
Integration Testing

Integration Testing

- **Integration testing** is a logical extension of **unit testing**.
 - In its simplest form, two units that have already been tested are combined into a component and the **interface** between them is tested.
 - A **component**, in this sense, refers to an integrated aggregate of more than one unit.
-

Integration Testing and Unit Testing

- Integration testing identifies problems that occur when units are combined.
 - By using a test plan that requires you to test each unit and ensure the viability of each before combining units, you know that any errors discovered when combining units are likely related to the interface between units.
 - This method reduces the number of possibilities to a far simpler level of analysis.
-

Interaction Errors

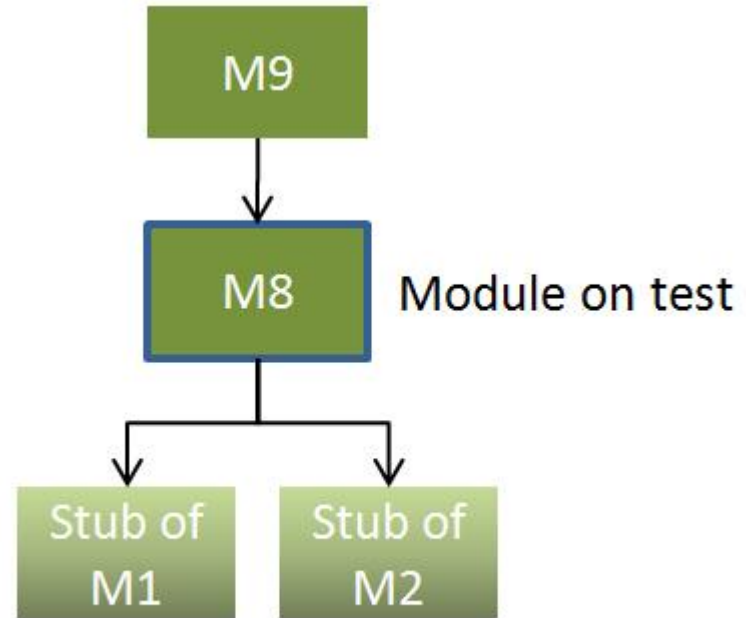
- **Interface Misuse** - A calling unit calls another unit and makes an error in its use of interface, probably by calling/passing parameters in the wrong sequence.
 - **Interface Misunderstanding** - A calling unit makes some assumption about the other units behavior which are incorrect.
-

Strategies in Integration Testing

- The **top-down approach** to integration testing requires the highest-level modules be tested and integrated first.
 - The **bottom-up approach** requires the lowest-level units be tested and integrated first.
 - The **sandwich approach** combines the top-down approach and the bottom-up approach.
-

Stubs

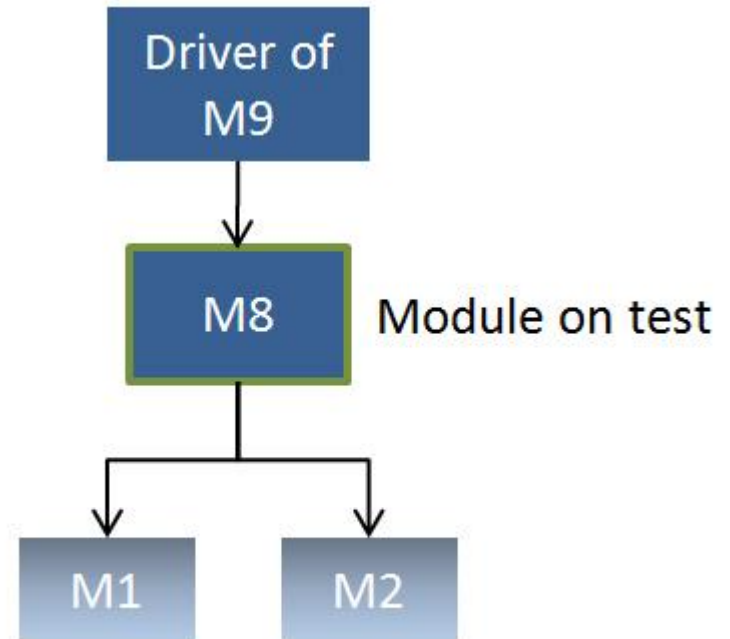
- A **stub** is a piece of code that simulates the activity of a unavailable lower level unit.
- Stubs are needed in top-down approach.



Implementing top-down tests

Drivers

- A **driver** is a piece of code that passes a test case to an available lower level unit.
- Drivers are needed in bottom-up approach.

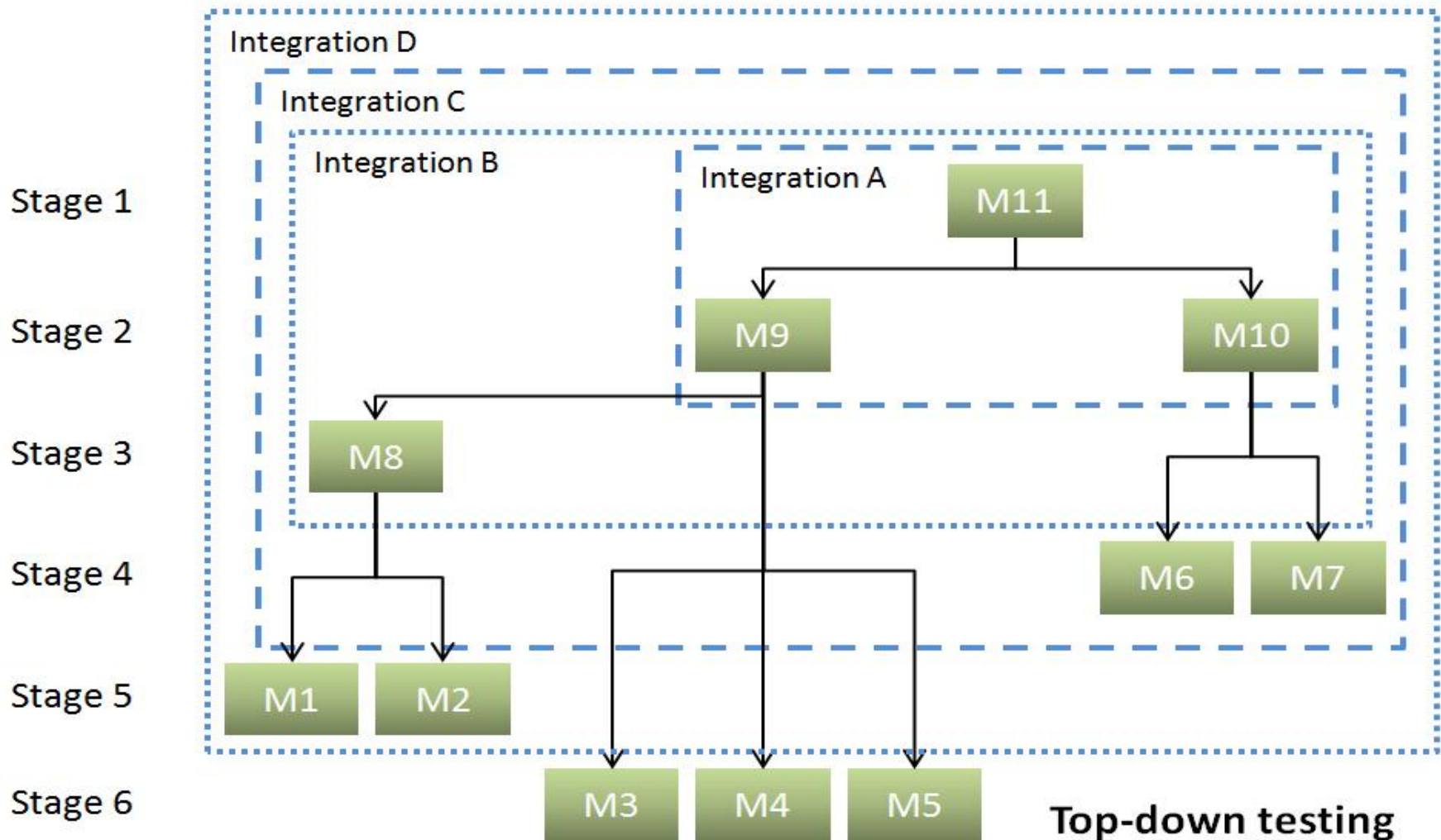


Implementing bottom-up tests

Top-Down Approach

- This allows high-level logic and data flow to be tested early in the process and it tends to minimize the need for drivers.
 - However, the need for stubs complicates test management and low-level utilities are tested relatively late in the development cycle.
 - Another disadvantage of top-down integration testing is its poor support for early release of limited functionality.
-

An Example



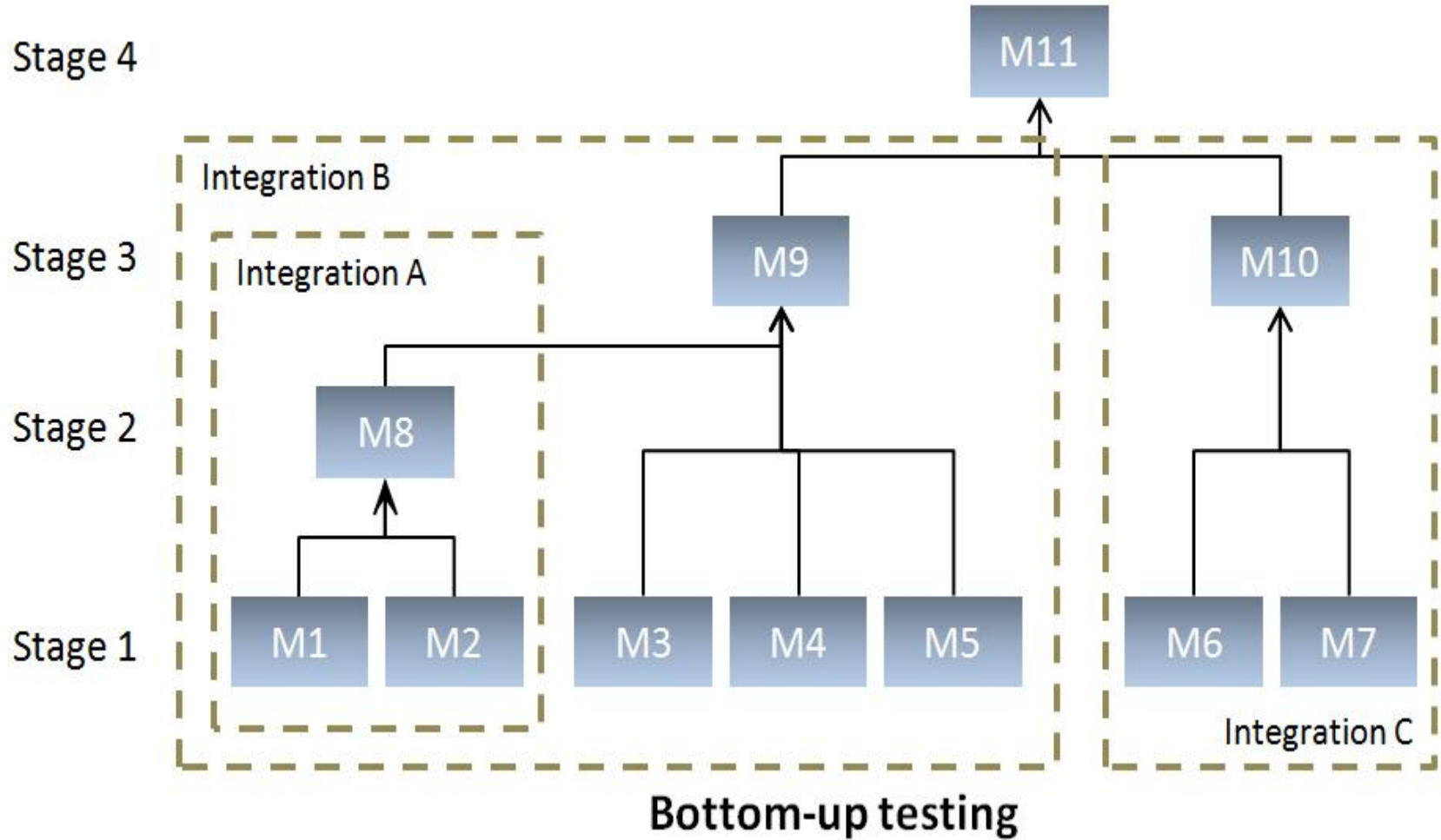
Top-Down Approach

- Top down testing can proceed in a **depth-first** or a **breadth-first** manner.
 - For depth-first integration each module is tested in increasing detail, replacing more and more levels of detail with actual code rather than stubs.
 - Alternatively breadth-first would proceed by refining all the modules at the same level of control throughout the application.
 - In practice a combination of the two techniques would be used.
-

Bottom-Up Approach

- These units are frequently referred to as utility modules.
 - By using this approach, utility modules are tested early in the development process and the need for stubs is minimized.
 - The downside, however, is that the need for drivers complicates test management and high-level logic and data flow are tested late.
 - Like the top-down approach, the bottom-up approach also provides poor support for early release of limited functionality.
-

Bottom-Up Approach



Sandwich Approach

- It focuses mainly upon testing the modules that contain **high degree of user interaction**.
 - The **input modules** are integrated in the **bottom-up** pattern, and the **output modules** are integrated in the **top-down** manner.
 - This approach is beneficial in the sense that it enables early release of a GUI-based application that enhances its functionality.
 - This approach is less systematic than the other two approaches.
-

Integration Testing Techniques

- **Coupling-based integration testing** — a structural testing technique.
 - **Interaction-based integration testing** — a functional testing technique.
-

Coupling-Based Integration Testing

- **Coupling** between two units measures the **dependency relations** between two units by reflecting the **interconnections** between units.
 - Integration faults are found exactly where couplings typically occur.
 - Coupling between two units increases the interconnections between the two units and increases the likelihood that a fault in one unit may affect the other.
-

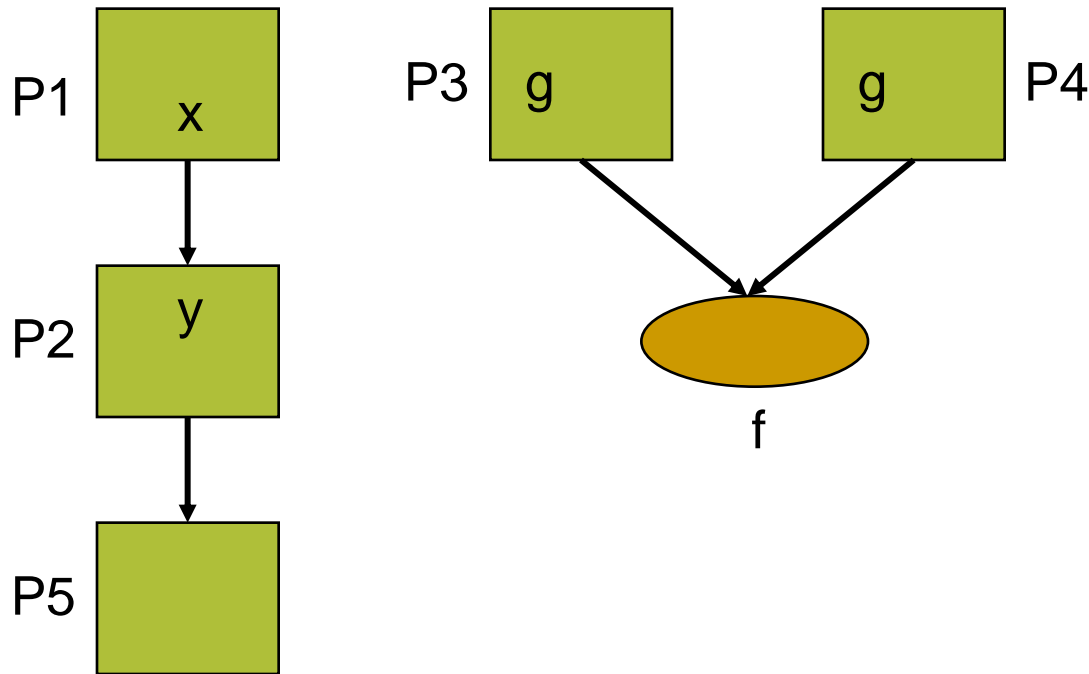
Coupling Types

- **Call coupling**: a unit A calls another unit B, but there are no parameters, common variable references, or common references to external media.
 - **Parameter coupling**: a unit A calls another unit B, and a variable in A is **passed** to B and is used in B.
-

Coupling Types

- **Shared data coupling:** a unit A calls another unit B, and both A and B refer to the same **non-local or global variable**.
 - **External device coupling:** a unit A calls another unit B, and both A and B access the same **external medium**.
-

A Call Graph Example



Basic Definitions

- If a unit A calls another unit B, A is called the **caller** and B is called the **callee**.
 - If A calls B, **actual parameters** of A are passed and assigned to **formal parameters** of B.
 - The **interface** between A and B is the mapping of actual to formal parameters.
-

Basic Definitions

- Assume that the control flow graph for each unit is present.
 - $\text{def}(P, V)$ is the set of nodes in P that contain a definition of a variable V .
 - $\text{use}(P, V)$ is the set of nodes in P that contain a use of a variable V .
 - **Call_site** is a node in the control flow graph of a unit A from which another unit B is called.
-

Basic Definitions

- **Call**($P_1, P_2, \text{call_site}, x \rightarrow y$): **TRUE** if unit P_1 calls unit P_2 at call_site and actual parameter x is mapped to formal parameter y .
 - **Return**(v): Nodes from which values for v are returned.
 - **Start**(P): The entry node of P .
-

An Example

```
procedure QUADRATIC is
begin
  GET(Control_Flag);
  if (Control_Flag = 1) then
    GET(X);
    GET(Y);
    GET(Z);
  else
    X := 10.0;
    Y := 0.0;
    Z := 12.0;
  end if;
  OK := TRUE;
  ROOT(X, Y, Z, R_1, R_2, OK);
```

An Example

```
ROOT(X, Y, Z, R_1, R_2, OK);  
if (OK) then  
    PUT(R_1);  
    PUT(R_2);  
else  
    PUT("No solution");  
end if;  
end QUADRATIC;
```

An Example

procedure ROOT(A, B, C: in FLOAT; ROOT_1, ROOT_2: out FLOAT;
Result: in out BOOLEAN) is

```
    ...  
D: REAL;  
    ...  
begin  
    D := B ** 2 - 4.0 * A * C;  
    if (Result and D < 0.0) then  
        Result := FALSE;  
        return;  
    end if;  
    ROOT_1 := (-B+sqrt(D))/(2.0*A);  
    ROOT_2 := (-B-sqrt(D))/(2.0*A);  
    Result:= TRUE;  
end ROOT;
```


Coupling-def

- A **coupling-def** is a node in a unit A that contains a definition that can reach a use in another unit B on at least one execution path.
- **Last-def-before-call, lbc-def(P_1 , call_site, x)**:
The set of nodes that defines **x** and for which there is a def-clear path from the node to the **call_site** in P_1 .

An Example

```
procedure QUADRATIC is
begin
  GET(Control_Flag);
  if (Control_Flag = 1) then
    GET(X);           -- last-def-before-call(X)
    GET(Y);           -- last-def-before-call(Y)
    GET(Z);           -- last-def-before-call(Z)
  else
    X := 10.0;        -- last-def-before-call(X)
    Y := 0.0;         -- last-def-before-call(Y)
    Z := 12.0;        -- last-def-before-call(Z)
  end if;
  OK := TRUE;        -- last-def-before-call(OK)
  ROOT(X, Y, Z, R_1, R_2, OK);  -- call-site
```

Coupling-def

- **Last-def-before-return, lbr-def(P_2, y):** The set of nodes that defines the returned variable y and for which there is a def-clear path from the node to the return statement in P_2 .
- **Shared-data-def, shared-last-def(P_3, P_4, g):** The set of nodes that defines a nonlocal or global variable g in P_3 that is used in P_4 , and for which there is a def-clear path from the def to the use.

An Example

procedure ROOT(A, B, C: in FLOAT; ROOT_1, ROOT_2: out FLOAT;
Result: in out BOOLEAN) is

```
...
D: REAL;
...
begin
  D := B ** 2 - 4.0 * A * C;
  if (Result and D < 0.0) then
    Result := FALSE;           -- last-def-before-return(Result)
    return;
  end if;
  ROOT_1 := (-B+sqrt(D))/(2.0*A); -- last-def-before-return(ROOT_1)
  ROOT_2 := (-B-sqrt(D))/(2.0*A); -- last-def-before-return(ROOT_2)
  Result:= TRUE;              -- last-def-before-return(Result)
end ROOT;
```

Coupling-use

- A **coupling-use** is a node in a unit B that contains a use that can be reached by a definition in another unit A on at least one execution path.
- **First-use-after-call**, **fac-use**(P_1 , **call_site**, **x**):
For call-by-reference, the set of nodes in P_1 that have uses of **x** and for which there is a def-clear path with no other uses or defs between the **call_site** for P_1 and these nodes.

An Example

```
ROOT(X, Y, Z, R_1, R_2, OK);    -- call-site
if (OK) then                    -- first-use-after-call(OK)
    PUT(R_1);                   -- first-use-after-call(R_1)
    PUT(R_2);                   -- first-use-after-call(R_2)
else
    PUT("No solution");
end if;
end QUADRATIC;
```

Coupling-use

- **First-use-in-callee, fic-use(P_2 , y)**: For call-by-value, the set of nodes for which parameter y in P_2 has a use, and there is a def-clear path with no other uses from the start statement to this use.
- **Shared-data-use, shared-first-use(P_4 , g)**: The set of nodes that uses a nonlocal or global variable g , and there is a def-clear path with no other uses from the start statement to this use.

An Example

procedure ROOT(A, B, C: in FLOAT; ROOT_1, ROOT_2: out FLOAT;
Result: in out BOOLEAN) is

```
...  
D: REAL;  
...  
begin  
  D := B ** 2 - 4.0 * A * C;           -- first-use-in-callee(A, B, C)  
  if (Result and D < 0.0) then       -- first-use-in-callee(Result)  
    Result := FALSE;  
    return;  
  end if;  
  ROOT_1 := (-B+sqrt(D))/(2.0*A);  
  ROOT_2 := (-B-sqrt(D))/(2.0*A);  
  Result:= TRUE;  
end ROOT;
```

External References

- **external-ref(i, j)**: For external coupling, the pair of references (i, j) to the same external file or device is called an external-reference.
 - In files and other external devices, considering definitions and uses does not make sense.
-

Coupling-Paths

- A **coupling path** is a path between two units from a definition to a use, or between two references, and that satisfies certain other requirements.
 - The other requirements depend on the type of coupling between the two units.
-

Parameter Coupling Paths

- **parameter-coupling**(P_1 , P_2 , **call_site**, x , y):
For each actual parameter x in P_1 , and each last definition of x before a **call_site**, there is a parameter coupling path from the last definition, to the **call_site**, and to each first use of the formal parameter y in P_2 .
-

An Example

procedure QUADRATIC is

...

begin

GET(Control_Flag);

if (Control_Flag = 1) then

GET(X); -- last-def-before-call(X)

GET(Y); -- last-def-before-call(Y)

GET(Z); -- last-def-before-call(Z)

else

X := 10.0; -- last-def-before-call(X)

Y := 0.0; -- last-def-before-call(Y)

Z := 12.0; -- last-def-before-call(Z)

end if;

OK := TRUE; -- last-def-before-call(OK)


ROOT(X, Y, Z, R_1, R_2, OK); -- call-site



An Example

procedure ROOT(A, B, C: in FLOAT; ROOT_1, ROOT_2: out FLOAT;
Result: in out BOOLEAN) is

```
...  
D: REAL;  
...  
begin  
  D := B ** 2 - 4.0 * A * C;           -- first-use-in-callee(A, B, C)  
  if (Result and D < 0.0) then       -- first-use-in-callee(Result)  
    Result := FALSE;                 -- last-def-before-return(Result)  
    return;  
  end if;  
  ROOT_1 := (-B+sqrt(D))/(2.0*A);    -- last-def-before-return(ROOT_1)  
  ROOT_2 := (-B-sqrt(D))/(2.0*A);   -- last-def-before-return(ROOT_2)  
  Result:= TRUE;                     -- last-def-before-return(Result)  
end ROOT;
```



An Example

procedure QUADRATIC is

...

begin

GET(Control_Flag);

if (Control_Flag = 1) then

GET(X); -- last-def-before-call(X)

GET(Y); -- last-def-before-call(Y)

GET(Z); -- last-def-before-call(Z)

else

X := 10.0; -- last-def-before-call(X)

Y := 0.0; -- last-def-before-call(Y)

Z := 12.0; -- last-def-before-call(Z)

end if;

OK := TRUE; -- last-def-before-call(OK)

ROOT(X, Y, Z, R_1, R_2, OK); -- call-site



An Example

procedure ROOT(A, B, C: in FLOAT; ROOT_1, ROOT_2: out FLOAT;
Result: in out BOOLEAN) is

```
...  
D: REAL;  
...  
begin  
  D := B ** 2 - 4.0 * A * C;           -- first-use-in-callee(A, B, C)  
  if (Result and D < 0.0) then       -- first-use-in-callee(Result)  
    Result := FALSE;                 -- last-def-before-return(Result)  
    return;  
  end if;  
  ROOT_1 := (-B+sqrt(D))/(2.0*A);    -- last-def-before-return(ROOT_1)  
  ROOT_2 := (-B-sqrt(D))/(2.0*A);    -- last-def-before-return(ROOT_2)  
  Result:= TRUE;                     -- last-def-before-return(Result)  
end ROOT;
```

An Example

procedure QUADRATIC is

...

begin

GET(Control_Flag);

if (Control_Flag = 1) then

GET(X); -- last-def-before-call(X)

GET(Y); -- last-def-before-call(Y)

GET(Z); -- last-def-before-call(Z)

else

X := 10.0; -- last-def-before-call(X)

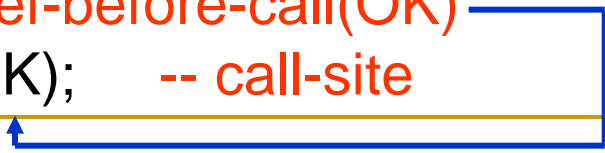
Y := 0.0; -- last-def-before-call(Y)

Z := 12.0; -- last-def-before-call(Z)

end if;

OK := TRUE; -- last-def-before-call(OK)

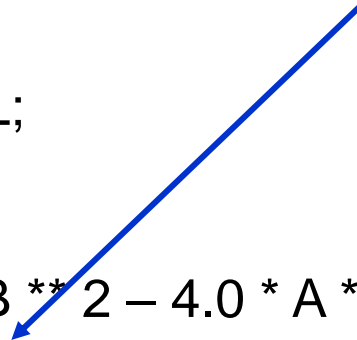
ROOT(X, Y, Z, R_1, R_2, OK); -- call-site



An Example

procedure ROOT(A, B, C: in FLOAT; ROOT_1, ROOT_2: out FLOAT;
Result: in out BOOLEAN) is

```
...  
D: REAL;  
...  
begin  
  D := B ** 2 - 4.0 * A * C;           -- first-use-in-callee(A, B, C)  
  if (Result and D < 0.0) then       -- first-use-in-callee(Result)  
    Result := FALSE;                 -- last-def-before-return(Result)  
    return;  
  end if;  
  ROOT_1 := (-B+sqrt(D))/(2.0*A);    -- last-def-before-return(ROOT_1)  
  ROOT_2 := (-B-sqrt(D))/(2.0*A);    -- last-def-before-return(ROOT_2)  
  Result:= TRUE;                     -- last-def-before-return(Result)  
end ROOT;
```



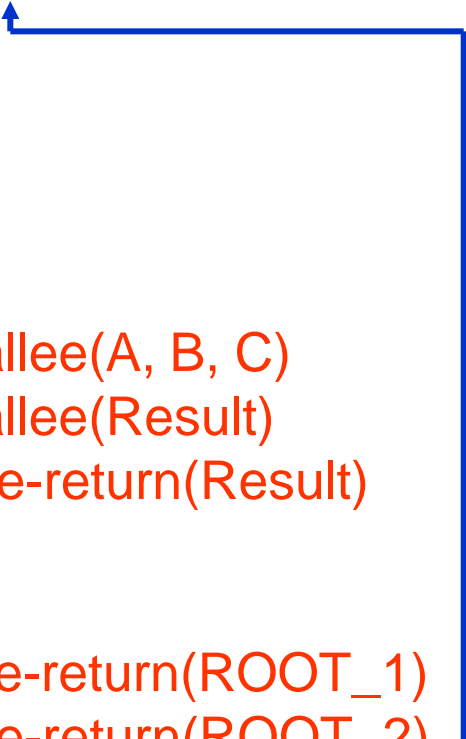
Parameter Coupling Paths

- **parameter-coupling(P_1 , P_2 , call_site, x, y)**: If a parameter x is call-by-reference, then there is also a parameter coupling path from each last definition before return of the formal parameter y in P_2 to each first use after the call of x in P_1 .

An Example

procedure ROOT(A, B, C: in FLOAT; ROOT_1, ROOT_2: out FLOAT;
Result: in out BOOLEAN) is

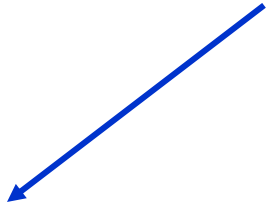
```
...
D: REAL;
...
begin
  D := B ** 2 - 4.0 * A * C;           -- first-use-in-callee(A, B, C)
  if (Result and D < 0.0) then        -- first-use-in-callee(Result)
    Result := FALSE;                 -- last-def-before-return(Result)
    return;
  end if;
  ROOT_1 := (-B+sqrt(D))/(2.0*A);     -- last-def-before-return(ROOT_1)
  ROOT_2 := (-B-sqrt(D))/(2.0*A);    -- last-def-before-return(ROOT_2)
  Result:= TRUE;                     -- last-def-before-return(Result)
end ROOT;
```



An Example

```
ROOT(X, Y, Z, R_1, R_2, OK);           -- call-site
if (OK) then
  PUT(R_1);
  PUT(R_2);
else
  PUT("No solution");
end if;
end QUADRATIC;
```

-- first-use-before-call(OK)
-- first-use-before-call(R_1)
-- first-use-before-call(R_2)



An Example

procedure ROOT(A, B, C: in FLOAT; ROOT_1, ROOT_2: out FLOAT;
Result: in out BOOLEAN) is

```
...
D: REAL;
...
begin
  D := B ** 2 - 4.0 * A * C;           -- first-use-in-callee(A, B, C)
  if (Result and D < 0.0) then       -- first-use-in-callee(Result)
    Result := FALSE;                -- last-def-before-return(Result)
    return;
  end if;
  ROOT_1 := (-B+sqrt(D))/(2.0*A);    -- last-def-before-return(ROOT_1)
  ROOT_2 := (-B-sqrt(D))/(2.0*A);    -- last-def-before-return(ROOT_2)
  Result:= TRUE;                     -- last-def-before-return(Result)
end ROOT;
```

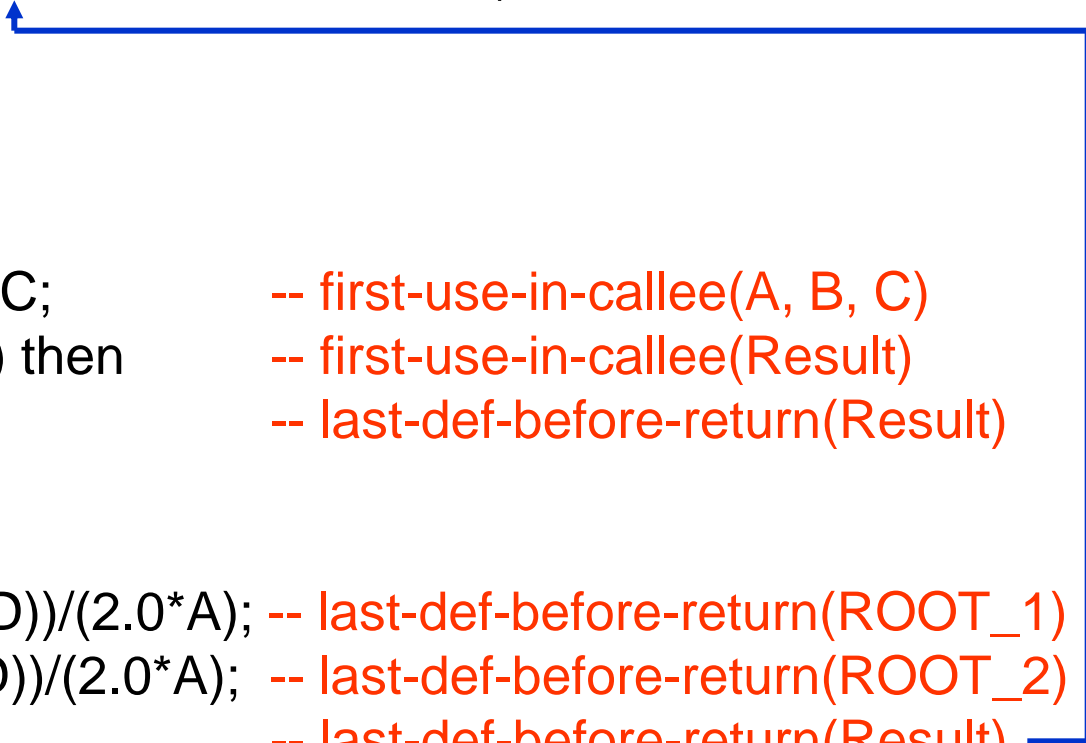
An Example

```
ROOT(X, Y, Z, R_1, R_2, OK);           -- call-site
if (OK) then                            -- first-use-before-call(OK)
    PUT(R_1);                           -- first-use-before-call(R_1)
    PUT(R_2);                           -- first-use-before-call(R_2)
else
    PUT("No solution");
end if;
end QUADRATIC;
```

An Example

procedure ROOT(A, B, C: in FLOAT; ROOT_1, ROOT_2: out FLOAT;
Result: in out BOOLEAN) is

```
...
D: REAL;
...
begin
  D := B ** 2 - 4.0 * A * C;           -- first-use-in-callee(A, B, C)
  if (Result and D < 0.0) then       -- first-use-in-callee(Result)
    Result := FALSE;                -- last-def-before-return(Result)
    return;
  end if;
  ROOT_1 := (-B+sqrt(D))/(2.0*A);    -- last-def-before-return(ROOT_1)
  ROOT_2 := (-B-sqrt(D))/(2.0*A);   -- last-def-before-return(ROOT_2)
  Result:= TRUE;                    -- last-def-before-return(Result)
end ROOT;
```



An Example

```
ROOT(X, Y, Z, R_1, R_2, OK);           -- call-site
if (OK) then                            -- first-use-before-call(OK)
    PUT(R_1);                            -- first-use-before-call(R_1)
    PUT(R_2);                            -- first-use-before-call(R_2)
else
    PUT("No solution");
end if;
end QUADRATIC;
```

Shared Data Coupling Paths

- **shared-data-coupling(P_3 , P_4 , g)**: For each nonlocal or global variable g that is defined in P_3 and used in P_4 , and each definition of g in P_3 , there is a shared data coupling path that is def-clear with respect to g from the definition to each first use of g in P_4 .
-

External Device Coupling Paths

- For each pair of references (i, j) to the same external device, an external device coupling path executes both i and j on the same execution path.
 - In files and other external devices, considering definitions and uses does not make sense.
-

Testing Coverage Criteria

- **Call coupling** requires that the set of paths executed by the test set covers all `call_sites` in the program.
 - **All-coupling-defs** requires that for each coupling-def of a variable `x`, the set of paths executed by the test set contains at least one coupling path to **at least one** reachable coupling-use.
-

Testing Coverage Criteria

- **All-coupling-uses** requires that for each coupling-def of a variable x , the set of paths executed by the test set contains at least one coupling path to **each** reachable coupling-use.
 - **All-coupling-paths** requires that all coupling paths be executed.
-

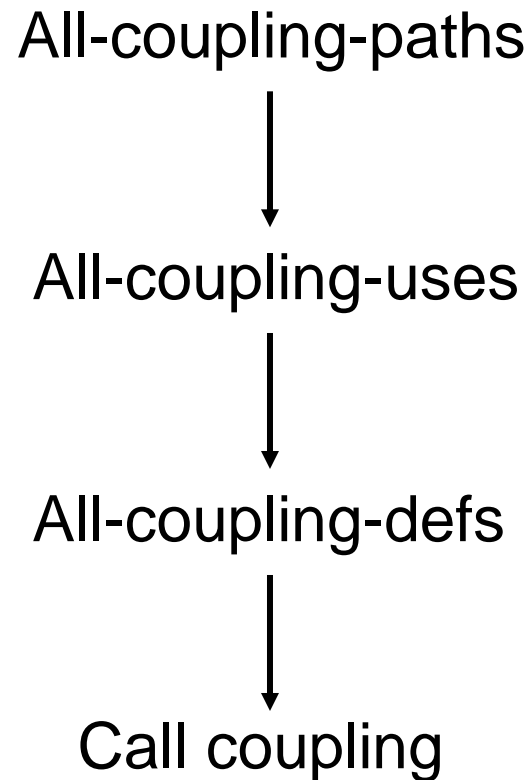
Handling Loops

- A **subpath set** is the set of nodes on some subpath.
 - There is a **many-to-one mapping** between subpaths and subpath sets.
 - If there is a loop within the subpath, the associated subpath set is the same no matter how many iterations of the loop are taken.
 - A **coupling path set** is the set of nodes on a coupling path.
-

Handling Loops

- **All-coupling-paths** requires that for each coupling-def of a variable x , the set of paths executed by the test set contains all **coupling path sets** from the coupling-def to **all** reachable coupling-uses.
 - If there is a loop, all-coupling-paths requires **two cases**: the loop body is not executed at all and the loop body is executed some arbitrary number of times.
-

Testing Coverage Criteria Hierarchy



Coupling Coverage Analysis

- **Structural coverage analysis** is needed to determine whether all couplings have been covered.
 - This analysis can be done on **coupling graphs**.
 - A coupling graph is a directed graph $C = (M, E, F, A)$.
-

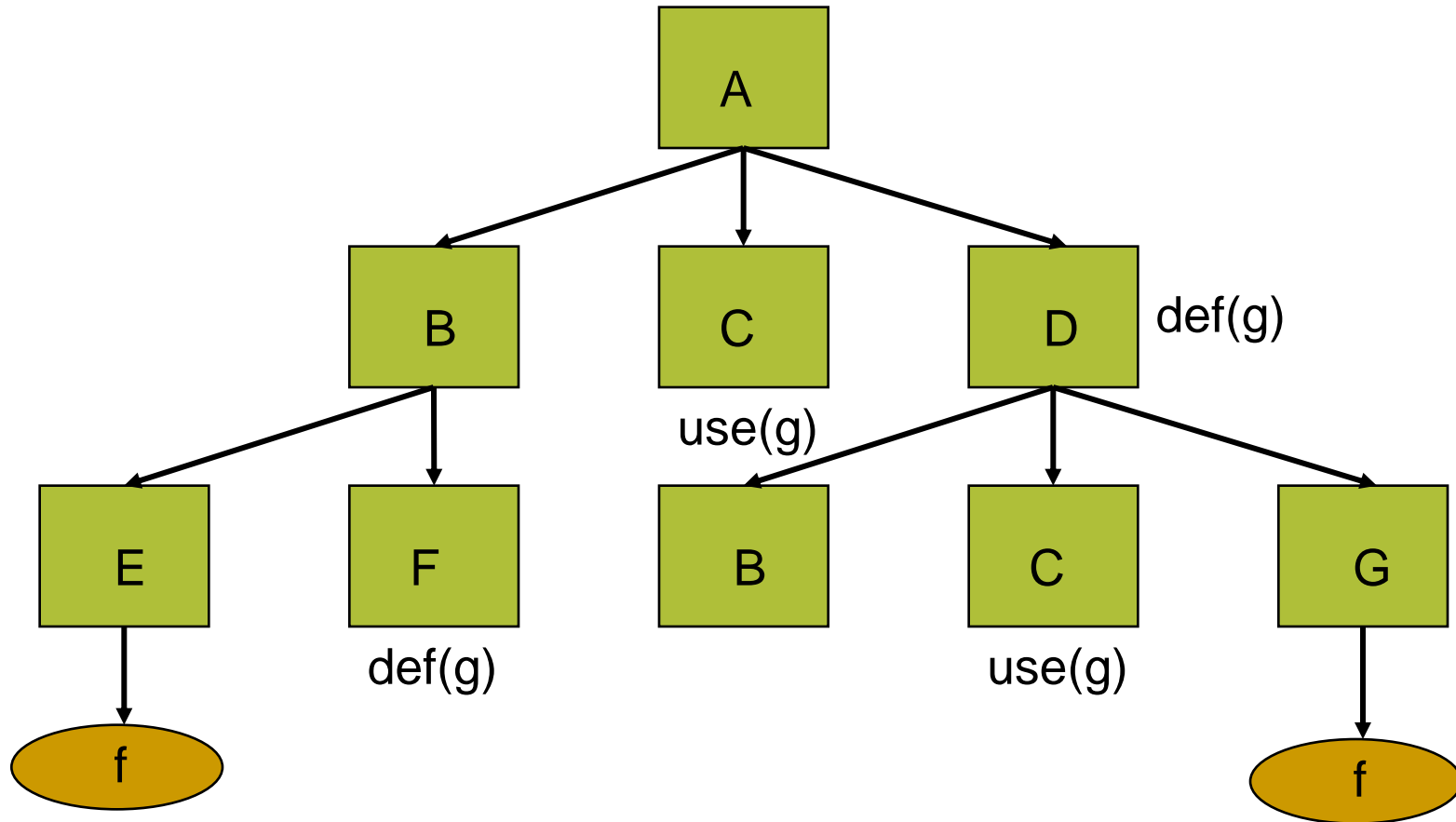
Coupling Graphs

- **M** is a finite set of nodes representing **units** in the program. Each node is depicted by a rectangle.
 - **F** is a finite set of nodes representing **external files** that a unit may write to or read from. It is represented by a circle.
-

Coupling Graphs

- E is finite set of **directed edges** that connect nodes in M and connect nodes in M to nodes in F . Edges between nodes in M are referred to as **call edges**. Edges between nodes in M and nodes in F are referred to as **shared device edges**.
- A is a set of **annotations** on nodes that reference nonlocal or global data.

An Example



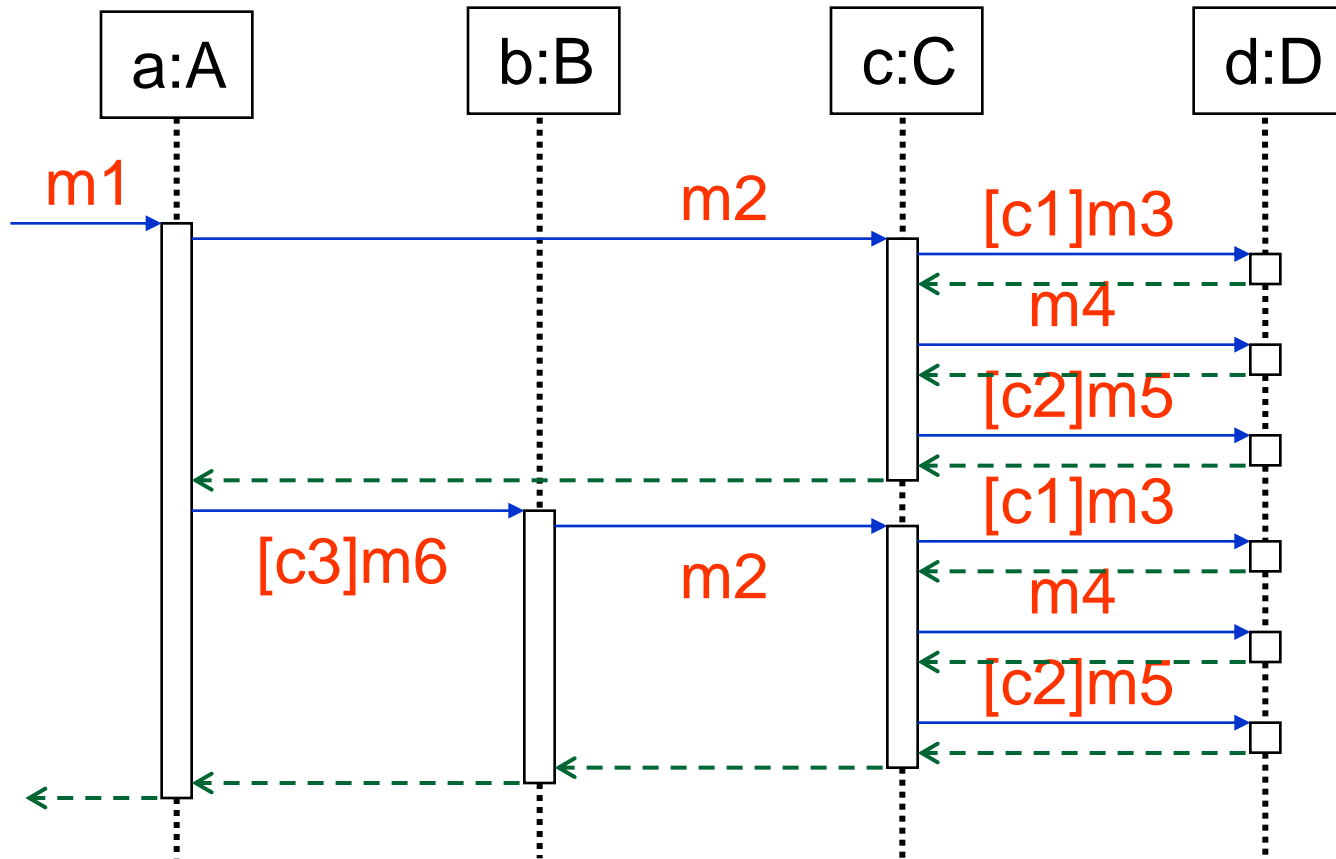
Coverage Measurement

- E: number of edges in coupling graph
 E_{covered} / E
 - CD = number of coupling defs
 CD_{covered} / CD
 - CU = number of coupling def-use pairs
 CU_{covered} / CU
 - CP = number of coupling path sets
 CP_{covered} / CP
-

Sequence Diagrams

- An **interaction** is a set of messages that are exchanged among several objects.
 - A **sequence diagram** displays an interaction as a two-dimensional chart.
 - The **vertical** dimension is the **time** axis; time proceeds down the page.
 - The **horizontal** dimension shows the **objects** in the interaction.
-

An Example



Lifelines of Objects

- Each object is represented by a vertical column containing a head symbol and a **vertical line**—a lifeline.
 - During the time an object exists, it is shown by a **dashed line**.
 - During the time a method of the object is active, it is drawn as a **double line**.
-

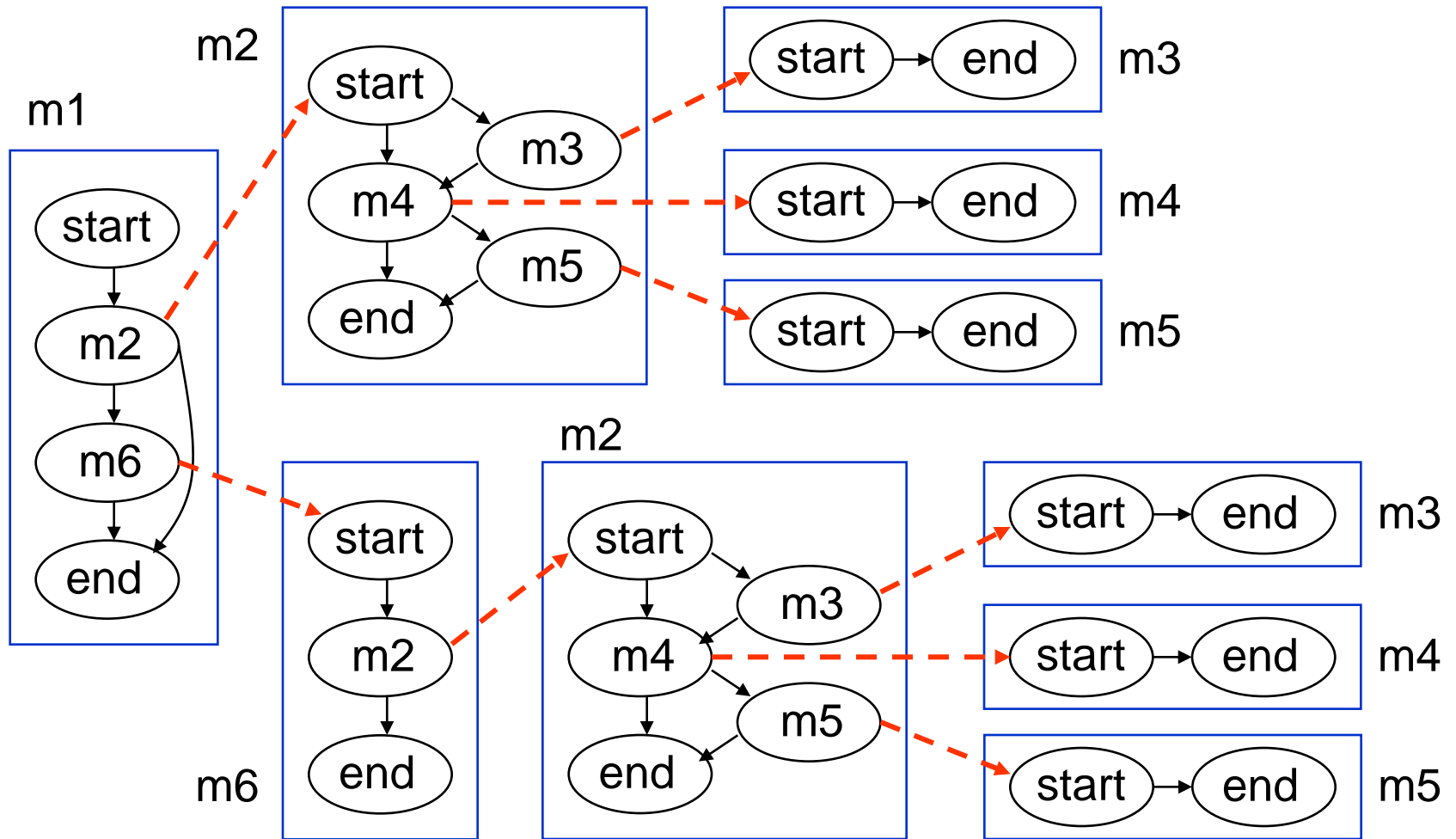
Messages

- A message (or a call) is shown as an **arrow**, with a **filled triangle arrowhead**, from the lifeline of the calling object to the lifeline of the called object.
 - The return of a call is shown by a **dashed arrow** with a **stick arrowhead**.
 - A message may contain a **guard**. The message is sent only when the condition in the guard is true.
-

Interprocedural Restricted Control-Flow Graph

- An **Interprocedural Restricted Control Flow Graph** (IRCFG) contains a set of **Restricted Control Flow Graphs** (RCFGs), together with edges connecting these RCFGs.
 - Each RCFG corresponding to a particular method and is similar to the CFG for that method, except that it is restricted to the flow of control that is relevant to **message** sending.
-

An Example



Testing Coverage Criteria

- **All-IRCFG-Paths Coverage**: requires coverage of the entire set of complete IRCGF paths.
- Each complete path is a start-to-end traversal of the IRCFG.
- An example,
($start_{m_1}, m_2, start_{m_2}, m_4, start_{m_4}, end_{m_4}, m_5,$
 $start_{m_5}, end_{m_5}, end_{m_2}, end_{m_1}$).
- The number of paths for the example is **20**.

Testing Coverage Criteria

- **All-RCFG-Paths Coverage**: requires coverage of all complete RCFG paths.
- A complete IRCFG path may cover several RCFG paths.
- $(start_{m_1}, m_2, start_{m_2}, m_4, start_{m_4}, end_{m_4}, m_5, start_{m_5}, end_{m_5}, end_{m_2}, end_{m_1})$ covers $(start_{m_1}, m_2, end_{m_1})$, $(start_{m_2}, m_4, m_5, end_{m_2})$, $(start_{m_4}, end_{m_4})$, $(start_{m_5}, end_{m_5})$.
- The number of paths for the example is **5**.

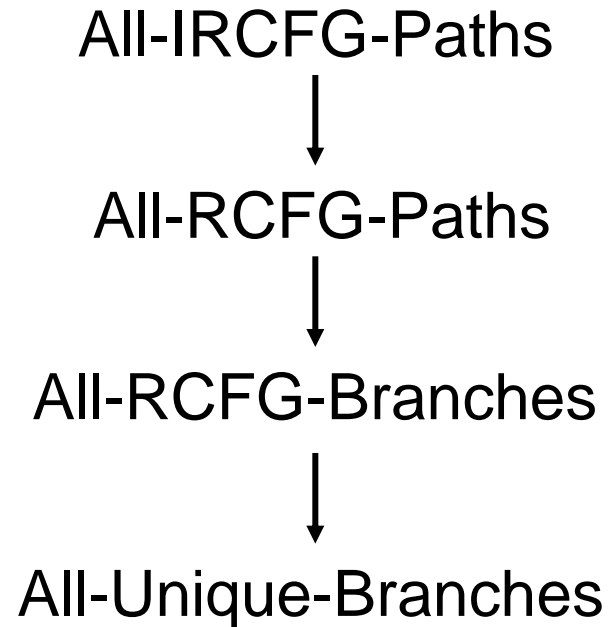
Testing Coverage Criteria

- **All-RCFG-Branched Coverage:** requires coverage of all RCFG edges.
- $(\text{start}_{m_1}, m_2, \text{start}_{m_2}, m_4, \text{start}_{m_4}, \text{end}_{m_4}, \text{end}_{m_2}, m_6, \text{start}_{m_6}, m_2, \text{start}_{m_2}, m_3, \text{start}_{m_3}, \text{end}_{m_3}, m_4, \text{start}_{m_4}, \text{end}_{m_4}, m_5, \text{start}_{m_5}, \text{end}_{m_5}, \text{end}_{m_2}, \text{end}_{m_6}, \text{end}_{m_1}),$
 $(\text{start}_{m_1}, m_2, \text{start}_{m_2}, m_4, \text{start}_{m_4}, \text{end}_{m_4}, \text{end}_{m_2}, m_6, \text{start}_{m_6}, m_2, \text{start}_{m_2}, m_4, \text{start}_{m_4}, \text{end}_{m_4}, \text{end}_{m_2}, \text{end}_{m_6}, \text{end}_{m_1}),$
 $(\text{start}_{m_1}, m_2, \text{start}_{m_2}, m_3, \text{start}_{m_3}, \text{end}_{m_3}, m_4, \text{start}_{m_4}, \text{end}_{m_4}, m_5, \text{start}_{m_5}, \text{end}_{m_5}, \text{end}_{m_2}, \text{end}_{m_1}).$
- The number of paths for the example is 3.

Testing Coverage Criteria

- **All-Unique-Branches Coverage**: requires coverage of all RCGF edges regardless of the calling context.
- $(\text{start}_{m_1}, m_2, \text{start}_{m_2}, m_4, \text{start}_{m_4}, \text{end}_{m_4}, \text{end}_{m_2}, m_6, \text{start}_{m_6}, m_2, \text{start}_{m_2}, m_3, \text{start}_{m_3}, \text{end}_{m_3}, m_4, \text{start}_{m_4}, \text{end}_{m_4}, m_5, \text{start}_{m_5}, \text{end}_{m_5}, \text{end}_{m_2}, \text{end}_{m_6}, \text{end}_{m_1})$,
 $(\text{start}_{m_1}, m_2, \text{start}_{m_2}, m_4, \text{start}_{m_4}, \text{end}_{m_4}, \text{end}_{m_2}, \text{end}_{m_1})$.
- The number of paths for the example is **2**.

Testing Coverage Criteria Hierarchy



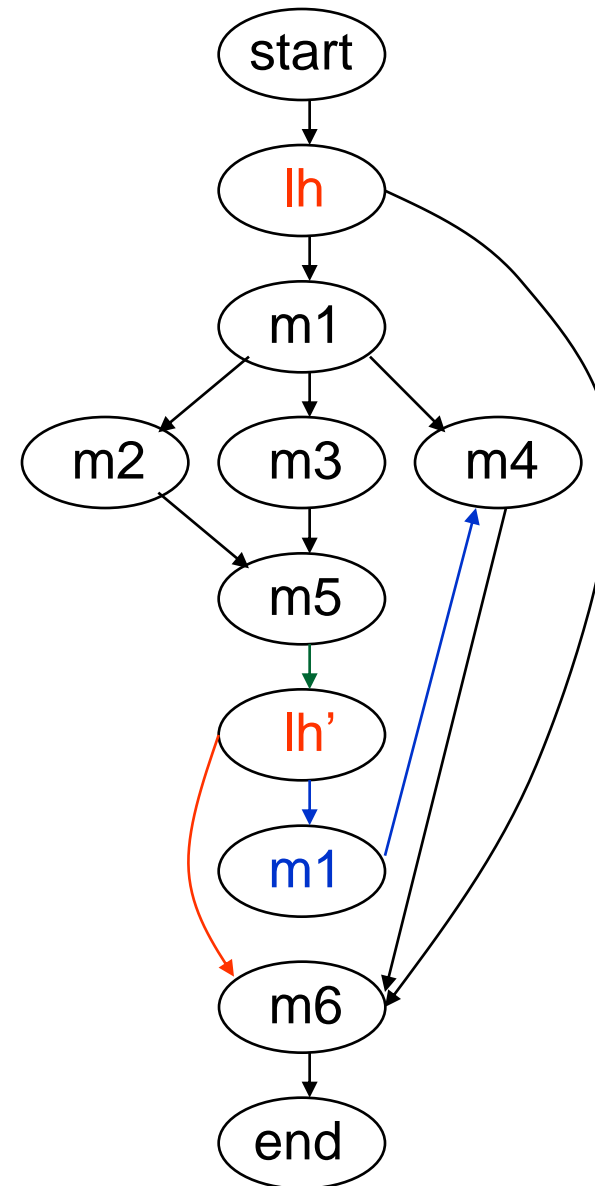
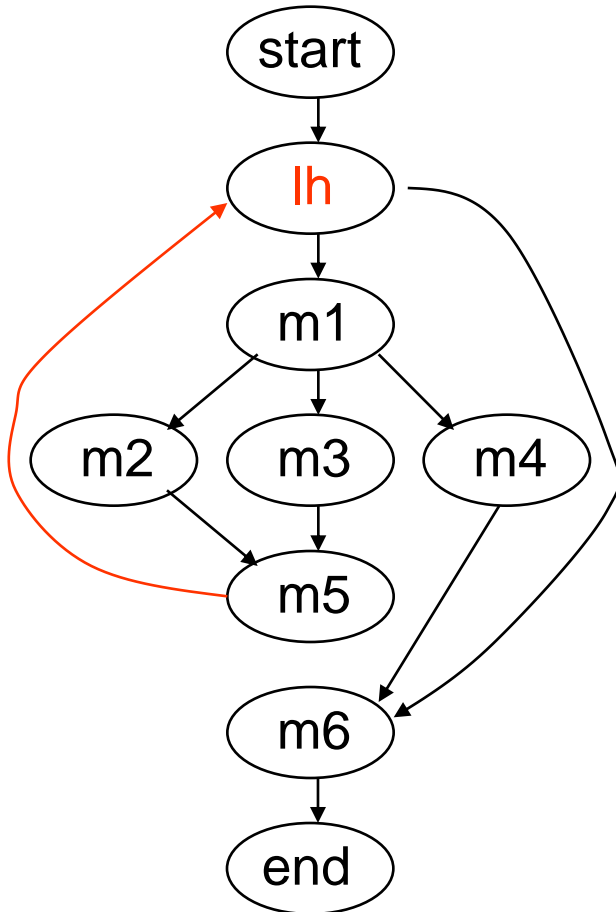
Handling Loops

- A path that completely bypasses the loop.
 - A path that iterates the loop some number of times and normally exits the loop.
 - A path that iterates the loop some number of times and takes one of the early exits of the loop.
-

Loop Transformation

- Assume that each loop contains an artificial loophead node.
 - Create a second loophead node for each loop which is directed to the next node of the loop.
 - Each back edge is redirected to the new loophead node.
 - The subpaths through the early exits are replicated.
-

An Example



(start, lh, m1, m2, m5,
lh, m1, m3, m5,
lh, m1, m4, m6, end).

(start, lh, m1, m2, m5, lh', m1, m4, m6, end),
(start, lh, m1, m3, m5, lh', m1, m4, m6, end).