White Box Coverage and Control Flow Graphs

Venezia Elodie

June 21, 2011

Contents

List of Figures

Introduction

White-box testing is a verification technique that can be used to examine if code works as expected. More precisely, according to IEEE (1990), the definition of white box testing is *testing that takes into account the internal mechanism of a system or component.* So, white box testing is performed based on knowledge of how the system is implemented. To describe the degree to which the source code of a program has been tested, we use different measures code coverage. In order to measure this code coverage and select tests based on the structure of a graph, we will write programs as graphs. This graph describes the logical structure of the program.
In a program, control flows arise from variables and procedures such as if- and while-constructs in C language. However, it's possible that some parts of the source code of a program are unreachable. This problem is due to the *semantic* of the program. In fact, there is two different approaches in the language which are the semantic which refers to the meaning of the language, and the syntax which as opposed refers to the form of the program. Since it's undecidable to detect the semantic part of a program and since the measure code coverage can be affected by it, we will describe two differents approach for white box coverage: syntactic and semantic approach. The syntactic approach will describe code coverage in a flow graph and the semantic approach will describe how syntactic coverage has to be changed to only take into account all feasible information flows.

Part I
Syntactic approach

1 Preliminaries

In this section, we will present the principles about graphs and notions of coverage.

1.1 Graphs and flow graphs

Definition 1.1. *A* directed graph *is a pair $G = (N, E, I)$, where N is a set of nodes and $E \subseteq (N \times N)$ a set of ordered pair of nodes called edges and $I \subseteq N$ a set of initial states.*

Definition 1.2. *Given a graph $G = (N, E, I)$, the* preset *and the* postset *of a node $n \in N$ are given by:*

$$pre(n) = \{n' \in N \mid (n', n) \in E\}$$

$$post(n) = \{n' \in N \mid (n, n') \in E\}$$

The number of edges entering and leaving a node $n \in N$ are denoted by:
$in(n) = |pre(n)|$ and $out(n) = |post(n)|$.

Definition 1.3. *Let $G = (N, E, I)$ be a directed graph. An* initial node *of the graph is node $n \in I$ such that every node is reachable from n. A* terminal node *is a node $n \in N$ such that $out(n) = 0$.*

Let T_G be the set of all terminal nodes of the graph G.

Definition 1.4. *Let $G = (N, E, I)$ be a directed graph. A* path *is a finite sequence of nodes $\pi = n_1 \dots n_k$ in N such that $\forall 1 \leq i \leq k-1 \,.\, (n_i, n_{i+1}) \in E$. The first node and the last node of π are: $first(\pi) = n_1$ and $last(\pi) = n_k$.*

Let Π be a set of paths in graph G. We define $N(\Pi) = |\{n \in N \mid \exists \pi \in \Pi \,.\, n \in \pi\}|$ as the number of different nodes in Π and $E(\Pi) = |\{(n, n') \in N \times N \mid \exists \pi \in \Pi \,.\, (n, n') \in \pi\}|$ as the number of different edges in Π.
We define $\Gamma(G)$ to be the set of all paths in G.

Definition 1.5. *A directed graph is* connected *if there is a path from any node to any other nodes in the graph.*

Definition 1.6. *A* flow graph *$G = (N, E, n_{init})$ is a connected directed graph such that there exists an unique initial node $n_{init} \in N$ and at least one terminal node in N.*

In a flow graph, depending on the predicates of a node, we can define different kinds of nodes.

Definition 1.7. *Given a graph $G = (N, E, n_1)$, a node is a* statement node *(*S-node*) if there is at most one edge leaving from it, otherwise it is a* decision node *(*D-node*).*

$$n \in N \text{ is an } (S\text{-node}) \text{ if } out(n) \leq 1$$

$$n \in N \text{ is } (D\text{-node}) \text{ if } out(n) > 1$$

Let S_G be the set of all (S-nodes) in the graph G and D_G the set of all (D-nodes). We represent S-nodes in graphical notations by rectangles and D-nodes by a diamonds.

Regarding a program, the flow graph is a representation of its possible control flows. More precisely, nodes correspond to statements and decisions and edges represent the possible flow of control between statements. For example, in C-functions, statements are expressions with a semi-colon such as *return-, continue-, if-, switch-, do-while-, for-, while-*.... An S-node represents a statement in the program such as modify an input and a D-node corresponds to decision in the program such as *if-, switch-, do-while-, for-, while-* statements.

In the program	**In the graph**	**Notation**
Statement	S-node	Rounded rectangle
Decision	D-node	Diamond
flow	Edges	Edges

If a path starts from the initial node to a terminal node of the graph G is called a *complete path.*

Definition 1.8. *Let $G = (N, E, r)$ be a flow graph with r the initial node and let $\pi = n_1 n_2 \dots n_k$ be a path in G. Then π is a complete path if $n_1 = r$ and $n_k \in T_G$*

Remark 1.9. *It's possible to have a program with loops and so the corresponding flow graph of this program will have an infinite set of paths and complete paths. Because we consider only the syntactic approach of a program. In order to have a finite set of complete paths, we introduce a* simple path *which can only traverse a loop twice. Let Λ be the set of all complete simple paths in a graph.*

Example 1.10. *Consider the following pseudo-code function and its corresponding flow graph:*

```
1.  int someFunction(int a, int b) {
2.     int result = 0;                    (*)
3.     if (a < b) {                       (**)
4.       System.exit(0);                  (*)
5.     }
6.     else {
7.       int c = a + b;                   (*)
8.       int i = 0;                       (*)
9.       while (i < c) {                  (**)
```

```
10.          result = (result + a) / b;      (*)
11.          i++;                            (*)
12.        }
13.      }
14.      return result;                      (*)
15.  }
```

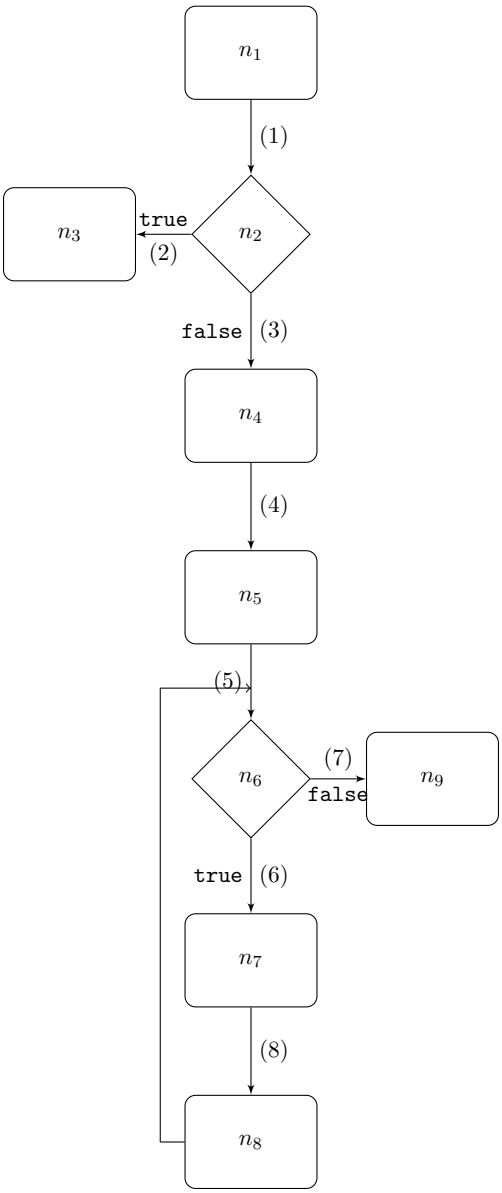


Figure 1: Control-flow graph for Example 1.10.

Note that this piece of code contains 7 *statements (indicated by asterisks) represented*

by S-nodes in the graph and 2 *decision points (indicated by double asterisks) represented by D-nodes.*

Because we only want to focus on the syntactic of the program, each label of a statement is ignored in the flow graph and replace by n_i *where* i *corresponds to the position of the statement in the code. Each edge is numberered. For instance the edge* (3) *represents the edge* (n_2, n_4)*. So, we will represent a edge by its number. Also, because a D-node is a decision in the program, its two possible flows/ edges correspond to the TRUE or FALSE value of the decision/condition. So, in the flow graph, the corresponding edges will be note by* `false` *or* `true`*.*

The flow graph that we obtain is $G = (N, E, n_1)$ *with* $N = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\}$ *and* $E = \{(1), (2), (3), (4), (5), (6), (7), (8), (9)\}$*. Note that* $T_G = \{n_3, n_9\}$*.*

We can notice that $\pi_1 = n_4 n_5 n_6 n_7$ *is a path in the flow graph with* n_4 *the first node of the path and* n_7 *the last one, and* $\pi_2 = n_1 n_2 n_3$ *or* $\pi_3 = n_1 n_2 n_4 n_5 n_8 n_6 n_9$ *are complete paths in the flow graph.*

1.2 Coverage measures

Because complete paths cover a certain number of nodes or edges, it's possible to define two percentages, representing the node coverage and edge coverage of a set of paths.

Definition 1.11. *Let* Π *be a set of complete paths in the graph* $G = (N, E, n_1)$*. The* Node coverage *of* Π *is the percentage of different nodes that is covered by at least one path in* Π*:*

$$NodeCov(G, \Pi) = \frac{N(\Pi)}{|N|} \times 100\%$$

Edge coverage *is the percentage of different edges that is covered by at least one path in* Π*:*

$$EdgeCov(G, \Pi) = \frac{E(\Pi)}{|E|} \times 100\%$$

In a graph, many different complete paths exist. Two percentages called *path coverage* and *simple path coverage* can be defined.

Definition 1.12. *Let* Π *be a set of complete paths in the graph* $G = (N, E, n_1)$ *and* Ω *be a set of complete simple paths in* G*.* Path coverage *corresponds to the percentage of complete paths that are contained in* Π

$$PathCov(G, \Pi) = \frac{|\Pi|}{|\Gamma(G)|} \times 100\%$$

Simple path coverage *corresponds to the percentage of complete simple paths that are contained in* Ω

$$PathCov_s(G, \Omega) = \frac{|\Omega|}{|\Lambda(G)|} \times 100\%$$

- $\pi_1 = n_1 n_2 n_3$
- $\pi_2 = n_1 n_2 n_4 n_5 n_6 n_7 n_8 n_9$
- $\pi_3 = n_1 n_2 n_4 n_5 n_6 n_9$

Example 1.13. *Consider the flow graph introduced in the example 1.10, and the followings paths*

The set of paths $\Pi = \{\pi_1, \pi_2, \pi_3\}$ *traverses every statement node, every edge and every complete path of the flow graph at least once. So* $NodeCov(G, \Pi) = EdgeCov(G, \Pi) = 100\%$. *If we apply only the path* π_1, *the nodes* n_1, n_2 *and* n_3 *are the only nodes traversed by the path. So, as the traversal of path* π_1 *visits* 3 *out of the* 9 *nodes, the statement coverage percentage is* $\frac{3}{9} \cdot 100\% = 33\%$. *Also only edges* (1) *and* (2) *are traversed by* π_2. *So* π_2 *visits* 2 *out of the* 9 *edges, the edge coverage percentage is* $\frac{2}{9} \cdot 100\% = 22\%$. *Due to the remark 1.9, the flow graph has 4 complete simple paths. The complete simple path* π_1 *only covers one possible path of the graph. So, the simple path coverage percentage is* $\frac{1}{4} \cdot 100\% = 25\%$ *and* $PathCov_s(G, \Pi) = 75\%$

□

2 Statement Coverage

A set of complete paths achieves complete statement coverage if it executes each statement in a flow graph at least once. For this purpose we only look at S-nodes. So we can create a reduced graph of the flow graph in order to keep only the S-nodes. This graph is called statement graph. By this reduction, complete node coverage in this new graph correspond to statement coverage in the flow graph.
In this section, we will define what the statement graph is and then how to apply criteria to the graph.

2.1 Statement Graph

A statement graph is a directed graph in which there are only statement nodes. So, from a flow graph, we have to delete decision nodes and edges between these decision nodes and other nodes. In order to maintain the connectivity of the graph, we have to add edges where D-nodes have been deleted. The statement graph is therefore defined as follows:

Definition 2.1. *Given a flow graph* $G = (N, E, n_1)$, *the* statement graph *of* G *is a graph* $G' = (S_G, E', I)$ *with*

$$E' = \{(n, m) \in (S_G \times S_G) \mid (n, m) \in E\} \cup$$

$$\{(n, m) \in (S_G \times S_G) \mid \exists d_1 \dots d_k \in D_G^+ \,.\, (n, d_1) \in E \wedge (d_k, n') \in E \wedge \forall 1 \leq i \leq k : (d_i, d_{i+1}) \in E\}$$

and $I = \{n_1 \mid n_1 \in S_G\} \vee \{n_i, n_j \mid n_1 \in D_G \wedge (n_1, n_i) \in E \wedge (n_1, n_j) \in E\}$

Example 2.2. *Consider the flow graph introduced in Example 1.10. To obtain the corresponding statement graph, we have to remove nodes* n_2 *and* n_6 *as well as, the edges* $(1), (2), (3), (5), (6), (7), (9)$ *and add the edges:*$(n_1, n_3), (n_1, n_4), (n_5, n_9), (n_5, n_7), (n_8, n_5)$.

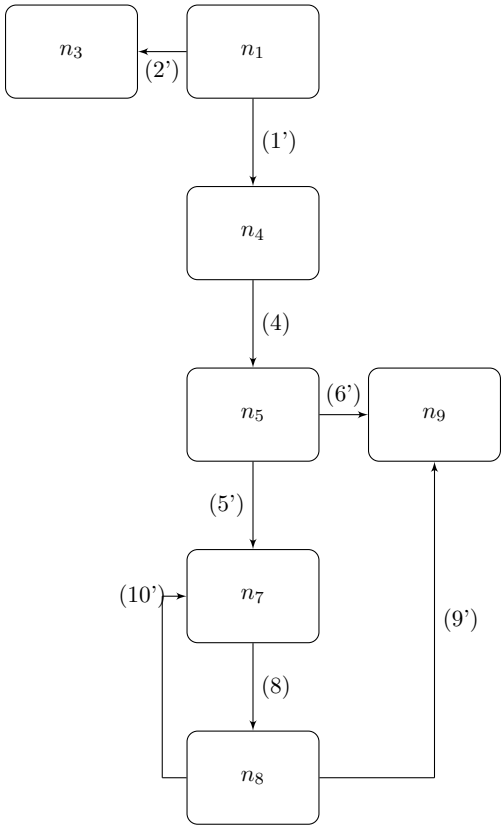


Figure 2: Statement graph for Example 2.2.

We obtain the graph $G_s = (N_s, E_s, n_1)$ *with* $N_s = n_1, n_3, n_4, n_5, n_7, n_8, n_9$ *and* $E_s = (1'), (2'), (4), (5'), (6'), (9'), (10')$. □

Remark 2.3. *As we can see in example 2.2, there still exist nodes with more than one edge leaving such as* n_1 *or* n_5*. So, due to the definition 1.7, those nodes are D-nodes. However, as we can see in the program or in the flow graph, those nodes are not decision/condition nodes. In fact, The definition of D-nodes and S-nodes is only available in a flow graph and not in a directed graph as the statement graph.*

2.2 Statement Coverage

There is complete statement coverage of a set of complete paths if all nodes from the statement graph are in a path of the set. It's possible to define statement coverage (as a percentage) for the flow graph because of the definition of the node coverage in the statement graph.

Definition 2.4. *Let* $G = (N, E, n)$ *a flow graph. Let* $G_s = (S_G, E_s, I)$ *be the statement graph of* G *and let* Π *be a set of complete paths in* G*, then*

$$StatCov(G, \Pi) = NodeCov(G_s, \Pi')$$

with $\Pi' = \Pi_{|N_s}$.

Remark 2.5. *Let* $G_s = (N_s, E_s, I)$ *be the statement graph and* Π *be a set of complete paths in* G_s*. There is the complete statement coverage for* Π *if for all nodes* $n \in G_s$*, there exists a path* $\pi \in \Pi$ *such that* $n \in \pi$ *.*

$$StatCov(G_s, \Pi) = 100\% \Leftrightarrow \forall n \in N_s \exists \pi \in \Pi \,.\, n \in \pi$$

Example 2.6. *Consider the flow graph* G *introduced in the example 1.10 and the corresponding statement graph* G_s *introduced in the example 2.2 and consider the two complete paths in* G *:*

- $\pi_1 = n_1 n_2 n_3$
- $\pi_2 = n_1 n_2 n_4 n_5 n_6 n_7 n_8 n_6 n_9$

To obtain the correpsonding path in the statement graph, we have to remove all decision nodes from these paths. So the corresponding complete paths in G_s *are:*

- $\pi_3 = n_1 n_3$
- $\pi_4 = n_1 n_4 n_5 n_7 n_8 n_9$

To achieve complete statement coverage, we could for instance apply the set of paths $\Pi = \{\pi_3, \pi_4\}$ *in* G_s*, as it traverses every statement node of the statement graph at least once. Moreover, the set* $\Pi' = \{\pi_1, \pi_2\}$ *also traverses all nodes in he flow graph, which that confirms the complete statement coverage. If we apply only the path* π_4*, the node* n_3 *would not be traversed anymore. So, as the traversal of path* π_1 *visits* 6 *out of the* 7 *nodes, the statement coverage percentage for the path* π_1 *is* $\frac{6}{7} \cdot 100\% = 85\%$*.*

□

3 Segment Coverage

A segment is a set of consecutive S-nodes that can be replaced by a single node. The graph that we obtain when replacing all segments by single nodes and changing the appropiate edges is called a segment graph. Based on this directed graph we can define segment coverage. In this section we will first define a segment graph and then define how to apply segment coverage to this graph.

3.1 Segment graph

First of all, in order to define a segment graph, we have to define what a segment is. A segment is a block of consecutives S-nodes, i.e., a sequence in which all nodes are the head and the tail of only one edge, except for the first node in the segment that can be the tail of more than one edge.

Definition 3.1. *In a flow graph $G = (N, E, n_1)$, a* segment *is a path $S = n_i \dots n_j$ in G such that $\forall i \leq k \leq j \,.\, n_i \in S_G$*

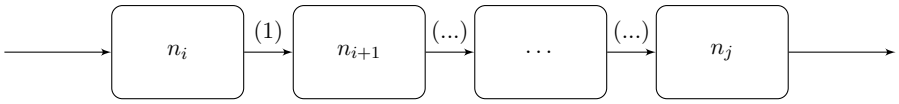


Figure 3: Example of a Segment in a flow graph

A segment S is a *maximal segment* if there is no other segment S' such that $S \subset S'$. So a segment $n_i \dots n_j$ is a maximal segment if there exists a path $\pi = n_{d_1} n_i \dots n_j n_{d_2}$ with n_{d_1} and n_{d_2} are D-nodes and $n_i \dots n_j$ are S-nodes. Let M_G be the set of all maximal segment in the flow graph.

Proposition 3.2. *Let $G = (N, E, n_1)$ be a flow graph. Each S-node exists in exactly one maximal segment.*

Proof. Let S and S' be two maximal segments. It follows that $S' \subset S$ or $S' \cap S = \emptyset$. If $S' \subset S$, then S' is not maximal. Therefore, $S' = S$ or $S' \cap S = \emptyset$. This means that two different maximal segments are disjoint. Let $n \in N$.
If $S = S'$ and $n \in S$ so $n \in S'$. But the maximal segments are equals, so n exists in one maximal segment.
If $S \cap S' = \emptyset$ and $n \in S$ so $n \notin S'$ and vice versa.
So a node exists in exactly one maximal segment. □

To create the segment graph of a flow graph, we have to replace all maximal segments S in the flow graph by single node n_S and changing some edges. We obtain the following definition:

Definition 3.3. *Let $G = (N, E, n_1)$ be a flow graph. $G' = (N', E', I)$ is the* segment graph *of G if:*

$$N' = \{n_S \mid S \in M_G\}$$

$$E' = \{(n_S, n'_S) \mid \{(last(S), first(S')) \in E\} \vee$$

$$\{\exists d_1 \dots d_k \in D_G^+ with (d_i, d_{i+1}) \in E \mid (last(S), d_1) \in E \wedge (d_k, first(S')) \in E)\}\}$$

$$and\ I = \{n_1 \mid n_1 \in S_G\} \vee \{n_i, n_j \mid n_1 \in D_G \wedge (n_1, n_i) \in E \wedge (n_1, n_j) \in E\}$$

Example 3.4. *Consider the flow graph introduced in example 1.10*
The segment graph is $G = (S_g, E_g, n_1)$ with $E_g = \{a, b, c, d, e, f\}$. □

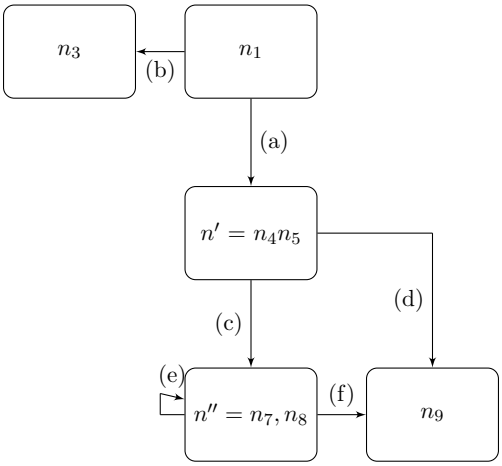


Figure 4: Segment graph for Example 3.4.

3.2 Segment coverage

Regarding the flow graph, segment coverage consists in: all segments of the graph have to be covered by a path, this criterion is also called *all-block* coverage. Because nodes in a segment graph correpsond to a segment in the flow graph, segment coverage in the segment graph corresponds to execute all nodes of this graph. So segment coverage for a flow graph corresponds to node coverage on the segment graph.

Definition 3.5. *Let $G = (N, E, n_1)$ be a flow graph and $G_s = (N_s, E_s, I)$ be the corresponding segment graph. Let Π be a set of paths in G_s.*

$$SegCov(G, \Pi) = NodeCov(G_s, \Pi')$$

with $\Pi' = \Pi_{|M_G}$

Proposition 3.6. *Let $G_s = (N_s, E_s, I)$ be a the segment graph.Let Π be a set of paths in G_s.*
Π implies segment coverage in G_s if all nodes $n \in N_s$ exist in a path in Π.

$$SegCov(G_s, \Pi) = 100\% \Leftrightarrow \forall n \in N_s \exists \pi \in \Pi \,.\, n \in \pi$$

Example 3.7. *Consider the flow graph introduced in example 1.10 and the segment graph introduced in example 3.4 and consider the two complete paths π_1, π_2 introduced in example 2.6. To obtain the corresponding paths in the statement graph, we have to restrict these paths to maximal segments. The corresponding complete paths in the segment graph are:*

- $\pi_5 = n_1 n' n'' n'' n_9$
- $\pi_6 = n_1 n_3$

To achieve complete segment coverage, we could for instance apply the set of paths $\Pi = \{\pi_5, \pi_6\}$, as it traverses every segment node of the segment graph at least once.

If we apply only the path π_5, the node n_3 would not be traversed anymore. So, as the traversal of path π_5 visits 3 out of the 4 nodes, the statement coverage percentage is $\frac{2}{3} \cdot 100\% = 66\%$.

□

4 Decision Coverage

It's also possible to define a new reduction of a flow graph so as to facilitate the definition of branch/decision coverage. In fact, we can create a graph that contains only D-nodes and terminal nodes. This new graph is called decision graph in which only decisions are kept.
In this section we will first define the decision graph and then define the criterion of decision coverage for the white box.

4.1 Decision graph

We define a *DD-path* (Decision-Decision path) to be a path $n_i \dots n_j$ where the start node n_i is a D-node and the last node n_j is D-nodes or a terminal node and all the other nodes $\{n_{i+1}, \dots, n_{j-1}\}$ in the path are S-nodes.
We assume the following proposition:

Proposition 4.1. *Let $G = (N, E, n_1)$ be a flow graph.*
1. *If $n, n' \in D_G$, then n can be the start node of out(n) DD-paths from n to n'*
2. *If a S-node exists in a DD-path then this S-node exists in exactly one DD-path.*

Proof. Let $G = (N, E, n_1)$ be a flow graph.
1. Let $b = |\text{post}(n)|$ and assume that there are b + 1 different DD-paths that start in n and end in n'. Then two DD-paths d_1, d_2 have the same first edge since there are only b possibilities to start a path from n. Let $d_1 = n_{11}n_{12} \dots n_{1k}$ and $d_2 = n_{21}n_{22} \dots n_{2x}$. Then $n_{11} = n_{21} = n, n_{12} = n_{22}$ and $n_{1k} = n_{2x} = n'$. Since $n_{12}, \dots, n_{1k-1}$ and $n_{22}, ..., n_{1x-1}$ are S-nodes, they have only one egde leaving from them, so postsets with only one element. It follows that $n_{13} = n_{23}, \dots, n_{1k-1} = n_{1h-1} and k - 1 = h - 1$. This means that both paths are equal.
2. Let d_1 and d_2 be two DD-paths of the graph. Because the first node and the last node of both d_1, d_2 are D-nodes and all the others in the DD-paths are S-nodes, if we remove these nodes, paths become segments as the definition in the previous section. We define also in the previous section, with the proposition 3.2 that each S segment, so by adding one start D-node and one final D-node to a segment, a S-node still exists in exactly one DD-path.

□

Let DD_G be the set of all DD-paths in a graph G.

To create the Decision graph we replace all S-nodes in all DD-path by one single edge and keep only D-nodes and terminal nodes.

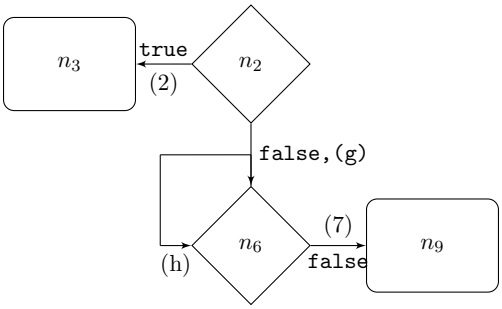


Figure 5: Decision graph for Example 4.3.

Definition 4.2. *Let $G = (N, E, n_1)$ be a flow graph. Then $G_d = (N_d, E_d, i)$ is the decision graph of G with:*

$$N_d = D_G \cup T_G \cup n_1$$

$$E_d = \{(n, n') \in E \,.\, \exists p_1, p_2 \in DD_G \mid first(p_1) = n \wedge last(p_2) = n'\}$$

and $I = \{n_1 \mid n_1 \in D_G\} \vee \{d \in D_g \mid n_1 \in S_G \wedge \exists n_1 \dots n_k d \in \Gamma(G) \,.\, n_1 \dots n_k \in S_G\}$

Example 4.3. *Consider the flow graph introduced in the example 1.10. The DD-paths in the flow graph are: $n_2n_4n_5n_6$ and $n_6n_7n_8n_6$. So we replace all S-nodes in thoses paths by single edges to obtain the decision graph $G_d = (N_d, E, d, n_2)$ with $N_d = \{n_2, n_3, n_6, n_9\}$ and $E_d = \{(2), (g), (h), (7)\}$.*

□

4.2 Decision coverage

Decision coverage or branch coverage consists in traverse every branch of a graph due to a set of complete paths, i.e, each possible outcome of each decision has to occur at least once on a path. Because in a decision graph, each edge corresponds to a branch of flow graph, and because in this graph there are only decision statements, each edge only corresponds to a possible outcome for a decision statement.

We can define the percentage of decision coverage for a flow graph based on edge coverage in the decision graph.

Definition 4.4. *Let $G = (N, E, n_i)$ be a flow graph and $G_d = (N_d, E_d, I)$ be the corresponding decision graph. Let Π be a set of paths in G*

$$DecCov(G, \Pi) = EdgeCov(G_d, \Pi')$$

with $\Pi' = \Pi_{|N_d}$

Definition 4.5. *Let $G = (N, E, I)$ be the decision graph. Let Π be a set of paths of G. Π satisfies the criteria of decision coverage if for all edges $e \in E_d$, there exists a path $p_k \in \Pi$ such that e is in p_k.*

$$\forall e \in E_d, \exists p_k \in \Pi \,.\, e \in p_k$$

Example 4.6. *Consider the flow graph and decision graph introcuded in example 4.3 and consider the two complete paths π_1 and π_2 introduced in example 2.6. To obtain the corresponding paths in the decision graph, we have to remove all S-nodes, except for terminal nodes. The two complete paths are:*

- $\pi_7 = n_2 n_6 n_6 n_9$
- $\pi_8 = n_2 n_3$

To achieve complete decision coverage, we could for instance apply the set of complete paths $\Pi = \{\pi_7, \pi_8\}$, as it traverses every edge of the decision graph at least once. If we apply the test suite π_7, the edge (2) would not be executed anymore. So, as the traversal of test case π_7 visits 3 out of the 4 statements, the statement coverage percentage is $\frac{3}{4} \cdot 100\% = 75\%$. □

5 Relations between coverage measures.

The first relation that we can establish is the one between the coverage measures.

Proposition 5.1. *Let $G = (N, E, n_1)$ be a flow graph and Π be a set of complete paths of G.*

$$\mathit{PathCov}(G, \Pi) = 100\% \Rightarrow \mathit{EdgeCov}(G, \Pi) = 100\% \Rightarrow \mathit{NodeCov}(G, \Pi) = 100\%$$

Proof. We assume that $\mathrm{PathCov}(G, \Pi) = 100\%$.
If Π covers all paths of the graph G, that means all edges of the graph are covered. Because edges are links between nodes, if all edges are covered by Π , Π contains all possibles nodes of G. □

Equivalences in the other direction are not always true as we can see it in the following example.

Example 5.2. *Consider the flow graph introduced in the example 1.10 and consider the set of complete paths Π composed of the four following complete paths:*

- $\pi_1 = n_1 n_2 n_3$
- $\pi_2 = n_1 n_2 n_4 n_5 n_6 n_9$
- $\pi_3 = n_1 n_2 n_4 n_5 n_6 n_7 n_8 n_6 n_9$
- $\pi_4 = n_1 n_2 n_4 n_5 n_6 n_7 n_8 n_6 n_7 n_8 n_9$

This set of path corresponds to all possible paths in the flow graph. So $\mathit{PathCov}(G, \Pi) = 100\%$. We notice that $N(\Pi) = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\} = N$ and $E(\Pi) = E$. So the set Π covers all nodes and all edges of the flow graph. So $\mathit{NodeCov}(G, \Pi) = 100\%$ and $\mathit{EdgeCov}(G, \Pi) = 100\%$.

If we consider the example 1.13, we notice that there is no relation possible between path coverage, node coverage and edge coverage if there is no complete path coverage. □

Statement coverage and Segment coverage both define a node coverage in different graphs. However, there exist a relation between these two coverages.

Proposition 5.3. *Let $G = (N, E, n_1)$ be a flow graph and Π be a set of complete paths of G.*

$$StatCov(G, \Pi) = 100\% \Leftrightarrow SegCov(G, \Pi) = 100\%$$

Proof. Let $G = (N, E, n_1)$ be a flow graph and $G_s = (N_s, E_s, I_s)$ and $G_g = (N_g, E_g, I_g)$ be respectivelly the statement graph and the segment graph of G
'$\Rightarrow$': We assume StatCov$(G, \Pi) = 100\%$. Let $n \in G_g$ Then there exists a maximal segment S in the control flow graph and the statement graph with $n = n_S$. Let $S = n_1, n_2, \ldots, n_k$ be the path of S-nodes. So there exists $\pi \in \Pi$ such that $S \subset \pi$. So there exists in G_s π' such as $n \in \pi'$ and $\pi' = \pi_{|N_g}$. Because it's true for all nodes is G_s, so all nodes of the segment graph are covered. So SegCov$(G, \Pi) = 100\%$

'$\Leftarrow$':We assume SegCov$(G, \Pi) = 100\%$. Let $n \in N_s$ and let S be the maximal segment that contains n. Then there exists a path d in Π such that $n \in d$. □

This proposition is true only is we considere 100 % of coverage. In fact,because one segment can define more than one statement, for a same path, the statement coverage can be different to the segment coverage.

Example 5.4. *Consider the flow graph G, statement graph G_s and segment graph G_g introduced before. Consider the following set of complete paths Π in G:*

- $\pi_1 = n_1 n_2 n_4 n_5 n_6 n_7 n_8 n_6 n_9$
- $\pi_2 = n_1 n_2 n_3$

The corresponding set of paths in the statement graph is: $\Pi_s = \{n_1 n_4 n_5 n_7 n_8 n_9, n_1 n_3\}$ which covers all nodes in the graph. So $NodeCov(G_s, \Pi_s) = 100\% = StatCov(G, \Pi)$

The corresponding set of paths in the segment graph is: $\Pi_g = \{n_1 n' n'' n_9, n_1 n_3\}$ which covers all nodes in the graph. So $NodeCov(G_g, \Pi_g) = 100\% = StatCov(G, \Pi)$.

However, if we consider the flow graph in the figure 5.4, we notice that if there is 50 % of segment coverage with the path $n_1 n_2 n_3$ there is no 50 % of statement coverage by 25 %. Vice vera, if there is 75 % of statement coverage with the path $n_1 n_2 n_4 n_5 n_6$, there is only 50 % of segment coverage.

□

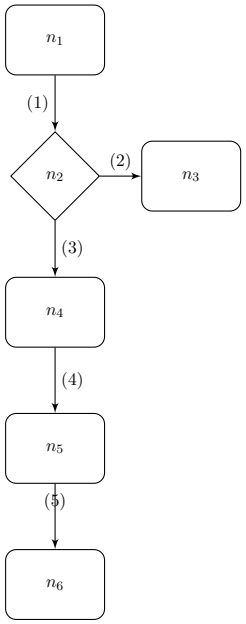


Figure 6: flow grpah for the example 5.4

It's also possible to define relations between decision coverage and statement coverage.

Proposition 5.5. *Let $G = (N, E, n_1)$ be a flow graph and Π be a set of complete paths of G.*

$$DecCov(G, \Pi) = 100\% \Rightarrow StatCov(G, \Pi) = 100\%$$

Proof. We assume that $\mathrm{DecCov}(G, \Pi) = 100\%$
Decision coverage or branch coverage corresponds in the decision graph as in the control flow graph, to cover all edges by paths. So $E(\Pi) = E$. So if Π covers all edges of G, it covers all nodes of G. So $N(\Pi) = N$. So $Stat(G, \Pi) = 100\%$ □

Example 5.6. *Consider the flow graph G in the the figure and consider the set of complete paths:*

- $\pi_1 = n_1 n_2 n_4 n_5 n_6$
- $\pi_2 = n_1 n_2 n_3$

The following figure represents the decision graph of G. In this graph, Π is equivalent to $\Pi_d = \{n_2 n_6, n_2 n_3\}$ which covers all edges. So $DecCov(G, \Pi) = 100\%$. The following figure represents the statement graph of G. In this graph, Π is equivalent to $\Pi_d = \{n_1 n4 n_5 n_6, n_1 n_3\}$ which covers all segments. So $StatCov(G, \Pi) = 100\%$.

However, if we only apply the path π_1, $DecCov(G, \pi_1) = 50\%$ and $StatCov(G, \pi_1) = 80\%$

The following example proves that there is no equivalence.

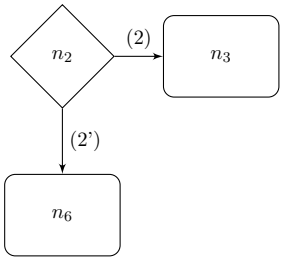


Table 1: the decision graph

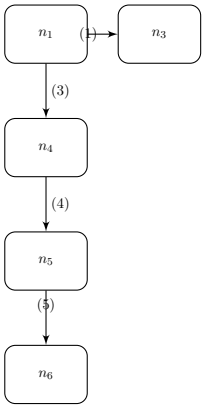


Table 2: The statement graph

If we consider the flow graph in the next figure 7, we notice that if there is 100 % statement coverage with the path $n_1n_2n_4n_5n_6$ there is only 50 % of decision coverage (the edge False of the decision node is not covered).

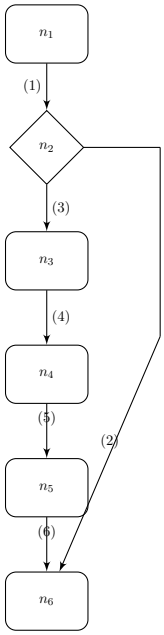


Figure 7: flow graph for the example 5.6

□

However, in some particular case, the proposition is true in the other direction. In fact, Segment coverage can imply decision coverage if there are segments in all possible branch of the graph.

Proposition 5.7. *Let $G = (N, E, n_1)$ be a flow graph without empty branches i.e there exist at least one node between two D-nodes or a D-node and a terminal node in G. Let Π be a set of complete paths of G.*

$$SegCov(G, \Pi) = 100\% \Leftrightarrow DecCov(G, \Pi) = 100\%$$

Example 5.8. *In order to prove the condition, we need to consider the following flow graph.*

In this flow graph in figure 8, we notice that this flow graph has no empty branch: there is at least one S-node between a decision node and a terminal node. If we apply the set of complete paths $\Pi = n_1n_2n_3, n_1n_2n_4n_5n_6$, $StatCov(G, \Pi) = 100\%$. Because this path travers all edges of the graph there is also $DecCov(G, \Pi) = 100\%$.
The condition that there must have no empty branch is present in the example 5.6.

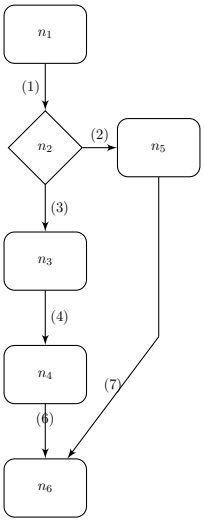


Figure 8: Flow graph for the example 5.8

□

To sum up, the figure 9 represents the relations between the coverage measures.

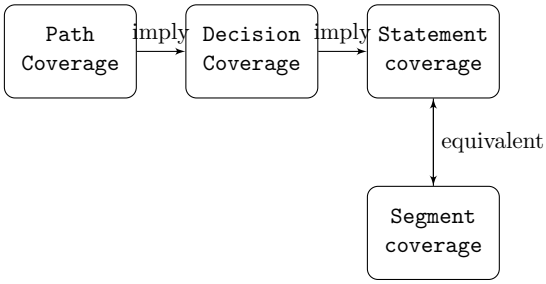


Figure 9: Representation of the relations between coverage measures

Part II
Semantic approach

6 From flow graph to control flow graph

This section will describe the relation between flow graphs defined in the previous part, control flow graphs and programs.

A *control flow graph* is a flow graph representing the possible control flow of a program and taking into account the label of each statement. So, a node in the control flow graph corresponds to a statement with its label in the program and edges represent the possible transfer of control flow between statements. The difference with the flow graph is that for the control flow graph we consider only the possible paths regarding inputs of the program. We define the *Label function* in order to associate each node to each label:

Definition 6.1. *Given a directed graph $G = (N, E, I)$ of a program p, the* Label function *is given by:*

$$L : N \to Stat^+$$

Where N is a set of nodes and $Stat^+$ is a set of statements of p.

In the previous part, to apply white box coverages to a flow graph, we were only using a set of complete paths. Because in this part we have to consider the semantic of the program, we have to define what is a test regarding the control flow graph of a program. In a program, a test case is a set of inputs. So when we pick a test case of the program, we have to plot its path through the control flow graph.

Definition 6.2. *Let p be a program and $G = (N, E, n_1)$ be its control flow graph. If t is a test case of p then there exists a corresponding complete path π in G*

According to the previous definition, a test suite which is a set of test cases, has in the control flow graph an equivalent Π set of complete paths. As a result, all test case for a program corresponds to all paths in the control flow graph

Example 6.3. *Consider the following pseudo-code function:*

```
 1.  int someFunction(int a, int b) {
 2.     int result = 0;                          (*)
 3.     if (a < b && a <> result) {                (**)
 4.       System.exit(0);                        (*)
 5.     }
 6.     else {
 7.       int c = a + b;                         (*)
 8.       int i = 0;                             (*)
 9.       while (i < c) {                        (**)
10.         result = (result + a) / b;           (*)
```

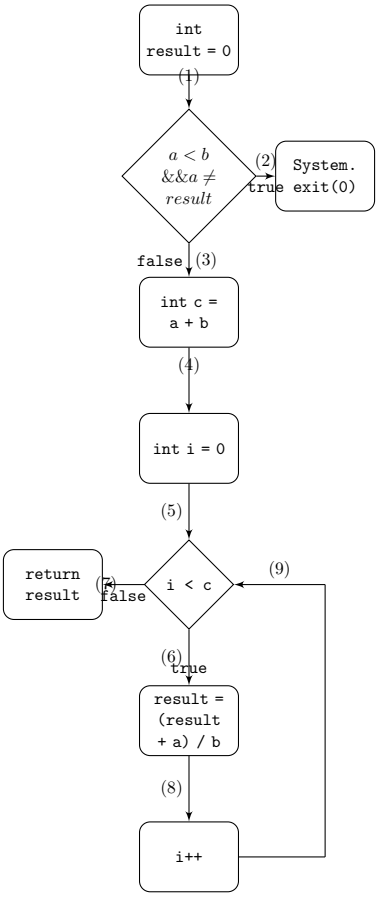


Figure 10: Control-flow graph for Example 6.3.

```
11.          i++;                                    (*)
12.        }
13.      }
14.      return result;                              (*)
15.  }
```

Note that the control flow graph is $G = (N, E, n_1)$ *with* $L(n_1) = \{int\ result = 0\}$*. Now, each test execution of the program can be seen as a traversal of this graph. Take for instance the test case* $t = (a = 1, b = 1 \mid 1)$*, were the values before the bar indicate the inputs, and the value after the bar indicates the expected result. It traverses the edges 1-2-5-6-7-8-9-10-8-9-10-11-12.* □

7 Statement coverage

The control flow graph is by definiion a flow graph, it's also possible to reduce it in a statement graph. However in this part, we want to focus on the semantic part of a

code of a program. In fact, in practice, many syntactic paths in the control flow graph representating a program are semantically infeasible, i.e, they may not be traversed by any execution of the program. We need in the graph all decision nodes in order to know if a path is feasible or not thanks its relation with the inputs. Yet, we will use the definitions defined in the previous part in order to define the semantic coverages. These definitions will have *syn* as exponent.

Statement coverage consists in a test suite that executes all statements at least once of a program. Regarding the control flow graph, statement coverage consists in: all nodes of the control flow graph have to be covered by a path that corresponding to a test in the test suite.

Definition 7.1. *Let $G = (N, E, n_1)$ be the control flow graph of the program p. Let T be a test suite for p and Π be the corresponding set of complete paths in G. Let θ be the set of all possible test cases for p and Θ be the corresponding set of possible complete paths in G.*

$$\text{StatCov}^{sem}(T, p) = \frac{\text{StatCov}^{syn}(G, \Pi)}{\text{StatCov}^{syn}(G, \Theta)}$$

Thus, semantic statement coverage is defined only on the possible nodes/statements that can be covers by a test or a path and not only in the whole program/graph.

Example 7.2. *Consider the program introduced in example 6.3, and consider the two test cases:*

- $t_1 = (a = 0, b = 1 \mid \text{error})$
- $t_2 = (a = 1, b = 1 \mid 1)$

To achieve complete statement coverage, we could for instance apply the test suite $T_1 = \{t_1, t_2\}$, as it traverses every S-node of the control flow graph at least once. If we apply the test suite $T_2 = \{t_2\}$, the second statement (line 4; the rounded rectangle in the upper right of the control flow graph) would not be executed anymore. So, as the traversal of test case T_2 visits 6 out of the 7 statements, the statement coverage percentage is $\frac{6}{7} \cdot 100\% = 85.7\%$. However, this percantage can be found with or without take into account the semantic of the program.

It is easy to write down a program for which not all statements are reachable. For instance:

```
1.  void someFunction(int a, int b) {
2.     if (a > b && a < b) {
3.       System.exit(0);
4.     }
5.     else {
6.       System.out.println("Succeeded.");
7.     }
8.  }
```

Clearly, any test case for this function achieves 50% *statement coverage: the statement on line 3 is unreachable and the statement on line 6 is always reached. So in this case,* $StatCov^{syn}(G, \Pi) = 50\%$ *whatever the set of complete paths if we take into account the semantic of the program. However,* $StatCov^{sem}(G, \Pi) = 100\%$. Π *can for instance corresponds to the test* $T = (a = 1, b = 2)$.

8 Decision coverage

For a program, decision coverage or branch coverage means to execute all branches of it so as to be sure that all possible outcomes of decision statements be executed. To have 100 % decision coverage there must exists a test suite such that it generates as set of paths that covers all branches/edges of the graph of the program wether for decision graph or control flow graph. In fact, the decision graph is just a reduction of the flow graph which preserves all possible outcome of decision statements only. However, as for the statemen coverage, we need to take into account the semantic of the program. The semantic decision coverage is defined regarding the possible outcomes of decision statements and not all of them such as in the syntactic approach. Yet, we will use the previous definition of decision coverage which will have *syn* as exponant, in order to define the semantic coverage.

Definition 8.1. *Let* $G = (N, E, n_1)$ *be the control flow graph of the program* p. *Let* T *be a test suite for* p *and let* Π *be the corresponding set of complete paths in* G. *Let* θ *be the set of all possible test cases for* p *and* Θ *be the corresponding set of possible complete paths in* G.

$$\mathrm{DecCov}^{sem}(T, p) = \frac{\mathrm{DecCov}^{syn}(G, \Pi)}{\mathrm{DecCov}^{syn}(G, \Theta)}$$

Example 8.2. *Consider the flow graph introduced in the figure of the previous example. As the statement in the line 3, the edge* (1) *is unreachable. If we apply the test* $T = (a = 1, b = 2)$ *or whatever of complete paths taking into account the semantic of the program,* $DecCov^{syn}(G, \Pi) = 50\%$ *but on the other hand* $DecCov^{sem}(p, T) = 100\%$.

9 Condition Coverages

Condition coverages group the condition coverage, the condition/decision coverage and the multiple condition coverage which are based on the execution on all conditions in the program. By definition, the flow graph reveals the presence of conditions, however it's possible that there are more than one condition in one statement. Take for example the line 2 of the program in example 7.2. Condition graph solves this problem by creating a new entity that represent the behavior of conditions in a decision node. In this section, we will first define a condition graph, and then apply criteria coverage in this graph.

9.1 Condition graph

In a control flow graph representing a program, only D-nodes are concerned by this new graph. In fact, a D-node represents a decision in the program. This D-node contains a certain numbers of relational expressions or Boolean variables called *compound condition.*

In our case, we will only consider that there are only D-node with 2 edges in a control flow graph.

Because each edge from a D-node is fixed by the outcome of the D-node's decision, we called T_n the edge leaving from a D-node n corresponding to the TRUE value of the decision and F_n the other edge corresponding to the FALSE value. As we already did it in previous examples.

Definition 9.1. *Let $G = (N, E, n_1)$ be a control flow graph. Let Π be a set of complete paths in G.*
We define $E_T(\Pi) = |\{T_n \in E \mid \exists \pi \in \Pi T_n \in \pi\}|$ as the number of differents `true edges` in the set of paths Π. Symmetrically, we define $E_F(\Pi))$ as the number of differents `false edges` in Π.

A D-node can contain more than one compound condition. In order to create the condition graph, we have to define a new entity called *C-node* (Condition node) that contains only one compound condition and then group them into a *condition block* equivalent to the D-node.

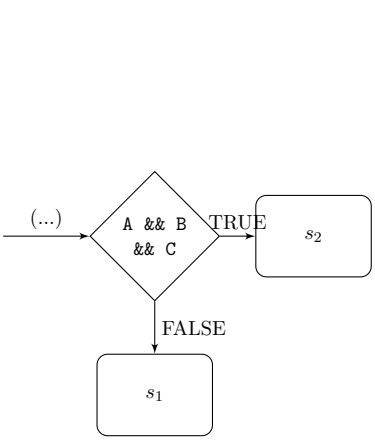
Definition 9.2. *The* C-node n_C *is a node containing only the condition C. C-node is a D-node.*

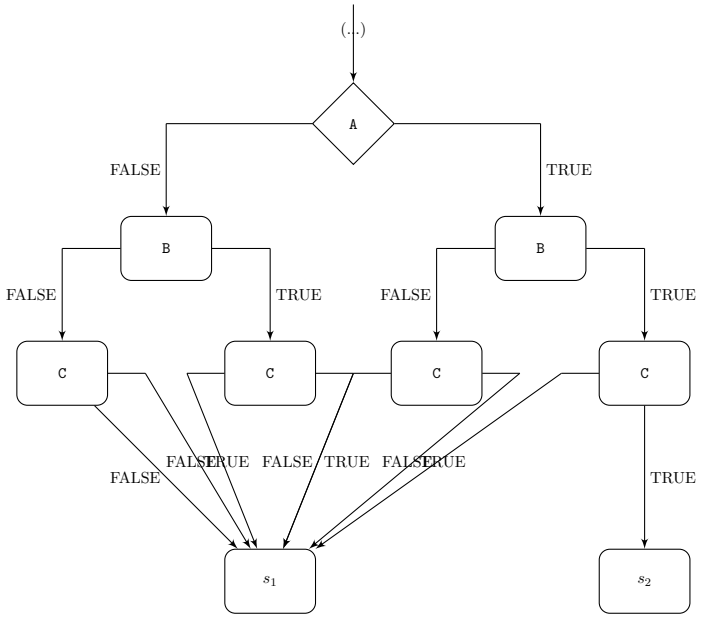
So in the control flow graph $G = (N, E, n_1)$, all D-nodes have to be replace by a condition block. The block corresponding to a node n with $(n, m), (n, p) =\in E$ is a tree of C-nodes in which:

- Each layer contains 2^{i-1} C-nodes of the condition C_i.
- Each edge e_T and e_F of each C-nodes n_i of the condition C_i are linking to a C-node n_{i+1} of the condition C_{i+1} in the next layer, except for the last layer.
- The condition block is linking to nodes m and p according to the relation between conditions.

Let C_G be the set of all condition blocks in the graph G.

Example 9.3. *Consider a simple D-node n with the conditions A,B,C such as A&&B&&C. From n, there are the two edges (n, s_1) and (n, s_2). The condition block of n will have three layers, one per conditions, in which there will have 2^{i-1} C-node of the corresponding condition where i represents to possition of the condition. For instance for the condition B, $i = 2$. Then the TRUE edge and the FALSE edge of each C-node is linked to a C-node of the next layer. The last layer is linked to the node s_1 and s_2 only if the conditions A, B, C are true. The following figures show how to create this condition block.*





In order to link the condition block with the whole function, we need to know which combinations of conditions'value lead to the value TRUE or FALSE of the starting D-node. It's possible for this to write the *truth table* of the combinations' relation. In fact due to the truth table, we can easily know which edges of the last layer have to be links with the next right node.

Example 9.4. *If we consider the decision node introduced in the example 9.3, in order to link this condition block to the whole control flow graph, we can create the table table of the conditions.*

The True table reveals that only the path crossing only `true` edges of C-node of the condition A, B, C, can lead to the node s_2. All others paths lead to s_1.

A	**B**	**C**	**A &&B&&C**
T	T	T	T
T	T	F	F
T	F	T	F
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

All elements of the condition graph were defined; we obtain the condition graph graph from the decision graph in which we replace all D-nodes by a condition block. So the condition graph contains only nodes that necessary to build the condition block, terminal nodes and edges necessary.

Example 9.5. *Consider the program and the flow graph introduced in example 6.3. To obtain the condition graph, we have to change the condition statement in the line 3 in a condition block. The condition statement in the line 9 contains already only one condition, so it's already a condition block.*

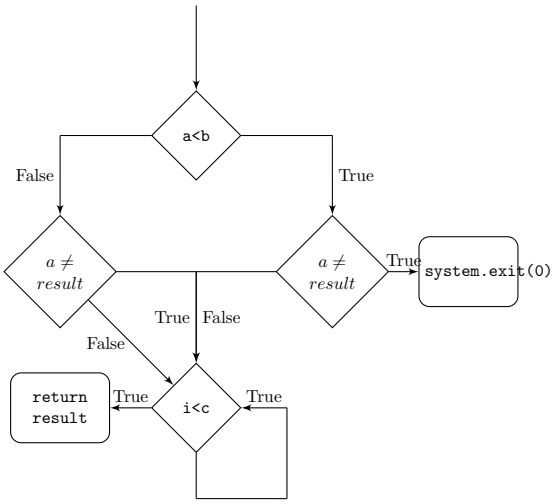


Figure 11: Condition graph for the program SomeFunction

9.2 Condition coverage

Based on this new graph, we can easily define the condition coverage. In fact, condition coverage consists in evaluate to TRUE and FALSE each condition independently of the value of the other conditions in a program. Because the condition graph represents each condition in one node due to the condition block, it's possible to find paths that evaluate the TRUE and FALSE of conditions.

Definition 9.6. *Let $G = (N, E)$ be the condition graph of the program p. Let T be a test suite for p and $\Pi = \{\pi_1, \ldots, \pi_k\}$ be the corresponding set of complete paths in G There is 100 % condition coverage in p by T if: For all D-nodes n in G, there exists paths π_i, π_j in Π such that* `true edge` *of n is in p_i and* `false edge` *of n is in p_j*

$$\forall n \in D_G, \exists i, j \in \{1, \ldots, k\} \,.\, p_i, p_j \in T \wedge T_n \in p_i \wedge F_n \in p_j$$

In this definition T_n represents the edge leaving the node n and corresponding to the TRUE value. And F_n represents the edge leaving the node n and corresponding to the FALSE value.

The percentage of condition coverage for a graph corresponds how much conditions the test suite tests. For 100 % condition coverage, the test case need to contain at least two tests: one which tests all `true` values and the second one which tests all `false` values.

Definition 9.7. *Let $G = (N, E, I)$ be the decision graph of a program p with C conditions. Let $G_c = (N_c, E_c)$ be the condition graph of p. Let T be a test suite for p and Π be the corresponding set of complete paths in G.*

$$\text{CondCov} = \frac{E_T(\Pi) + E_F(\Pi)}{2C} \times 100\%$$

Example 9.8. *Consider the condition graph introduced in the example 9.5. This program consists of 3 conditions, so $C = 3$. A test suite that wants to achieve complete condition coverage should make sure that each of the conditions is evaluated to* `true` *at least once and to* `false` *at least once. For this, consider the following test cases:*

- $t_1 = (a = 1, b = 2 \mid System.exit(0))$
- $t_2 = (a = 2, b = 1 \mid 6)$

In each test case, the values before the bar indicate the inputs, and the value after the bar indicates the expected result.

Applying the test suite $T = \{t_1, t_2\}$ yields complete condition coverage. Therefore, all conditions have been evaluated at least once to `true` *and once to* `false`*, so $E_T(T) = 3$, $E_F(T) = 3$ and therefore the condition coverage percentage is $\frac{3+3}{2 \cdot 3} \cdot 100\% = 100\%$.*

Note that the test suite $T_2 = \{t_1\}$ does not have complete condition coverage. Although all conditions at least once hold and at least once do not hold for the inputs provided by these test cases, not all of the decision nodes containing them are indeed reached. For T_2, the condition $a \neq result$ is not evaluate to `true` *and the the condition $a¡b$ is not evaluate to* `false`*. So we have $E_T(T_2) = 2$ and $E_F(T_2) = 2$, so the condition coverage percentage is $\frac{4+2}{2 \cdot 3} \cdot 100\% = 66\%$.*

□

9.3 Decision-Condition Coverage

Condition-decision coverage combines the requirements for decision coverage with those for condition coverage. That is, there must be sufficient test cases to execute the decision outcome between `true` and `false` and to execute each condition value between `true` and `false`. So a test suite T corresponds to a condition-decision coverage is a test that satisfies condiction coverage and decision coverage.

Definition 9.9. *Let $G = (N, E)$ be a condition graph of a program p. Let T be a test suite for p*

T is condition-decision coverage $\Leftrightarrow T$ is condition coverage $\wedge T$ is decision coverage

Definition 9.10. *Let $G = (N, E)$ be a condition graph of a program p. Let T be a test suite for p.*

$$DecConCov(p, T) = \frac{CondCov(p, T) + DecCov(p, T)}{2}$$

Example 9.11. *Consider the complete paths in example 9.8. we know that CondCov(p,T)=100 %, however, the test suite T doesn't allow to have complete decision-condition coverage. In fact, the* `true` *value of the D-node corresponding to the condition block is not evaluated. So, DecCov(p,T) = 75 %. We can determine decision-condition coverage for T and the control flow graph:* $\frac{100+75}{2} = 88\%$.

□

9.4 Multiple condition coverage

For *multi-condition coverage*, we do not only require every condition to be evaluated to `true` and to `false` at least once; we require every *combination* of such valuations of conditions within a decision to occur at least once. So, for a decision `if (a && b)` to be completely covered with respect to this metric, we should make sure that the decision is reached at least once while `a` is `true` and `b` is `true`, once while `a` is `true` and `b` is `false`, once while `a` is `false` and `b` is `true`, and once while `a` is `false` and `b` is `false`. So if there are n conditions is the function f, they will be 2^n tests. Based on the condition graph, Multiple condition coverage correspond to execute all possibles edges in the graph. In fact, in the graph edges represent the outcome for each conditions of the function so all edges in the graph represent all possible combinaisons of conditions.

Definition 9.12. *Let $G = (N, E)$ be the condition graph of a program p. Let T be a test case of p and Π the corresponding set of complete paths in G.*
There is multiple condition coverage by T in p if:

$$\forall p \in \Gamma(G) \,.\, p \in \Pi$$

So we can define the percentage of multiple condition coverage in a program using the edge coverage defined previously in the paper:

Definition 9.13. *Let $G = (N, E)$ be the condition graph of a program p. Let T be a test case of p and Π the corresponding set of complete paths in G. Let Θ be the corresponding set of possible complete paths in G.*

$$\mathit{MultiCov}(p, T) = \frac{\mathrm{EdgeCov}^{syn}(G, \Pi)}{\mathrm{EdgeCov}^{syn}(G, \Theta)}$$

or

$$\mathit{MultiCov}(p, T) = \frac{\mathrm{EdgeCov}^{syn}(G, \Pi)}{2 \times |D_G|}$$

Example 9.14. *Consider the program and the condition graph introduced in the example* **??**. *The graph contains 4 D-nodes, so $|D_G| = 4$. Each of the four decisions consists of one condition (by definition of the condition graph). However, by definition of the program or the control flow grapn, there are $2^2 + 2 = 8$ combinations of conditions. The test case $T = \{t_1, t_2, t_3\}$ introduced in example 9.8 had complete condition coverage, but does not have complete multi-condition coverage. The following table provides the condition combinations that occur: In fact, the test suite doesn't evaluate the combinaison 'false,*

Test	`a < b`	$a \neq result$	`i < c`
t_1	`true`	`true`	-
t_2	`false`	`true`	`true`, `false`
t_3	`true`	`false`	`true` , `false`

false' for the two first conditions. Moreover, if we consider Π *as the corresponding set of complete paths of* T*,* $E(\Pi) = 7$ *out of 8. So, we covered* 7 *out of the* 8 *combinations, yielding a multi-condition coverage of* $\frac{7}{8} \cdot 100\% = 88\%$*.*

10 Relations of semantic coverage measures

The relations of syntactic coverage measures are still available with the semantic coverage measure. However, Condition coverages are stronger than decision coverage or statement coverage.

The relationship of all the code coverage metrics are represented in the figure 12

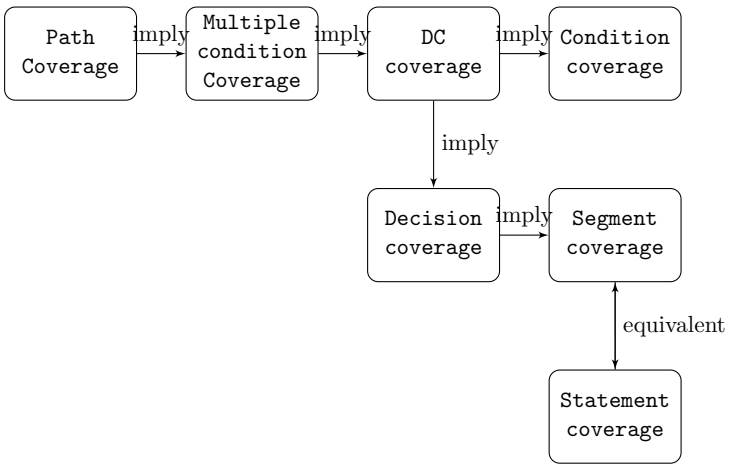


Figure 12: Representation of the relations between code coverage metrics

Conclusion

White box testing is a dynamic testing and based on knowledge of how a program works. There are four main theoretical techniques: Statement, branch, condition, and paths coverage. In order to define each one, we have created a flow graph adapted to each criteria. For instance, the statement graph only contains nodes that we need for the statement coverage. Thoses graphs represent the possible control flow in the program. However, syntactically indicated behaviors (statements, edges, etc.) are often impossible and that imply unreachable code, infeasible edges, paths that must be detected by white box coverage. In this paper, we decide to define the semantic coverage based on the syntactic one and the feasible paths of the program. One other solution can be to change all graphs so as to remove all unreachable paths, so as it defined in the paper *Path-Sensitive Analysis through Infeasible-Path Detection and Syntactic Language Refinement*.

References

http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf
Control flowgraphs and code coverage by ROBERT GOLD
Flow graph reducibility by Matthew S. Hecht and Jeffrey D. Ullman
Software verification and validation by Lionel Briand
Data flow analysis techniques for test data selection by Sandra Rapps and Elaine J. Weyuker
An integrated approach to software engineering, chapter TESTING by P. Jalote
On partitioning program graphs by Michael R. Paige
Test coverage by Martijn Adolfsen
Path-Sensitive Analysis through Infeasible-Path Detection and Syntactic Language Refinement by Gogul Balakrishnan, Sriram Sankaranarayanan, Franjo Ivan, Ou Wei, and Aarti Gupta.