AUTOMATED BUG NEIGHBORHOOD ANALYSIS FOR IDENTIFYING INCOMPLETE BUG FIXES

Mijung Kim,* Saurabh Sinha,† Carsten Gorg,* Hina Shah,* Mary Jean Harrold,* and Mangala Gowri Nanda†

* Georgia Institute of Technology

† IBM Research – India

Supported by NSF under CCF-0429117, CCF-0541049, and CCF-0725202, and IBM by a Software Quality Innovation Faculty Award

Existing Techniques

- Automated techniques to detect Java runtime exceptions (e.g., null-pointer exceptions)
  - ESC/Java [Flanagan et al. PLDI 2002]
  - SALSA [Loginov et al. ISSTA 2008]
  - XYLEM [Nanda and Sinha ICSE 2009]
  - XYLEM w/ Stack trace [Sinha et al. ISSTA 2009]

Existing Techniques

- Automated techniques to detect Java runtime exceptions (e.g., null-pointer exceptions)

Limitations
Techniques don't identify whether and how bugs are fixed

Existing Techniques

- Automated techniques to detect Java runtime exceptions (e.g., null-pointer exceptions)

Limitations
Techniques don't identify whether and how bugs are fixed

- Research that has investigated bug fixes
  - Evaluating static analysis defect warnings on production software [Ayewah, et al. PASTE 2007]
  - Tracking defect warnings across versions [Spacco, Hovemeyer, and Pugh MSR 2006]

Existing Techniques

- Automated techniques to detect Java runtime exceptions (e.g., null-pointer exceptions)

Limitations
Techniques don't identify whether and how bugs are fixed

- Research that has investigated bug fixes

Limitations
Techniques don't identify whether attempted bug fixes are complete

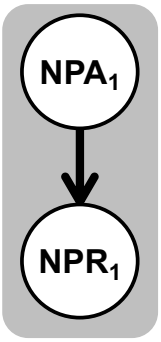
Incomplete Bug Fixes

- NPA : Null-Pointer Assignment
- NPR: Null-Pointer deReference

Incomplete Bug Fixes

- NPA : Null-Pointer Assignment
- NPR: Null-Pointer deReference

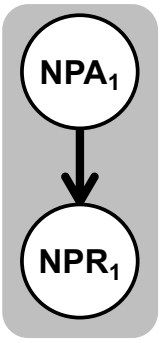
```
   foo(int i, j) {
[1]   x = null; // NPA1
[2]   if ( j > 10 ) {

[3]       x.m1(); // NPR1
[4]       x.m2();
      } else {
[5]       x.m3();
[6]       x.m4();
      }
   }
```



Incomplete Bug Fixes

- NPA : Null-Pointer Assignment
- NPR: Null-Pointer deReference

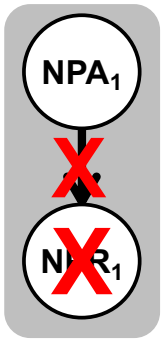
```
   foo'(int i, j) {
[1]   x = null; // NPA1
[2]   if ( j > 10 ) {
[2a]     if (x != null) // FIX
[3]         x.m1(); // NPR1
[4]       x.m2();
      } else {
[5]       x.m3();
[6]       x.m4();
      }
   }
```



Incomplete Bug Fixes

- NPA : Null-Pointer Assignment
- NPR: Null-Pointer deReference

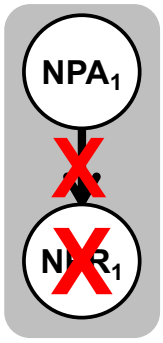
```
   foo'(int i, j) {
[1]   x = null; // NPA1
[2]   if ( j > 10 ) {
[2a]     if (x != null) // FIX
[3]          x.m1(); // NPR1
[4]        x.m2();
      } else {
[5]        x.m3();
[6]        x.m4();
      }
   }
```



Incomplete Bug Fixes

- NPA : Null-Pointer Assignment
- NPR: Null-Pointer deReference

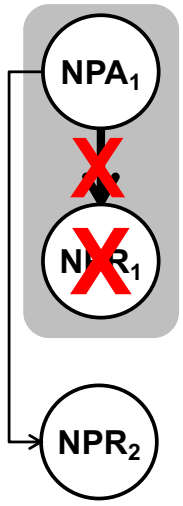
```
   foo'(int i, j) {
[1]   x = null; // NPA1
[2]   if ( j > 10 ) {
[2a]     if (x != null) // FIX
[3]          x.m1(); // NPR1
[4]        x.m2(); // NPR2
      } else {
[5]        x.m3();
[6]        x.m4();
      }
   }
```



Incomplete Bug Fixes

- NPA : Null-Pointer Assignment
- NPR: Null-Pointer deReference

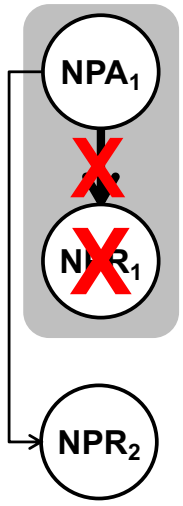
```
   foo'(int i, j) {
[1]   x = null; // NPA1
[2]   if ( j > 10 ) {
[2a]     if (x != null) // FIX
[3]          x.m1(); // NPR1
[4]       x.m2(); // NPR2
      } else {
[5]       x.m3();
[6]       x.m4();
      }
   }
```



Incomplete Bug Fixes

- NPA : Null-Pointer Assignment
- NPR: Null-Pointer deReference

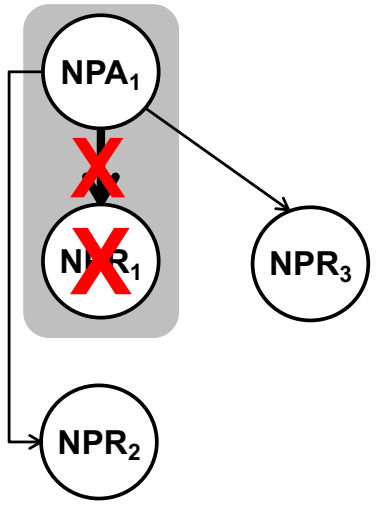
```
   foo'(int i, j) {
[1]   x = null; // NPA1
[2]   if ( j > 10 ) {
[2a]     if (x != null) // FIX
[3]          x.m1(); // NPR1
[4]       x.m2(); // NPR2
      } else {
[5]       x.m3(); // NPR3
[6]       x.m4();
      }
   }
```



Incomplete Bug Fixes

- NPA : Null-Pointer Assignment
- NPR: Null-Pointer deReference

```
   foo'(int i, j) {
[1]   x = null; // NPA1
[2]   if ( j > 10 ) {
[2a]     if (x != null) // FIX
[3]          x.m1(); // NPR1
[4]       x.m2(); // NPR2
      } else {
[5]       x.m3(); // NPR3
[6]       x.m4();
      }
   }
```

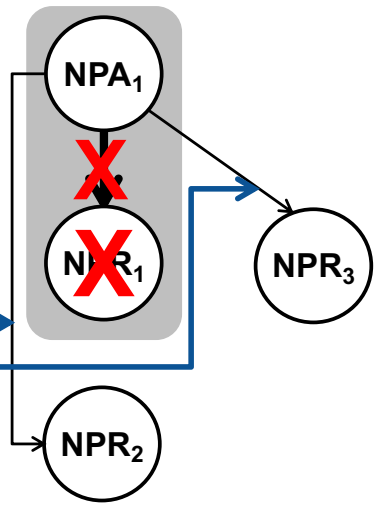


Incomplete Bug Fixes

- NPA : Null-Pointer Assignment
- NPR: Null-Pointer deReference

```
   foo'(int i, j) {
[1]   x = null; // NPA1
[2]   if ( j > 10 ) {
[2a]     if (x != null) // FIX
[3]          x.m1(); // NPR1
[4]       x.m2(); // NPR2
      } els
[5]       x
[6]       x
      }
   }
```

Potential unfixed pairs related to NPA_1
→ **Incomplete fix**



Our Work

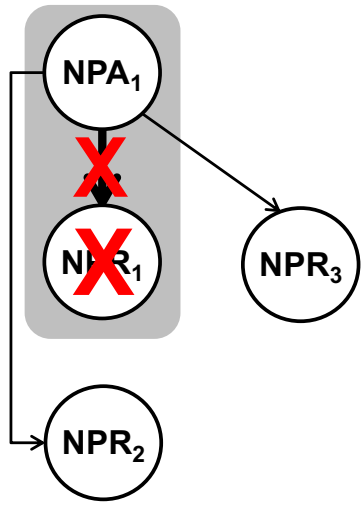
- Bug neighborhood analysis
  - Computes potential unfixed (NPA, NPR) pairs
- Classification of attempted bug fixes as complete or incomplete
- Empirical studies using open-source and commercial software
  - Neighborhood can be large and complex
  - Attempted bug fixes can be incomplete

Outline

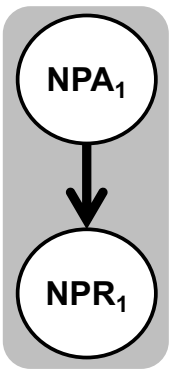
- Bug Neighborhoods (BN)
- Technique
- Empirical Evaluation
- Conclusion

Bug Neighborhood (BN)

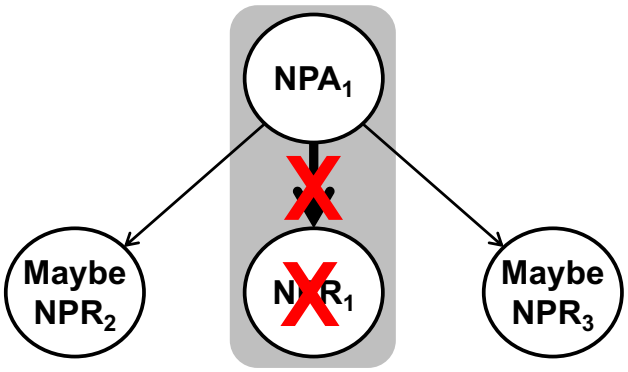
- Potential unfixed (NPA, NPR) pairs related to NPA_1 and NPR_1
- BN is induced by 4 types of NPAs and NPRs identified by our approach
  - Maybe NPR
  - Forward NPR
  - Maybe NPA
  - Backward NPA



Maybe NPRs

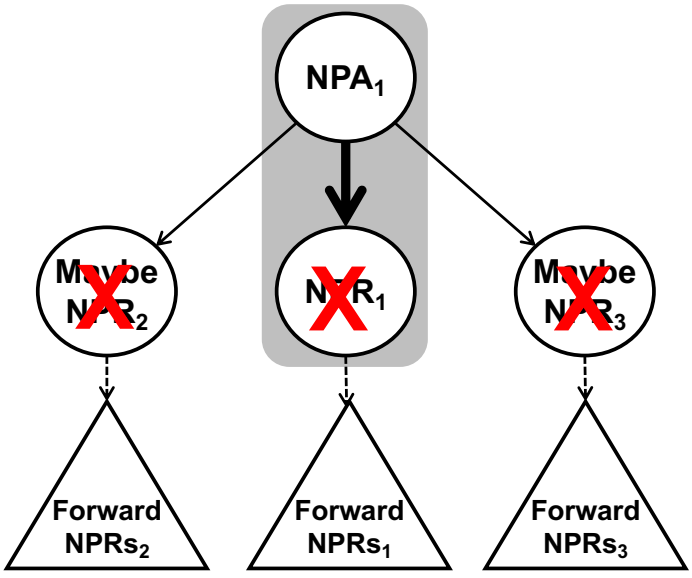


Maybe NPRs



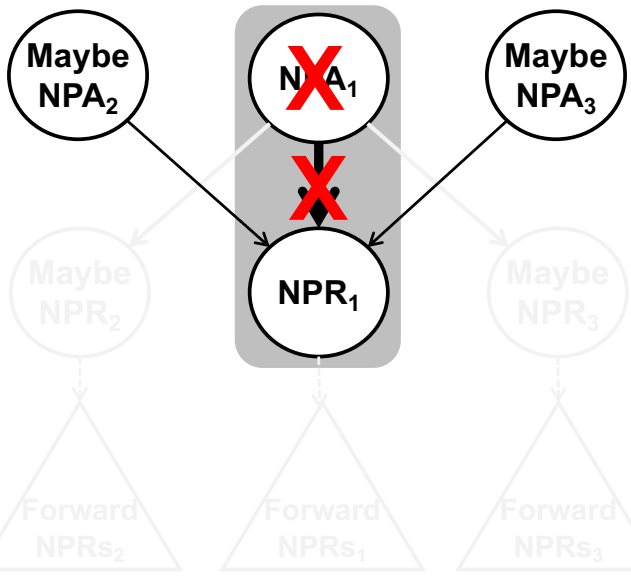
- NPRs that might be reached by NPA_1 in other executions

Forward NPRs



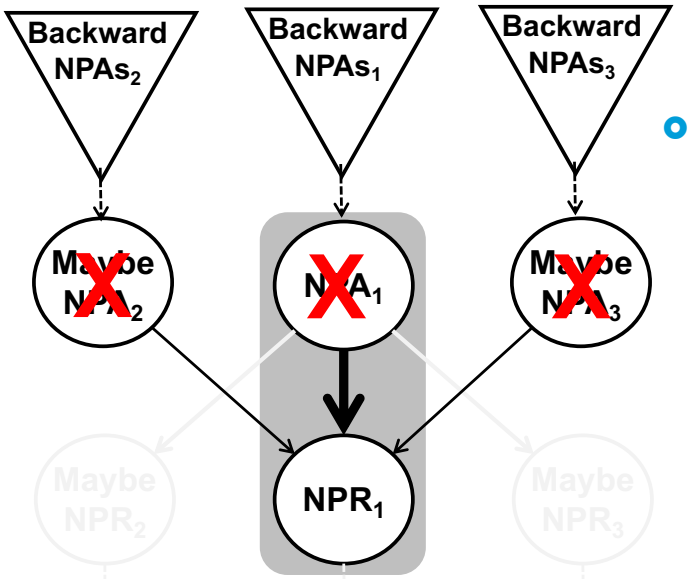
- NPRs that might be reachable from NPA_1 after NPR_1 and Maybe NPRs are fixed

Maybe NPAs



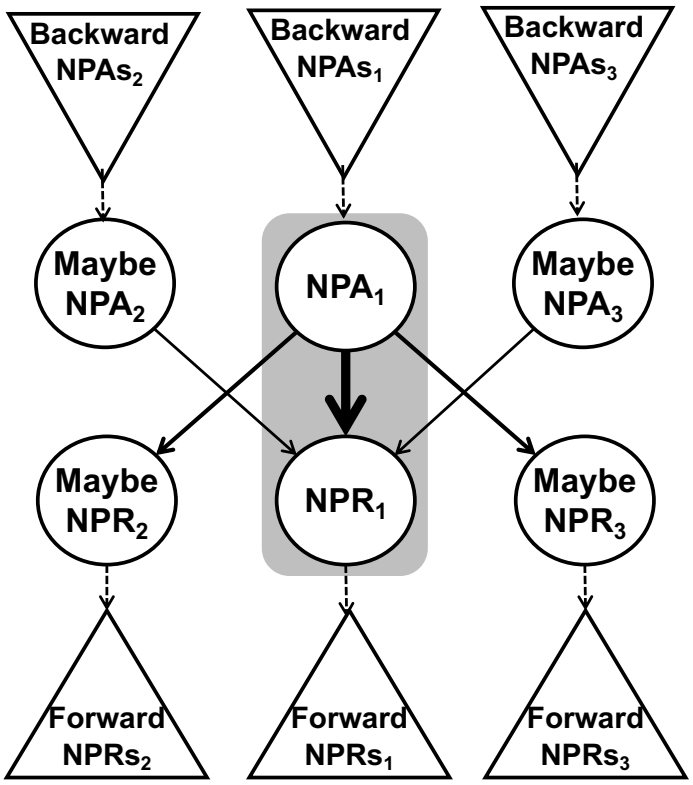
- NPAs that might reach NPR_1 in other executions

Backward NPAs



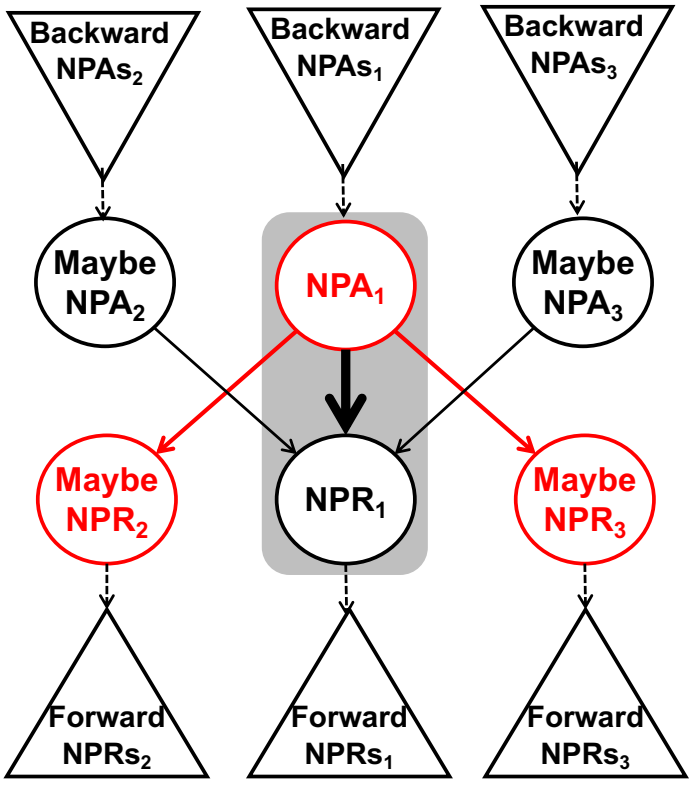
- NPAs that might reach NPR_1 after NPA_1 and Maybe NPAs are fixed

Bug Neighborhood (BN)



BN for (NPA_1, NPR_1)
(NPA1, Maybe NPR2)
(NPA1, Maybe NPR3)
(Maybe NPA2, NPR1)
(Maybe NPA3, NPR1)
(NPA1, Forward NPR1)
…
(NPA1, Forward NPRn)
(Backward NPA_1, NPR_1)
…
(Backward NPA_n, NPR_1)

Bug Neighborhood (BN)



BN for (NPA_1, NPR_1)
(NPA1, Maybe NPR2)
(NPA1, Maybe NPR3)
(Maybe NPA2, NPR1)
(Maybe NPA3, NPR1)
(NPA1, Forward NPR1)
…
(NPA1, Forward NPRn)
(Backward NPA_1, NPR_1)
…
(Backward NPA_n, NPR_1)

Definitions

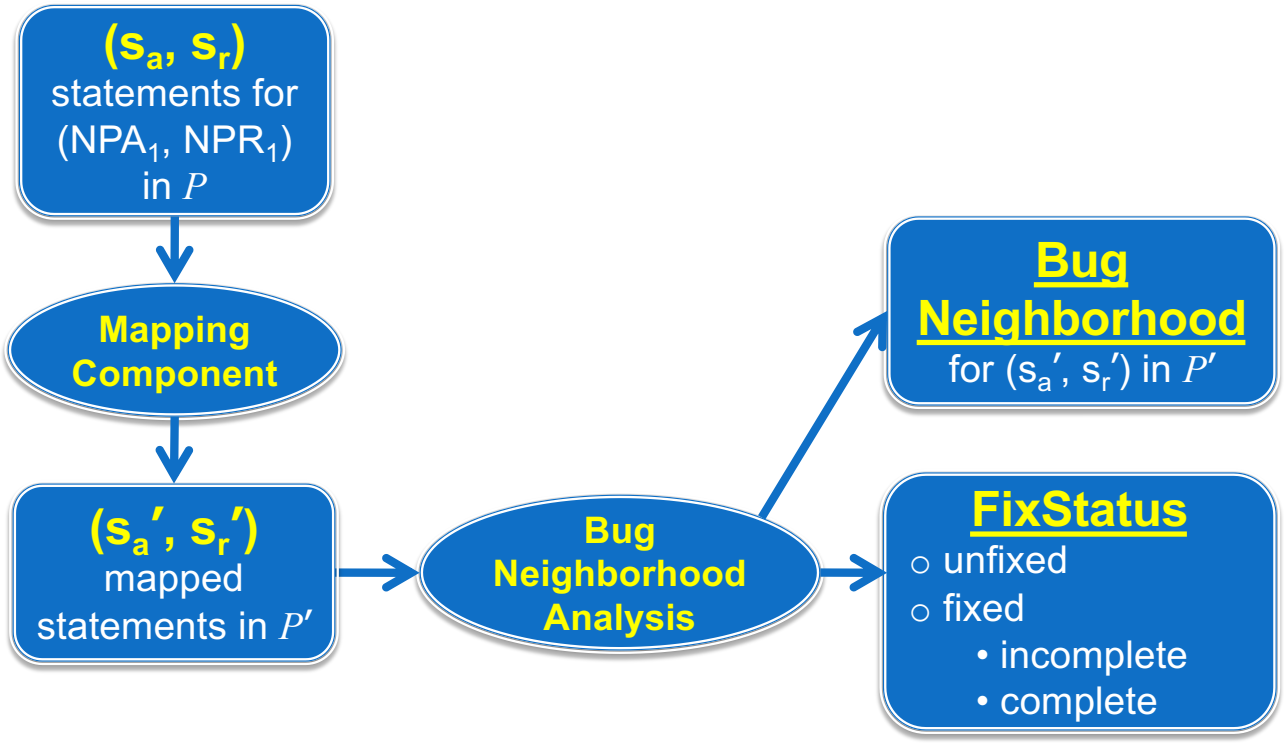
- BN for (NPA_1, NPR_1): set of potential unfixed pairs related to (NPA_1, NPR_1)
- Size of BN for (NPA_1, NPR_1): number of pairs in BN

Definitions

- BN for (NPA_1, NPR_1): set of potential unfixed pairs related to (NPA_1, NPR_1)
- Size of BN for (NPA_1, NPR_1): number of pairs in BN

- Complete fix: attempted fix (i.e. change from P to P') for (NPA_1, NPR_1) in which the BN for associated pair in P' is empty

Algorithm



Example

```
    foo( int i, j) {
1]    x = null; // NPA1
2]    if ( j > 10 ) {

3]       x.m1(); // NPR1
4]       x.m2(); // F-NPR
      } else {
5]       x.m3(); // M-NPR
6]       x.m4(); // F-NPR
    }
  }
```

Input
(S_a, S_r): (1, 3)

Example

```
  foo( int i, j) {
[1]    x = null; // NPA1
[2]    if ( j > 10 ) {

[3]       x.m1(); // NPR1
[4]       x.m2(); // F-NPR
       } else {
[5]       x.m3(); // M-NPR
[6]       x.m4(); // F-NPR
      }
   }
```

Input
(S_a, S_r): (1, 3)

```
  foo'( int i, j) {
[1]    x = null; // NPA1
[2]    if ( j > 10 ) {
[2a]      if (x != null) // FIX
[3]          x.m1(); // NPR1
[4]       x.m2(); // F-NPR
       } else {
[5]       x.m3(); // M-NPR
[6]       x.m4(); // F-NPR
      }
   }
```

(S_a', S_r'): (1, 3)

BugNeighborhoodAnalysis

- Reaching NPAs for 3: None

```
  foo'( int i, j) {
[1]    x = null; // NPA1
[2]    if ( j > 10 ) {
[2a]      if (x != null) // FIX
[3]          x.m1(); // NPR1
[4]       x.m2(); // F-NPR
       } else {
[5]       x.m3(); // M-NPR
[6]       x.m4(); // F-NPR
      }
   }
```

(S_a', S_r'): (1, 3)

BugNeighborhoodAnalysis

- Reaching NPAs for 3: None
- Reachable NPRs for 1: 4, 5, 6

BugNeighborhood

```
  foo'( int i, j) {
[1]    x = null; // NPA1
[2]    if ( j > 10 ) {
[2a]     if (x != null) // FIX
[3]        x.m1(); // NPR1
[4]      x.m2(); // F-NPR
       } else {
[5]      x.m3(); // M-NPR
[6]      x.m4(); // F-NPR
     }
   }
```

(S_a', S_r'): (1, 3)

BugNeighborhoodAnalysis

- Reaching NPAs for 3: None
- Reachable NPRs for 1: 4, 5, 6

BugNeighborhood

- (1,3) ∉ *BugNeighborhood*

  ∴ *FixStatus* = *fixed but,* ***incomplete***

```
  foo'( int i, j) {
[1]    x = null; // NPA1
[2]    if ( j > 10 ) {
[2a]     if (x != null) // FIX
[3]        x.m1(); // NPR1
[4]      x.m2(); // F-NPR
       } else {
[5]      x.m3(); // M-NPR
[6]      x.m4(); // F-NPR
     }
   }
```

(S_a', S_r'): (1, 3)

BugNeighborhoodAnalysis

- Reaching NPAs for 3: None
- Reachable NPRs for 1: 4, 5, 6

BugNeighborhood

- (1,3) ∉ *BugNeighborhood*

  ∴ *FixStatus* = *fixed but,* ***incomplete***

```
  foo'( int i, j) {
[1]    x = null; // NPA1
[2]    if ( j > 10 ) {
[2a]     if (x != null) // FIX
[3]        x.m1(); // NPR1
[4]      x.m2(); // F-NPR
       } else {
[5]      x.m3(); // M-NPR
[6]      x.m4(); // F-NPR
     }
   }
```

(S_a', S_r'): (1, 3)

Empirical Setup: Tools

- Finding (NPA, NPR) pairs: XYLEM
  [Nanda and Sinha ICSE2009]

- Integrated our analysis into XYLEM
  - Mapping pairs between P and P'
  - Identifying BN for a (S_a', S_r') pair in P'
    (No Backward NPAs implementation)

Empirical Setup: Subjects

Subject	Classes	Methods	Bytecode Instructions	(NPA, NPR) Pairs
Ant-1.6.0	1858	17204	443254	167
Lucene-2.2.0	381	2815	72691	86
Tomcat-4.1.27	260	4077	101075	97
App-A	278	3933	98225	63
App-B	169	1876	46286	119
App-C	2488	13746	340896	107

Study 1: BN Categories and Sizes

Goal: To examine characteristics of BNs

Method

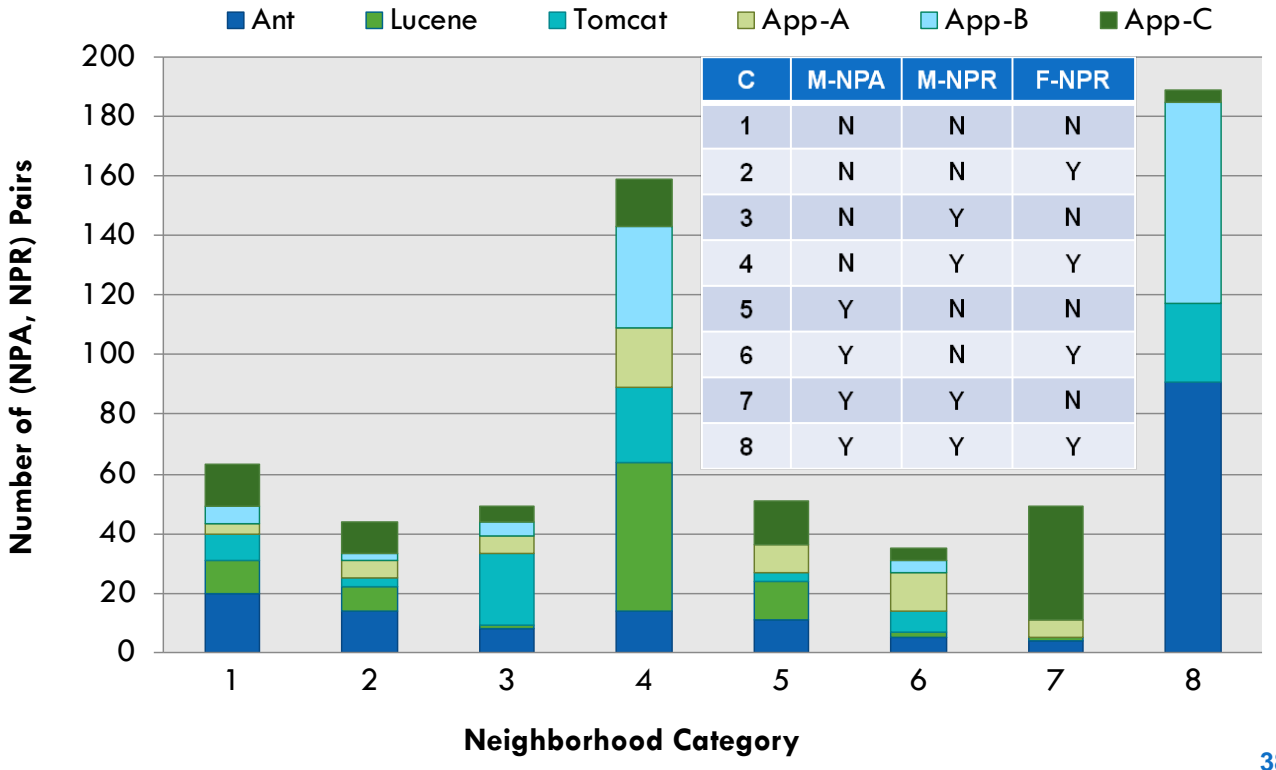
For each (NPA, NPR)

- Compute BN and BN size
- Classify
  - BN in 1 of 8 categories
  - BN size in 1 of 3 categories

BN Categories

Category	Maybe NPA Present	Maybe NPR Present	Forward NPR Present
1	NO	NO	NO
2	NO	NO	YES
3	NO	YES	NO
4	NO	YES	YES
5	YES	NO	NO
6	YES	NO	YES
7	YES	YES	NO
8	YES	YES	YES

Occurrences of Each BN Category

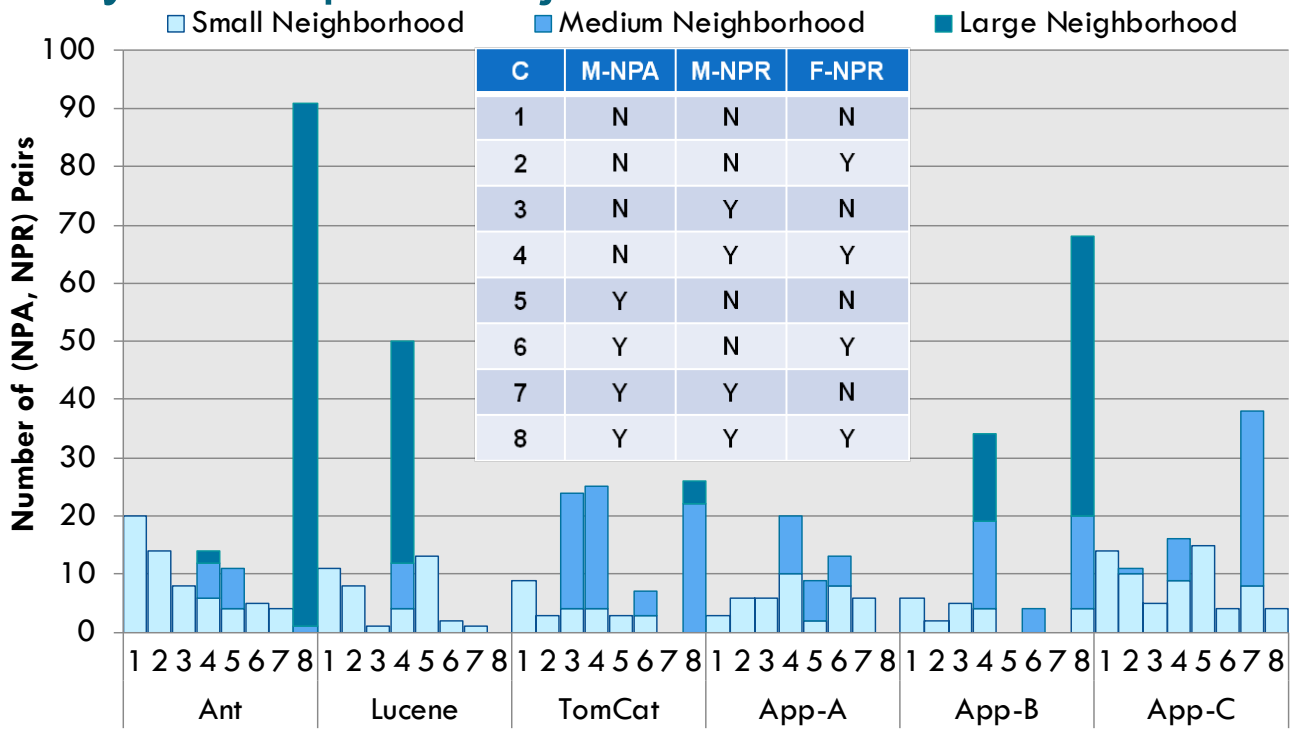


C	M-NPA	M-NPR	F-NPR
1	N	N	N
2	N	N	Y
3	N	Y	N
4	N	Y	Y
5	Y	N	N
6	Y	N	Y
7	Y	Y	N
8	Y	Y	Y

BN Size Categories

- Small: pairs ≤ 5
- Medium: $5 <$ pairs ≤ 15
- Large: pairs > 15

Occurrences of Each BN Category by Sizes per Subject



C	M-NPA	M-NPR	F-NPR
1	N	N	N
2	N	N	Y
3	N	Y	N
4	N	Y	Y
5	Y	N	N
6	Y	N	Y
7	Y	Y	N
8	Y	Y	Y

Neighborhood Categories per Subjects

Study 2: Completeness of Fixes

Goal: To investigate existence and frequency of incomplete fixes

Method

For each (NPA, NPR) in P'

- Verified mapping from P (manually)
- If mapping is accurate, compute BN for (NPA, NPR) pair (manually verified)

Completeness of Attempted Bug Fixes

Subject	# Incomplete Fixes / # Attempted Fixes	BN Categories of Incomplete Fixes	BN Sizes of Incomplete Fixes
Ant	4 / 26	C2, C4, C4, C5	1 x Small, 3 x Medium
Lucene	3 / 17	C4, C4, C4	1 x Small, 2 x Large
Tomcat	0 / 9	—	—
App-A	0 / 7	—	—

C	M-NPA	M-MPR	F-NPR
2	N	N	Y
4	N	Y	Y
5	Y	N	N

Future Work

- Implement Backward NPAs and precise mapping component
- Perform more experiments with more subjects
  - Correlation between incomplete fixes and BN category or size
  - Comparison of results from open-source projects and industrial projects
- Guide developers in addressing an incomplete fix and making it complete

Contributions

- New bug neighborhood analysis
  - Determines completeness of attempted bug fixes
- Technique that helps developers prevent incomplete fixes in new revisions
- Empirical studies that show
  - large and complex BNs do occur frequently
  - Attempted bug fixes are incomplete in practice

Q & A