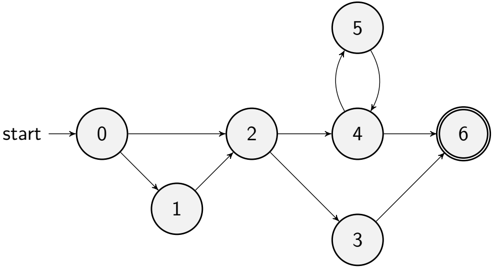
Consider a path $p = [0, 2, 4, 6]$ in the graph G.

G = 

- $|p| = 3$

Paths – Example

Consider a path $p = [0, 2, 4, 6]$ in the graph G.

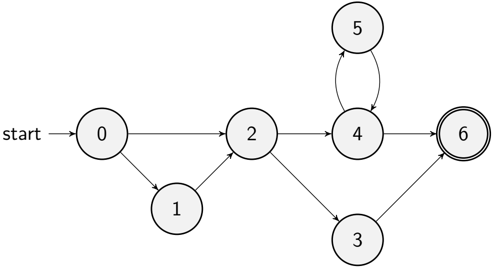
$G =$ 

- $|p| = 3$
- $[0, 2, 6]$ is a **subpath** of p (i.e., $[0, 2, 6] \preceq p$)

Paths – Example

Consider a path $p = [0, 2, 4, 6]$ in the graph G.

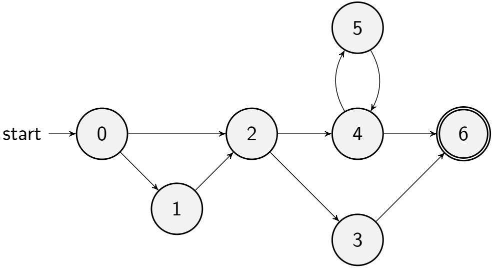
$G =$



- $|p| = 3$
- $[0, 2, 6]$ is a **subpath** of p (i.e., $[0, 2, 6] \preceq p$)
- p is a **test path**

Paths – Example

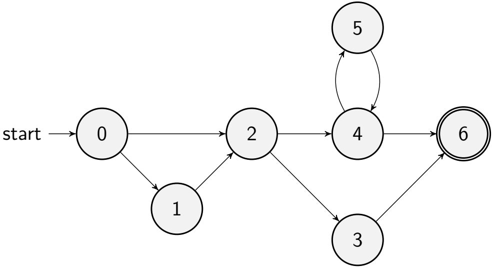
Consider a path $p = [0, 2, 4, 6]$ in the graph G.

$G =$ 

- $|p| = 3$
- $[0, 2, 6]$ is a **subpath** of p (i.e., $[0, 2, 6] \preceq p$)
- p is a **test path**
- p **visits** nodes 0, 2, 4, and 6

Paths – Example

Consider a path $p = [0, 2, 4, 6]$ in the graph G.

G = 

- $|p| = 3$
- $[0, 2, 6]$ is a **subpath** of p (i.e., $[0, 2, 6] \preceq p$)
- p is a **test path**
- p **visits** nodes 0, 2, 4, and 6
- p **visits** edges $(0, 2)$, $(2, 4)$, and $(4, 6)$

Graph Coverage Criterion

Definition (Graph Coverage Criterion)

A **graph coverage criterion** $C_G = (R_G, \sim_G)$ for a given graph G is defined with:

- a set of **test requirements (TRs)** R_G, and
- a **cover relation** $\sim \subseteq P_G \times R_G$ between paths and test requirements.

- A **test case** t **covers** a TR r if its test path satisfies the TR.

$$t \sim r \quad \iff \quad \text{path}(t) \sim r$$

Graph Coverage Criterion

Definition (Graph Coverage Criterion)

A **graph coverage criterion** $C_G = (R_G, \sim_G)$ for a given graph G is defined with:

- a set of **test requirements (TRs)** R_G, and
- a **cover relation** $\sim \subseteq P_G \times R_G$ between paths and test requirements.

- A **test case** t **covers** a TR r if its test path satisfies the TR.

$$t \sim r \quad \iff \quad \text{path}(t) \sim r$$

- A **test suite** T **satisfies** the coverage criterion C_G if it covers all TRs.

$$T \vdash C_G \quad \iff \quad \forall r \in R_G.\ \exists t \in T.\ t \sim r$$

Graph Coverage Criterion

Definition (Graph Coverage Criterion)

A **graph coverage criterion** $C_G = (R_G, \sim_G)$ for a given graph G is defined with:

- a set of **test requirements (TRs)** R_G, and
- a **cover relation** $\sim \subseteq P_G \times R_G$ between paths and test requirements.

- A **test case** t **covers** a TR r if its test path satisfies the TR.

$$t \sim r \quad \iff \quad \mathrm{path}(t) \sim r$$

- A **test suite** T **satisfies** the coverage criterion C_G if it covers all TRs.

$$T \vdash C_G \quad \iff \quad \forall r \in R_G.\ \exists t \in T.\ t \sim r$$

- A **structural coverage criterion** is defined on a graph in terms of **nodes**, **edges**, and **paths**.

Graph Coverage Criterion

Definition (Graph Coverage Criterion)

A **graph coverage criterion** $C_G = (R_G, \sim_G)$ for a given graph G is defined with:

- a set of **test requirements (TRs)** R_G, and
- a **cover relation** $\sim \subseteq P_G \times R_G$ between paths and test requirements.

- A **test case** t **covers** a TR r if its test path satisfies the TR.

$$t \sim r \quad \iff \quad \text{path}(t) \sim r$$

- A **test suite** T **satisfies** the coverage criterion C_G if it covers all TRs.

$$T \vdash C_G \quad \iff \quad \forall r \in R_G.\ \exists t \in T.\ t \sim r$$

- A **structural coverage criterion** is defined on a graph in terms of **nodes**, **edges**, and **paths**.
- A **data-flow coverage criterion** is defined on a graph annotated with references to **variables**.

Definition (Node Coverage (NC))

The **node coverage** criterion $C_G = (R_G, \sim)$ is defined with:

- the set of **TRs** is a set of nodes $R_G = N$
- a path p **covers** a node n if p visits n

Definition (Node Coverage (NC))

The **node coverage** criterion $C_G = (R_G, \sim)$ is defined with:

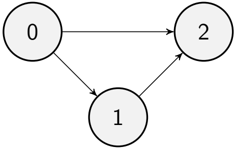
- the set of **TRs** is a set of nodes $R_G = N$
- a path p **covers** a node n if p visits n

Definition (Edge Coverage (EC))

The **edge coverage** criterion $C_G = (R_G, \sim)$ is defined with:

- the set of **TRs** is a set of edges $R_G = E$
- a path p **covers** an edge (n, m) if p visits (n, m)

NC and EC are only different when there is an edge and another subpath between a pair of nodes (e.g., an `if-else` statement).



Structural Coverage – k-Limiting Path Coverage

Definition (k-Limiting Path Coverage (k-PC))

The k-**limiting path coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of paths whose lengths are bounded by k:

$$R_G = \{p \in P_G \mid |p| \leq k\}$$

- a path p **covers** a path q if q is a subpath of p:

$$p \sim q \iff q \preceq p$$

Structural Coverage – k-Limiting Path Coverage

Definition (k-Limiting Path Coverage (k-PC))

The k-**limiting path coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of paths whose lengths are bounded by k:

$$R_G = \{p \in P_G \mid |p| \leq k\}$$

- a path p **covers** a path q if q is a subpath of p:

$$p \sim q \iff q \preceq p$$

Definition (Edge-Pair Coverage (EPC))

The **edge-pair coverage** criterion is 2-limiting path coverage.

Structural Coverage – k-Limiting Path Coverage

Definition (k-Limiting Path Coverage (k-PC))

The k-**limiting path coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of paths whose lengths are bounded by k:

$$R_G = \{p \in P_G \mid |p| \leq k\}$$

- a path p **covers** a path q if q is a subpath of p:

$$p \sim q \iff q \preceq p$$

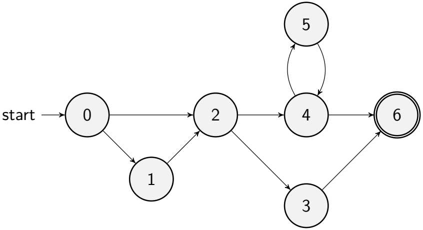
Definition (Edge-Pair Coverage (EPC))

The **edge-pair coverage** criterion is 2-limiting path coverage.

Definition (Complete Path Coverage (CPC))

The **complete path coverage** criterion is ∞-limiting path coverage.

Structural Coverage – Examples

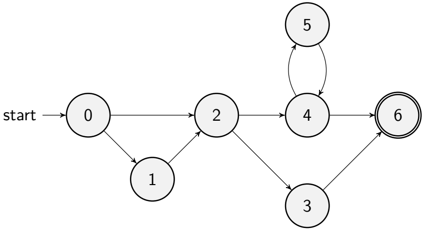
$G =$ 

- Node Coverage (NC)

$$\text{TRs } R_G = \{0, 1, 2, 3, 4, 5, 6\}$$
$$\text{Test Paths} = \{[0, 1, 2, 3, 6], [0, 1, 2, 4, 5, 4, 6]\}$$

Structural Coverage – Examples

$G =$



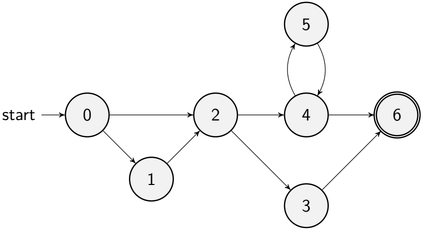
- Node Coverage (NC)

$$\text{TRs } R_G = \{0, 1, 2, 3, 4, 5, 6\}$$
$$\text{Test Paths} = \{[0, 1, 2, 3, 6], [0, 1, 2, 4, 5, 4, 6]\}$$

- Edge Coverage (EC)

$$\text{TRs } R_G = \{\ldots, (0, 1), (0, 2), (1, 2), (2, 3), (2, 4), (3, 6), (4, 5), (4, 6)\}$$
$$\text{Test Paths} = \{[0, 1, 2, 3, 6], [0, 1, 2, 4, 5, 4, 6]\}$$

Structural Coverage – Examples



$G =$

- Edge-Pair Coverage (EPC)

$$
\begin{aligned}
&\text{TRs } R_G = \{ \\
&\quad \ldots, [0,1,2], [0,2,3], [0,2,4], [1,2,3], [1,2,4], [2,3,6], \\
&\quad [2,4,5], [2,4,6], [4,5,4], [5,4,5], [5,4,6] \\
&\} \\
&\text{Test Paths} = \{ \\
&\quad [0,1,2,3,6], [0,1,2,4,6], [0,2,3,6], [0,2,4,5,4,5,4,6] \\
&\}
\end{aligned}
$$

Structural Coverage – Loops in Graphs

- If a graph contains a loop, it has an **infinite** number of paths

Structural Coverage – Loops in Graphs

- If a graph contains a loop, it has an **infinite** number of paths
- In this case, the **complete path coverage** (CPC) is not feasible.

Structural Coverage – Loops in Graphs

- If a graph contains a loop, it has an **infinite** number of paths
- In this case, the **complete path coverage** (CPC) is not feasible.
- One possible solution is to use a **specified path coverage (SPC)** criterion with a set of paths manually specified by the tester.

Structural Coverage – Loops in Graphs

- If a graph contains a loop, it has an **infinite** number of paths
- In this case, the **complete path coverage** (CPC) is not feasible.
- One possible solution is to use a **specified path coverage (SPC)** criterion with a set of paths manually specified by the tester.
- However, it is highly **dependent** on the tester's **expertise**.

Structural Coverage – Loops in Graphs

- If a graph contains a loop, it has an **infinite** number of paths
- In this case, the **complete path coverage** (CPC) is not feasible.
- One possible solution is to use a **specified path coverage (SPC)** criterion with a set of paths manually specified by the tester.
- However, it is highly **dependent** on the tester's **expertise**.
- Attempts to deal with loops:
  - 1970s: Execute cycles once ([4, 5, 4] in the previous example)
  - 1980s: Execute each loop, exactly once
  - 1990s: Execute loops 0 times, once, more than once
  - 2000s: **Prime paths**

Definition (Simple Path)

A **simple path** is a path that does not contain a repeated node, except for the start and final nodes. In other words,

- No internal loops
- A loop is a simple path

Definition (Simple Path)

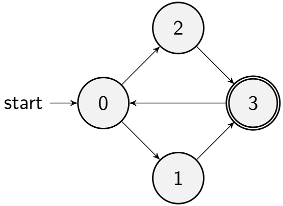
A **simple path** is a path that does not contain a repeated node, except for the start and final nodes. In other words,

- No internal loops
- A loop is a simple path

Definition (Prime Path)

A **prime path** is a simple path that is not a subpath of other simple paths.

Structural Coverage – Simple and Prime Paths



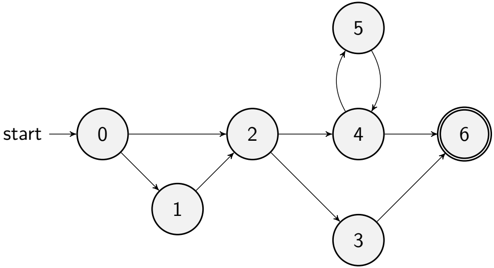
- **23 Simple Paths**:

$$[0], [1], [2], [3], [0, 1], [0, 2], [1, 3], [2, 3], [3, 0],$$
$$[0, 1, 3], [0, 2, 3], [1, 3, 0], [2, 3, 0], [3, 0, 1], [3, 0, 2],$$
$$[0, 1, 3, 0], [0, 2, 3, 0], [1, 3, 0, 1], [1, 3, 0, 2],$$
$$[2, 3, 0, 1], [2, 3, 0, 2], [3, 0, 1, 3], [3, 0, 2, 3]$$

- **8 Prime Paths**:

$$[0, 1, 3, 0], [0, 2, 3, 0], [1, 3, 0, 1], [1, 3, 0, 2],$$
$$[2, 3, 0, 1], [2, 3, 0, 2], [3, 0, 1, 3], [3, 0, 2, 3]$$

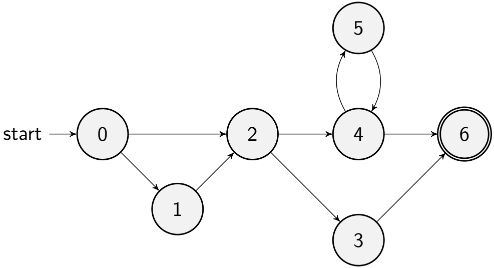
Structural Coverage – Simple and Prime Paths



- **38 Simple Paths**
- **9 Prime Paths**:

$[0, 1, 2, 3, 6]$	$[5, 4, 6]$
$[0, 1, 2, 4, 5]$	$[4, 5, 4]$
$[0, 1, 2, 4, 6]$	$[5, 4, 5]$
$[0, 2, 3, 6]$	
$[0, 2, 4, 5]$	
$[0, 2, 4, 6]$	

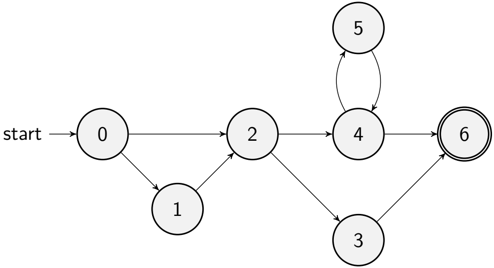
Structural Coverage – Simple and Prime Paths



- **38 Simple Paths**
- **9 Prime Paths**:

$[0, 1, 2, 3, 6]$	$[5, 4, 6]$	$[0, 2, 4, 6]$ executes the loop **0 times**
$[0, 1, 2, 4, 5]$	$[4, 5, 4]$	
$[0, 1, 2, 4, 6]$	$[5, 4, 5]$	
$[0, 2, 3, 6]$		
$[0, 2, 4, 5]$		
$[0, 2, 4, 6]$		

Structural Coverage – Simple and Prime Paths

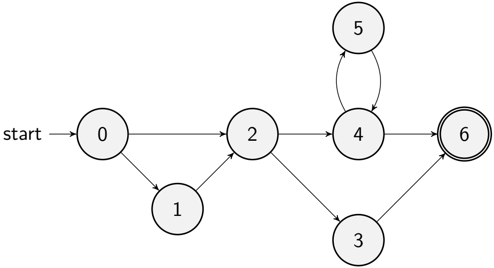


- **38 Simple Paths**
- **9 Prime Paths**:

$[0, 1, 2, 3, 6]$ $[5, 4, 6]$
$[0, 1, 2, 4, 5]$ $[4, 5, 4]$
$[0, 1, 2, 4, 6]$ $[5, 4, 5]$
$[0, 2, 3, 6]$
$[0, 2, 4, 5]$
$[0, 2, 4, 6]$

$[0, 2, 4, 6]$ executes the loop **0 times**

$[4, 5, 4]$ executes the loop **once**



- **38 Simple Paths**
- **9 Prime Paths**:

$[0, 1, 2, 3, 6]$	$[5, 4, 6]$
$[0, 1, 2, 4, 5]$	$[4, 5, 4]$
$[0, 1, 2, 4, 6]$	$[5, 4, 5]$
$[0, 2, 3, 6]$	
$[0, 2, 4, 5]$	
$[0, 2, 4, 6]$	

$[0, 2, 4, 6]$ executes the loop **0 times**

$[4, 5, 4]$ executes the loop **once**

$[5, 4, 5]$ executes the loop **more than once**

Structural Coverage – Prime Path Coverage

Definition (Prime Path Coverage (PPC))

The **prime path coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of prime paths:

$$R_G = \{p \in P_G \mid p \text{ is a prime path}\}$$

- a path p **covers** a prime path q if q is a subpath of p:

$$p \sim q \iff q \preceq p$$

Structural Coverage – Round Trip Paths

Definition (Round-Trip Path)

A **round-trip path** is a prime path that starts and ends at the same node.

Structural Coverage – Round Trip Paths

Definition (Round-Trip Path)

A **round-trip path** is a prime path that starts and ends at the same node.

Definition (Complete Round-Trip Path Coverage (CRPC))

The **complete round-trip path coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of all round-trip paths:

$$R_G = \{p \in P_G \mid p \text{ is a round-trip path}\}$$

- a path p **covers** a round-trip path q if q is a subpath of p:

$$p \sim q \iff q \preceq p$$

Structural Coverage – Round Trip Paths

Definition (Simple Round-Trip Path Coverage (SRPC))

The **simple round-trip path coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of nodes visited by at least one round-trip path:

$$R_G = \{n \in N \mid \exists p \in P_G.\ p \text{ is a round-trip path} \land n \in p\}$$

- a path p **covers** a node n if at least one round-trip path for n is a subpath of p:

$$p \sim n \iff \exists q \in P_G.\ q \text{ is a round-trip path} \land n \in q \land q \preceq p$$

Structural Coverage – Round Trip Paths

Definition (Simple Round-Trip Path Coverage (SRPC))

The **simple round-trip path coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of nodes visited by at least one round-trip path:

$$R_G = \{n \in N \mid \exists p \in P_G.\ p \text{ is a round-trip path} \land n \in p\}$$

- a path p **covers** a node n if at least one round-trip path for n is a subpath of p:

$$p \sim n \iff \exists q \in P_G.\ q \text{ is a round-trip path} \land n \in q \land q \preceq p$$

- CRPC and SRPC omit nodes and edges not in round-trip paths
- In other words, they only focus on loops

Structural Coverage – Touring

Prime paths do not have **internal loops**!

Definition (Tour)

A test pest p **tours** a path q if q is a subpath of p

Structural Coverage – Touring

Prime paths do not have **internal loops**!

Definition (Tour)

A test pest p **tours** a path q if q is a subpath of p

Definition (Tour with Sidetrips)

A test pest p **tours** a path q with **sidetrips** if and only if every **edge** in q is also in p in **the same order**.

Structural Coverage – Touring

Prime paths do not have **internal loops**!

Definition (Tour)

A test pest p **tours** a path q if q is a subpath of p

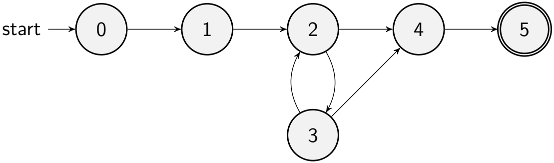
Definition (Tour with Sidetrips)

A test pest p **tours** a path q with **sidetrips** if and only if every **edge** in q is also in p in **the same order**.

Definition (Tour with Detours)

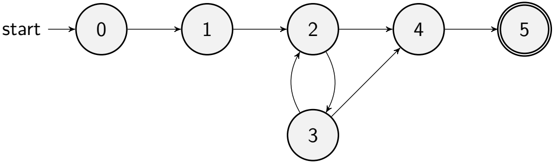
A test pest p **tours** a path q with **detours** if and only if every **node** in q is also in p in **the same order**.

Structural Coverage – Touring



$[0, 1, 2, 4, 5]$ **tours** $[1, 2, 4]$

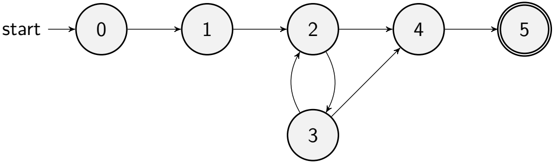
Structural Coverage – Touring



$[0, 1, 2, 4, 5]$ **tours** $[1, 2, 4]$

$[0, 1, 2, 3, 2, 4, 5]$ does **not tour** $[1, 2, 4]$ but **tours** it with **sidetrips** ([2, 3, 2] is a sidetrip)

Structural Coverage – Touring



$[0, 1, 2, 4, 5]$ **tours** $[1, 2, 4]$

$[0, 1, 2, 3, 2, 4, 5]$ does **not tour** $[1, 2, 4]$ but **tours** it with **sidetrips** ([2, 3, 2] is a sidetrip)

$[0, 1, 2, 3, 4, 5]$ does **not tour** $[1, 2, 4]$ with sidetrips but **tours** it with **detours** ([2, 3, 4] is a detour)

Structural Coverage – Infeasibility

- An **infeasible** test requirement **cannot be satisfied**
  - Unreachable statements (dead code)
  - A subpath that can only be executed with a contradiction

Structural Coverage – Infeasibility

- An **infeasible** test requirement **cannot be satisfied**
  - Unreachable statements (dead code)
  - A subpath that can only be executed with a contradiction
- Most **coverage criteria** have some infeasible test requirements

Structural Coverage – Infeasibility

- An **infeasible** test requirement **cannot be satisfied**
  - Unreachable statements (dead code)
  - A subpath that can only be executed with a contradiction
- Most **coverage criteria** have some infeasible test requirements
- It is usually **undecidable** whether all test requirements are feasible

Structural Coverage – Infeasibility

- An **infeasible** test requirement **cannot be satisfied**
  - Unreachable statements (dead code)
  - A subpath that can only be executed with a contradiction
- Most **coverage criteria** have some infeasible test requirements
- It is usually **undecidable** whether all test requirements are feasible
- When **sidetrips** or **detours** are not allowed, many structural coverage criteria have **more infeasible test requirements**

Structural Coverage – Infeasibility

PLRG

- An **infeasible** test requirement **cannot be satisfied**
  - Unreachable statements (dead code)
  - A subpath that can only be executed with a contradiction
- Most **coverage criteria** have some infeasible test requirements
- It is usually **undecidable** whether all test requirements are feasible
- When **sidetrips** or **detours** are not allowed, many structural coverage criteria have **more infeasible test requirements**
- **Practical solutions**: (1) try to satisfy as many test requirements as possible **without** sidetrips or detours, (2) **allow** sidetrips or detours to try to satisfy not yet satisfied test requirements

Data-Flow Coverage

Our goal it try to **ensure** that values are computed and **used** correctly.

Data-Flow Coverage

Our goal it try to **ensure** that values are computed and **used** correctly.

Definition (Def)

A **definition** of a variable is a **location** in the program where a value is assigned to the variable.

Data-Flow Coverage

Our goal it try to **ensure** that values are computed and **used** correctly.

Definition (Def)

A **definition** of a variable is a **location** in the program where a value is assigned to the variable.

Definition (Use)

A **use** of a variable is a **location** in the program where the value of the variable is accessed.

Data-Flow Coverage

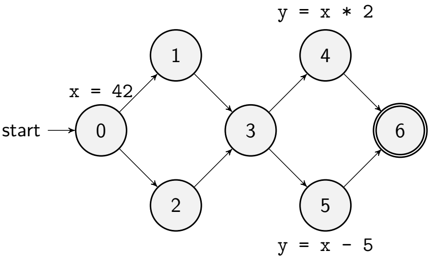
Our goal it try to **ensure** that values are computed and **used** correctly.

Definition (Def)

A **definition** of a variable is a **location** in the program where a value is assigned to the variable.

Definition (Use)

A **use** of a variable is a **location** in the program where the value of the variable is accessed.



$$def(0) = \{x\}$$
$$def(4) = \{y\}$$
$$def(5) = \{y\}$$
$$use(4) = \{x\}$$
$$use(5) = \{x\}$$

DU-Pairs and DU-Paths

Definition (DU-Pair)

A **du-pair** is a pair of a locations (l, l') such that a variable x is defined at l and used at l'.

DU-Pairs and DU-Paths

Definition (DU-Pair)

A **du-pair** is a pair of a locations (l, l') such that a variable x is defined at l and used at l'.

Definition (Def-Clear Path)

A path from l to l' is **def-clear** with respect to a variable x if x is not given another value on any of the nodes or edges in the path.

DU-Pairs and DU-Paths

Definition (DU-Pair)

A **du-pair** is a pair of a locations (l, l') such that a variable x is defined at l and used at l'.

Definition (Def-Clear Path)

A path from l to l' is **def-clear** with respect to a variable x if x is not given another value on any of the nodes or edges in the path.

Definition (DU-Path)

A **du-path** is a simple subpath that is def-clear with respect to x from a def of x to a use of x.

- $du(n, n', x)$ is the set of du-paths from n to n' with respect to x
- $du(n, x)$ is the set of du-paths from n to any use of x

Data-Flow Coverage

Definition (DU-Tour)

A test path p **du-tours** a du-path q with respect to x if p tours d and the subpath taken is def-clear with respect to x

Data-Flow Coverage

Definition (DU-Tour)

A test path p **du-tours** a du-path q with respect to x if p tours d and the subpath taken is def-clear with respect to x

Sidetrips or **detours** can be used, just as with previous touring.

Data-Flow Coverage

PLRG

Definition (DU-Tour)

A test path p **du-tours** a du-path q with respect to x if p tours d and the subpath taken is def-clear with respect to x

Sidetrips or **detours** can be used, just as with previous touring.

Definition (All-Defs Coverage (ADC))

The **all-defs coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of pairs of nodes and variables such that

$$R_G = \{(n, x) \mid n \in N \wedge |du(n, x)| > 0\}$$

- a path p **covers** a pair (n, x) if p du-tours a du-path in $du(n, x)$

$$p \sim (n, x) \iff \exists q \in du(n, x).\ p \text{ du-tours } q$$

Data-Flow Coverage

Definition (All-Ues Coverage (AUC))

The **all-uses coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of triples of two nodes and variables such that

$$R_G = \{(n, n', x) \mid n, n' \in N \land |du(n, n', x)| > 0\}$$

- a path p **covers** (n, n', x) if p du-tours a du-path in $du(n, n', x)$

$$p \sim (n, n', x) \iff \exists q \in du(n, n', x).\ p \text{ du-tours } q$$

Data-Flow Coverage

PLRG

Definition (All-Ues Coverage (AUC))

The **all-uses coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of triples of two nodes and variables such that

$$R_G = \{(n, n', x) \mid n, n' \in N \land |du(n, n', x)| > 0\}$$

- a path p **covers** (n, n', x) if p du-tours a du-path in $du(n, n', x)$

$$p \sim (n, n', x) \iff \exists q \in du(n, n', x).\ p \text{ du-tours } q$$

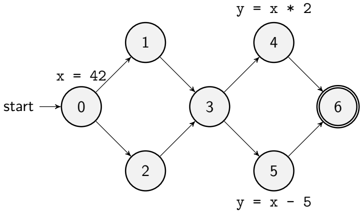
Definition (All-DU-Paths Coverage (ADUPC))

The **all-du-paths coverage** criterion $C_G = (R_G, \sim)$ is

- the set of **TRs** is a set of du-paths: $R_G = \{q \in P_G \mid q \text{ is a du-path}\}$
- a path p **covers** a du-path q if p du-tours q:

$$p \sim q \iff p \text{ du-tours } q$$

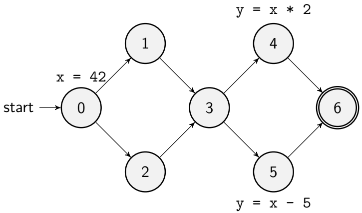
Data-Flow Coverage – Example



- **All-Defs Coverage (ADC)**

$$\text{TRs } R_G = \{(0, x)\}$$
$$\text{Test Paths} = \{[0, 1, 3, 4, 6]\}$$

Data-Flow Coverage – Example



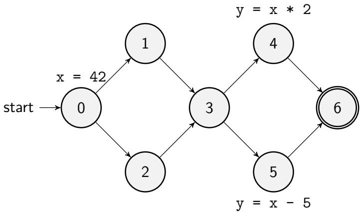
- **All-Defs Coverage (ADC)**

$$\text{TRs } R_G = \{(0, x)\}$$
$$\text{Test Paths} = \{[0, 1, 3, 4, 6]\}$$

- **All-Uses Coverage (AUC)**

$$\text{TRs } R_G = \{(0, 4, x), (0, 5, x)\}$$
$$\text{Test Paths} = \{[0, 1, 3, 4, 6], [0, 1, 3, 5, 6]\}$$

Data-Flow Coverage – Example



- **All-Defs Coverage (ADC)**

$$\text{TRs } R_G = \{(0, x)\}$$
$$\text{Test Paths} = \{[0, 1, 3, 4, 6]\}$$

- **All-Uses Coverage (AUC)**

$$\text{TRs } R_G = \{(0, 4, x), (0, 5, x)\}$$
$$\text{Test Paths} = \{[0, 1, 3, 4, 6], [0, 1, 3, 5, 6]\}$$

- **All-DU-Paths Coverage (ADUPC)**

$$\text{TRs } R_G = \{[0, 1, 3, 4], [0, 1, 3, 5], [0, 2, 3, 4], [0, 2, 3, 5]\}$$
$$\text{Test Paths} = \{[0, 1, 3, 4, 6], [0, 1, 3, 5, 6], [0, 2, 3, 4, 6], [0, 2, 3, 5, 6]\}$$

Subsumption Relationships

Definition (Subsumption)

A coverage criterion $C_G = (R_G, \sim)$ **subsumes** another coverage criterion $C'_G = (R'_G, \sim')$ if and only if any test suite T satisfying C_G satisfies C'_G.

Subsumption Relationships

PLRG

Definition (Subsumption)

A coverage criterion $C_G = (R_G, \sim)$ **subsumes** another coverage criterion $C'_G = (R'_G, \sim')$ if and only if any test suite T satisfying C_G satisfies C'_G.

- **Edge Coverage (EC)** subsumes **Node Coverage (NC)**

Subsumption Relationships

PLRG

Definition (Subsumption)

A coverage criterion $C_G = (R_G, \sim)$ **subsumes** another coverage criterion $C'_G = (R'_G, \sim')$ if and only if any test suite T satisfying C_G satisfies C'_G.

- **Edge Coverage (EC)** subsumes **Node Coverage (NC)**
- **Edge-Pair Coverage (EPC)** subsumes **Edge Coverage (EC)**

Subsumption Relationships

Definition (Subsumption)

A coverage criterion $C_G = (R_G, \sim)$ **subsumes** another coverage criterion $C'_G = (R'_G, \sim')$ if and only if any test suite T satisfying C_G satisfies C'_G.

- **Edge Coverage (EC)** subsumes **Node Coverage (NC)**
- **Edge-Pair Coverage (EPC)** subsumes **Edge Coverage (EC)**
- **Prime Path Coverage (PPC)** subsumes **Edge Coverage (EC)**

Subsumption Relationships

Definition (Subsumption)

A coverage criterion $C_G = (R_G, \sim)$ **subsumes** another coverage criterion $C'_G = (R'_G, \sim')$ if and only if any test suite T satisfying C_G satisfies C'_G.

- **Edge Coverage (EC)** subsumes **Node Coverage (NC)**
- **Edge-Pair Coverage (EPC)** subsumes **Edge Coverage (EC)**
- **Prime Path Coverage (PPC)** subsumes **Edge Coverage (EC)**
- However, **Prime Path Coverage (PPC)** does **not** subsume **Edge-Pair Coverage (EPC)**
    - If a node n has an self-cycle, EPC will require non-prime path $[n, n, m]$

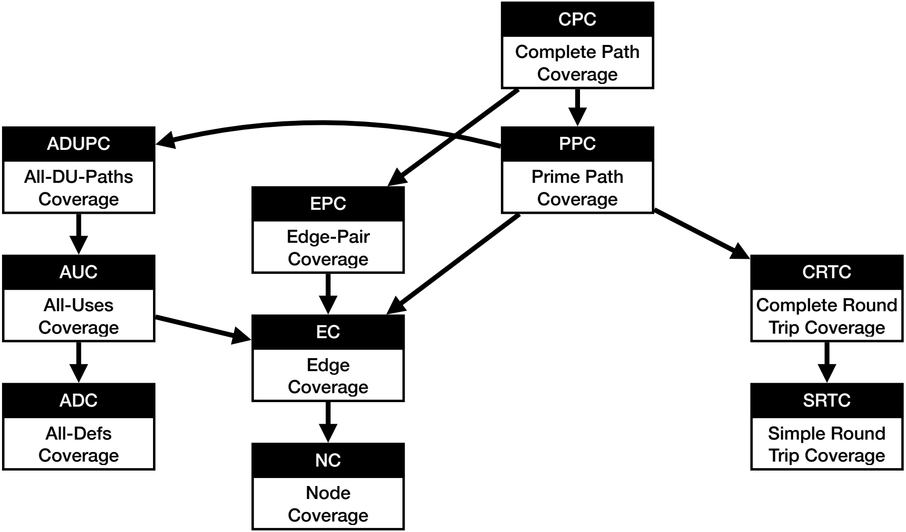
Subsumption Relationships

Definition (Subsumption)

A coverage criterion $C_G = (R_G, \sim)$ **subsumes** another coverage criterion $C'_G = (R'_G, \sim')$ if and only if any test suite T satisfying C_G satisfies C'_G.

- **Edge Coverage (EC)** subsumes **Node Coverage (NC)**
- **Edge-Pair Coverage (EPC)** subsumes **Edge Coverage (EC)**
- **Prime Path Coverage (PPC)** subsumes **Edge Coverage (EC)**
- However, **Prime Path Coverage (PPC)** does **not** subsume **Edge-Pair Coverage (EPC)**
  - If a node n has an self-cycle, EPC will require non-prime path $[n, n, m]$
- **Complete Round-Trip Path Coverage (CRPC)** and **Simple Round-Trip Path Coverage (SRPC)** do not subsume **Node Coverage (NC)**

Subsumption Relationships



Contents

Logic Coverage

- Logic expressions show up in many situations

Logic Coverage

- Logic expressions show up in many situations

- Covering logic expressions is required by US Federal Aviation Administration for safety critical software

Logic Coverage

- Logic expressions show up in many situations
- Covering logic expressions is required by US Federal Aviation Administration for safety critical software
- Logical expressions can come from many sources

Logic Coverage

- Logic expressions show up in many situations

- Covering logic expressions is required by US Federal Aviation Administration for safety critical software

- Logical expressions can come from many sources

- Tests are intended to choose some subset of the total number of truth assignments to the expressions

Logic Predicates and Clauses

- A **predicate** is an expression that evaluates to a **boolean** value
- Predicates can contain
  - **Boolean variables**
  - non-boolean variables with a **comparison** operator
  - Boolean **function** calls

Logic Predicates and Clauses

- A **predicate** is an expression that evaluates to a **boolean** value
- Predicates can contain
  - **Boolean variables**
  - non-boolean variables with a **comparison** operator
  - Boolean **function** calls
- Internal structure is created by logical operators
  - $\neg$ – the **negation** operator
  - $\wedge$ – the **conjunction** operator
  - $\vee$ – the **disjunction** operator
  - $\oplus$ – the **exclusive or** operator
  - $\Rightarrow$ – the **implication** operator
  - $\Leftrightarrow$ – the **equivalence** operator

Logic Predicates and Clauses

- A **predicate** is an expression that evaluates to a **boolean** value
- Predicates can contain
  - **Boolean variables**
  - non-boolean variables with a **comparison** operator
  - Boolean **function** calls
- Internal structure is created by logical operators
  - $\neg$ – the **negation** operator
  - $\wedge$ – the **conjunction** operator
  - $\vee$ – the **disjunction** operator
  - $\oplus$ – the **exclusive or** operator
  - $\Rightarrow$ – the **implication** operator
  - $\Leftrightarrow$ – the **equivalence** operator
- A **clause** is a predicate without logical operators

Examples

$$(a < b) \vee f(z) \wedge D \wedge (m \geq n \times o)$$

- Four clauses:
    - $(a < b)$ – relational expression
    - $f(z)$ – boolean-value function
    - D – boolean variable
    - $(m \geq n \times o)$ – relational expression

Examples

$$(a < b) \vee f(z) \wedge D \wedge (m \geq n \times o)$$

- Four clauses:
  - $(a < b)$ – relational expression
  - $f(z)$ – boolean-value function
  - D – boolean variable
  - $(m \geq n \times o)$ – relational expression
- Most predicates have few clauses

Predicate and Clause Coverage

Abbreviations:

- P is the set of **predicates**
- p is a single **predicate** in P
- C is the set of **clauses**
- C_p is the set of **clauses** in predicate p
- c is a single **clause** in C

Predicate and Clause Coverage

Abbreviations:

- P is the set of **predicates**
- p is a single **predicate** in P
- C is the set of **clauses**
- C_p is the set of **clauses** in predicate p
- c is a single **clause** in C

Definition (Predicate Coverage (PC))

For each predicate $p \in P$, test requirements are the **truth** or **falsity** of p.

It is sometimes called **decision coverage**.

Predicate and Clause Coverage

Abbreviations:

- P is the set of **predicates**
- p is a single **predicate** in P
- C is the set of **clauses**
- C_p is the set of **clauses** in predicate p
- c is a single **clause** in C

Definition (Predicate Coverage (PC))

For each predicate $p \in P$, test requirements are the **truth** or **falsity** of p.

It is sometimes called **decision coverage**.

Definition (Clause Coverage (CC))

For each clause $c \in C$, test requirements are the **truth** or **falsity** of c.

It is sometimes called **condition coverage**.

Predicate and Clause Coverage – Example

$$p = ((a < b) \vee D) \wedge (m \geq n \times o)$$

Predicate and Clause Coverage – Example

$$p = ((a < b) \vee D) \wedge (m \geq n \times o)$$

- **Predicate Coverage (PC)**

a	b	D	m	n	o	p
5	10	T	1	1	1	T
10	5	F	1	1	1	F

Predicate and Clause Coverage – Example

$$p = ((a < b) \lor D) \land (m \geq n \times o)$$

- **Predicate Coverage (PC)**

a	b	D	m	n	o	p
5	10	T	1	1	1	T
10	5	F	1	1	1	F

- **Clause Coverage (CC)**

a	b	D	m	n	o	$(a < b)$	D	$(m \geq n \times o)$
5	10	F	1	1	1	T	F	T
10	5	T	1	2	2	F	T	F

Predicate and Clause Coverage

PLRG

- PC does not fully exercise all the clauses in the predicate, especially in the presence of **short circuit** evaluation

Predicate and Clause Coverage

- PC does not fully exercise all the clauses in the predicate, especially in the presence of **short circuit** evaluation

- CC does not subsume PC. For example, the following test suite satisfies CC but not PC:

$A \vee B$

A	B	$A \vee B$
T	F	T
F	T	T

- The simplest solution is to test **all combinations**

Predicate and Clause Coverage

- PC does not fully exercise all the clauses in the predicate, especially in the presence of **short circuit** evaluation

Predicate and Clause Coverage

- PC does not fully exercise all the clauses in the predicate, especially in the presence of **short circuit** evaluation

- CC does not subsume PC. For example, the following test suite satisfies CC but not PC:

$A \vee B$

A	B	$A \vee B$
T	F	T
F	T	T

Predicate and Clause Coverage

- PC does not fully exercise all the clauses in the predicate, especially in the presence of **short circuit** evaluation

- CC does not subsume PC. For example, the following test suite satisfies CC but not PC:

$A \vee B$

A	B	$A \vee B$
T	F	T
F	T	T

- The simplest solution is to test **all combinations**

Combinatorial Coverage

Definition (Combinatorial Coverage (CoC))

For each predicate $p \in P$, test requirements in **combinatorial coverage (CoC)** are the all **combinations** of the truth or falsity of the clauses in C_p.

Combinatorial Coverage

Definition (Combinatorial Coverage (CoC))

For each predicate $p \in P$, test requirements in **combinatorial coverage (CoC)** are the all **combinations** of the truth or falsity of the clauses in C_p.

For example, we need the following combinations for the predicate p

$$p = ((a < b) \vee D) \wedge (m \geq n \times o)$$

$(a < b)$	D	$(m \geq n \times o)$	p
T	T	T	T
T	T	F	F
T	F	T	T
T	F	F	F
F	T	T	T
F	T	F	F
F	F	T	F
F	F	F	F

Determination

- This is simple, neat, clean, but quite **expensive**!

Determination

- This is simple, neat, clean, but quite **expensive**!

- 2^N tests required for N clauses

Determination

- This is simple, neat, clean, but quite **expensive**!
- 2^N tests required for N clauses
- Let's test each clause **independently** from the other clauses using whether each clause **determines** the entire predicate.

Determination

- This is simple, neat, clean, but quite **expensive**!

- 2^N tests required for N clauses

- Let's test each clause **independently** from the other clauses using whether each clause **determines** the entire predicate.

Definition (Determination)

A clause c in predicate p, called the **major clause**, **determines** p if and only if the values of the remaining **minor clauses** c' are such that changing the value of p.

Determination – Examples

- A (or B) **determines** $A \vee B$ if B (or A) is `false`, and A (or B) **determines** $A \wedge B$ if B (or A) is `true`.

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

Determination – Examples

- A (or B) **determines** $A \vee B$ if B (or A) is `false`, and
  A (or B) **determines** $A \wedge B$ if B (or A) is `true`.

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

- A (or B) always **determines** $A \oplus B$, and
  A (or B) always **determines** $A \Leftrightarrow B$.

A	B	$A \oplus B$
T	T	F
T	F	T
F	T	T
F	F	F

A	B	$A \Leftrightarrow B$
T	T	T
T	F	F
F	T	F
F	F	T

Active Clause Coverage (ACC)

Definition (Active Clause Coverage (ACC))

For each predicate $p \in P$, test requirements in **active clause coverage (ACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of c.

- For example, $p = A \vee B$, and pick A (or B) as the major clause.
  1. $A = \texttt{true}$ and $B = \texttt{false}$ (A determines p)
  2. $A = \texttt{false}$ and $B = \texttt{false}$ (A determines p)
  3. $A = \texttt{false}$ and $B = \texttt{true}$ (B determines p)
  4. ~~$A = \texttt{false}$ and $B = \texttt{false}$ (B determines p)~~

Active Clause Coverage (ACC)

Definition (Active Clause Coverage (ACC))

For each predicate $p \in P$, test requirements in **active clause coverage (ACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of c.

- For example, $p = A \vee B$, and pick A (or B) as the major clause.
  1. $A =$ `true` and $B =$ `false` (A determines p)
  2. $A =$ `false` and $B =$ `false` (A determines p)
  3. $A =$ `false` and $B =$ `true` (B determines p)
  4. ~~$A =$ `false` and $B =$ `false` (B determines p)~~

- The last one is **duplicate** and can be omitted.

Active Clause Coverage (ACC)

Definition (Active Clause Coverage (ACC))

For each predicate $p \in P$, test requirements in **active clause coverage (ACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of c.

- For example, $p = A \vee B$, and pick A (or B) as the major clause.
  1. $A = \texttt{true}$ and $B = \texttt{false}$ (A determines p)
  2. $A = \texttt{false}$ and $B = \texttt{false}$ (A determines p)
  3. $A = \texttt{false}$ and $B = \texttt{true}$ (B determines p)
  4. ~~$A = \texttt{false}$ and $B = \texttt{false}$ (B determines p)~~
- The last one is **duplicate** and can be omitted.
- Another name of ACC is **modified condition/decision coverage (MC/DC)**, which is required by the **US Federal Aviation Administration** for **safety critical software**.

Active Clause Coverage (ACC) – Ambiguity

- **Ambiguity** – Do the **minor clauses** have to have the **same values** when the **major clause** is `true` or `false`?

Active Clause Coverage (ACC) – Ambiguity

- **Ambiguity** – Do the **minor clauses** have to have the **same values** when the **major clause** is `true` or `false`?

- For example, $p = A \lor (B \land C)$, and pick A as the major clause.
  1. $A =$ `true`, $B =$ `false`, $C =$ `true`
  2. $A =$ `false`, $B =$ `false`, $C =$ `false` (is $C =$ `false` allowed?)

Active Clause Coverage (ACC) – Ambiguity

- **Ambiguity** – Do the **minor clauses** have to have the **same values** when the **major clause** is `true` or `false`?

- For example, $p = A \vee (B \wedge C)$, and pick A as the major clause.
  1. $A = $ `true`, $B = $ `false`, $C = $ `true`
  2. $A = $ `false`, $B = $ `false`, $C = $ `false` (is $C = $ `false` allowed?)

- This question caused a **confusion** among testers for years

Active Clause Coverage (ACC) – Ambiguity

- **Ambiguity** – Do the **minor clauses** have to have the **same values** when the **major clause** is `true` or `false`?

- For example, $p = A \vee (B \wedge C)$, and pick A as the major clause.
  1. $A =$ `true`, $B =$ `false`, $C =$ `true`
  2. $A =$ `false`, $B =$ `false`, $C =$ `false` (is $C =$ `false` allowed?)

- This question caused a **confusion** among testers for years

- Consider this carefully leads to three separate coverage criteria:
  - Minor clauses **do not need** to be the same
  - Minor clauses **must** be the same
  - Minor clauses **force the predicate** to have different values

General Active Clause Coverage (GACC)

Definition (General Active Clause Coverage (GACC))

For each predicate $p \in P$, test requirements in **general active clause coverage (GACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** do **not need** to be the same when the **major clause** is `true` or `false`.

General Active Clause Coverage (GACC)

Definition (General Active Clause Coverage (GACC))

For each predicate $p \in P$, test requirements in **general active clause coverage (GACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** do **not need** to be the same when the **major clause** is `true` or `false`.

- Unfortunately, GACC does **not subsume predicate coverage (PC)**.

General Active Clause Coverage (GACC)

Definition (General Active Clause Coverage (GACC))

For each predicate $p \in P$, test requirements in **general active clause coverage (GACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** do **not need** to be the same when the **major clause** is `true` or `false`.

- Unfortunately, GACC does **not subsume predicate coverage (PC)**.
- For example, the following selection satisfies GACC but not PC:

A	B	$A \Leftrightarrow B$
T	F	T
T	F	F
F	T	F
F	F	T

Restricted Active Clause Coverage (RACC)

Definition (Restricted Active Clause Coverage (RACC))

For each predicate $p \in P$, test requirements in **restricted active clause coverage (RACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** **must** be the same when the **major clause** is `true` or `false`.

Restricted Active Clause Coverage (RACC)

Definition (Restricted Active Clause Coverage (RACC))

For each predicate $p \in P$, test requirements in **restricted active clause coverage (RACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** **must** be the same when the **major clause** is `true` or `false`.

- This has been a **common interpretation** by aviation developers

Restricted Active Clause Coverage (RACC)

PLRG

Definition (Restricted Active Clause Coverage (RACC))

For each predicate $p \in P$, test requirements in **restricted active clause coverage (RACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** **must** be the same when the **major clause** is `true` or `false`.

- This has been a **common interpretation** by aviation developers
- RACC often leads to **infeasible test requirements**

Restricted Active Clause Coverage (RACC)

PLRG

Definition (Restricted Active Clause Coverage (RACC))

For each predicate $p \in P$, test requirements in **restricted active clause coverage (RACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** **must** be the same when the **major clause** is `true` or `false`.

- This has been a **common interpretation** by aviation developers
- RACC often leads to **infeasible test requirements**
- There is **no logical reason** for such a strict restriction

Restricted Active Clause Coverage (RACC)

Definition (Restricted Active Clause Coverage (RACC))

For each predicate $p \in P$, test requirements in **restricted active clause coverage (RACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** **must** be the same when the **major clause** is `true` or `false`.

- This has been a **common interpretation** by aviation developers
- RACC often leads to **infeasible test requirements**
- There is **no logical reason** for such a strict restriction
- Our **goal** is just to subsume **predicate coverage (PC)**

Correlated Active Clause Coverage (CACC)

Definition (Correlated Active Clause Coverage (CACC))

For each predicate $p \in P$, test requirements in **correlated active clause coverage (CACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** **force the predicate** to have different values when the **major clause** is `true` or `false`.

Correlated Active Clause Coverage (CACC)

Definition (Correlated Active Clause Coverage (CACC))

For each predicate $p \in P$, test requirements in **correlated active clause coverage (CACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** **force the predicate** to have different values when the **major clause** is `true` or `false`.

- A **more recent** interpretation

Correlated Active Clause Coverage (CACC)

Definition (Correlated Active Clause Coverage (CACC))

For each predicate $p \in P$, test requirements in **correlated active clause coverage (CACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** **force the predicate** to have different values when the **major clause** is `true` or `false`.

- A **more recent** interpretation
- CACC **implicitly** allows minor clauses to have different values

Correlated Active Clause Coverage (CACC)

PLRG

Definition (Correlated Active Clause Coverage (CACC))

For each predicate $p \in P$, test requirements in **correlated active clause coverage (CACC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ determine p and (2) the truth or falsity of p. The values chosen for the **minor clauses** **force the predicate** to have different values when the **major clause** is `true` or `false`.

- A **more recent** interpretation
- CACC **implicitly** allows minor clauses to have different values
- CACC **explicitly** subsumes **predicate coverage (PC)**

RACC vs. CACC

A	B	C	$A \wedge (B \vee C)$
T	T	T	T
F	T	T	F
T	T	F	T
F	T	F	F
T	F	T	T
F	F	T	F

A	B	C	$A \wedge (B \vee C)$
T	T	T	T
T	T	F	T
T	F	T	T
F	T	T	F
F	T	F	F
F	F	T	F

- We pick A as the major clause
- The left table shows that there are only **three combinations** allowed in **RACC**
- The right table shows that there are **nine combinations** allowed in **CACC** by selecting any three cases for each truth value of A

Inactive Clause Coverage (ICC)

- **Inactive clause coverage (ICC)** is the **dual** of **active clause coverage (ACC)**

Inactive Clause Coverage (ICC)

- **Inactive clause coverage (ICC)** is the **dual** of **active clause coverage (ACC)**

- ACC criteria ensure that **major** clauses **determine** the predicate

Inactive Clause Coverage (ICC)

- **Inactive clause coverage (ICC)** is the **dual** of **active clause coverage (ACC)**
- ACC criteria ensure that **major** clauses **determine** the predicate
- ICC criteria ensure that **major** clauses **do not determine** the predicate

Inactive Clause Coverage (ICC)

- **Inactive clause coverage (ICC)** is the **dual** of **active clause coverage (ACC)**
- ACC criteria ensure that **major** clauses **determine** the predicate
- ICC criteria ensure that **major** clauses **do not determine** the predicate

Definition (Inactive Clause Coverage (ICC))

For each predicate $p \in P$, test requirements in **inactive clause coverage (ICC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ **not determine** p and (2) the truth or falsity of c.

General and Restricted ICC

PLRG

Definition (General Inactive Clause Coverage (GICC))

For each predicate $p \in P$, test requirements in **general inactive clause coverage (GICC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ **not determine** p and (2) the truth or falsity of c. The values chosen for the **minor clauses** do **not need** to be the same when the **major clause** is `true` or `false`.

General and Restricted ICC

Definition (General Inactive Clause Coverage (GICC))

For each predicate $p \in P$, test requirements in **general inactive clause coverage (GICC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ **not determine** p and (2) the truth or falsity of c. The values chosen for the **minor clauses** do **not need** to be the same when the **major clause** is `true` or `false`.

Definition (Restricted Inactive Clause Coverage (RICC))

For each predicate $p \in P$, test requirements in **restricted inactive clause coverage (RICC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ **not determine** p and (2) the truth or falsity of c. The values chosen for the **minor clauses** **must** be the same when the **major clause** is `true` or `false`.

General and Restricted ICC

Definition (General Inactive Clause Coverage (GICC))

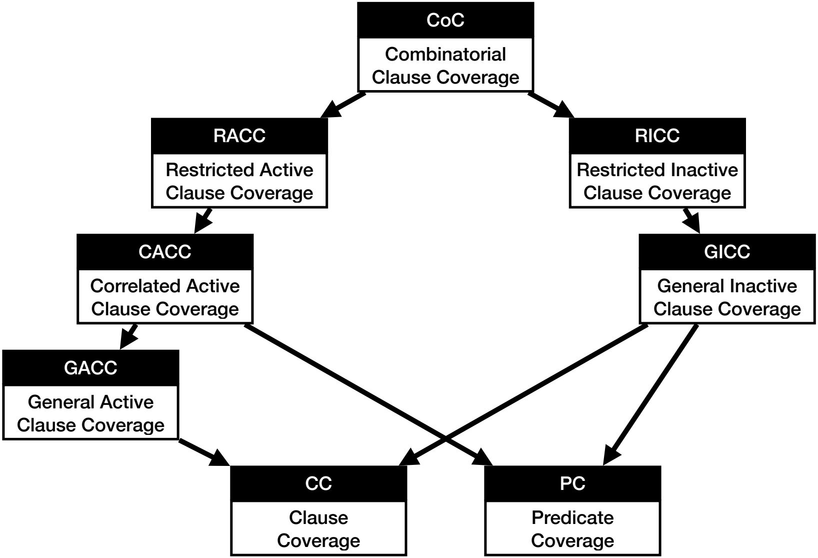
For each predicate $p \in P$, test requirements in **general inactive clause coverage (GICC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ **not determine** p and (2) the truth or falsity of c. The values chosen for the **minor clauses** do **not need** to be the same when the **major clause** is `true` or `false`.

Definition (Restricted Inactive Clause Coverage (RICC))

For each predicate $p \in P$, test requirements in **restricted inactive clause coverage (RICC)** are pairs of (1) **conditions** that make each selected **major clause** $c \in C_p$ **not determine** p and (2) the truth or falsity of c. The values chosen for the **minor clauses** **must** be the same when the **major clause** is `true` or `false`.

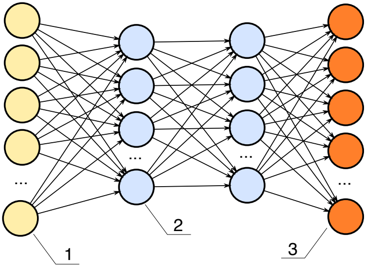
- Unlike ACC, the notion of **correlation** is not relevant to ICC (major clause c does not determine p, so cannot correlate with it)

Subsumption Relationships



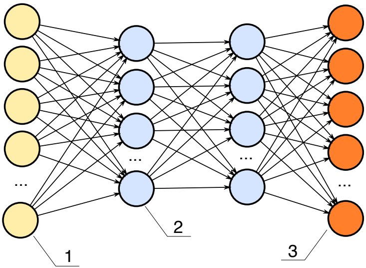
Contents

Deep Neural Network (DNN)



How to define coverage criteria for **deep neural networks (DNN)**?

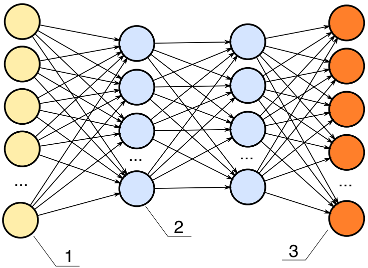
Deep Neural Network (DNN)



How to define coverage criteria for **deep neural networks (DNN)**?

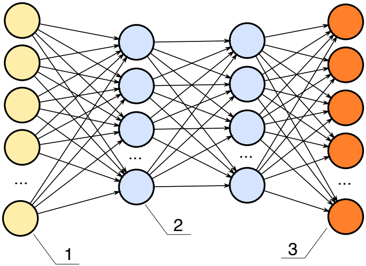
Neuron Coverage!

Deep Neural Network (DNN)



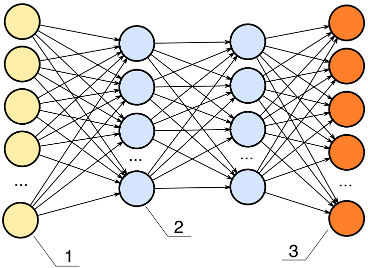
- A **DNN classifier** can be formalized as a function $f : X \to Y$, a mapping form a set of inputs X into a set of labels Y.

Deep Neural Network (DNN)



- A **DNN classifier** can be formalized as a function $f : X \to Y$, a mapping form a set of inputs X into a set of labels Y.
- The output of the DNN classifier is a **probability distribution** $P(Y \mid x)$, which is the probability that an input vector $x \in X$ belongs to each class of labels in Y.

Deep Neural Network (DNN)



- A **DNN classifier** can be formalized as a function $f : X \to Y$, a mapping form a set of inputs X into a set of labels Y.
- The output of the DNN classifier is a **probability distribution** $P(Y \mid x)$, which is the probability that an input vector $x \in X$ belongs to each class of labels in Y.
- A DNN classifier f usually contains an input layer, a number of hidden layers, and an output layer; each layer consists of many **neurons**.

Deep Neural Network (DNN)

- The **parameters** θ of the DNN are the **weights** of each connected edge between neurons of two adjacent layers.

Deep Neural Network (DNN)

- The **parameters** θ of the DNN are the **weights** of each connected edge between neurons of two adjacent layers.
- For an input vector x, the output of DNN $f_\theta(x)$ can be computed as the **weighted sum** of the outputs of all the neurons.

Deep Neural Network (DNN)

- The **parameters** θ of the DNN are the **weights** of each connected edge between neurons of two adjacent layers.
- For an input vector x, the output of DNN $f_\theta(x)$ can be computed as the **weighted sum** of the outputs of all the neurons.
- Given a training dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, the goal is to learn to optimize the following **objective**:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i)$$

where $\mathcal{L}$ is a **loss function** to calculate the penalties for incorrect classifications.

Deep Neural Network (DNN)

- The **parameters** θ of the DNN are the **weights** of each connected edge between neurons of two adjacent layers.
- For an input vector x, the output of DNN $f_\theta(x)$ can be computed as the **weighted sum** of the outputs of all the neurons.
- Given a training dataset $D = \{(x_i, y_i)\}_{i=1}^N$, the goal is to learn to optimize the following **objective**:

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i)$$

  where $\mathcal{L}$ is a **loss function** to calculate the penalties for incorrect classifications.
- For a neuron n in a DNN model f, the **output** of n for an input x is denoted as $f_\theta(n, x)$.

Neuron Coverage (NC)

PLRG

Definition (Activate Neuron)

A neuron is **activated** if the weighted sum of its inputs exceeds a certain threshold t.

$$AC(x, t) = \{n \mid f_\theta(n, x) > t\}$$

Neuron Coverage (NC)

PLRG

Definition (Activate Neuron)

A neuron is **activated** if the weighted sum of its inputs exceeds a certain threshold t.

$$AC(x, t) = \{n \mid f_\theta(n, x) > t\}$$

Definition (Neuron Coverage (NC))

For a DNN model f and a threshold t, the test requirements in **neuron coverage (NC)** are the **activation** of each neuron n in the DNN model f.

$$NC(T, t) = \frac{|\{n \mid \exists x \in T.\ f_\theta(n, x) > t\}|}{|N|}$$

k-Multisection Neuron Coverage (KMNC)

PLRG

For a neuron n, the lower and upper boundary of its output values on **training data** can be denoted as low_n and up_n, respectively.

Definition (k-Multisection Neuron Coverage (KMNC))

For a DNN model f and a number of sections k, the test requirements in k-**multisection neuron coverage (KMNC)** are k **equal sections** of $[low_n, up_n]$ for each neuron n in the DNN model f.

$$KMNC = \frac{\sum_{n \in N} |\{S_m^n \mid \exists x \in T.\ f_\theta(n, x) \in S_m^n\}|}{|N|}$$

where

$$S_m^n = \left[low_n + \frac{m \times (up_n - low_n)}{k}, low_n + \frac{(m+1) \times (up_n - low_n)}{k} \right]$$

Neuron Boundary Coverage (NBC)

For new test inputs T, the output values of neurons may fall into $(-\infty, low_n)$ or $(up_n, +\infty)$. instead of the derived boundary $[low_n, up_n]$.

Definition (Upper or Lower Neuron Coverage (UNC or LNC))

For a DNN model f and a number of sections k, the test requirements in **upper neuron coverage (UNC)** (or **lower neuron coverage (LNC)**) are the **upper boundary** (or **lower boundary**) of the output values of each neuron n in the DNN model f.

$$UNC = \frac{|\{n \mid \exists x \in T.\ f_\theta(n, x) > up_n\}|}{|N|}$$

$$LNC = \frac{|\{n \mid \exists x \in T.\ f_\theta(n, x) < low_n\}|}{|N|}$$

Neuron Boundary Coverage (NBC)

For new test inputs T, the output values of neurons may fall into $(-\infty, low_n)$ or $(up_n, +\infty)$. instead of the derived boundary $[low_n, up_n]$.

Definition (Upper or Lower Neuron Coverage (UNC or LNC))

For a DNN model f and a number of sections k, the test requirements in **upper neuron coverage (UNC)** (or **lower neuron coverage (LNC)**) are the **upper boundary** (or **lower boundary**) of the output values of each neuron n in the DNN model f.

$$UNC = \frac{|\{n \mid \exists x \in T.\ f_\theta(n, x) > up_n\}|}{|N|}$$

$$LNC = \frac{|\{n \mid \exists x \in T.\ f_\theta(n, x) < low_n\}|}{|N|}$$

UNC is often called **strong neuron activation coverage (SNAC)**.
Neuron Boundary Coverage (NBC) is combination of UNC and LNC.

$$NBC = (|UNC| + |LNC|)/2$$

Top-k Neuron Coverage (TKNC)

PLRG

Definition (Top-k Neuron Coverage (TKNC))

TKNC is a layer-level coverage testing criterion that measures the ratio of neurons that have at least been the **most active** k **neurons** of **each layer** on a given test set T once.

$$TKNC = \frac{\left|\bigcup_{x \in T} \bigcup_{1 \leq l \leq L} top_k(x, l)\right|}{|N|}$$

where L denotes the number of layers in the DNN model f, and $top_k(x, l)$ denotes the neurons which have largest k output values in the l-th layer of the DNN model f for the input x.

Top-k Neuron Coverage (TKNC)

Definition (Top-k Neuron Coverage (TKNC))

TKNC is a layer-level coverage testing criterion that measures the ratio of neurons that have at least been the **most active k neurons** of **each layer** on a given test set T once.

$$TKNC = \frac{|\bigcup_{x \in T} \bigcup_{1 \le l \le L} top_k(x, l)|}{|N|}$$

where L denotes the number of layers in the DNN model f, and $top_k(x, l)$ denotes the neurons which have largest k output values in the l-th layer of the DNN model f for the input x.

We can extend this criterion to the **Top-k Neuron Pattern Coverage (TKNPC)** by considering the **combination** of the most active neurons in each layer.

Contents

Feature-Sensitive Coverage

- **[PLDI'23]** J. Park et al. *"Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations."*

- It suggests a new way to refine a given graph coverage criterion using **feature-sensitive** information.

- Slides: **link**

Summary

1. Graph Coverage
   - Structural Coverage
   - Data-Flow Coverage
   - Subsumption Relationships
2. Logic Coverage
   - Simple Logic Expression Coverage
   - Active Clause Coverage
   - Inactive Clause Coverage
   - Subsumption Relationships
3. Neuron Coverage
4. Feature-Sensitive Coverage

Next Lecture

PLRG

- Coverage Criteria (Homework)

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr