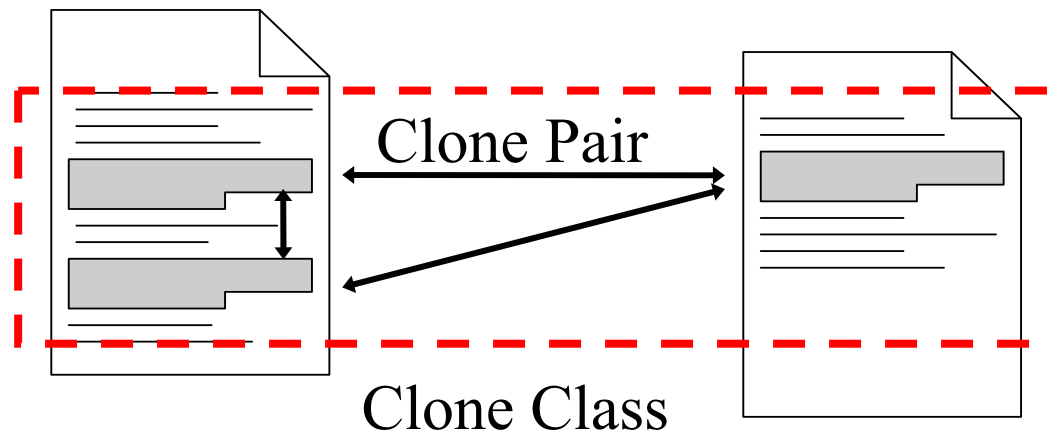
A Mutation / Injection-based Automatic Framework for Evaluating Code Clone Detection Tools

Chanchal K. Roy
James R. Cordy
Queen’s University, Canada
April 4, 2009

Introduction

- What are "Code Clones"?
  - A code fragment which has identical or similar code fragment (s) in source code



Introduction

- Intentional copy/paste is a common reuse technique in software development
- Previous studies report 7% - 30% cloned code software systems [Baker WCRE'95, Roy and Cordy WCRE'08]
- Unfortunately, clones are harmful in software maintenance and evolution [Juergens et al. ICSE'09]

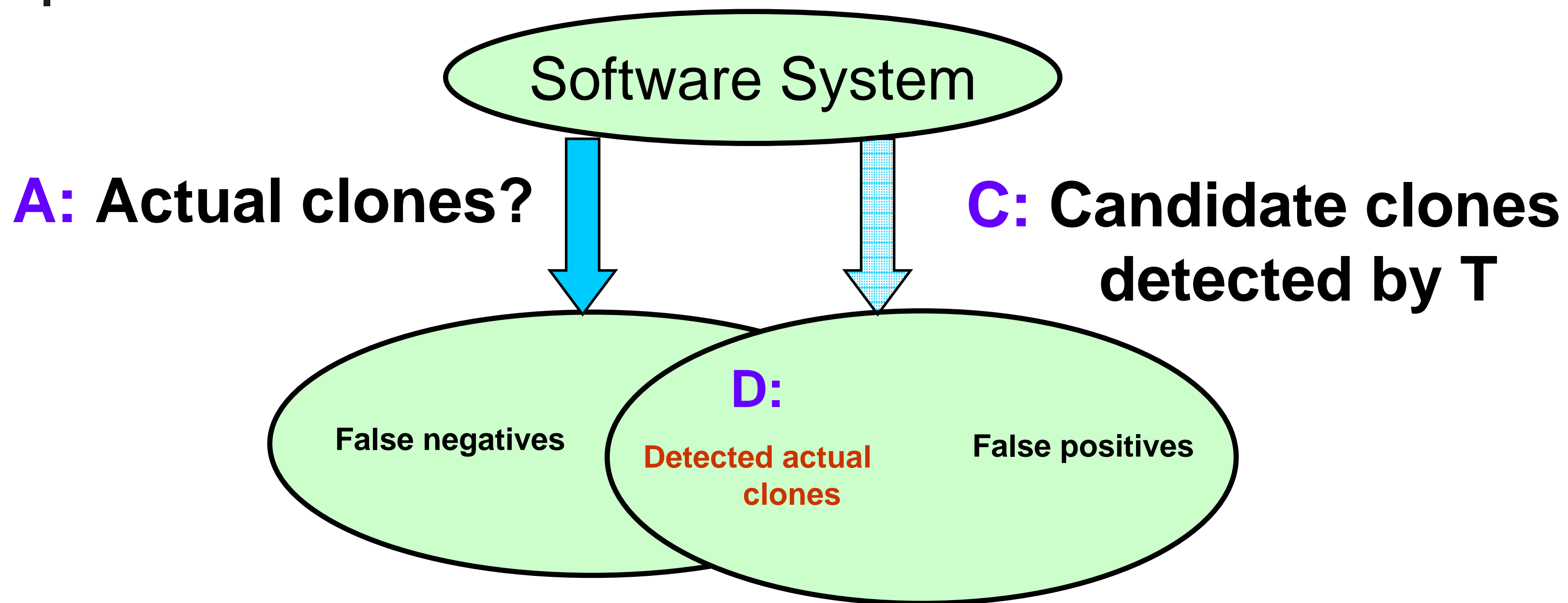
Introduction: Existing Methods

- In response, many methods proposed:
  - Text-based: Duploc [Ducasse et al. ICSM'99], NICAD [Roy and Cordy, ICPC'08]
  - Token-based: Dup [Baker, WCRE'95], CCFinder [Kamiya et al., TSE'02], CP-Miner [Li et al., TSE'06]
  - Tree-Based: CloneDr [Baxter et al. ICSM'98], Asta [Evans et al. WCRE'07], Deckard [Jiang et al. ICSE'07], cpdetector [Falke et al. ESE'08]
  - Metrics-based: Kontogiannis WCRE'97, Mayrand et al. ICSM'96
  - Graph-based: Gabel et al. ICSE'08, Komondoor and Horwitz SAS'01, Dublix [Krinke WCRE'01]

Introduction: Lack of Evaluation

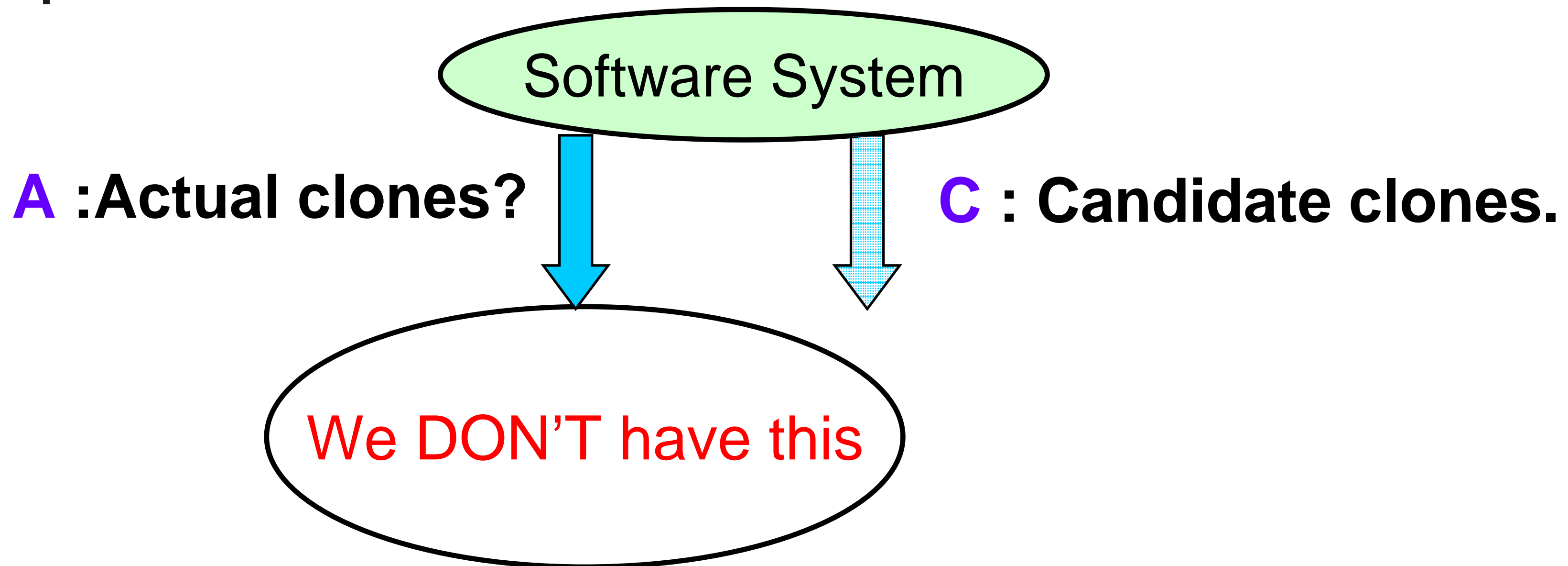
- Marked lack of in-depth evaluation of the methods in terms of
  - precision and
  - recall
- Existing tool comparison experiments (e.g., Bellon et al. TSE'07) or individual evaluations have faced serious challenges [Baker TSE'07, Roy and Cordy ICPC'08, SCP'09]

Introduction: Precision and Recall



$$\text{Recall} = \frac{|D|}{|A|} \qquad \text{Precision} = \frac{|D|}{|C|}$$

Primary Challenge: Lack of a Reliable Reference Set



We still don't have this actual/reliable clone set for any system

Challenges in Oracling a System

- No crisp definition of code clones
- Huge manual effort
  - May be possible for small system
- What about for large systems?
  - Even the relatively small cook system yields nearly a million function pairs to sort through
  - Not possible for human to do error-free

Challenges in Evaluation

- Union of results from different tools can give good relative results
  - but no guarantee that the subject tools indeed detect all the clones
- Manual validation of the large candidate clone set is difficult
  - Bellon [TSE'07] took 77 hours for only 2% of clones
- No studies report the reliability of judges

Lack of Evaluation for Individual Types of Clones

- No work reports precision and recall values for different types of clones except,
  - Bellon et al. [TSE’07]: Types I, II and III
  - Falke et al. [ESE’08]: Types I and II
- Limitations reported
  - Baker [TSE’07]
  - Roy and Cordy ICPC’08, SCP’09

In this paper…

- A mutation-based framework that automatically and efficiently
  - measures and
  - compares precision and recall of the tools for different fine-grained types of clones.
- A taxonomy of clones
  - => Mutation operators for cloning
  - => Framework for tool comparison

An Editing Taxonomy of Clones

- Definition of clone is inherently vague
  - Most cases detection dependent and task-oriented
- Some taxonomies proposed
  - but limited to function clones and still contain the vague terms, “similar” and “long differences” [Roy and Cordy SCP’09, ICPC’08]
- We derived the taxonomy from the literature and validated with empirical studies [Roy and Cordy WCRE’08]
- Applicable to any granularity of clones

Exact Software Clones

Changes in layout and formatting

```
void  sumProd(int n) {      //s0
int  sum=0;                 //s1
int product =1;             //s2
for (int i=1; i<=n; i++) {  //s3
     sum=sum + i;           //s4
     product = product * i; //s5
     fun(sum, product); }}  //s6
```

Type I

Reuse by copy and paste

Changes in whitespace

```
void  sumProd(int n) {      //s0
   int  sum=0;                 //s1
   int product =1;             //s2
   for (int i=1; i<=n; i++) { //s3
        sum=sum + i;           //s4
        product = product * i; //s5
        fun(sum, product); }}  //s6
```

Changes in comments

```
void  sumProd(int n) {      //s0
int  sum=0;                 //s1'
int product =1;             //s2
for (int i=1; i<=n; i++) {  //s3'
     sum=sum + i;           //s4
     product = product * i; //s5'
     fun(sum, product); }}  //s6
```

Changes in formatting

```
void  sumProd(int n) {      //s0
int  sum=0;                 //s1
int product =1;             //s2
for (int i=1; i<=n; i++)    //s3
  { sum=sum + i;            //s4
     product = product * i; //s5
     fun(sum, product); }}  //s6
```

Near-Miss Software Clone

Renaming Identifiers and Literal Values

Type II

```
void  sumProd(int n) {     //s0
int  sum=0;                //s1
int product =1;            //s2
for (int i=1; i<=n; i++) {  //s3
      sum=sum + i;          //s4
      product = product * i; //s5
      fun(sum, product); }} //s6
```

Reuse by copy and paste

Renaming of identifiers

Renaming of Literals and Types

sumProd =>addTimes

sum => add

product => times

0=>0.0

1=>1.0

int=>double

```
void  addTimes(int n) {  //s0
int  add=0;              //s1
int times =1;            //s2
for (int i=1; i<=n; i++) { //s3
      add=add + i;        //s4
      times = times * i;  //s5
      fun(add, times); }} //s6
```

```
void  sumProd(int n) {     //s0
double  sum=0.0;           //s1
double product =1.0;       //s2
for (int i=1; i<=n; i++) {  //s3
      sum=sum + i;          //s4
      product = product * i; //s5
      fun(sum, product); }} //s6
```

Near-Miss Software Clone

Statements added/deleted/modified in copied fragments

Type III

```
void  sumProd(int n) {      //s0
int  sum=0;                 //s1
int product =1;             //s2
for (int i=1; i<=n; i++) {  //s3
     sum=sum + i;           //s4
     product = product * i; //s5
     fun(sum, product); }}  //s6
```

Reuse by copy and paste

Modification of lines

```
void  sumProd(int n) {            //s0
int  sum=0;                       //s1
int product =1;                   //s2
for (int i=1; i<=n; i++) {        //s3
     if  (i % 2 == 0) sum+= i;    //s4m
     product = product * i;       //s5
     fun(sum, product); }}        //s6
```

Addition of new of lines

```
void  sumProd(int n) {         //s0
int  sum=0;                    //s1
int product =1;                //s2
for (int i=1; i<=n; i++)       //s3
     if  (i % 2 == 0){         //s3b
     sum=sum + i;              //s4
     product = product * i;    //s5
     fun(sum, product); }}     //s6
```

Deletions of lines

```
void  sumProd(int n) {      //s0
int  sum=0;                 //s1
int product =1;             //s2
for (int i=1; i<=n; i++) {  //s3
     sum=sum + i;           //s4
     //s5  line deleted
     fun(sum, product); }}  //s6
```

Near-Miss Software Clone

Statements reordering/control replacements

```
void  sumProd(int n) {      //s0
int  sum=0;                 //s1
int product =1;             //s2
for (int i=1; i<=n; i++) {  //s3
     sum=sum + i;           //s4
     product = product * i; //s5
     fun(sum, product); }}  //s6
```

Type IV

Reuse by copy and paste

Reordering of Statements

```
void  sumProd(int n) {      //s0
int product =1;             //s2
int sum=0;                  //s1
for (int i=1; i<=n; i++) {  //s3
     sum=sum + i;           //s4
     product = product * i; //s5
     fun(sum, product); }}  //s6
```

Control Replacements

```
void  sumProd(int n) {      //s0
int  sum=0;                 //s1
int product =1;             //s2
int i = 0;                  //s7
while (i<=n) {              //s3'
     sum=sum + i;           //s4
     product = product * i; //s5
     fun(sum, product);     //s6
     i =i + 1; }}           //s8
```

Mutation Operators for Cloning

- For each of the fine-grained clone types of the clone taxonomy,
  - We built mutation operators for cloning
  - We use TXL [Cordy SCP'06] in implementing the operators
- Tested with
  - C, C# and Java

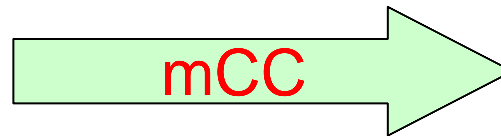
Mutation Operators for Cloning

- For Type I Clones

Name	Random Editing Activities
mCW	Changes in whitespace
mCC	Changes in comments
mCF	Changes in formatting

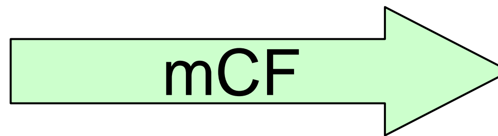
mCC: Changes in Comments

Line	Original		Mutated	Line
1	if (x==5)		if (x==5)	1
2	{		{	2
3	a=1;		a=1;	3
4	}		}	4
5	else		else //C	5
6	{		{	6
7	a=0;		a=0;	7
8	}		}	8



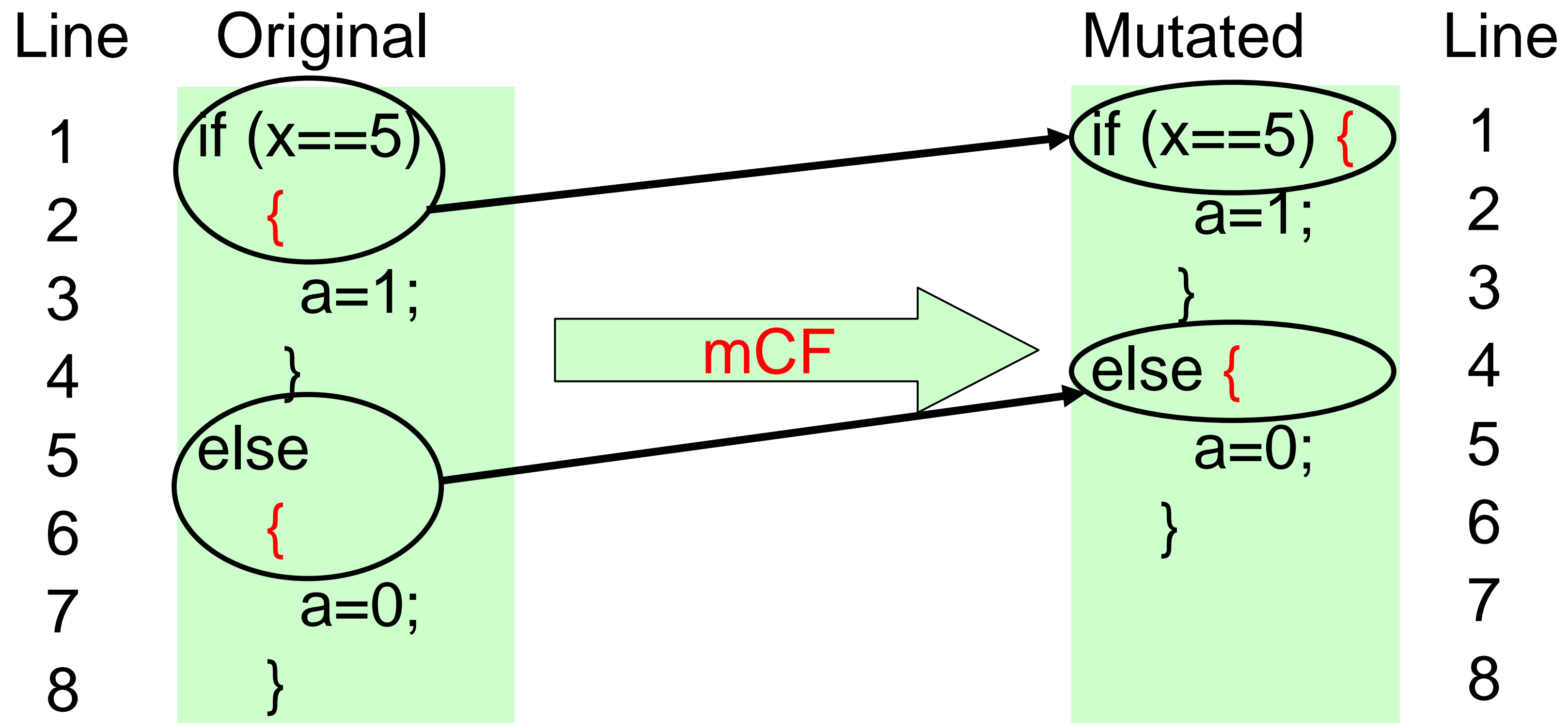
mCF: Changes in formatting

Line	Original
1	if (x==5)
2	{
3	a=1;
4	}
5	else
6	{
7	a=0;
8	}



Mutated	Line
if (x==5) {	1
a=1;	2
}	3
else {	4
a=0;	5
}	6
	7
	8

mCF: Changes in formatting



One or more changes can be made at a time

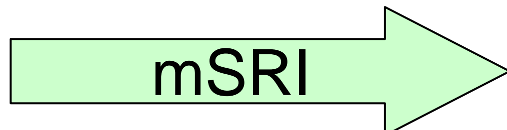
Mutation Operators for Cloning

- For Type II Clones

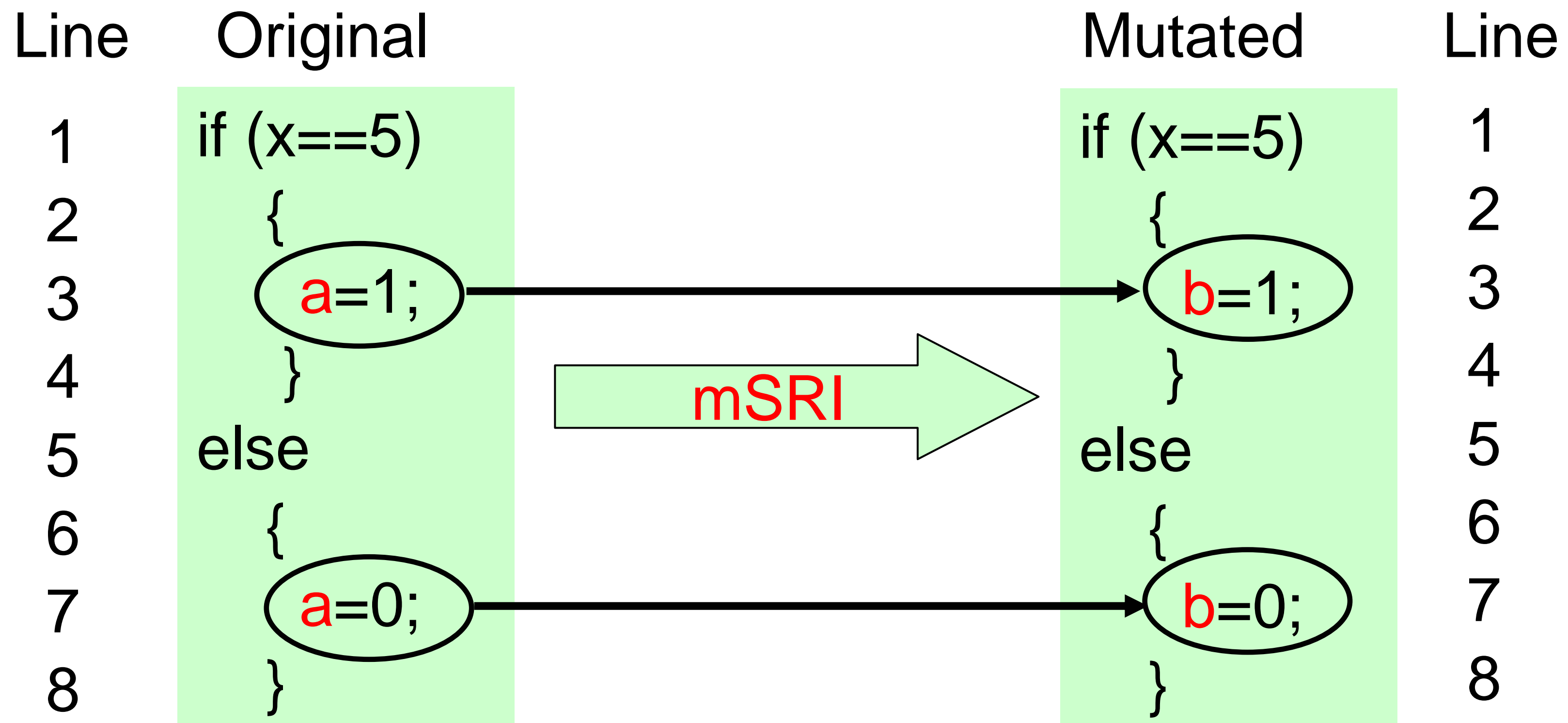
Name	Random Editing Activities
mSRI	Systematic renaming of identifiers
mARI	Arbitrary renaming of identifiers
mRPE	Replacement of identifiers with expressions (systematically/non-systematically)

mSRI: Systematic renaming of identifiers

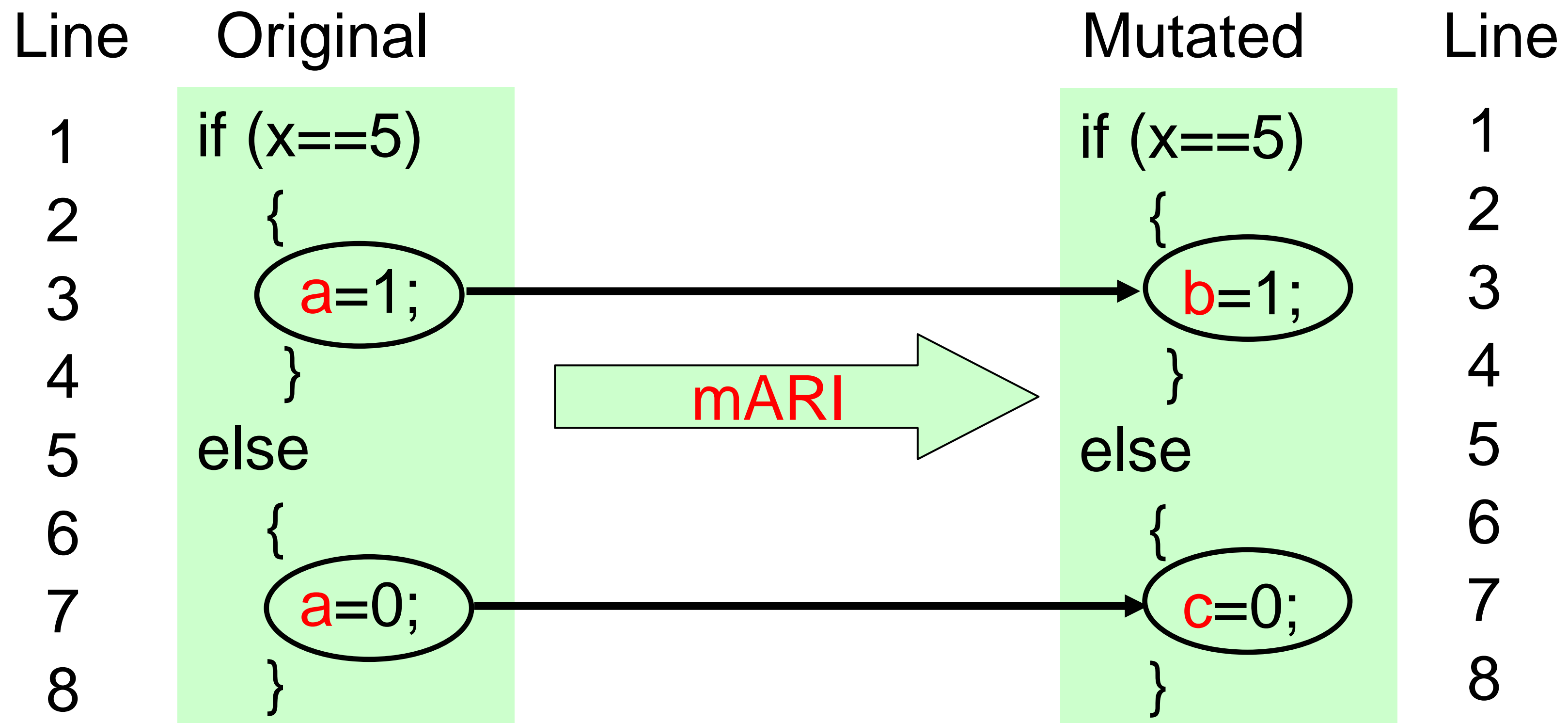
Line	Original	Mutated	Line
1	if (x==5)	if (x==5)	1
2	{	{	2
3	a=1;	b=1;	3
4	}	}	4
5	else	else	5
6	{	{	6
7	a=0;	b=0;	7
8	}	}	8



mSRI: Systematic renaming of identifiers



mARI: Arbitrary renaming of identifiers



Mutation Operators for Cloning

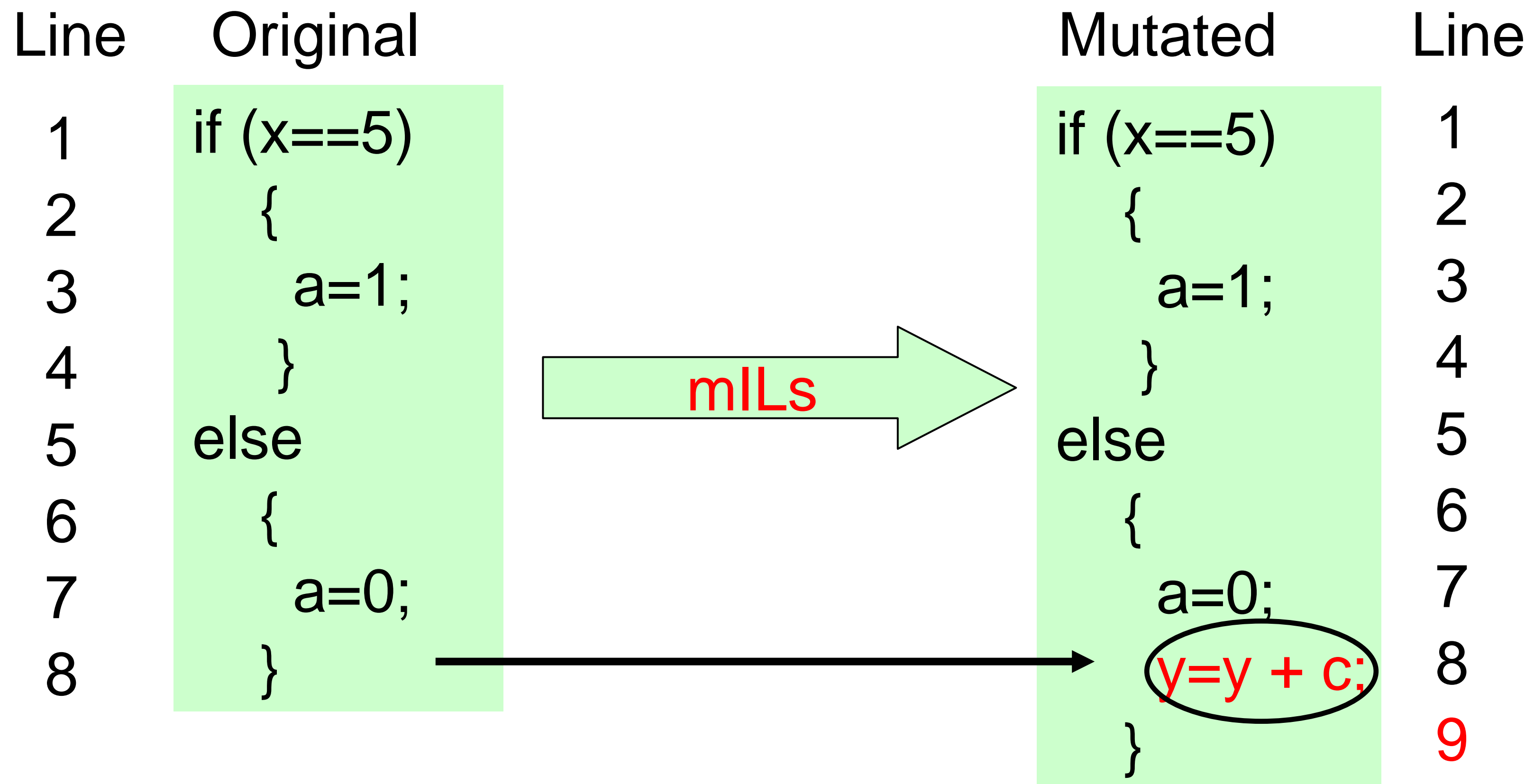
- For Type III Clones

Name	Random Editing Activities
mSIL	Small insertions within a line
mSDL	Small deletions within a line
mILs	Insertions of one or more lines
mDLs	Deletions of one or more lines
mMLs	Modifications of whole line(s)

mSIL: Small Insertion within a Line

Line	Original		Mutated	Line
1	if (x==5)		if (x==5)	1
2	{		{	2
3	a=1;	→	a=1 + x;	3
4	}	mSIL	}	4
5	else		else	5
6	{		{	6
7	a=0;		a=0;	7
8	}		}	8

mILs: Insertions of One or More Lines



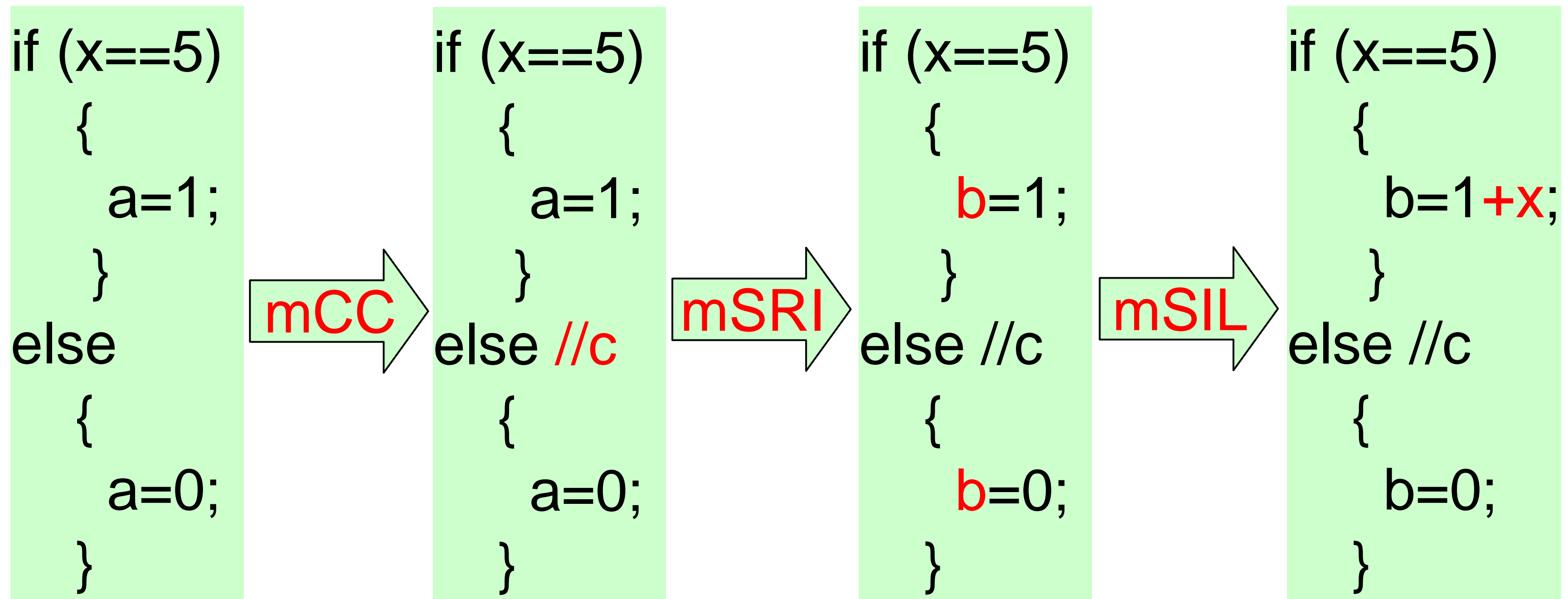
Mutation Operators for Cloning

- For Type IV Clones

Name	Random Editing Activities
mRDs	Reordering of declaration statements
mROS	Reordering of other statements (Data-dependent and/or in-dependent statements)
mCR	Replacing one type of control by another

Mutation Operators for Cloning

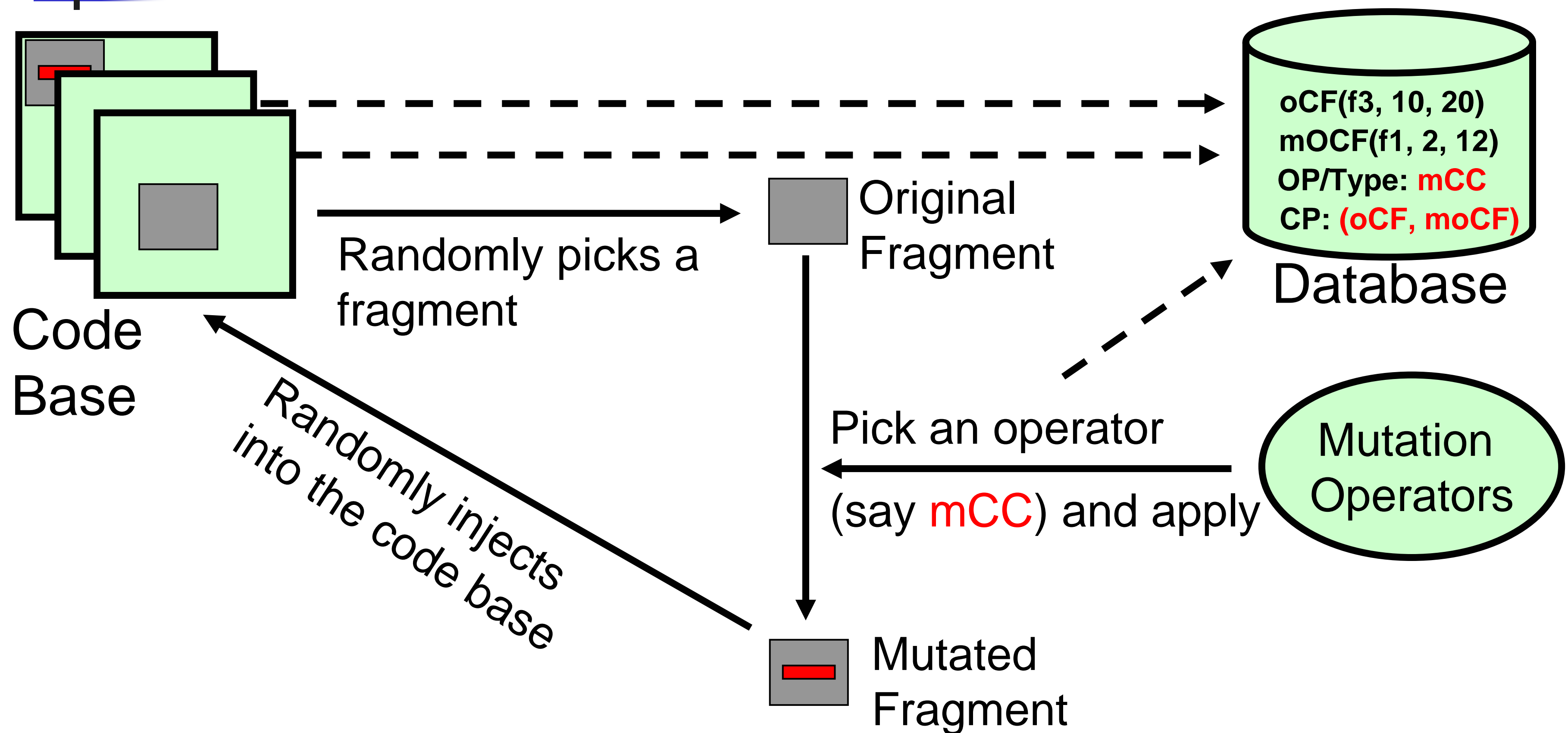
- Combinations of mutation operators



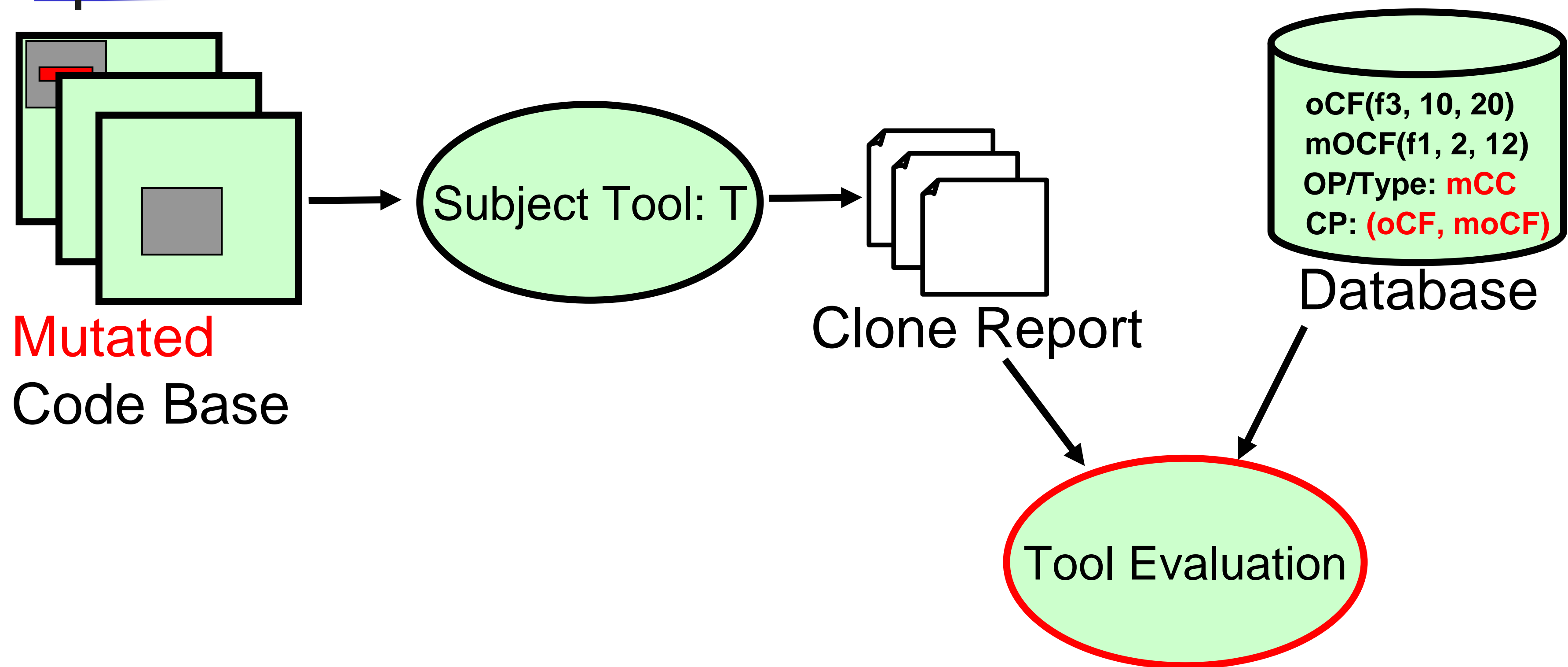
The Evaluation Framework

- Generation Phase
  - Create artificial clone pairs (using mutation analysis)
  - Injecte to the code base
- Evaluation Phase
  - How well and how efficiently the known clone pairs are detected by the tool(s)

Generation Phase: Base Case



Evaluation Phase: Base Case



Unit Recall

- For known clone pair, (oCF, moCF), of type mCC, the unit recall is:

$$UR_{T}^{(oCF,\ moCF)} = \begin{cases} 1, & \text{if (oCF, moCF) is killed by T} \\ & \text{in the mutated code base} \\ 0, & \text{otherwise} \end{cases}$$

Definition of killed(oCF, moCF)

- (oCF, moCF) has been detected by the subject tool, T
  - That is a clone pair, (CF1, CF2) detected by T matches or subsumes (oCF, moCF)
  - We use source coordinates of the fragments to determine this
  - First match the full file names of the fragments, then check for begin-end line numbers of the fragments within the files

Unit Precision

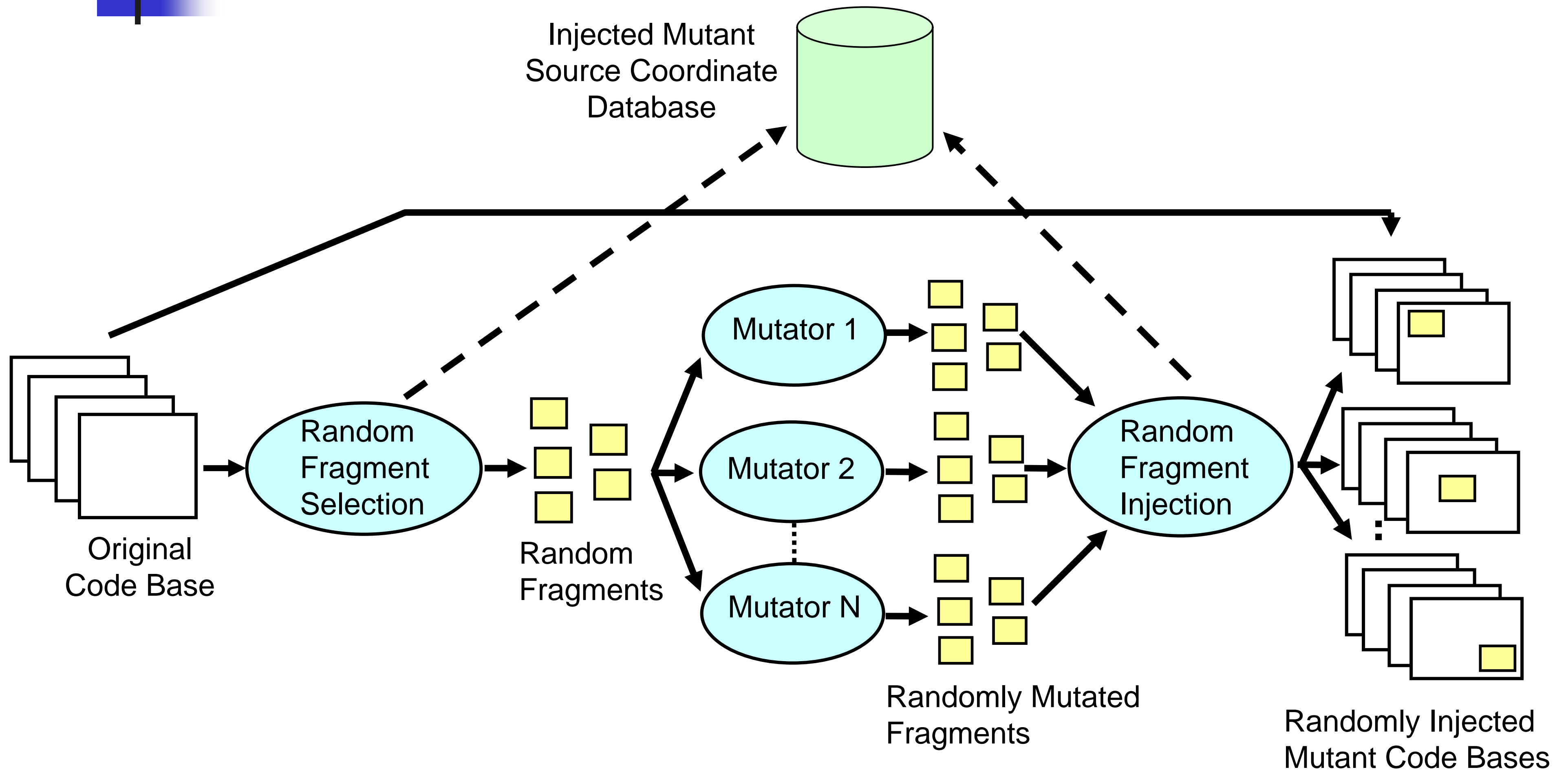
- Say, for moCF, T reports k clone pairs,
  - (moCF, CF1), (moCF, CF1),…, (moCF, CFk)
- Also let, v of them are valid clone pairs, then
  - For known clone pair, (oCF, moCF), of type mCC, the unit precision is:

$$UP_{T}^{(oCF,\ moCF)} = \frac{\mathbf{v}\text{: Total no of valid clone pairs with moCF}}{\mathbf{k}\text{:Total no of clone pairs with moCF}}$$

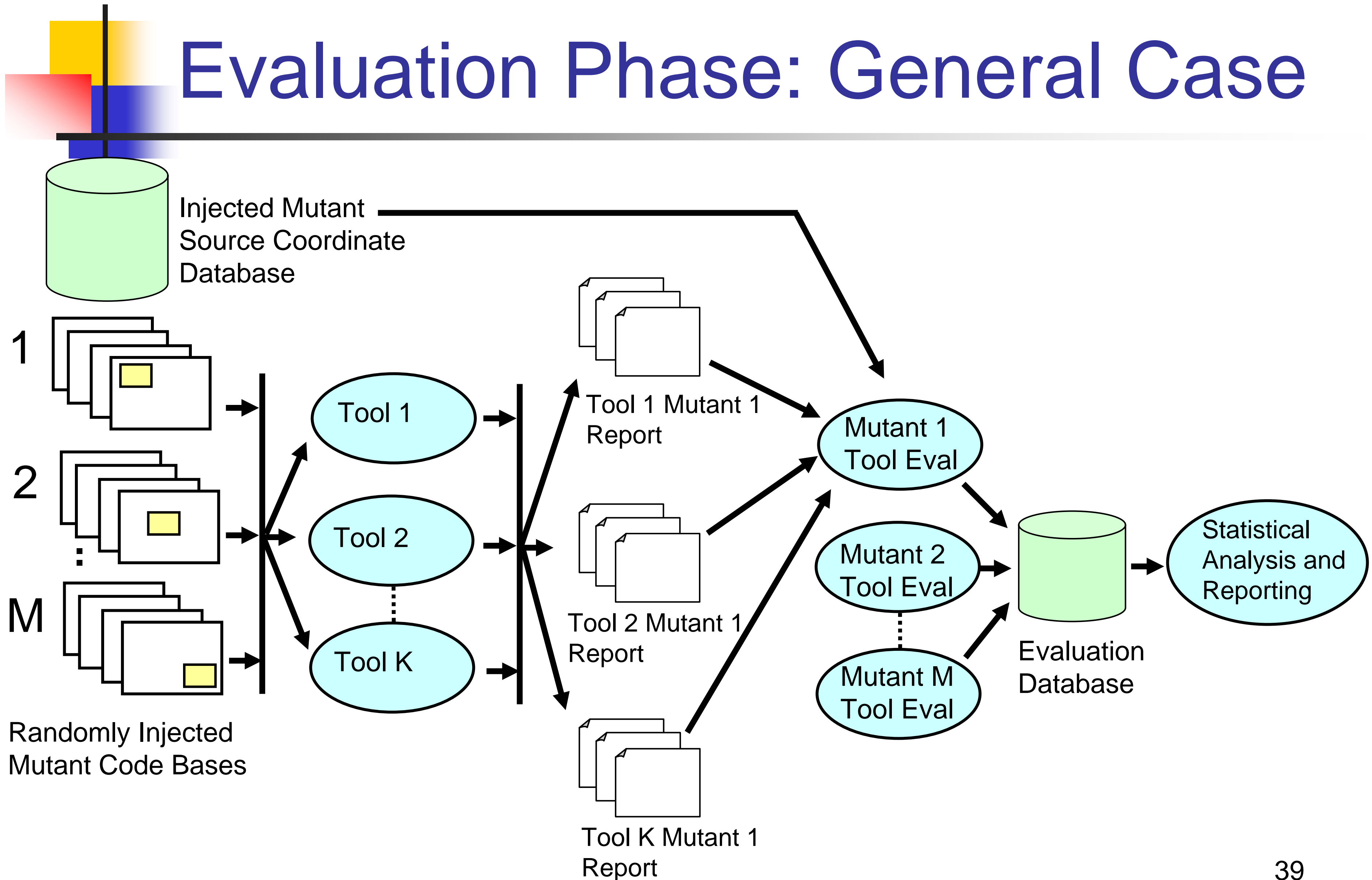
Automatic Validation of Known Clone Pairs

- Built a clone pair validator based on NICAD (Roy and Cordy ICPC'08)
- Unlike NICAD, it is not a clone detector
  - It only works with a specific given clone pair
  - It is aware of the mutation operator applied
  - Depending on the inserted clone, detection parameters are automatically tuned

Generation Phase: General Case



Evaluation Phase: General Case



Recall

- With mCC, m fragments are mutated and each injected n times to the code base

$$R_T^{mCC} = \frac{\sum_{i=1}^{m * n} UR_T^{(oCF_i, moCF_i)}}{m * n}$$

$$R_T^{Type\ I} = \frac{\sum_{i=1}^{(m * n) * (3 + 4)} UR_T^{(oCF_i, moCF_i)}}{(m * n) * (3 + 4)}$$

Overall Recall

- l clone mutation operators and c of their combinations applied n times to m selected code fragments, so

$$R_T^{overall} = \frac{\sum_{i=1}^{(m * n) * (l + c)} UR_T^{(oCF_i,\, moCF_i)}}{(m * n) * (l + c)}$$

Precision

- With mCC, m fragments are mutated and each injected n times to the code base

$$P_{T}^{mCC} = \frac{\sum_{i=1}^{m*n} \mathbf{v}_i}{\sum_{i=1}^{m*n} \mathbf{k}_i} \qquad P_{T}^{Type\ I} = \frac{\sum_{i=1}^{m*n*(3+4)} \mathbf{v}_i}{\sum_{i=1}^{m*n*(3+4)} \mathbf{k}_i}$$

Overall Precision

- l clone mutation operators and c of their combinations applied n times to m selected code fragments, so

$$P_T^{Overall} = \frac{\sum_{i=1}^{m * n * (l + c)} \mathbf{v}_i}{\sum_{i=1}^{m * n * (l + c)} \mathbf{k}_i}$$

Example Use of the Framework

- Select one or more subject systems
- Case one: Evaluate single tool
  - We evaluate NICAD [Roy and Cordy ICPC’08]
- Case two: Compare a set of tools
  - Basic NICAD [Roy and Cordy WCRE’08]
  - Flexible Pretty-Printed NICAD [Roy and Cordy ICPC’08]
  - and Full NICAD [Roy and Cordy ICPC’08]

Subject Systems

Language	Code Base	LOC	#Methods
C	GZip-1.2.4	8K	117
	Apache-httpd-2.2.8	275K	4301
	Weltab	11K	123
Java	Netbeans-Javadoc	114K	972
	Eclipse-jdtcore	148K	7383
	JHotdraw 5.4b	40K	2399

Recall Measurement

Clone Type	Standard Pt-Printing	Flexible Pt-Printing	Full NICAD
Type I	100%	100%	100%
Type II	29%	27%	100%
Type III	80%	85%	100%
Type IV	67%	67%	77%
Overall	84%	87%	96%

Precision Measurement

Clone Type	Standard Pt-Printing	Flexible Pt-Printing	Full NICAD
Type I	100%	100%	100%
Type II	94%	94%	97%
Type III	85%	81%	96%
Type IV	81%	79%	89%
Overall	90%	89%	95%

Other Issues

- Time and memory requirements
  - Can report fine-grained comparative timing and memory requirements for subject tools
- Scalability of the framework
  - Can work subject system of any size, depending on the scalability of the subject tools
  - Uses multi-processing to balance the load
- Adapting tools to the framework
  - The subject tool should run in command line
  - Should provide textual reports of the found clones

Related Work: Tool Comparison Experiments

- Baily and Burd SCAM’02
  - 3 CDs + 2 PDs
- Bellon et al. TSE’07
  - Extensive to-date
  - 6 CDs, C and Java systems
- Koschke et al. WCRE’06
- Rysselberge and Demeyer ASE’04

Related Work: Single Tool Evaluation Strategies

- Text-based
  - Only example-based evaluation, no precision and recall evaluations except NICAD
- Token-based
  - CP-Miner, for precision by examining 100 randomly selected fragments
- AST-based
  - Cpdetector, in terms of both precision and recall (Type I & II).
  - Deckard, for precision with examing 100 segments
- Metrics-based
  - Kontogianis evaluated with IR-approach, system was oracled manually.
- Graph-based
  - Gabel et al., precision examined 30 fragments per experiment.

Conclusion and Future Work

- Existing evaluation studies have several limitations
  - Baker TSE’07, Roy and Cordy SCP’09/ICPC’08
- We provided a mutation/injection-based automatic evaluation framework
  - Evaluates precision and recall single tool
  - Compare tools for precision and recall
- Effectiveness of this framework has been shown comparing NICAD variants
- We are planning to conduct a mega tool comparison experiment with the framework

Acknowledgements

- Inspirations
  - Rainer Koschke for Dagstuhl seminar 2006
  - Stefan Bellon et al., for beginning the road to meaningful evaluation & comparison
- Tool & method authors
  - For useful answers to our questions and worries
- Anonymous referees & colleagues
  - For help in presenting, tuning and clarifying several of the papers of this work.

Questions?