universitätbonn ROOTS

MDSE Lab 2013 Results

– Presented 20.8. 2014 –

Priyanka Dank

2517008
priyankadank@gmail.com

Cor 112 – Repeated Conditional Tests

Normal Case

```
int x,y;
String result;
void dontfindRpc()
{
if (x == 10 || y == 10)
{
result = "No Repeated Conditions";
}
}
```

Problem Case

```
int x,y;
String result;
void dontfindRpc()
{
if (x == 10 || x == 10)
{
result = “Repeated Conditional Test";
}
}
```

Background

- Conditions are different as variables used are different

Problems

- Conditions are the same
- Compiler does not compare the first condition with rest of the conditions in the statement.

Goal = Detect repetitive conditions!

Analysis

find_repetitive_condition

Step1: Detect the Conditions (if/while/dowhile)

- Detect all possible types of the conditions with all possible conditional operators

Possible types of the condition
if
while
dowhile

Possible conditional operators
AND (&&)
OR (||)
bitwise AND (&)
bitwise OR (|)

```
if (x == 10 || y==20 || x==10 || z == 30)
{
Some Code
}
```

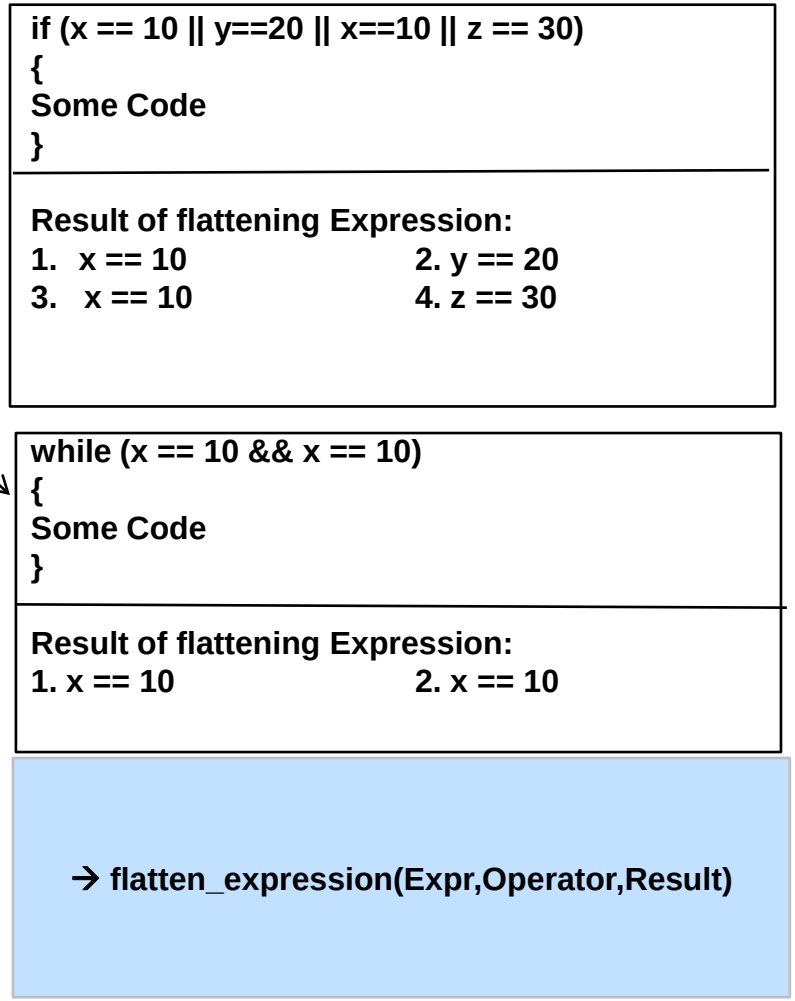
```
while (x == 10 && x == 10)
{
Some Code
}
```

```
do
{
Some Code
}
 while (x == 10 || x == 10);
```

→ **conditional_statement(Condition)**

Step 2: Flatten the conditions

- Two patterns of the condition
  - Conditions with many siblings
  - Conditions with one sibiling
- Need to flatten the whole expression
- Can be achieved by recursively calling the flatten_expression predicate for left hand side conditions and for right hand side conditions of the operator



Step 2: Flatten the conditions

flatten_expression(Expr,Operator,Result)

Top Level Call : second argument is free since everything is needed to be flattened

flatten_expression(+Expr,- Operator,? Result)

Recursive Call : second argument is bound since subconditions of the same operator to be flattened.

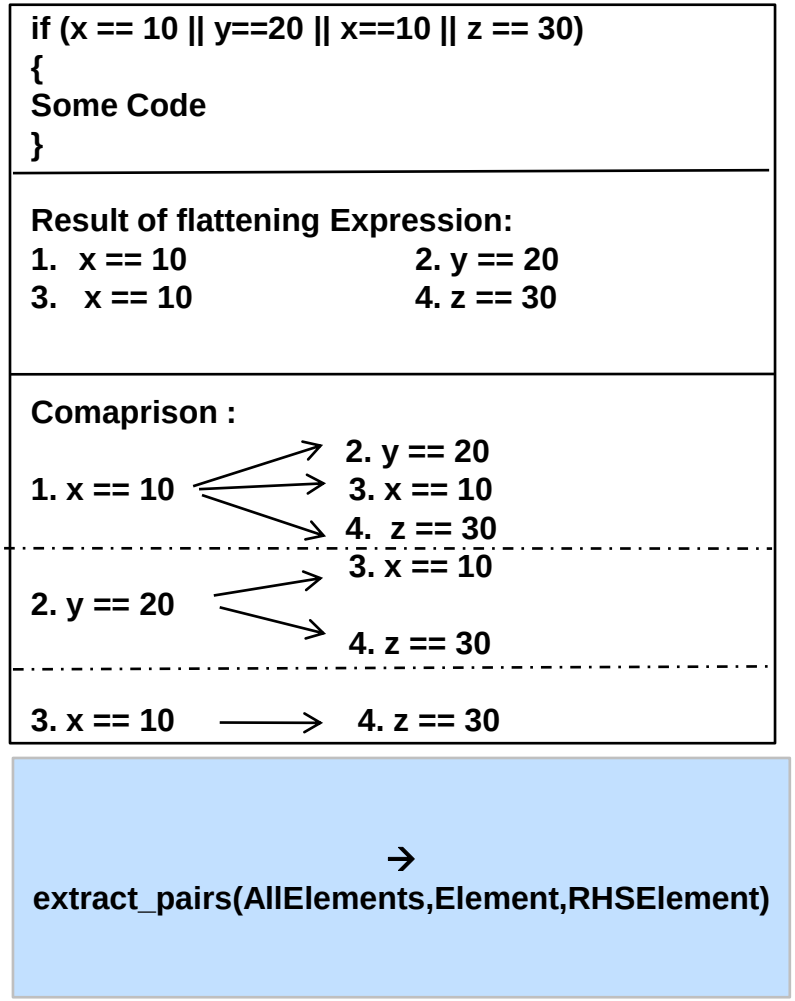
flatten_expression(+Expr,+Operator,? Result)

```
while ( (x == 10 || this.x == 20) && (x == 10 || this.x == 30) )
{
result = “No Repeated Contions";
}
```

Thus, false positives are avoided

Step 3: Consider the elements for comparison

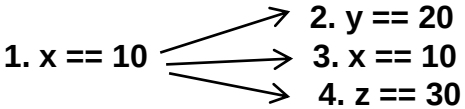
- Need to compare all elements of the condition
- Consider the leftmost element and pair it with each element further right
- Continue recursively with extracting pairs from the elements on the right



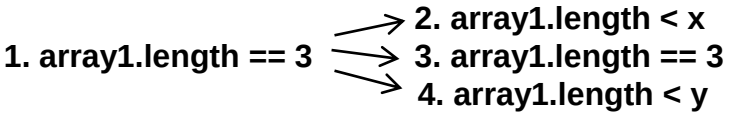
Step 4: Compare the extracted pairs

- Need to compare
  - Operator
  - Left-hand side expression
  - Right-hand side expression
- Need to consider different structures of left hand side and right hand side expressions
- Need to develop generalize predicate based on the structure

Comaprison example I:



Comaprison example II:



- **compare operationT element to compare operators**
- **recursively comapre left hand side and right hand side structures**

→ **same_structure(FirstElement,SecondElement)**

Implementation: Covered Variants

- Conditions
  - if
  - while
  - do...while
- Patterns of the Condition
  - with many siblings
  - with one sibling
- Conditional Operators
  - AND (&&)
  - OR (||)
  - bitwise AND (&)
  - bitwise OR (|)
- Comparision Operators
  - '>'
  - '<'
  - '>='
  - '<='
  - '=='
  - '!='
- Structures for the comparison
  - Operation
  - Field access
  - Parameter
  - Literal

Transformation

delete_repetitive_condition

Step 1 : Observe the associated elements

- Repetitive condition is already detected using analysis
- Need to find who is associated with the repetitive condition i.e. Sibling
- Need to find who is referring to the repetitive condition i.e. parent

```
if (x == 10 || y==20 || x==10 || z == 30)
{
Some Code
}
```

```
if (x == 10 || y==20 || x==10 || z == 30)
{
Some Code
}
```

```
if (x == 10 || y==20 || x==10 || z == 30)
{
Some Code
}
```

→ operationT(_,_,_,[Sibling,RC],_,_)

→ ast_parent(RepetitiveCondition,Parent)

Step 2 : Delete the repetitive condition

Possible Options

- Delete the repetitive condition
- Problem: Undeleted references
- Delete the reference to the repetitive condition as well.
- Problem: Sibling will be deleted as well. Loss of Data

```
if (x == 10 || y==20 || x==10 || z == 30)
{
Some Code
}
```

```
if (x == 10 || y==20 ||  || z == 30)
{
Some Code
}
```

```
if (z == 30)
{
Some Code
}
```

Step 2 : Delete the repetitive condition

- Need to find grandparent of the repetitive condition.
- Need to create the lnik between grandparent and siblings to overcome problems : undeleted references and loss of data
- Now the repetitive condition and its parent can be deleted

```
if (x == 10 || y==20 || x==10 || z == 30)
{
Some Code
}
```

```
if (x == 10 || y==20 || x==10 || z == 30)
{
Some Code
}
```

```
if (x == 10 || y==20 || x==10 || z == 30)
{
Some Code
}
```

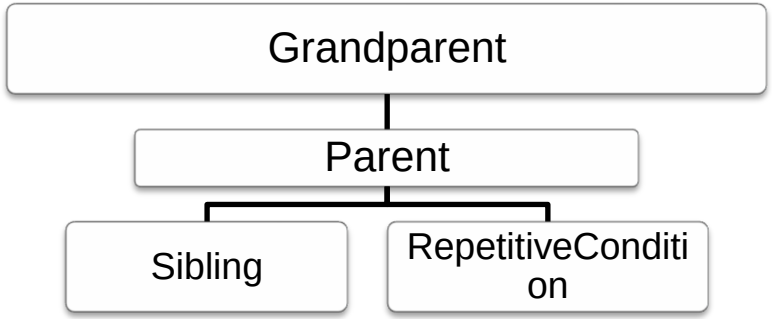
→ replace_value_in_term(GP, P, Sibling, NewGP)

→ deepDelete(RepetitiveCondition)
→ delete(operationT(Parent,_,_,_,_,_))

Overview of Transformation

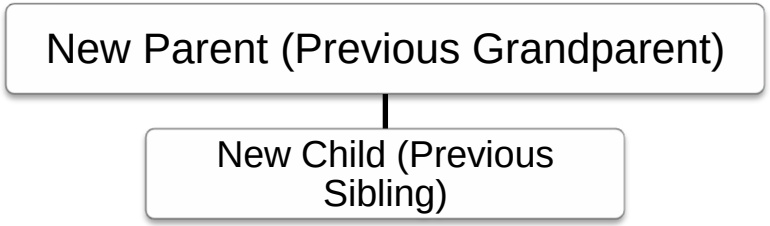
Before Transformation

```
if (x == 10 || y==20 || x==10 || z == 30)
{
Some Code
}
```



After Transformation

```
if (x == 10 || y==20 || z == 30)
{
Some Code
}
```



Evaluation

find_repetitive_condition

JTransformer Detector over FindBugs

- FindBugs detector (Repeated Conditional Tests) can not detect the repetitive condition which uses local vaiables but the developed JTransformer detector (find_repetitive_condition) can.

```
void findRpc_inIf_onLocal()
{
int x=0,y=0;
if (x == 10 || x == 10)
{
result = "Error Repeated Contional
Test";
}
if(x==y || x==y)
{
}
}
```

Limitations

- Both FindBugs detector (Repeated Conditional Tests) and the developed JTransformer detector (find_repetitive_condition) can not detect the repetitive conditions containing subconditions

```
void find_repeated_subconditions(int x)
{
while ( (x == 10 || this.x == 20) && (x ==
10 || this.x == 20) )
{
result = "Error  Repeated Contional
Test";
}
}
```

Evaluation – Used Benchmark Projects

Benchmark		Startup time	
Project	**Version**	**Initial Factbase Creation**	**Restart From Cache**
MDSE	commit number		
Apache Tomcat Container	v5.1.15	90.992	65.130
Argo UML	v1.2	220.29	202.705
AWT	v1.14	114.778	100.507
JHotDraw	v6.0	90.618	81.959

Evaluation – Factbase Statistics

Benchmark		KB	# of PEFs			
Project	Version	Prolog Process	Total	Classes	Methods	Fields
MDSE	1.0	25.281	27907	575	4425	941
JHotDraw	v6.0	18.021	138709	1879	15997	4858
Apache Tomcat Container	v5.1.15	19.208	178138	1739	16775	7575
AWT	v1.14	15.864	297206	1774	15359	6424
Argo UML	v1.2	22.328	464287	3899	32419	8756

Evaluation – Precision and Recall

FindBugs

Benchmark		Repeated Conditional Tests				
Project	Version	Correct	Wrong	Missed	Precision	Recall
MDSE	1.0	5	0	6	100%	45.4%
JHotDraw	v6.0	0	0	0	0%	0%
e Tomcat Container	v5.1.15	0	0	0	0	0
AWT	v1.14	0	0	0	0	0
Argo UML	v1.2	4	0	0	100%	100%

JTransformer

Benchmark		Find_repetitive_condition				
Project	Version	Correct	Wrong	Missed	Precision	Recall
MDSE	1.0	11	0	0	100%	100%
e Tomcat Container	v5.1.15	0	0	0	0	0
AWT	v1.14	0	0	0	0	0
Argo UML	v1.2	4	0	0	100%	100%

Evaluation – Speed

FindBugs

Benchmark		Miliseconds
Project	Version	find_repetitive_condition
MDSE	1.0	4400
JHotDraw	v6.0	1300
Apache Tomcat Container	v5.1.15	8500
AWT	v1.14	5300
Argo UML	v1.2	11800

JTransformer

Benchmark		Miliseconds
Project	Version	find_repetitive_condition
MDSE	commit number	15
JHotDraw	v6.0	10
Apache Tomcat Container	v5.1.15	16
AWT	v1.14	47
Argo UML	v1.2	45

Overview: All Detectors / Fixes

FindBugs detector	JTransformer detector	Examples
RpC_REPEATED_CONDITIONAL_TEST	Cor 112 - RpC: Repeated conditional tests	if (x == 0 || x == 0)
RV_RETURN_VALUE_IGNORED	Cor 111 - RV: Method ignores return value	dateString.trim();
VA_FORMAT_STRING_BAD_ARGUMENT	Cor 047 - FS: Format string placeholder incompatible with passed argument	System.out.printf("%d\n"," hello");
VA_FORMAT_STRING_BAD_CONVERSION	Cor 048 - FS: The type of a supplied argument doesn't match format specifier	String.format("%d","1");
VA_FORMAT_STRING_MISSING_ARGUMENT	Cor 052 - FS: Format string references missing argument	System.out.printf("%d");

THANK YOU