Proposal of Execution Paths Indication Method for Integration Testing by Using an Automatic Visualization Tool 'Avis'

Yoshihiro Kita, Tetsuro Katayama, and Shigeyuki Tomita

University of Miyazaki, Japan

5th World Congress for Software Quality - Shanghai, China

Agenda

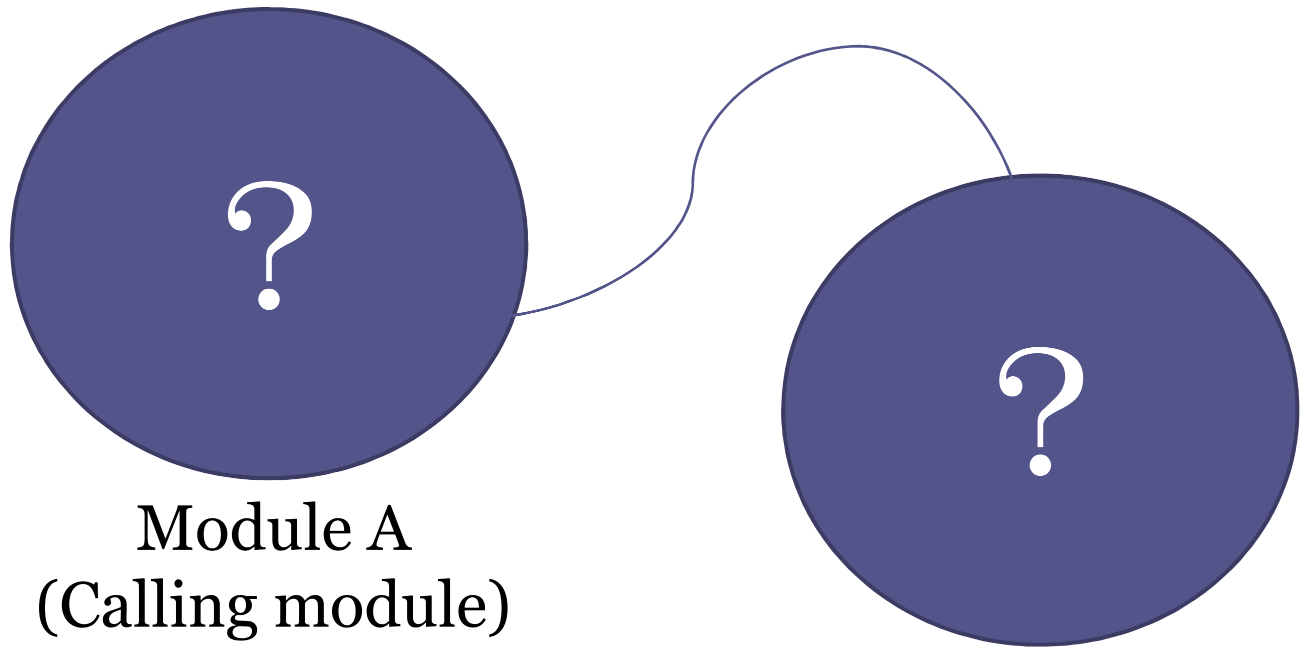
1. Background
2. Goal & Approach
3. Avis
4. Evaluation of Proposal method
5. Discussion
6. Conclusion & Future works

Background

- Integration Testing
  - A part of software testing process
  - Verification targets
    - Completeness of functionality
    - Data manipulation
    - Inter-module interfaces

Background

- Verification of inter-module interfaces by black-box testing.

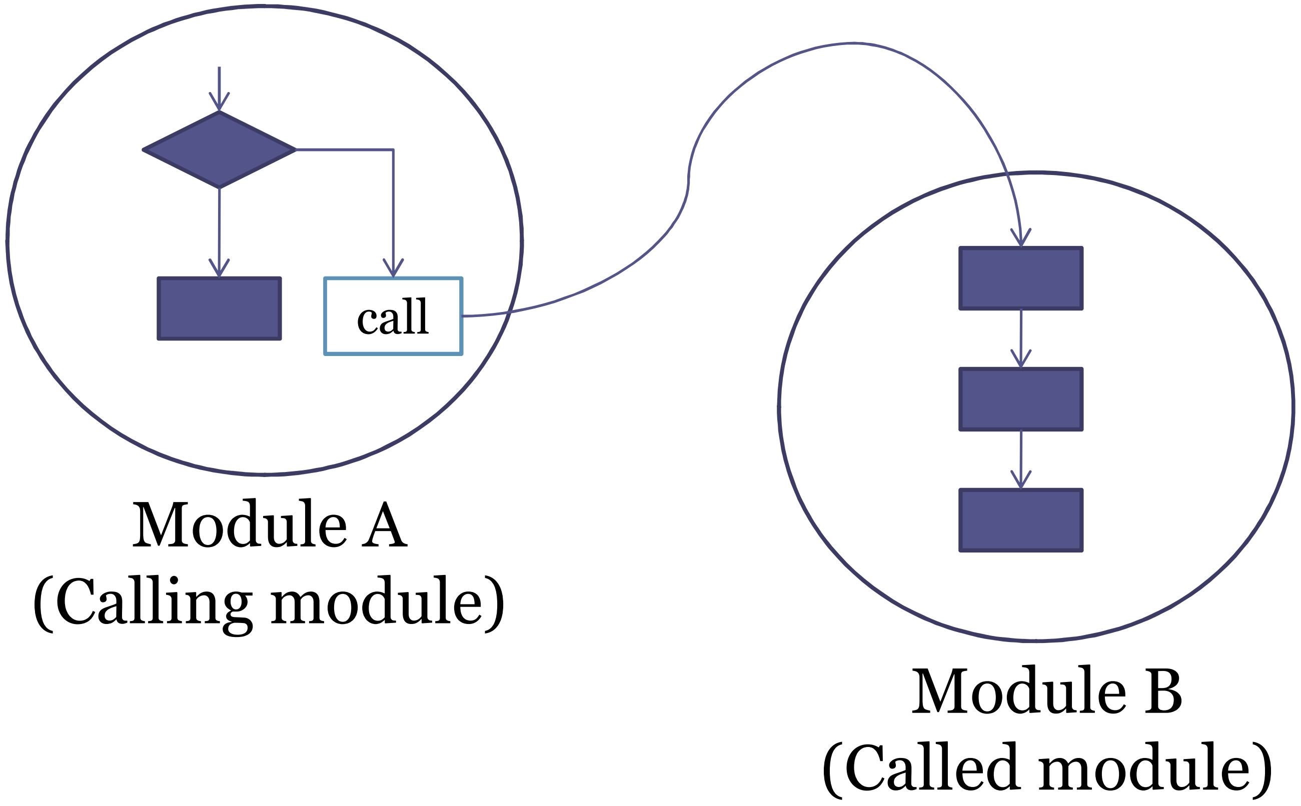


Module A
(Calling module)

Module B
(Called module)

Background

- Verification of inter-module interfaces by white-box testing.

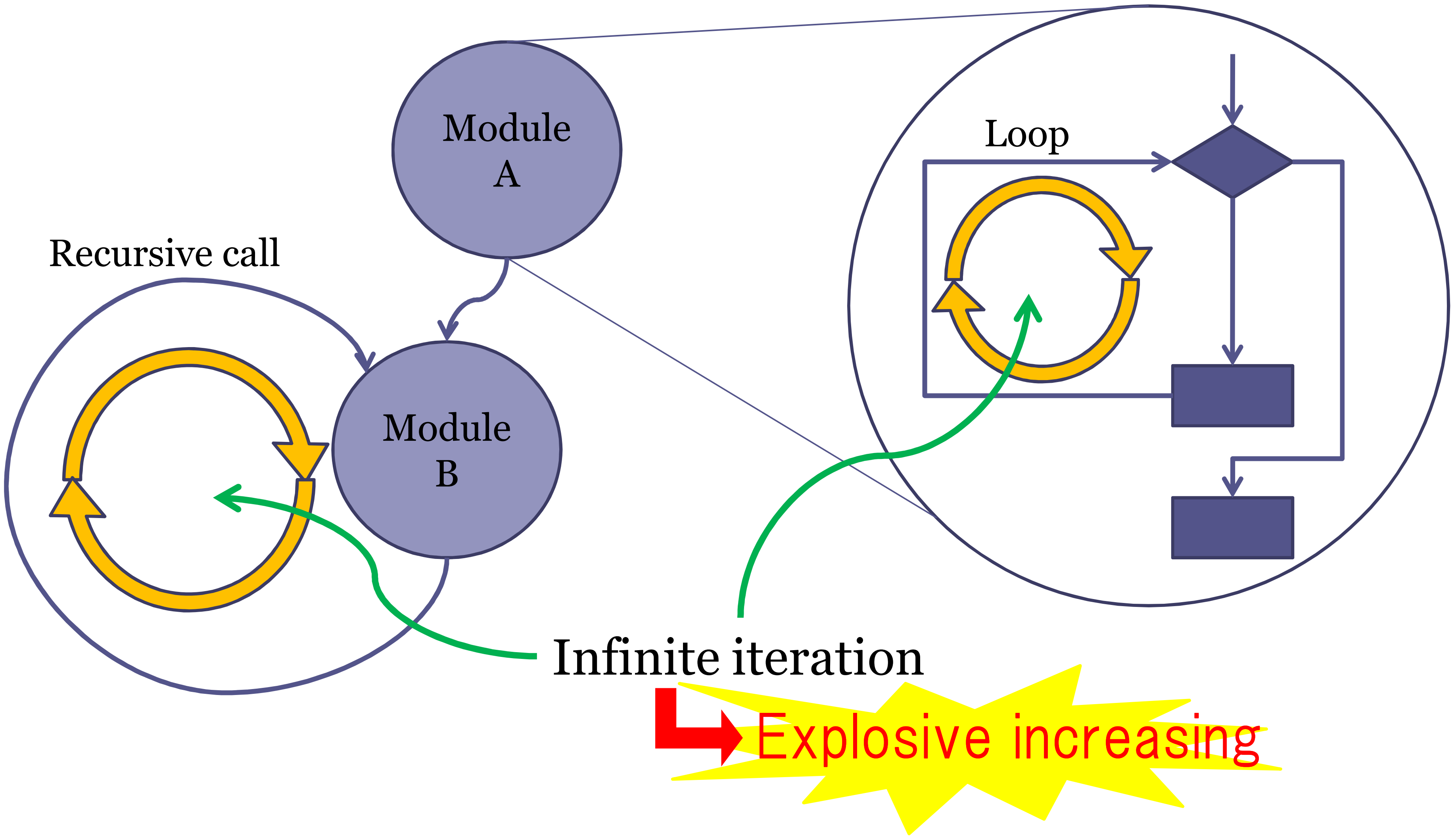


Background

- A problem of integration testing by white-box testing
  - Explosive increasing number of execution paths for covering all modules and call-pairs.

It is impossible to execute all paths.

Explosive increasing number of execution paths



Goal & Approach

- Goal

  Verification support of inter-module interfaces by white-box testing in integration testing

- Approach

  Indicate the minimum set of execution paths by using automatic visualization tool ‘Avis’.

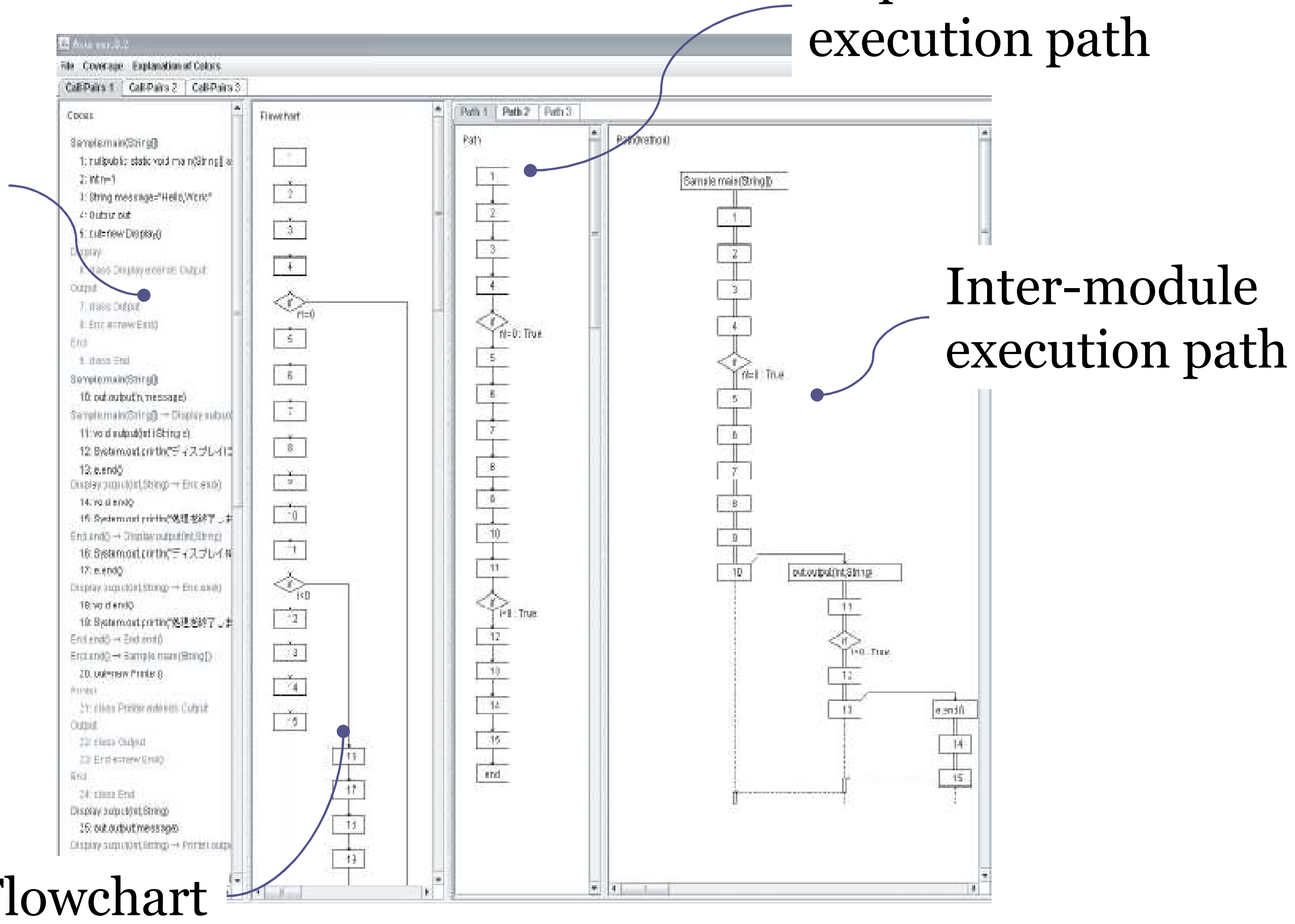
Avis

- Automatic Visualization Tool for Programs

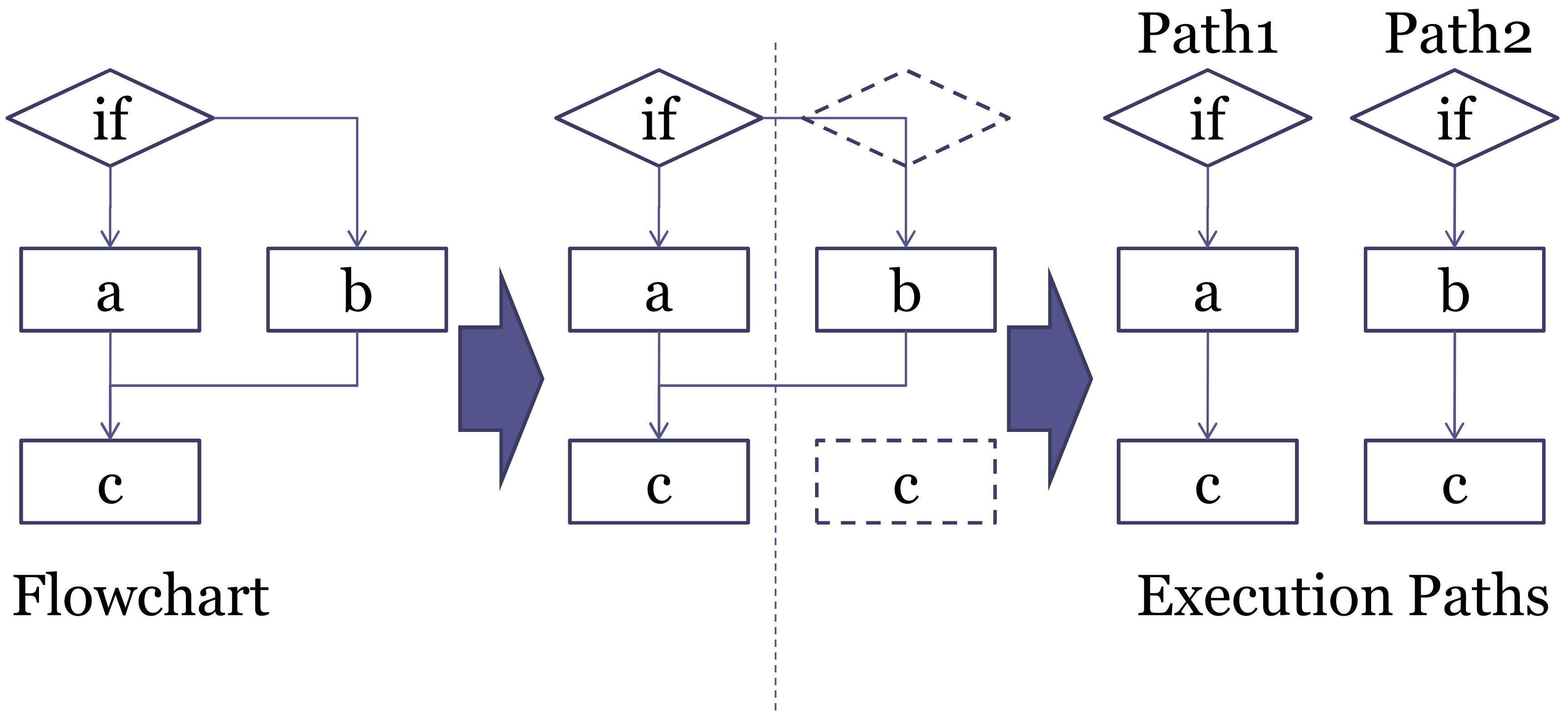
- Input
  - Source code of Java program

- Outputs
  - Flowchart
  - Sequential execution paths
  - Inter-module execution paths

View of Avis


Sequential execution path
Code
Inter-module execution path
Flowchart
File Coverage Explanation of Colors
Call-Pairs 1
Call-Pairs 2
Call-Pairs 3
Codes
Flowchart
Path 1
Path 2
Path 3
Path
Path(method)
Sample.main(String[])
out.output(int,String)
e.end()
end

Paths Generation

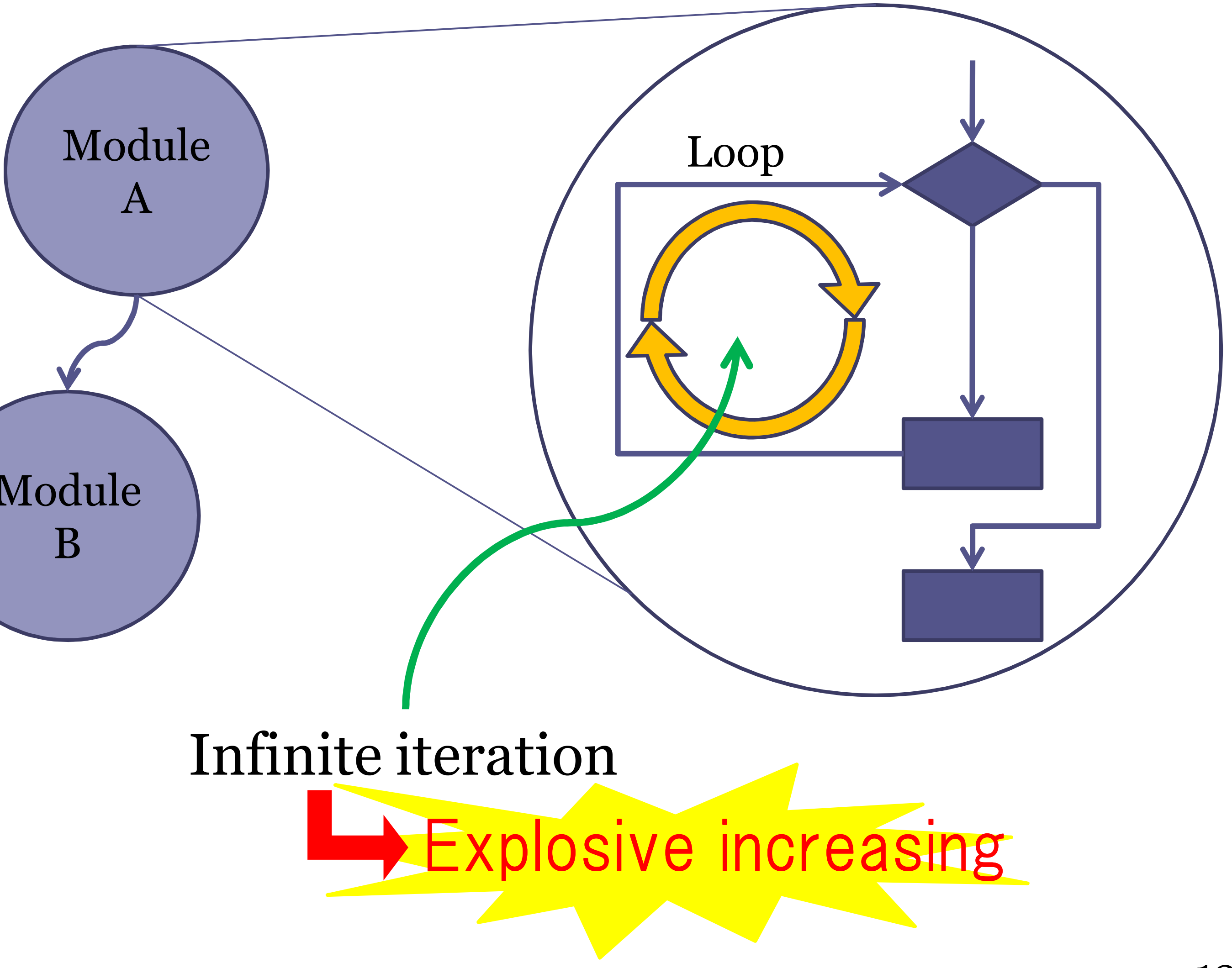


Separation

Solution of explosive increasing number of paths

- Criteria for generating execution paths
- Criteria for integration testing
- Algorithm for abstracting execution paths

Explosive increasing number of execution paths

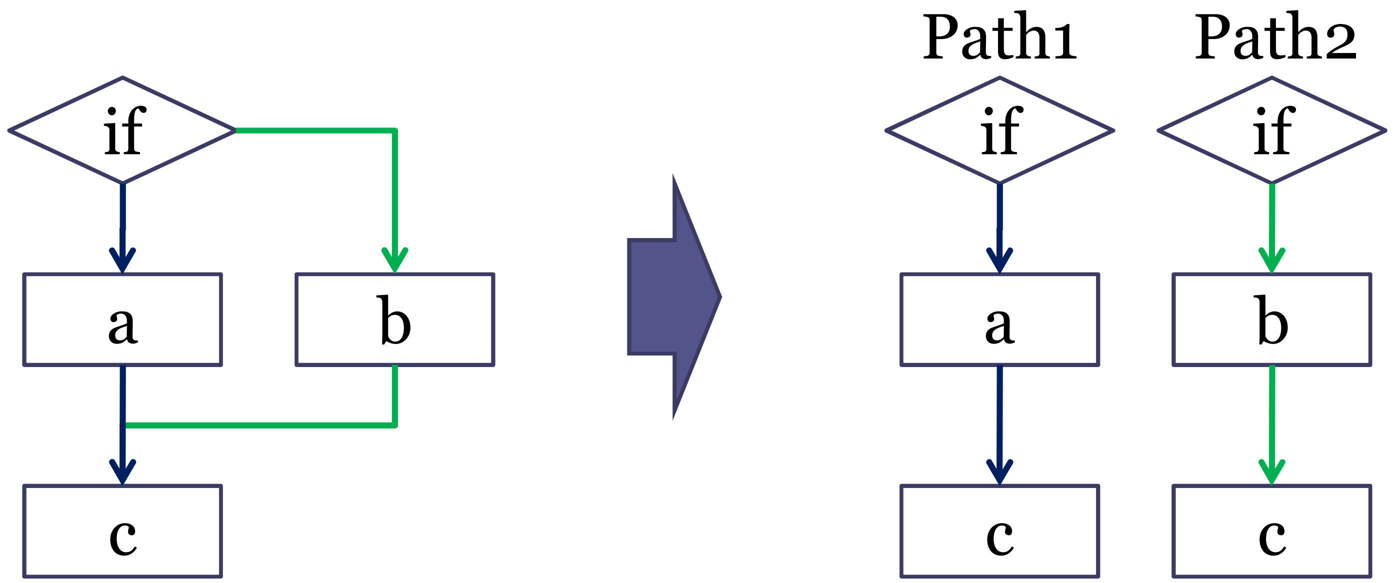


Criteria for generating execution paths

- Avis's criteria for reducing the number of paths based on branch coverage (C1)
  - Criterion for branch covering
  - Criterion for loop covering

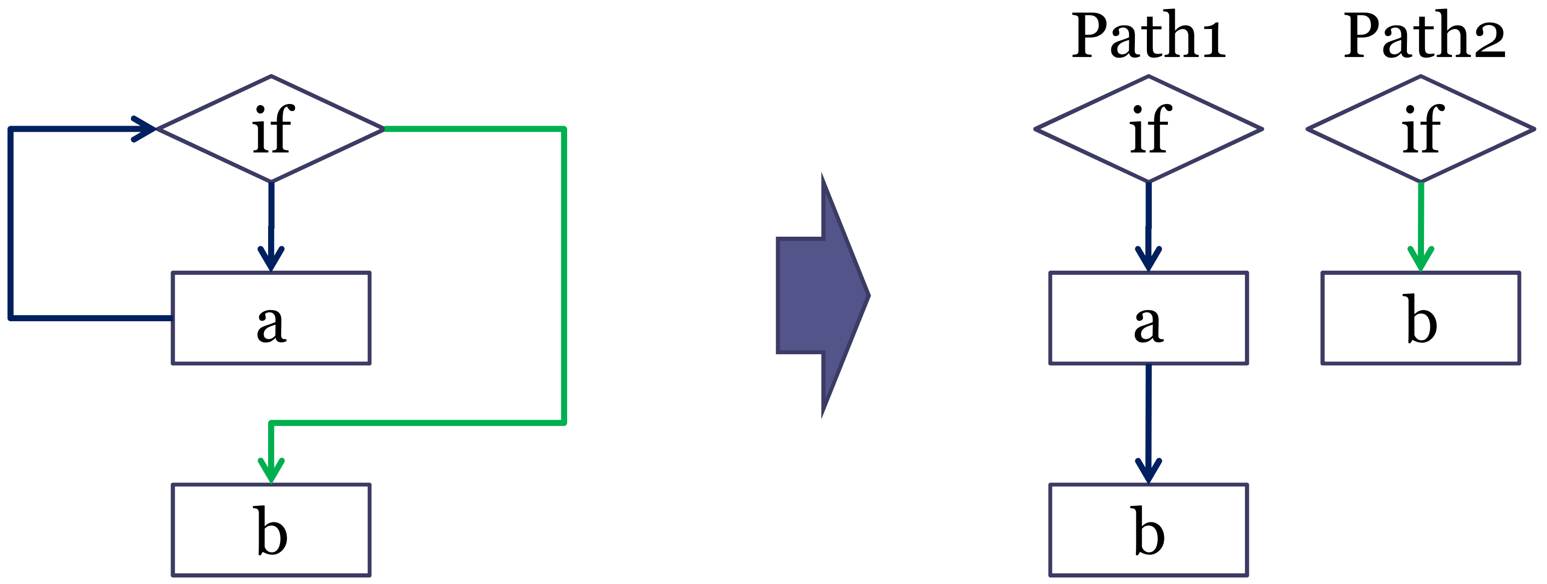
Criterion for branch covering

- All branches are executed at least once.



Criterion for loop covering

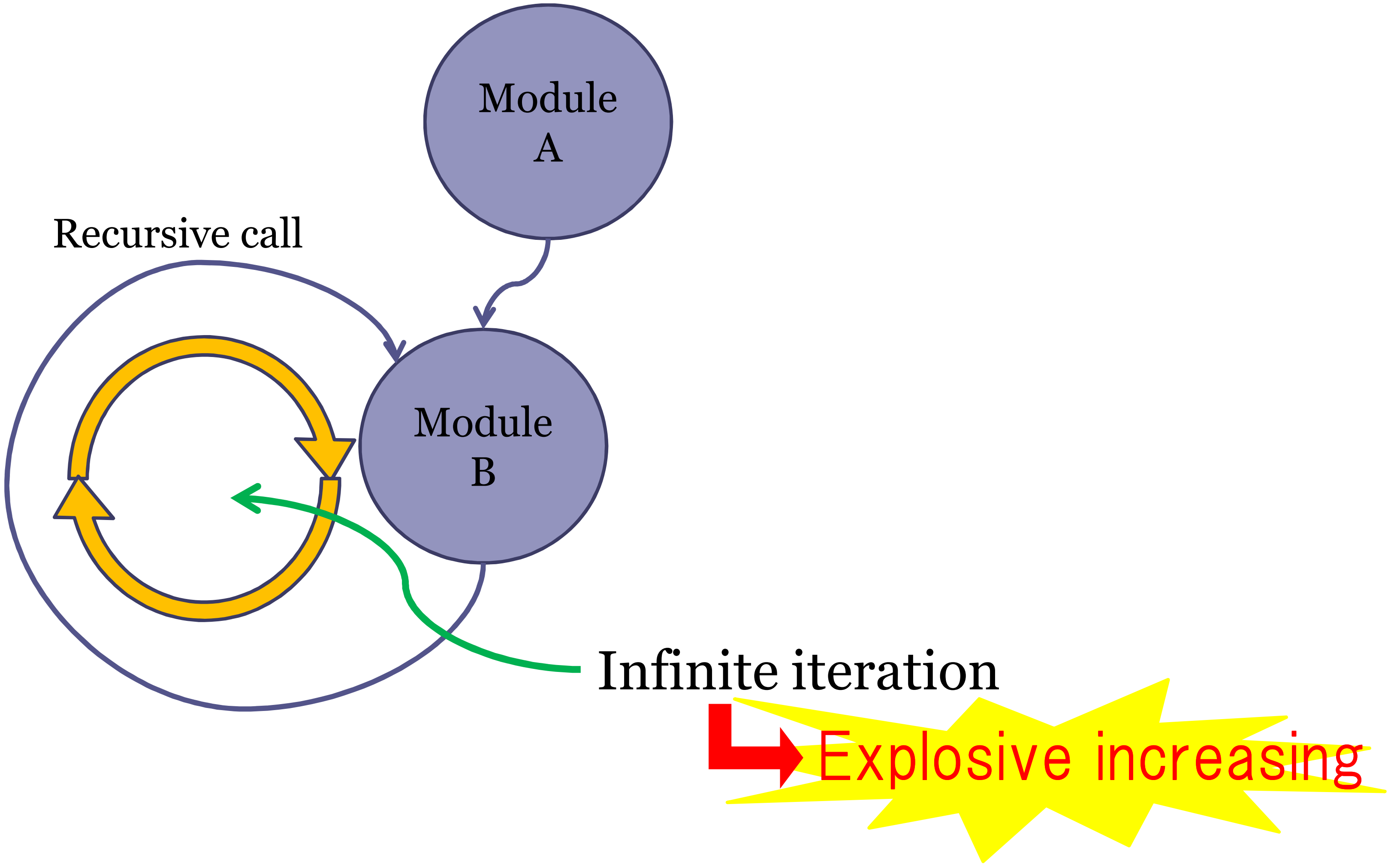
- All loops are iterated zero times (i.e. no iteration) and once.



Solution of explosive increasing number of paths

- Criteria for generating execution paths
- Criteria for integration testing
- Algorithm for abstracting execution paths

Explosive increasing number of execution paths

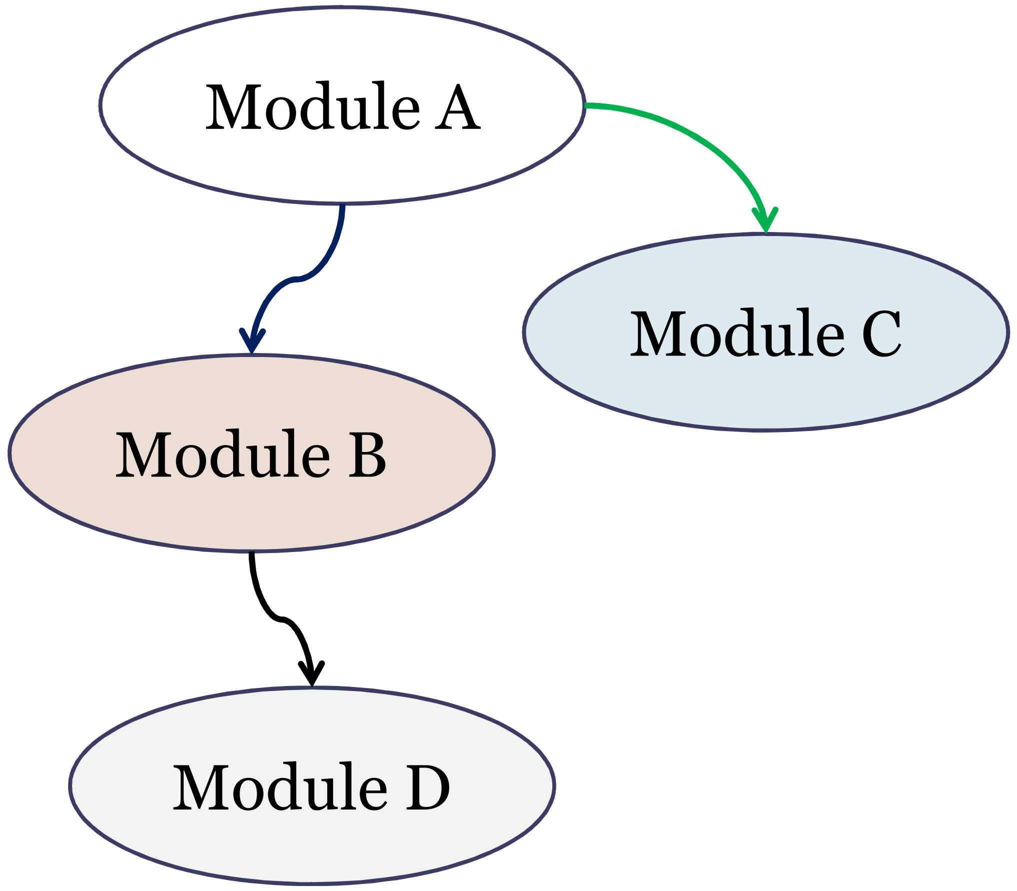


Criteria for integration testing

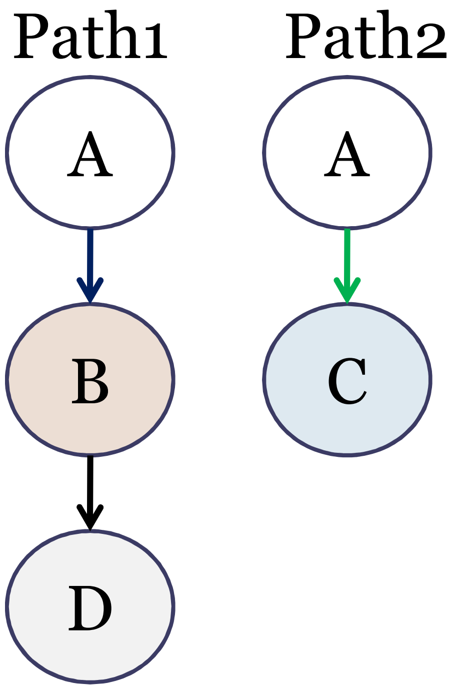
- Avis's new criteria for reducing the number of paths based on Module coverage(S0) and Call-pair coverage(S1) in integration testing.
  - Criterion for module covering
  - Criterion for recursive call covering

Criterion for module covering

- All modules are executed at least once.

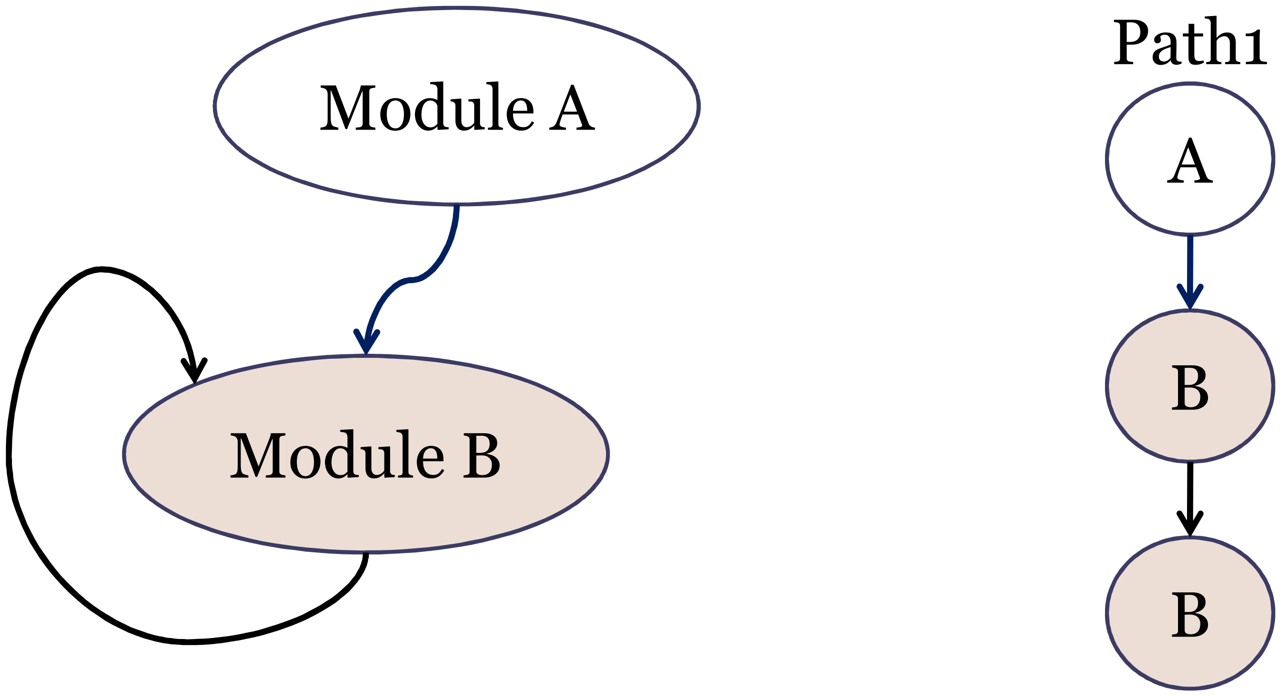


$$\begin{tikzcd}
A \arrow[d] & A \arrow[d] \\
B \arrow[d] & C \\
D &
\end{tikzcd}$$



Criterion for recursive call covering

- All recursive calls are iterated once.



Solution of explosive increasing number of paths

- Criteria for generating execution paths
- Criteria for integration testing
- Algorithm for abstracting execution paths

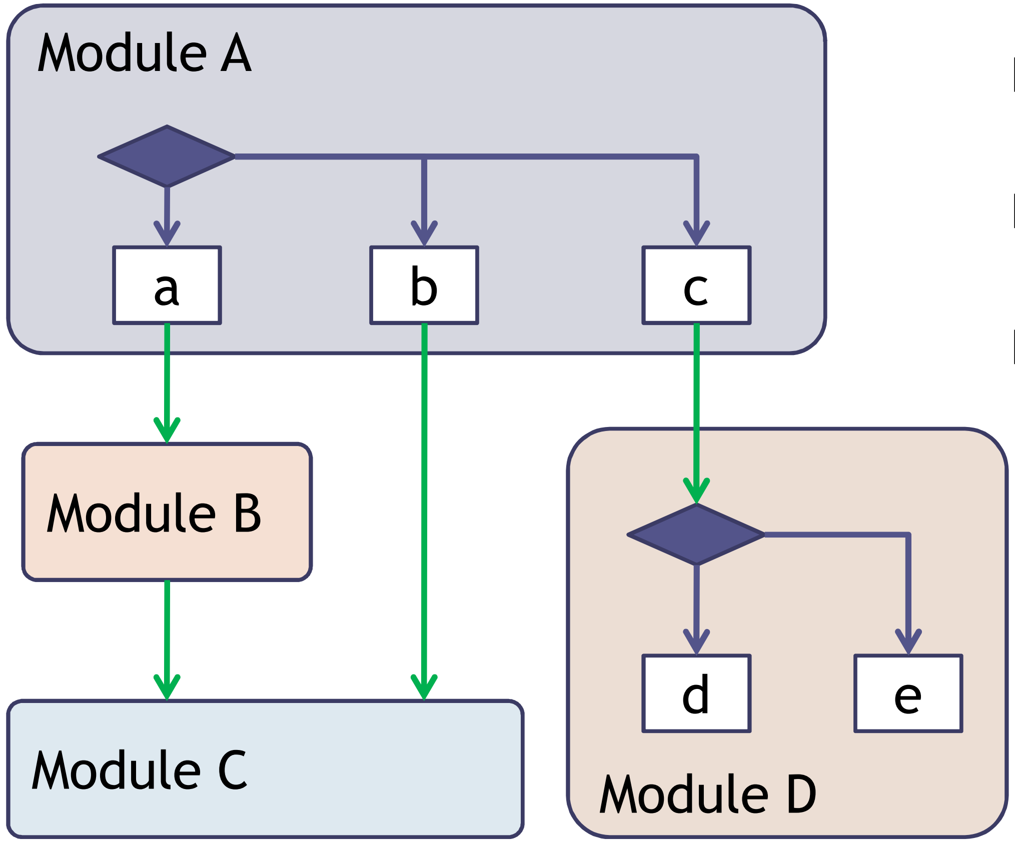
Useless of execution paths

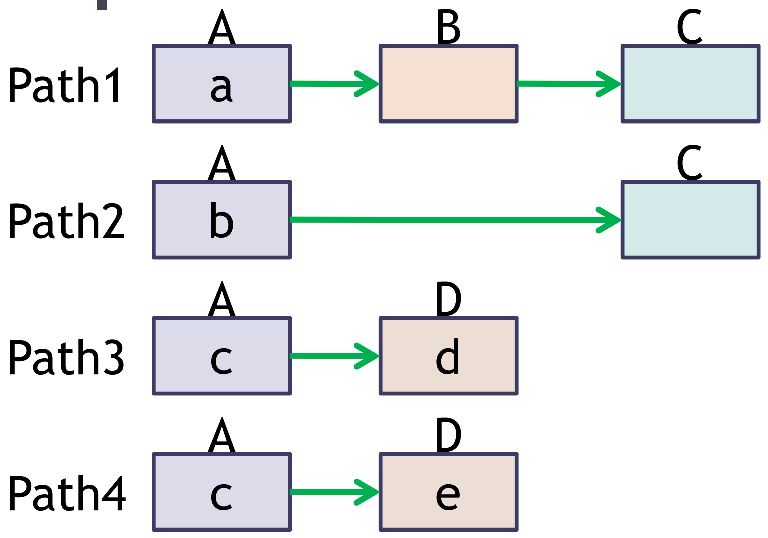
- The execution paths are generated by Avis include useless paths for indicating

⇩

Abstraction of the useful execution paths

abstracting execution paths





Path1, Path2 and Path3
are abstracted paths
for covering all call-pairs.

Path1 and Path3 are
abstracted paths
for covering all modules.

Practicality of execution paths

- Infeasible paths may be generated in the following cases.
  - Paths disregard the dependence of branch conditions.
  - Paths include the code that it is impossible to execute (called ‘dead-code’).

Evaluation of proposal method

- Practicality of Avis
  - Runtime of Avis
- Abstraction of execution paths
  - Comparison between the number of modules (or call-pairs) and the number of paths generated by Avis
- Rate of executable paths
  - Number of executable paths among execution paths
- Usefulness of indicating infeasible paths
  - Number of dead-code included in infeasible paths

Practicality of Avis

- Runtime of Avis

Java Program	Lines of code	Runtime(sec)
Program A: Calendar	80	0.21
Program B: Simulation of bounding balls	155	0.36
Program C: Card game (Black jack)	722	1.22
Program D: Parser (A part of Avis)	8,034	9.73

Abstraction of execution paths

- Comparison between the number of modules and the number of execution path satisfying S0

Java Program	Number of modules	Number of execution paths satisfying S0	Reduction ratio(%)
Program A	7	3	57.1
Program B	23	14	39.1
Program C	76	38	50.0
Program D	573	352	38.6

Abstraction of execution paths

- Comparison between the number of call-pairs and the number of execution path satisfying S1

Java Program	Number of call-pairs	Number of execution paths satisfying S1	Reduction ratio(%)
Program A	17	3	82.4
Program B	66	16	75.8
Program C	259	45	82.6
Program D	2,277	372	83.7

Rate of executable paths

- Number of executable paths among execution paths satisfying So

Java Program	Num. of paths	Num. of executable paths	Num. of covering modules
Program A	3	3 (100.0%)	7 (100.0%)
Program B	14	13 (92.0%)	22 (95.7%)
Program C	38	27 (71.1%)	61 (80.3%)
Program D	352	217 (61.6%)	409 (71.4%)

Rate of executable paths

- Number of executable paths among execution paths satisfying S1

Java Program	Num. of paths	Num. of executable paths	Num. of covering call-pairs
Program A	3	3 (100.0%)	7 (100.0%)
Program B	16	14 (87.5%)	63 (95.5%)
Program C	45	29 (64.4%)	217 (83.8%)
Program D	372	223 (59.9%)	1,558 (68.4%)

Usefulness of indicating infeasible paths

- Number of dead-code included in infeasible paths

Java Program	Num. of infeasible paths(S0)	Num. of dead-codes	Num. of infeasible paths(S1)	Num. of dead-codes
Program A	0	-	0	-
Program B	1	0	2	0
Program C	11	0	16	0
Program D	135	7	149	9

Discussion

- Result of evaluation
  - Reducing the number of execution paths by proposed criteria.
  - Minimum set of execution paths can cover all modules and call-pairs.
  - Infeasible paths include dead-code.

Conclusion

- Goal

  Verification support of inter-module interfaces by white-box testing in integration testing

- Proposal

  Indicate the minimum set of execution paths by using automatic visualization tool 'Avis'.

- Out Comes
  - Minimum set of execution paths can cover both all modules and call-pairs.
  - The set of execution paths contain dead-code can be specified.

Future works

- Measures for infeasible paths
- Applications of Avis to software testing education
- Applications of Avis to other programming languages for integration testing

Thank you for your attention!!