Practical Techniques for Improving the Efficiency and Usability of Mutation Analysis for Java Programs

Gregory M. Kapfhammer†

Department of Computer Science
Allegheny College
http://www.cs.allegheny.edu/∼gkapfham/

University of Sheffield – February 3, 2012

† Joint with René Just and Franz Schweiggert (University of Ulm) and Jonathan Miller Kauffman (Allegheny College)

Accessing the Presentation

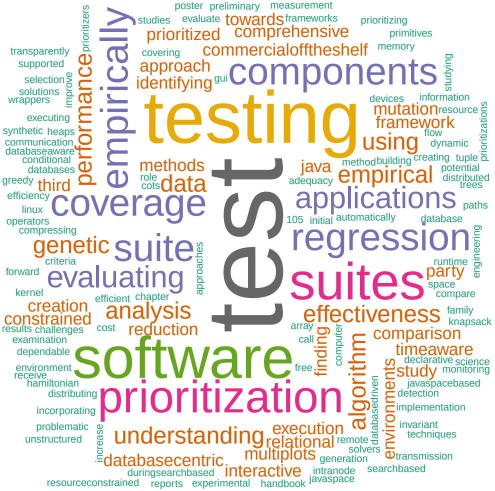
Scan this QR Code with your smartphone!

... or, visit this Web site:

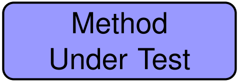
`http://is.gd/rekiwo`

... or, ask me for a USB drive!

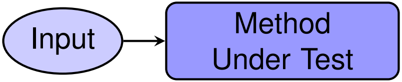
Presenter Introduction: Gregory M. Kapfhammer



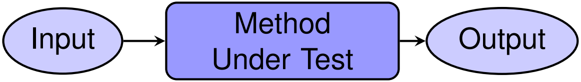
What is a Test Case?



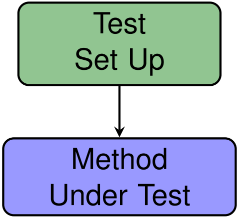
What is a Test Case?



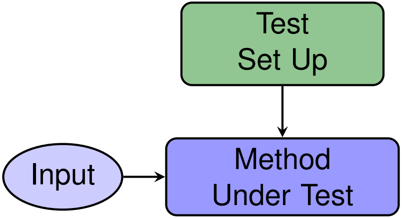
What is a Test Case?



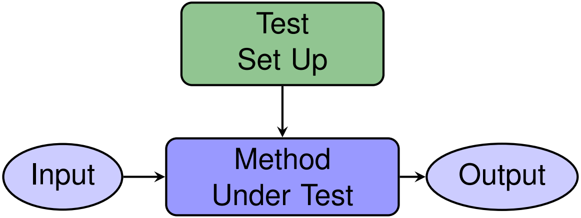
What is a Test Case?



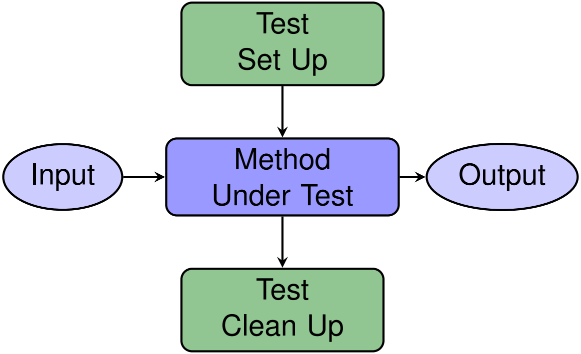
What is a Test Case?



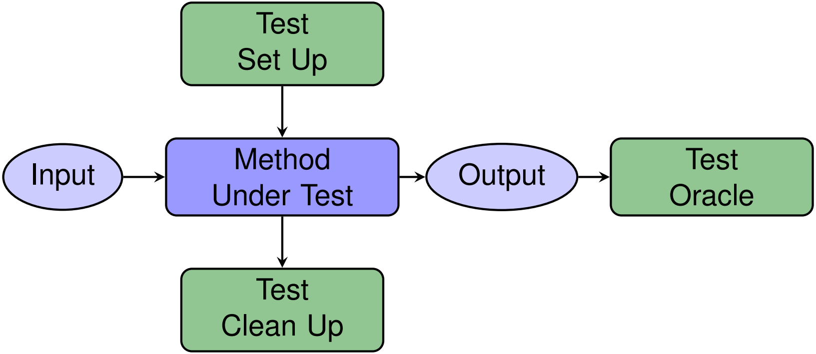
What is a Test Case?



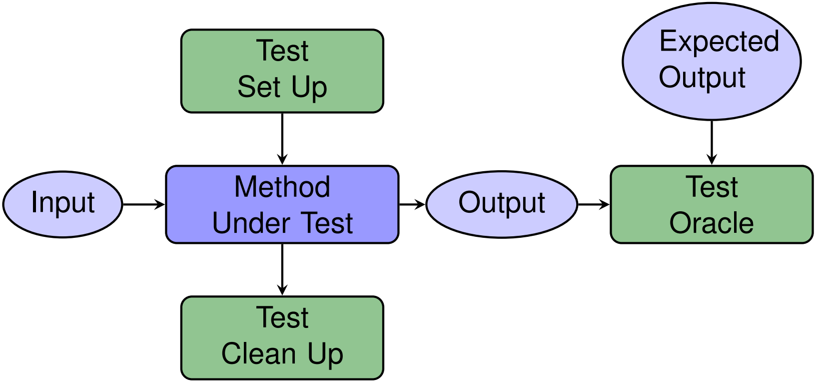
What is a Test Case?



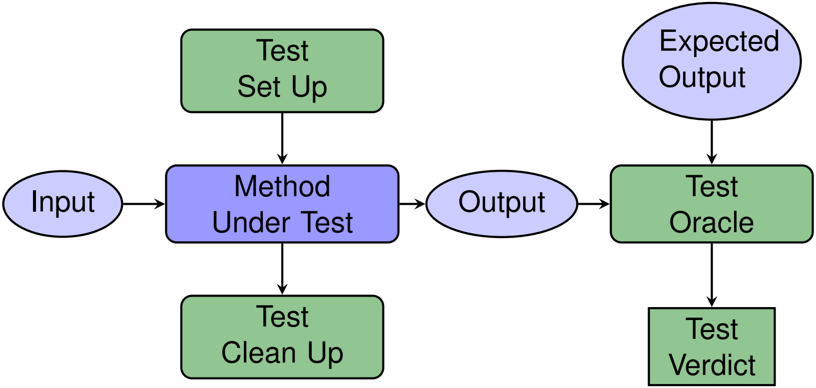
What is a Test Case?



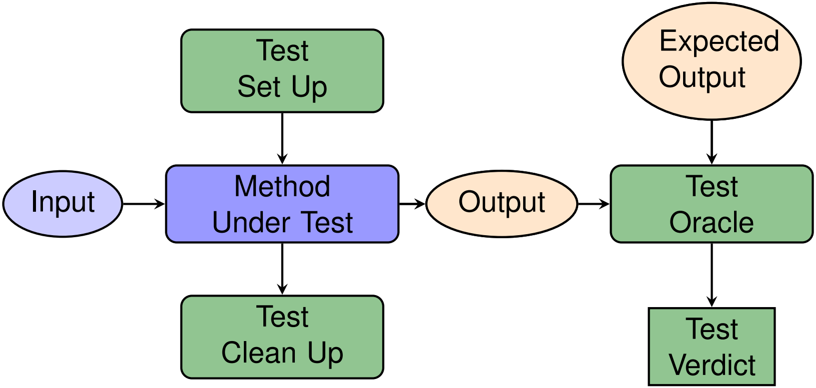
What is a Test Case?



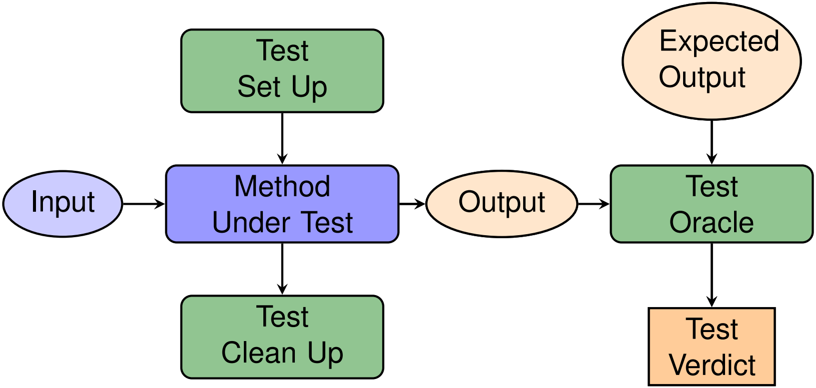
What is a Test Case?



What is a Test Case?

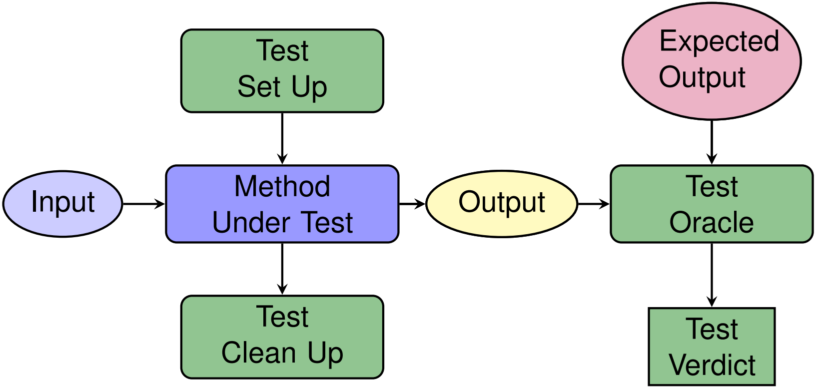


What is a Test Case?

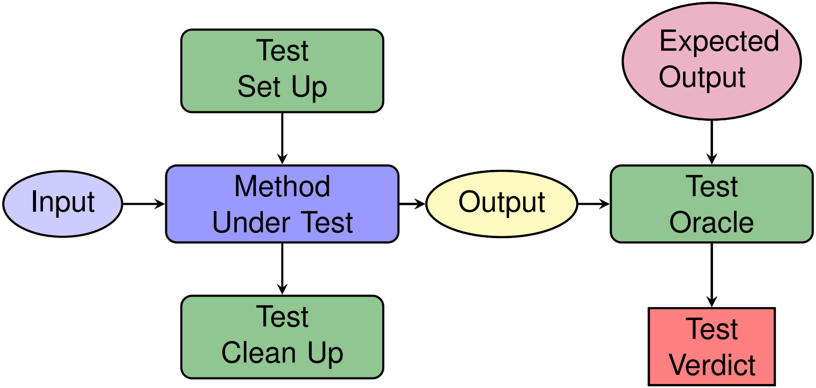


The test case passes and the code is correct!

What is a Test Case?



What is a Test Case?



The test case fails and a defect is found!

What is a Test Suite?

T_1 T_2

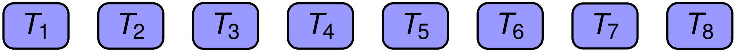
What is a Test Suite?

T_1 T_2 T_3 T_4

What is a Test Suite?

T_1 T_2 T_3 T_4 T_5 T_6

What is a Test Suite?



What is a Test Suite?

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

R_1 R_2

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

R_1 R_2 R_3 R_4

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

R_1 R_2 R_3 R_4 R_5 R_6

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

R_1 R_2 R_3 R_4 R_5 R_6 F_1 F_2

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

R_1 R_2 R_3 R_4 R_5 R_6 F_1 F_2 F_3 F_4

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

R_1 R_2 R_3 R_4 R_5 R_6 F_1 F_2 F_3 F_4 B_1 B_2

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

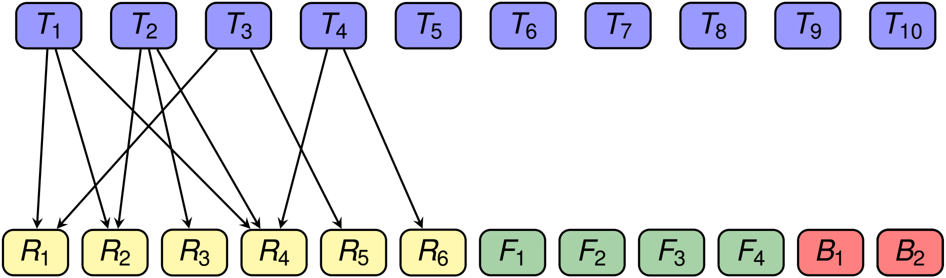
T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

R_1 R_2 R_3 R_4 R_5 R_6 F_1 F_2 F_3 F_4 B_1 B_2

Requirements $R = \{R_1, \ldots, R_6\}$, Features $F = \{F_1, \ldots, F_4\}$, Bug Fixes $B = \{B_1, B_2\}$

What is a Test Suite?

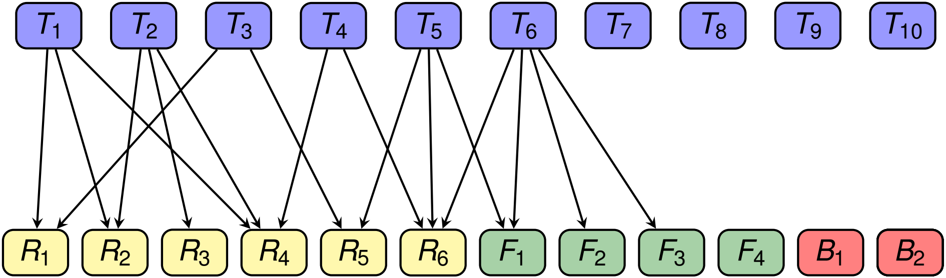
Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$



Requirements $R = \{R_1, \ldots, R_6\}$, Features $F = \{F_1, \ldots, F_4\}$, Bug Fixes $B = \{B_1, B_2\}$

What is a Test Suite?

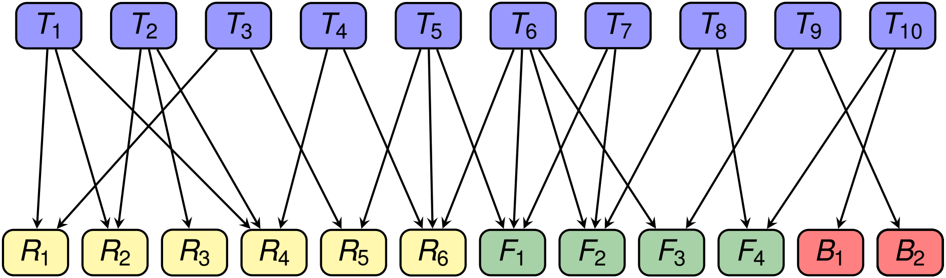
Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$



Requirements $R = \{R_1, \ldots, R_6\}$, Features $F = \{F_1, \ldots, F_4\}$, Bug Fixes $B = \{B_1, B_2\}$

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$



Requirements $R = \{R_1, \ldots, R_6\}$, Features $F = \{F_1, \ldots, F_4\}$, Bug Fixes $B = \{B_1, B_2\}$

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

R_1 R_2 R_3 R_4 R_5 R_6 F_1 F_2 F_3 F_4 B_1 B_2

Requirements $R = \{R_1, \ldots, R_6\}$, Features $F = \{F_1, \ldots, F_4\}$, Bug Fixes $B = \{B_1, B_2\}$

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

How Good is Test Suite T?

R_1 R_2 R_3 R_4 R_5 R_6 F_1 F_2 F_3 F_4 B_1 B_2

Requirements $R = \{R_1, \ldots, R_6\}$, Features $F = \{F_1, \ldots, F_4\}$, Bug Fixes $B = \{B_1, B_2\}$

What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10}

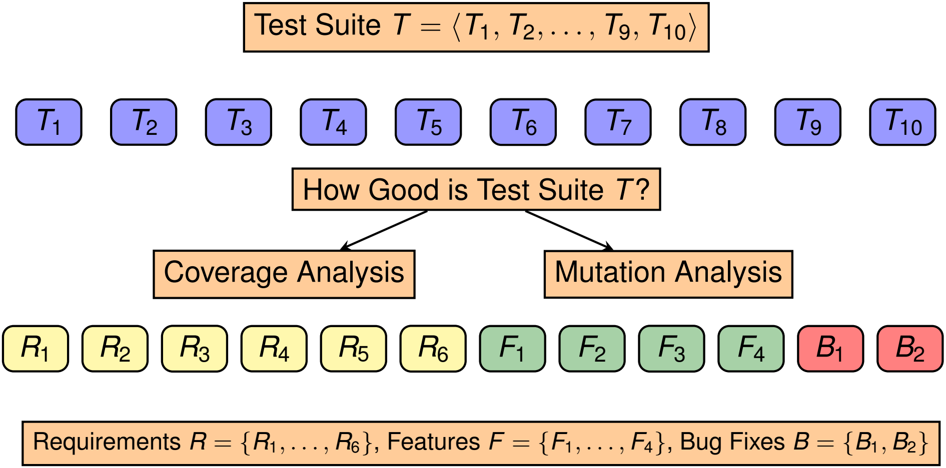
How Good is Test Suite T?

Coverage Analysis

R_1 R_2 R_3 R_4 R_5 R_6 F_1 F_2 F_3 F_4 B_1 B_2

Requirements $R = \{R_1, \ldots, R_6\}$, Features $F = \{F_1, \ldots, F_4\}$, Bug Fixes $B = \{B_1, B_2\}$

What is a Test Suite?



Overview of Mutation Analysis

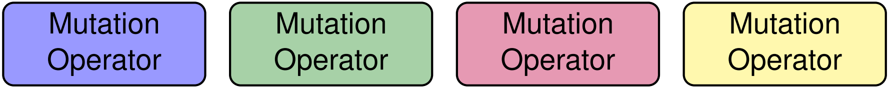
Mutation Operator

Overview of Mutation Analysis

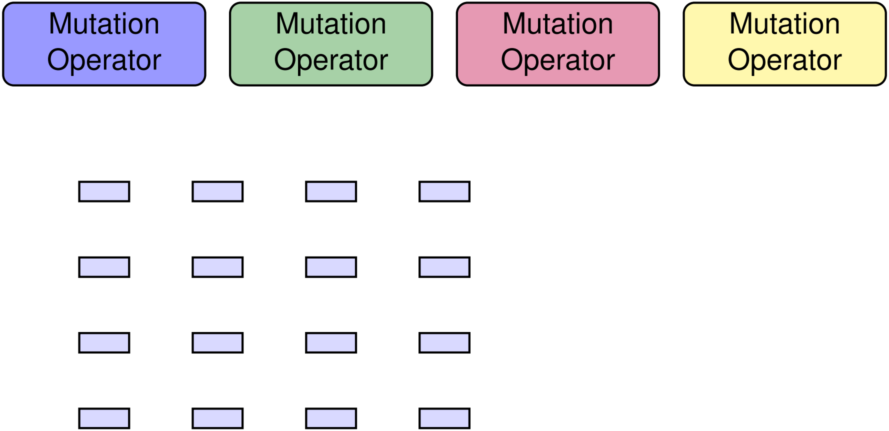
Mutation Operator

Mutation Operator

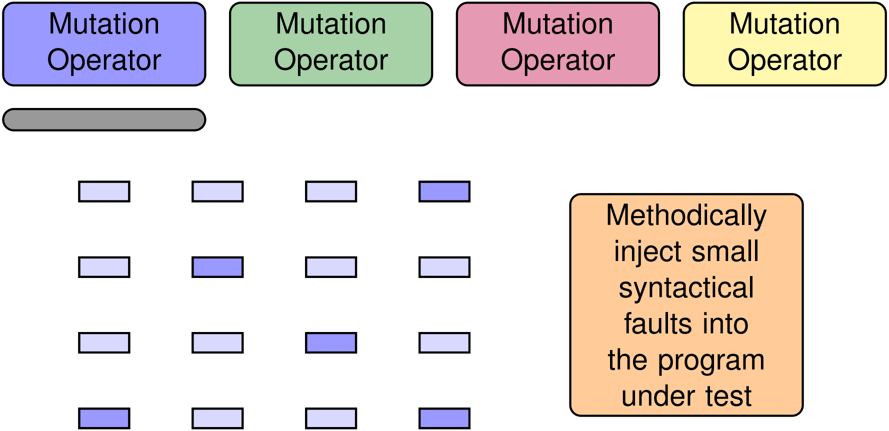
Overview of Mutation Analysis



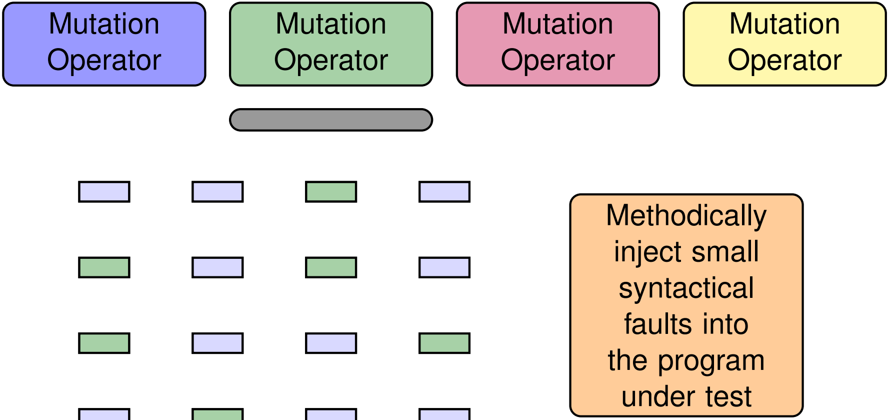
Overview of Mutation Analysis



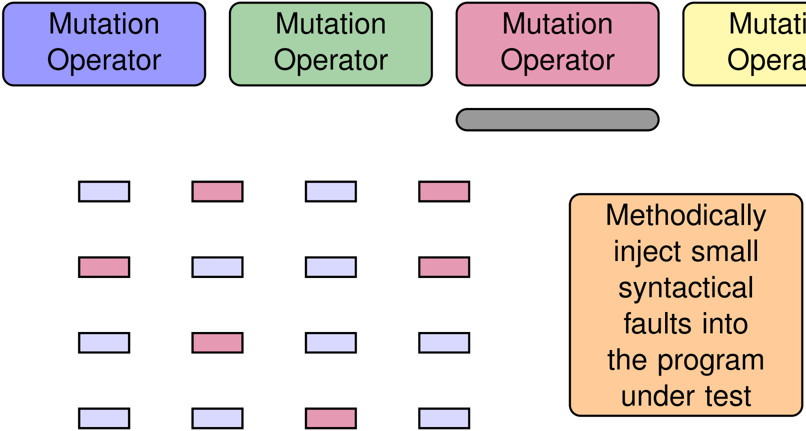
Overview of Mutation Analysis



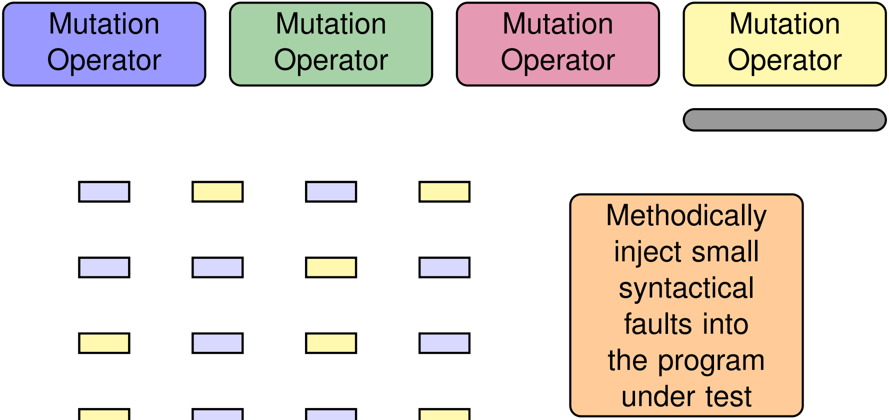
Overview of Mutation Analysis



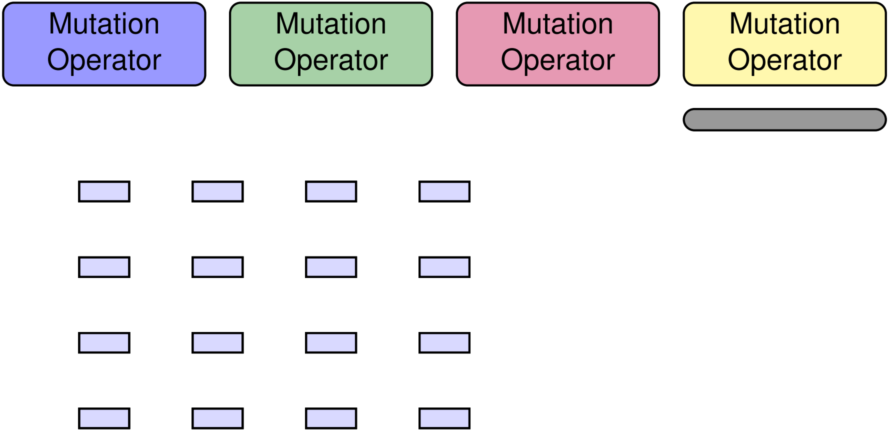
Overview of Mutation Analysis



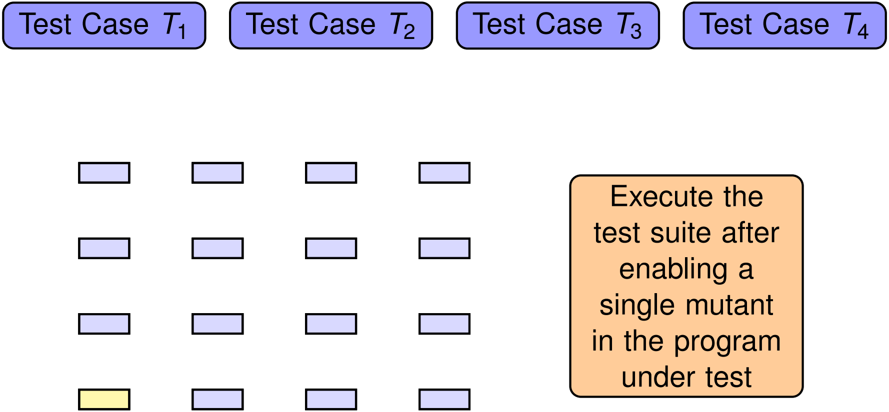
Overview of Mutation Analysis



Overview of Mutation Analysis



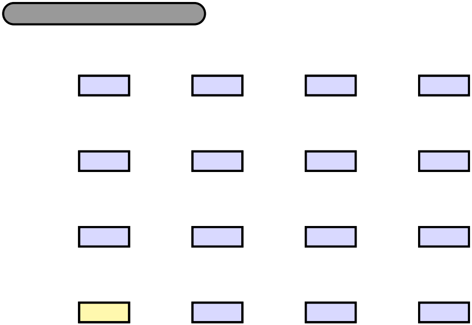
Overview of Mutation Analysis



Execute the test suite after enabling a single mutant in the program under test

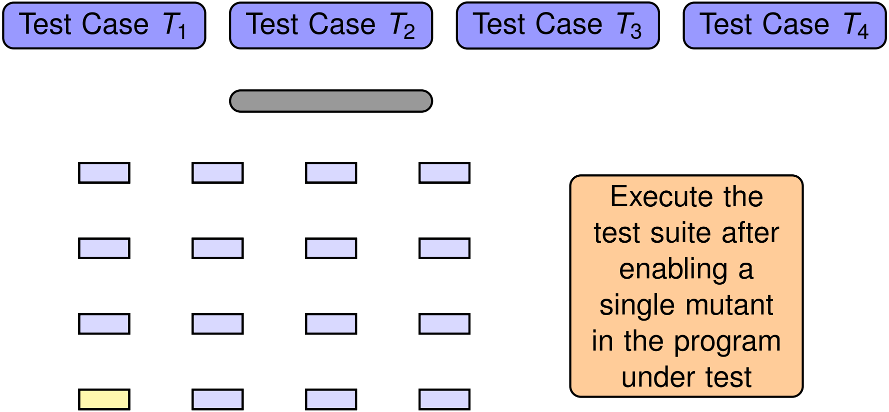
Overview of Mutation Analysis

Test Case T_1 | Test Case T_2 | Test Case T_3 | Test Case T_4

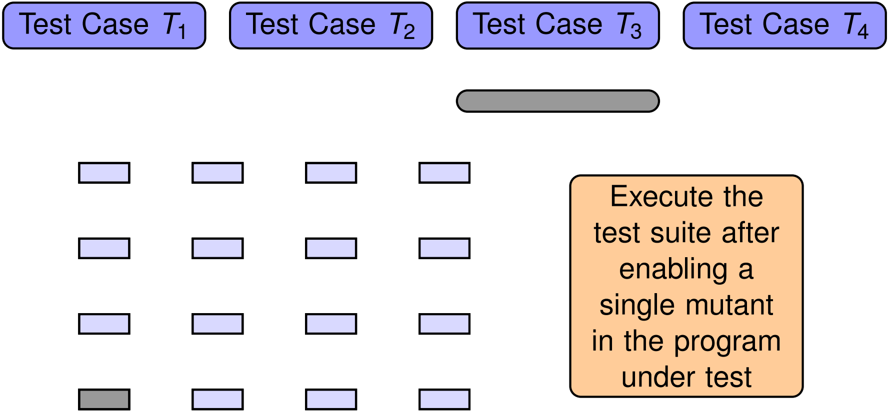


Execute the test suite after enabling a single mutant in the program under test

Overview of Mutation Analysis

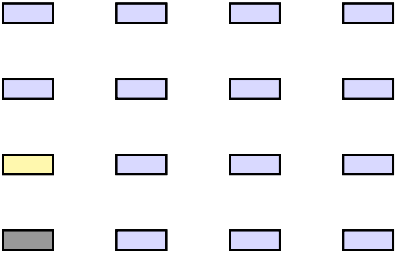


Overview of Mutation Analysis



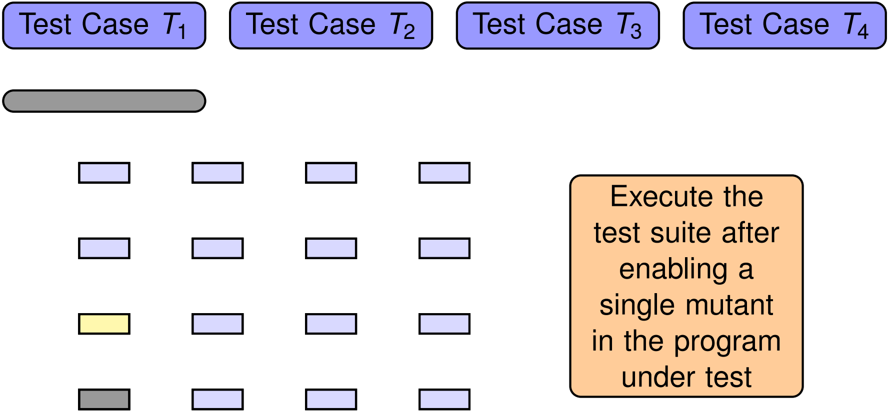
Overview of Mutation Analysis

Test Case T_1 Test Case T_2 Test Case T_3 Test Case T_4

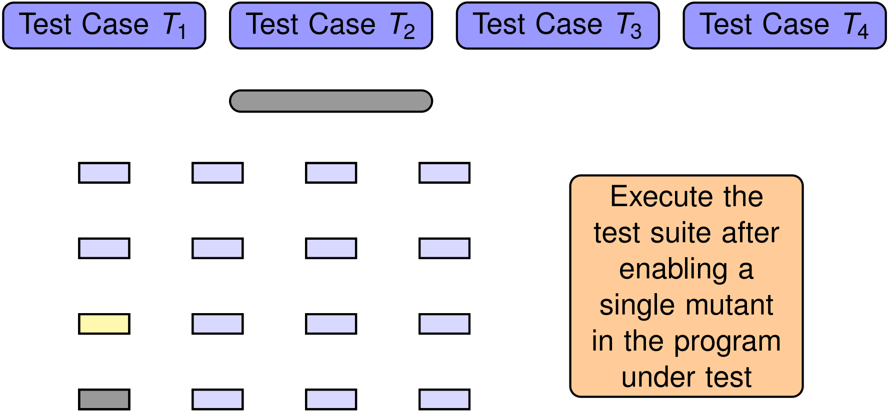


Execute the test suite after enabling a single mutant in the program under test

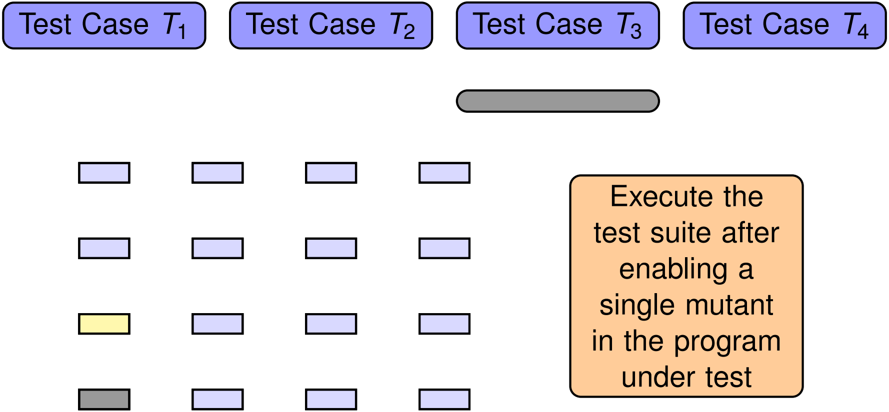
Overview of Mutation Analysis



Overview of Mutation Analysis

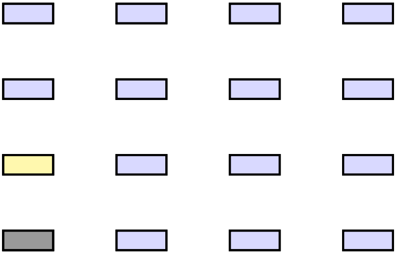


Overview of Mutation Analysis



Overview of Mutation Analysis

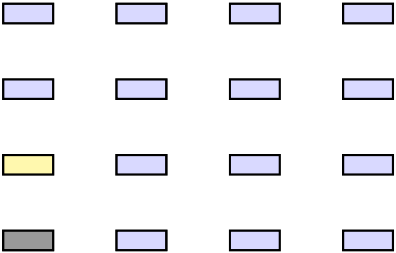
Test Case T_1 | Test Case T_2 | Test Case T_3 | Test Case T_4



Execute the test suite after enabling a single mutant in the program under test

Overview of Mutation Analysis

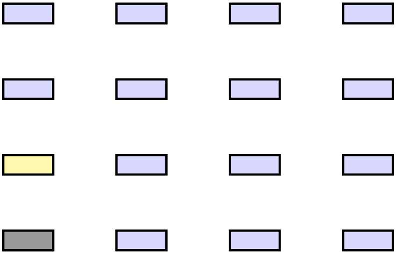
Test Case T_1 | Test Case T_2 | Test Case T_3 | Test Case T_4



The test suite *cannot* kill the mutant – either a test suite weakness or an equivalent mutant!

Overview of Mutation Analysis

Test Case T_1 Test Case T_2 Test Case T_3 Test Case T_4

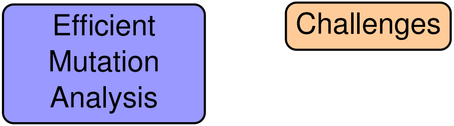


Repeat this process for *all* of the test cases and mutants – calculate mutation score when finished

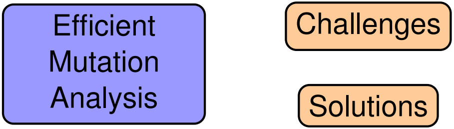
Contributions of this Presentation

Efficient Mutation Analysis

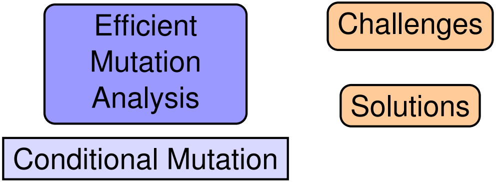
Contributions of this Presentation



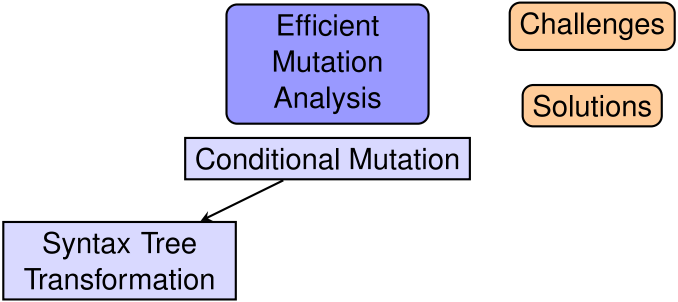
Contributions of this Presentation



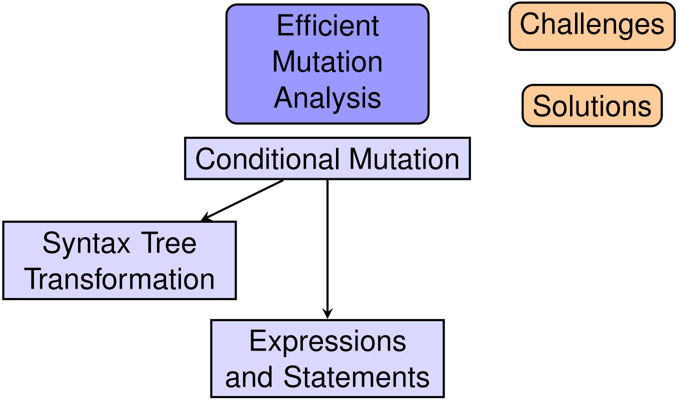
Contributions of this Presentation



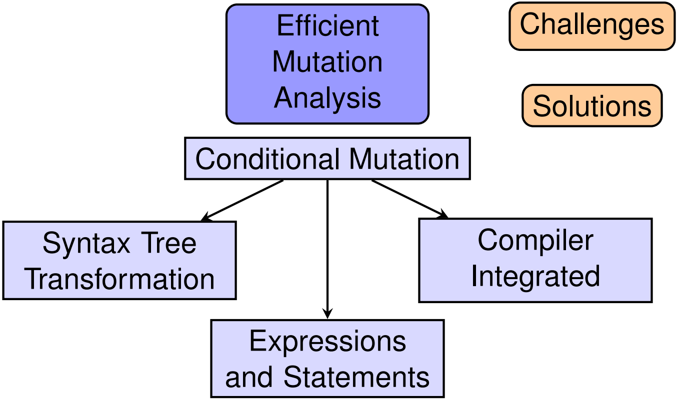
Contributions of this Presentation



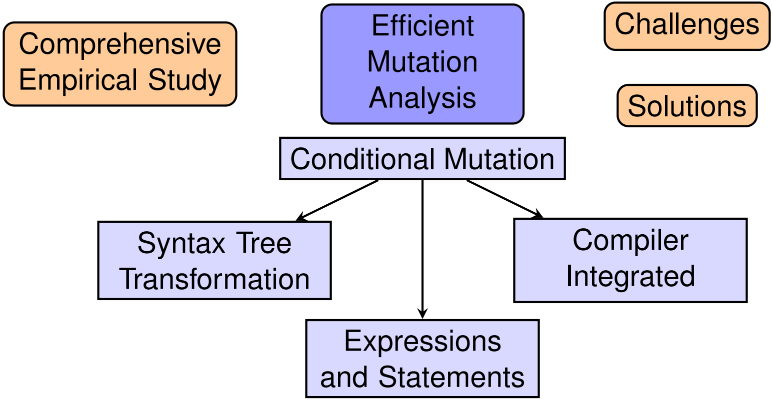
Contributions of this Presentation



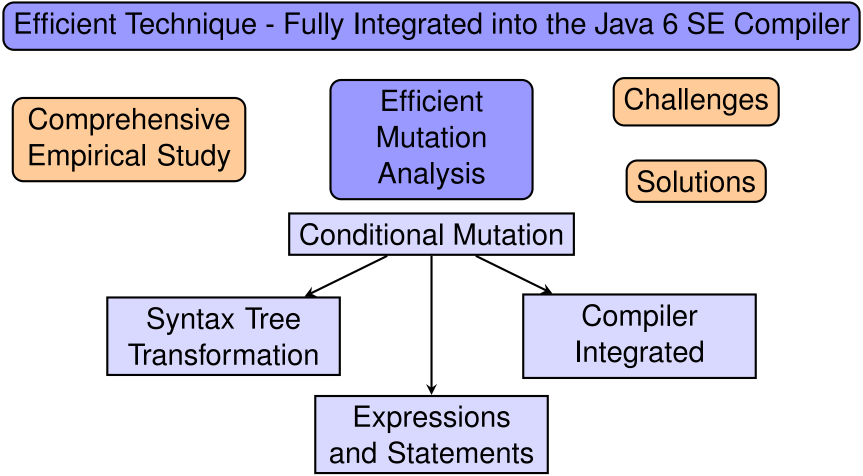
Contributions of this Presentation



Contributions of this Presentation



Contributions of this Presentation



Understanding Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

Understanding Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

Methodically inject small syntactical faults into the program under test

Understanding Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

Understanding Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

⟹

- y = a - x;
- y = a + x;
- y = a / x;

```
public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

⟹

- if(b < a)
- if(b != a)
- if(b == a)

Understanding Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
       max=b;
   }

   return max;
}
```

Unbiased and powerful method for assessing oracles and input values

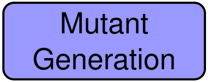
Understanding Mutation Analysis

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
   int max = a;

   if(b>a){
      max=b;
   }

   return max;
}
```

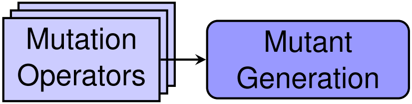
Unbiased and powerful method for assessing oracles and input values

Useful method for fault seeding during the empirical study of testing techniques

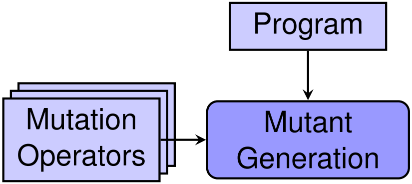
Mutation Analysis Challenges



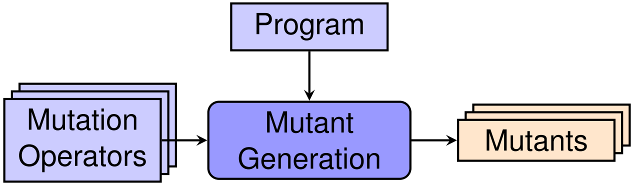
Mutation Analysis Challenges



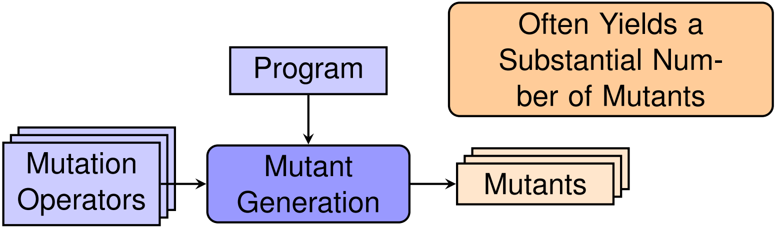
Mutation Analysis Challenges



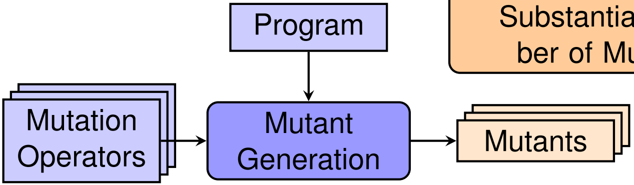
Mutation Analysis Challenges



Mutation Analysis Challenges



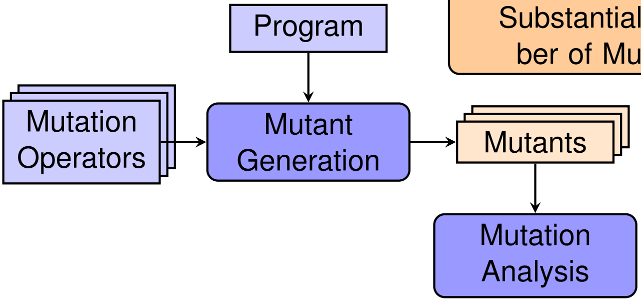
Mutation Analysis Challenges



Often Yields a Substantial Number of Mutants

High Time Overhead for Generation

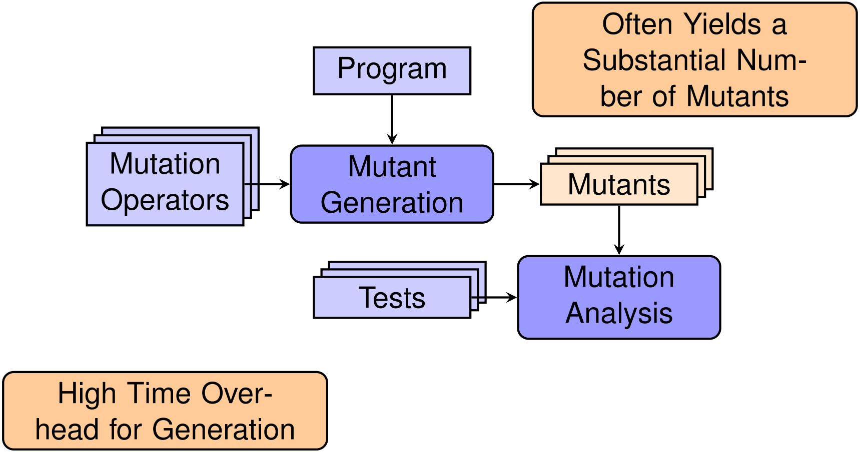
Mutation Analysis Challenges



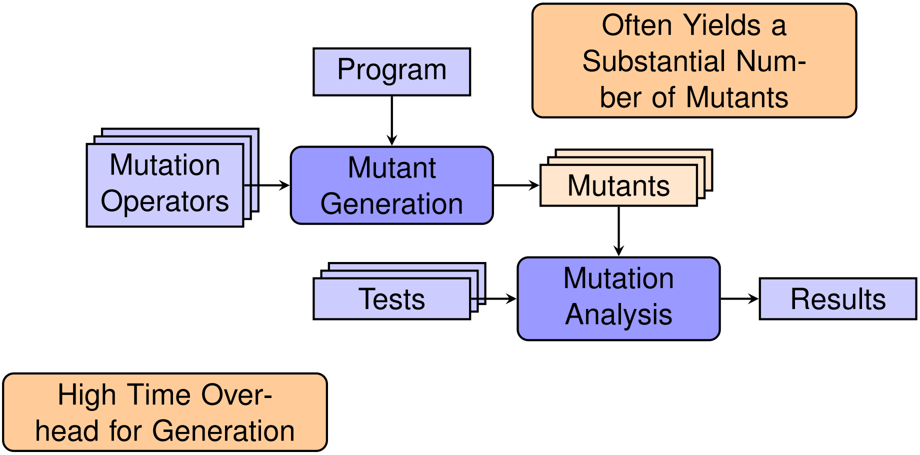
Often Yields a Substantial Number of Mutants

High Time Overhead for Generation

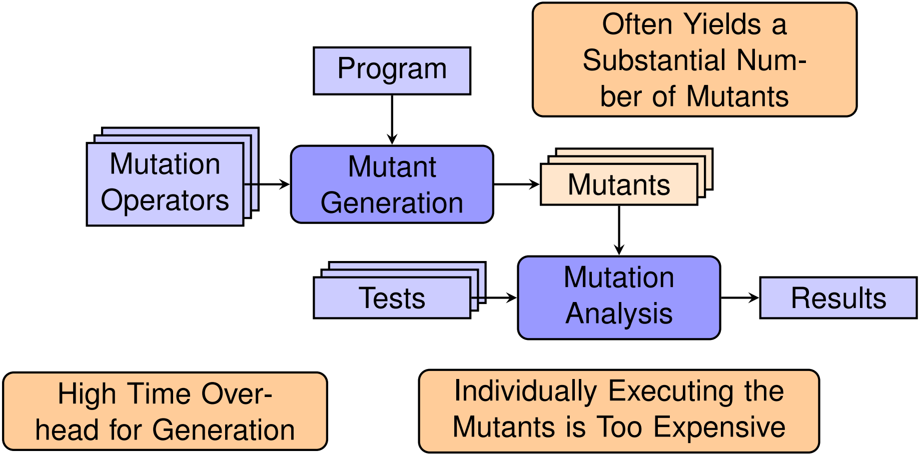
Mutation Analysis Challenges



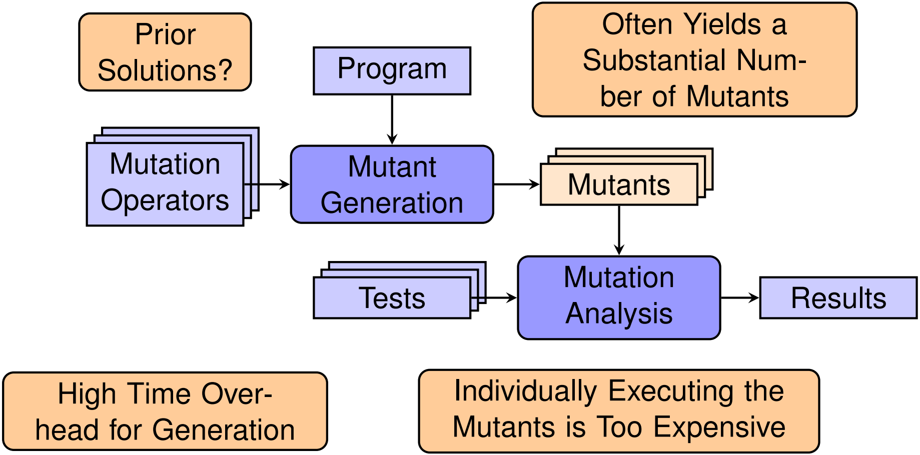
Mutation Analysis Challenges



Mutation Analysis Challenges



Mutation Analysis Challenges



Prior Work in Mutation Analysis

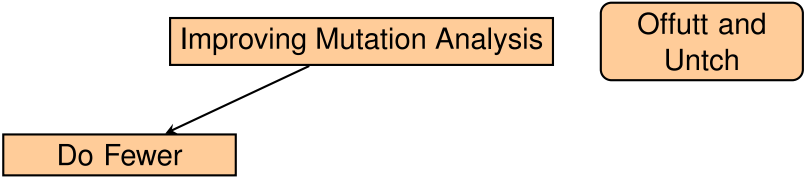
Improving Mutation Analysis

Prior Work in Mutation Analysis

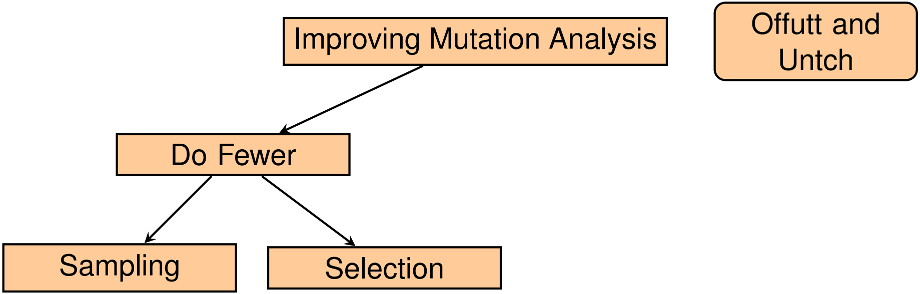
Improving Mutation Analysis

Offutt and Untch

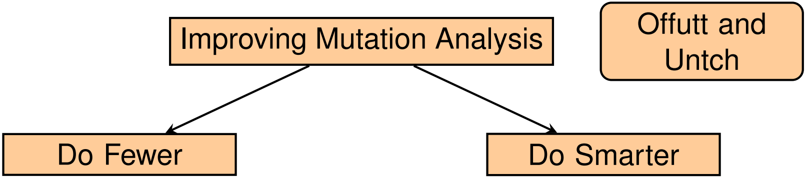
Prior Work in Mutation Analysis



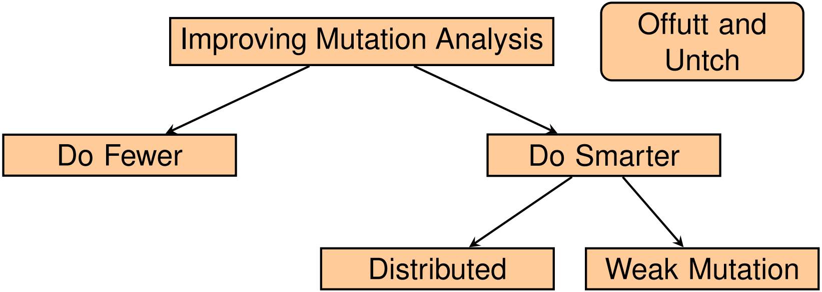
Prior Work in Mutation Analysis



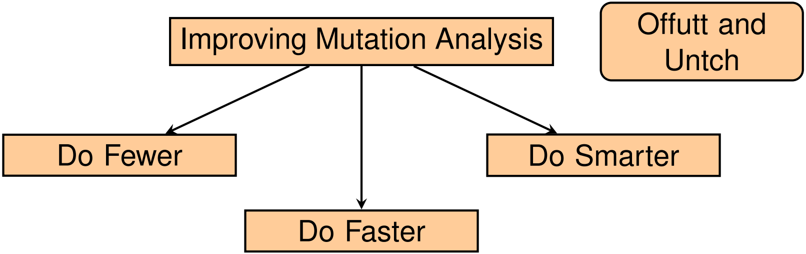
Prior Work in Mutation Analysis



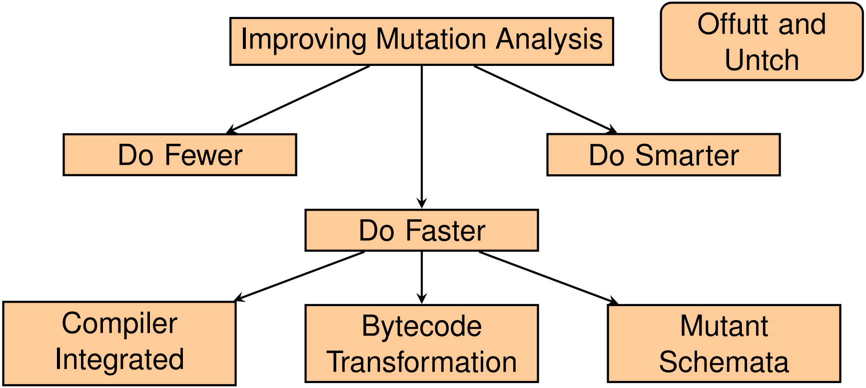
Prior Work in Mutation Analysis



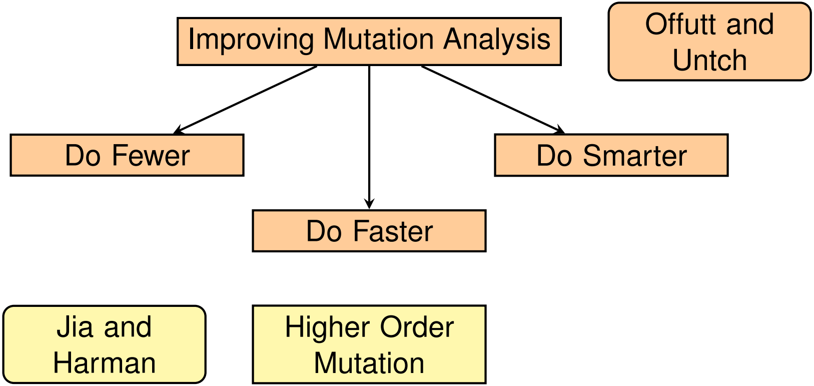
Prior Work in Mutation Analysis



Prior Work in Mutation Analysis



Prior Work in Mutation Analysis



Practical Mutation Analysis

Practical (adjective):

1. Of or concerned with the actual doing or use of something rather than with theory and ideas
2. (of an idea, plan, or method) Likely to succeed or be effective in real circumstances; feasible
3. Suitable for a particular purpose

Practical Mutation Analysis

Practical (adjective):

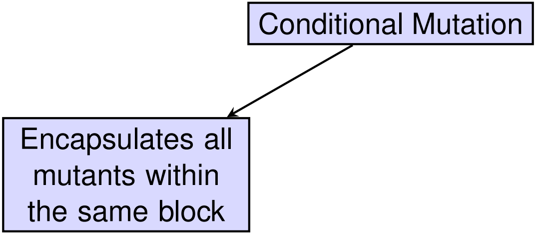
1. Of or concerned with the actual doing or use of something rather than with theory and ideas
2. (of an idea, plan, or method) Likely to succeed or be effective in real circumstances; feasible
3. Suitable for a particular purpose

What are the practical techniques that MAJOR employs for improving the efficiency and usability of mutation analysis?

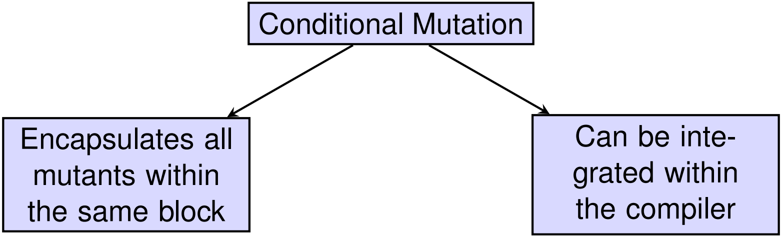
Conditional Mutation

Conditional Mutation

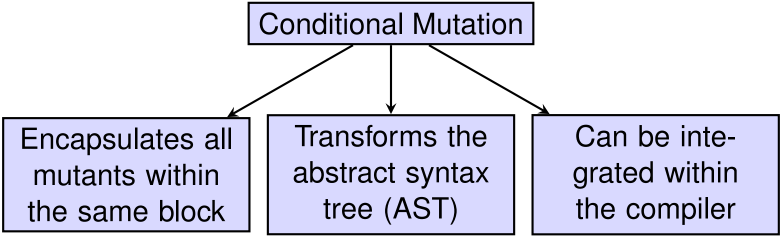
Conditional Mutation



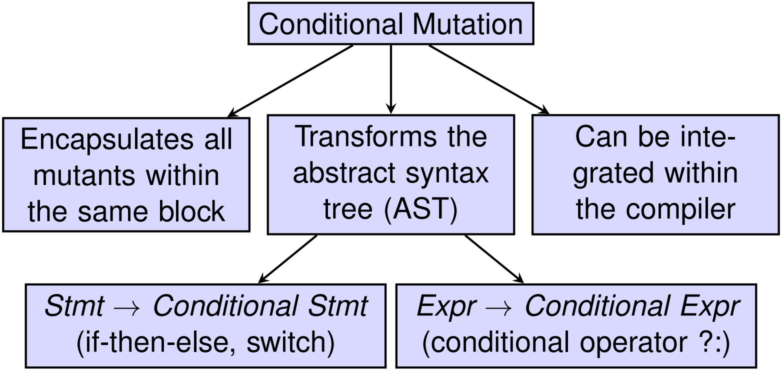
Conditional Mutation



Conditional Mutation



Conditional Mutation



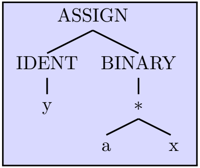
Transforming the AST

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

Transforming the AST

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

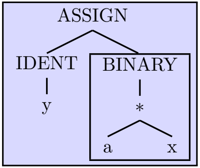
⇓



Transforming the AST

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

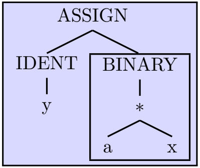
⇓



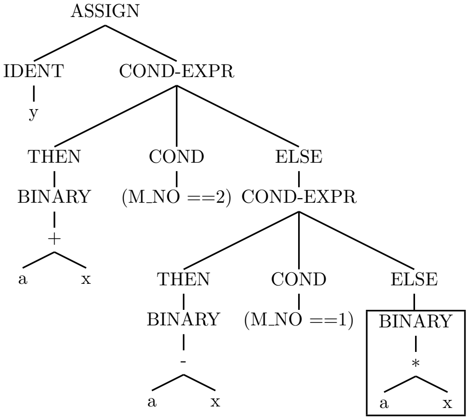
Transforming the AST

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

⇓



⟹



Source Code View of Inserting Mutants

```
public int eval(int x){
    int a=3, b=1, y;


    y = a * x;


    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Source Code View of Inserting Mutants

```
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Source Code View of Inserting Mutants

```
public int eval(int x){
    int a=3, b=1, y;

    y = [a * x];

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Source Code View of Inserting Mutants

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Source Code View of Inserting Mutants

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Source Code View of Inserting Mutants

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Source Code View of Inserting Mutants

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

Mutants that are not executed cannot be killed

1. Define mutation operators $MOP(x * y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Collecting and Using Mutation Coverage

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                   a * x;

    y += b;
    return y;
}
```

Mutants that are not executed cannot be killed

Collecting and Using Mutation Coverage

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
        (M_NO==0 &&
         COVERED(1,3))?
              a * x : a * x;

    y += b;

    return y;
}
```

Mutants that are not executed cannot be killed

Determine covered mutants with additional instrumentation

Collecting and Using Mutation Coverage

```
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
        (M_NO==0  &&
         COVERED(1,3))?
              a * x : a * x;

    y += b;

    return y;
}
```

Mutants that are not executed cannot be killed

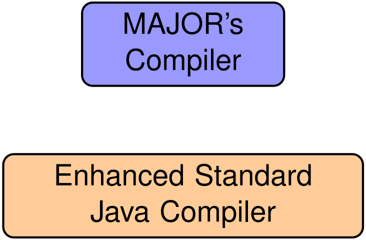
Determine covered mutants with additional instrumentation

Only execute and investigate the covered mutants

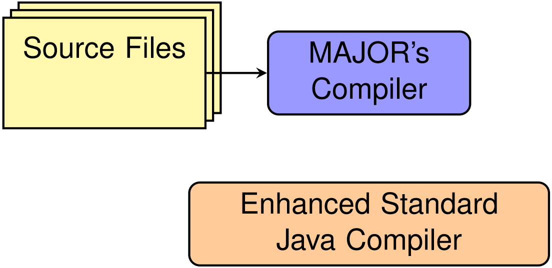
MAJOR's Compiler

MAJOR's Compiler

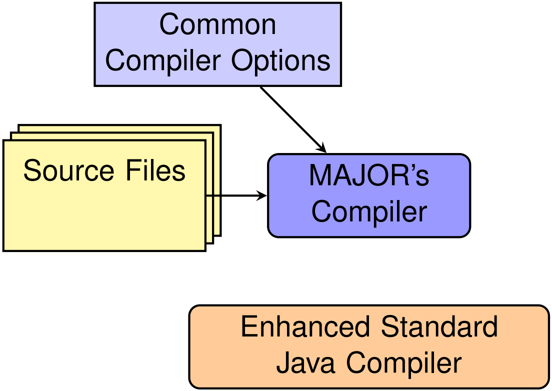
MAJOR's Compiler



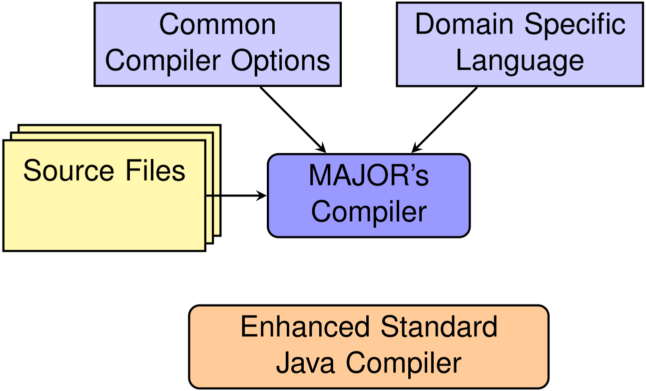
MAJOR's Compiler



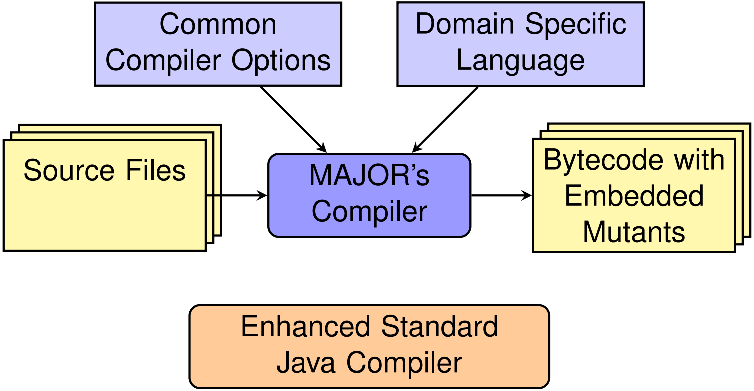
MAJOR's Compiler



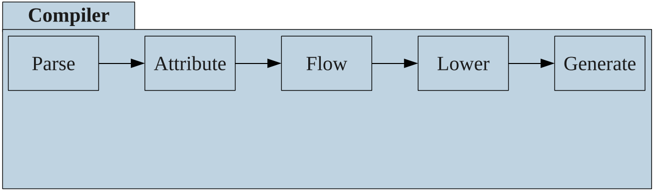
MAJOR’s Compiler



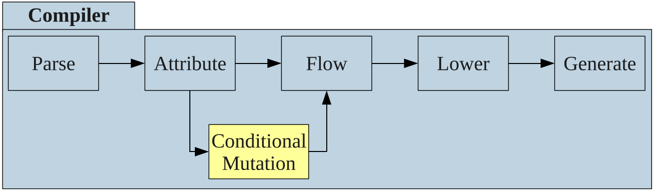
MAJOR's Compiler



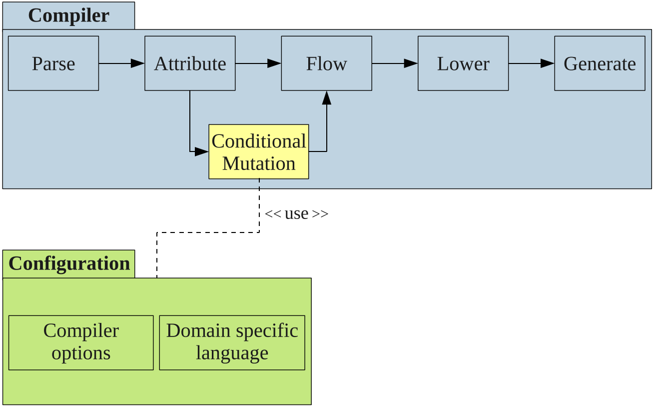
Integration into the Java Compiler



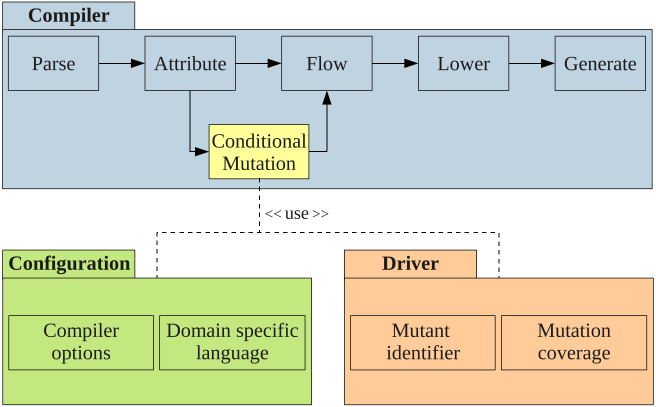
Integration into the Java Compiler



Integration into the Java Compiler



Integration into the Java Compiler



MAJOR's Domain Specific Language

```
// variable declaration
listCOR={&&, ||, ==, !=};
// Define replacement list
BIN(+)<"org"> -> {-,*};
BIN(*)<"org"> -> {/,%};
// Define own operator
myOp{
  BIN(&&) -> listCOR;
  BIN(||) -> listCOR;
  COR;
  LVR;
}
// Enable built-in operator AOR
AOR<"org">;
// Enable operator myOp
myOp<"java.lang.System@println">;
```

MAJOR’s Domain Specific Language

```
// variable declaration
listCOR={&&, ||, ==, !=};
// Define replacement list
BIN(+)<"org"> -> {-,*};
BIN(*)<"org"> -> {/,%};
// Define own operator
myOp{
  BIN(&&) -> listCOR;
  BIN(||) -> listCOR;
  COR;
  LVR;
}
// Enable built-in operator AOR
AOR<"org">;
// Enable operator myOp
myOp<"java.lang.System@println">;
```

Specify mutation operators in detail

MAJOR’s Domain Specific Language

```
// variable declaration
listCOR={&&, ||, ==, !=};
// Define replacement list
BIN(+)<"org"> -> {-,*};
BIN(*)<"org"> -> {/,%};
// Define own operator
myOp{
  BIN(&&) -> listCOR;
  BIN(||) -> listCOR;
  COR;
  LVR;
}
// Enable built-in operator AOR
AOR<"org">;
// Enable operator myOp
myOp<"java.lang.System@println">;
```

Specify mutation operators in detail

Define own mutation operator groups

MAJOR’s Domain Specific Language

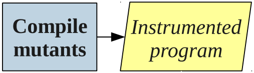
```
// variable declaration
listCOR={&&, ||, ==, !=};
// Define replacement list
BIN(+)<"org"> -> {-,*};
BIN(*)<"org"> -> {/,%};
// Define own operator
myOp{
  BIN(&&) -> listCOR;
  BIN(||) -> listCOR;
  COR;
  LVR;
}
// Enable built-in operator AOR
AOR<"org">;
// Enable operator myOp
myOp<"java.lang.System@println">;
```

Specify mutation operators in detail

Define own mutation operator groups

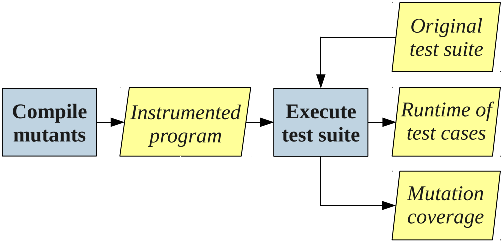
Enable operators for a specific package, class, or method

Optimized Mutation Analysis Process



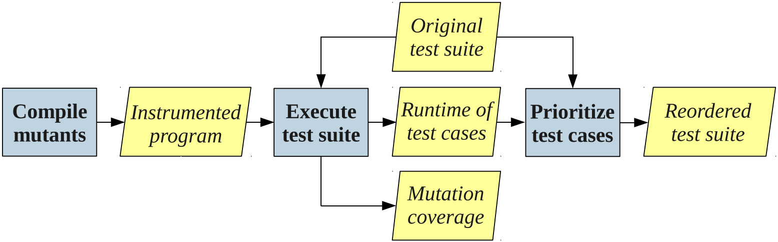
1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite

Optimized Mutation Analysis Process



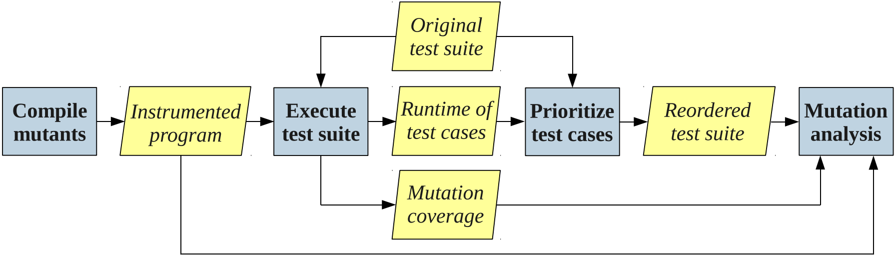
1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite

Optimized Mutation Analysis Process



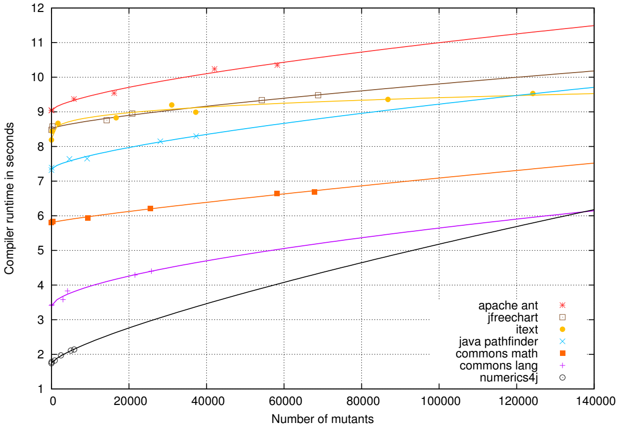
1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite

Optimized Mutation Analysis Process



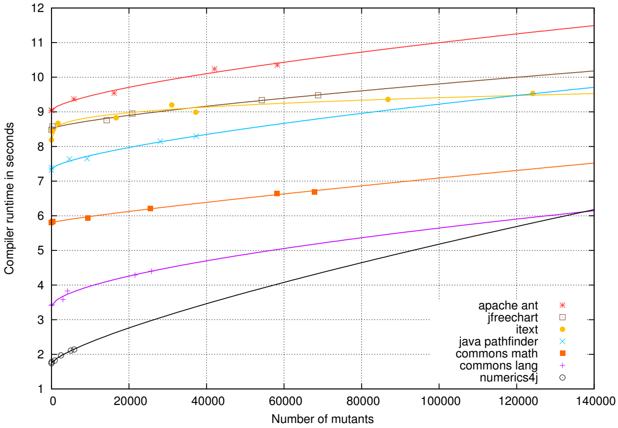
1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite

Mutant Generation and Compilation



Overhead for generating and compiling mutants is negligible

Mutant Generation and Compilation



Overhead for generating and compiling mutants is negligible

Time and Space Overhead

Application	***Mutants***	***Runtime of test suite***			***Memory consumption***	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Time and Space Overhead

Application	***Mutants***	***Runtime of test suite***			***Memory consumption***	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Time and Space Overhead

Application	***Mutants***	***Runtime of test suite***			***Memory consumption***	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Time and Space Overhead

Application	***Mutants***	***Runtime of test suite***			***Memory consumption***	
		original	*instrumented*		*original*	*instrumented*
			wcs	*wcs+cov*		
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

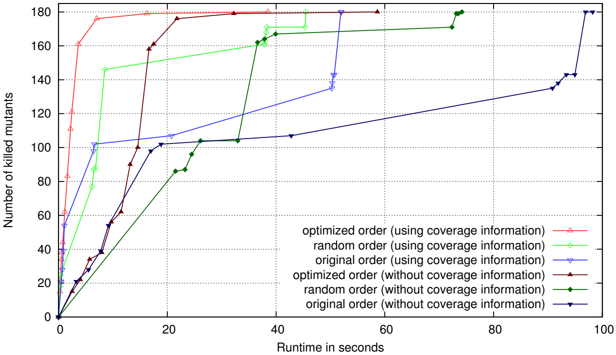
- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Time and Space Overhead

Application	***Mutants***	***Runtime of test suite***			***Memory consumption***	
		original	*instrumented* *wcs*	*instrumented* *wcs+cov*	*original*	*instrumented*
aspectj	406,382	4.3	4.8	5.0	559	813
apache ant	60,258	331.0	335.0	346.0	237	293
jfreechart	68,782	15.0	18.0	23.0	220	303
itext	124,184	5.1	5.6	6.3	217	325
java pathfinder	37,331	17.0	22.0	29.0	182	217
commons math	67,895	67.0	83.0	98.0	153	225
commons lang	25,783	10.3	11.8	14.8	104	149
numerics4j	5,869	1.2	1.3	1.6	73	90

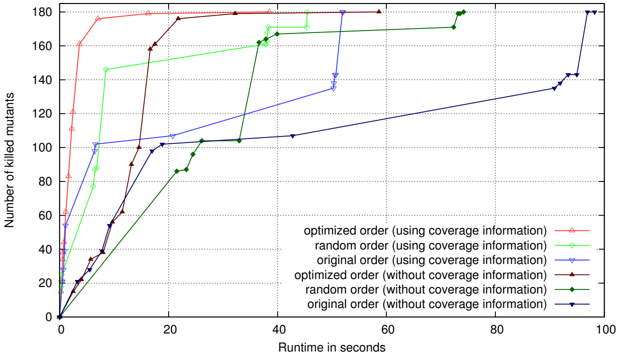
- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

Evaluating and Improving Mutation Analysis



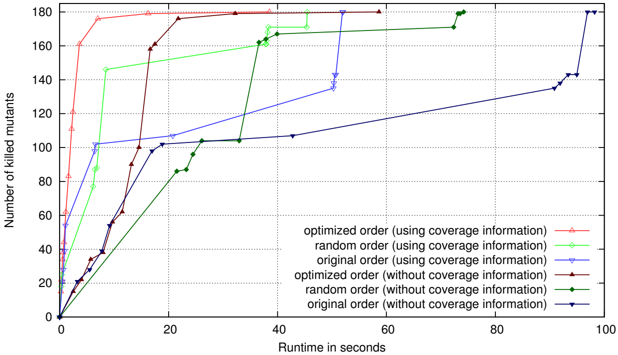
- Mutation analysis is not feasible without coverage information
- Reordering the test suite significantly speeds up the process, especially if runtimes of tests differ by orders of magnitude

Evaluating and Improving Mutation Analysis



- Mutation analysis is not feasible without coverage information
- Reordering the test suite significantly speeds up the process, especially if runtimes of tests differ by orders of magnitude

Evaluating and Improving Mutation Analysis



- Mutation analysis is not feasible without coverage information
- Reordering the test suite significantly speeds up the process, especially if runtimes of tests differ by orders of magnitude

Revisiting Practical Mutation Analysis

Practical (adjective):

1. Of or concerned with the actual doing or use of something rather than with theory and ideas
2. (of an idea, plan, or method) Likely to succeed or be effective in real circumstances; feasible
3. Suitable for a particular purpose

Revisiting Practical Mutation Analysis

Practical (adjective):

1. Of or concerned with the actual doing or use of something rather than with theory and ideas
2. (of an idea, plan, or method) Likely to succeed or be effective in real circumstances; feasible
3. Suitable for a particular purpose

The evidence suggests that MAJOR is “likely to succeed or be effective” in real-world software testing circumstances

Reviewing MAJOR's Contributions

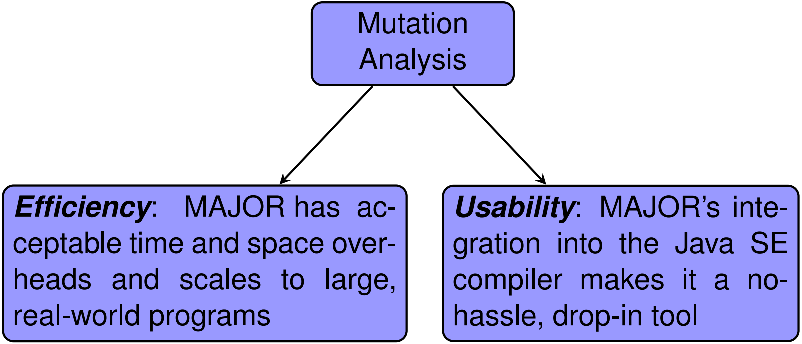
Mutation Analysis

Reviewing MAJOR's Contributions

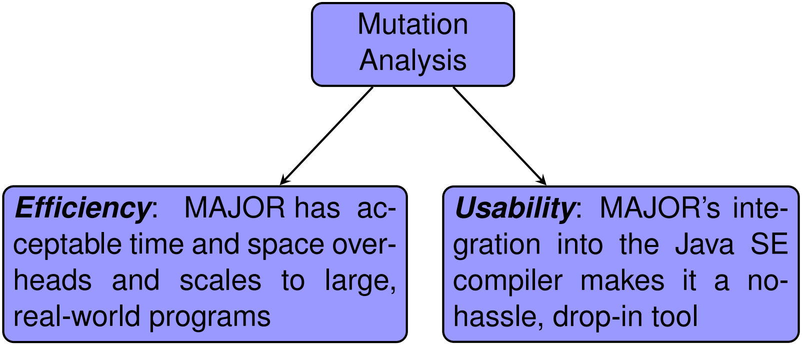
Mutation Analysis

Efficiency: MAJOR has acceptable time and space overheads and scales to large, real-world programs

Reviewing MAJOR’s Contributions



Reviewing MAJOR's Contributions



We will release MAJOR as free and open source software

Conclusion

Key Concepts and Features:

- Compiler-integrated solution
- Conditional mutation with the abstract syntax tree
- Furnishes its own domain specific language
- Collects and leverages mutation coverage information

Conclusion

Key Concepts and Features:

- Compiler-integrated solution
- Conditional mutation with the abstract syntax tree
- Furnishes its own domain specific language
- Collects and leverages mutation coverage information

Characteristics of MAJOR:

- Fast and scalable technique
- Configurable and extensible mutation tool
- Enables an optimized workflow for mutation analysis

Practical Techniques for Improving the Efficiency and Usability of Mutation Analysis for Java Programs

Gregory M. Kapfhammer

Department of Computer Science
Allegheny College
http://www.cs.allegheny.edu/~gkapfham/

Thank you for your attention!
I welcome your questions and comments.