Context-Based Detection of Clone-Related Bugs

Lingxiao Jiang, Zhendong Su, Edwin Chiu
University of California at Davis

Outline

- Introduction and samples of cloning errors
- Definitions of inconsistencies
- Inconsistency filters for bug detection
- Empirical evaluation
- Conclusion and future work

Code Clones are Common

- Replicating similar functionalities helps improve development productivity
- Many clone detection techniques exist
  - CCFinder: 29% cloned code in JDK
  - CP-Miner: 23% cloned code in the Linux kernel 2.6.6
- Clone removal may not always be beneficial
  - Kim, Sazawal, Notkin, ESEC/FSE 2005
  - Rajapakse, Jarzabek, ICSE 2007

Clone-Related Bugs

- Copy-paste may create many inconsistencies among code clones
- Inconsistencies among clones can be indications of potential defects

Clone-Related Bugs – Examples 1/3

```
TableItem item = Get…… ;


for (int i=0; i<colCnt; i++)
{
   Point start = new Point (item..... ;
   start =
       fTableViewer.getTable()……;
   ......
   if (start.x < point.x && end.x > point.x)
       return i;
}
```

```
TableItem item = Get..... ;
if (item != null)
{
  for (int i=0; i<colCnt; i++)
  {
      Point start = new Point (item..... ;
      start =
          fTableViewer.getTable()……;
      ......
      if (start.x < point.x && end.x > point.x)
          return i;
  }
}
```

Clone-Related Bugs – Examples 2/3

```
if ( length >= 9 &&
     strncmp (buffer, "EESOXSCSI", 9)
     == 0 )
{
     buffer += 9;
     length -= 9;

     if ( length >= 5 &&
          strncmp(buffer, "term=", 5) == 0)
     {
        ......
     }
     else
        ret = -EINVAL;
}
else
     ret = -EINVAL;
```

```
if ( length >= 11 &&
     strcmp (buffer, "CUMANASCSI2“)
     == 0)
{
     buffer += 11;
     length -= 11;

     if ( length >= 5 &&
          strncmp(buffer, "term=", 5) == 0)
     {
        ......
     }
     else
        ret = -EINVAL;
}
else
     ret = -EINVAL;
```

Clone-Related Bugs – Examples 3/3

```
if ( l_stride != NULL )
    mpz_cdiv_q ( X1, X1, l_stride->value.integer );
```

```
if ( l_stride != NULL )
    mpz_cdiv_q ( X2, X2, r_stride->value.integer );
```

Outline

- Introduction and samples of cloning errors
- Definitions of inconsistencies
- Inconsistency filters for bug detection
- Empirical evaluation
- Conclusion and future work

Types of Inconsistencies 1/3

Type-1:

- Given two clones, their surrounding contextual nodes are different types
- E.g., `function-def` vs. `if-statement`

```
TableItem item = Get…… ;


for (int i=0; i<colCnt; i++)
{
  Point start = new Point (item..... ;
  start =
      fTableViewer.getTable()……;
  ......
  if (start.x < point.x && end.x > point.x)
      return i;
}
```

```
TableItem item = Get..... ;
if (item != null)
{
  for (int i=0; i<colCnt; i++)
  {
     Point start = new Point (item..... ;
     start =
        fTableViewer.getTable()……;
      ......
      if (start.x < point.x && end.x > point.x)
        return i;
  }
}
```

Types of Inconsistencies 2/3

Type-2:

- Given two clones, their surrounding conditions are different
- E.g., `strncmp` vs. `strcmp`

```
if ( length >= 9 &&
    strncmp (buffer, "EESOXSCSI", 9) == 0 )
{
      buffer += 9;
      length -= 9;

      if ( length >= 5 &&
          strncmp(buffer, "term=", 5) == 0)
      {
         ......
      }
      else
         ret = -EINVAL;
}
else
      ret = -EINVAL;
```

```
if ( length >= 11 &&
    strcmp (buffer, "CUMANASCSI2“) == 0)
{
      buffer += 11;
      length -= 11;

      if ( length >= 5 &&
          strncmp(buffer, "term=", 5) == 0)
      {
         ......
      }
      else
         ret = -EINVAL;
}
else
      ret = -EINVAL;
```

Types of Inconsistencies 3/3

Type-3:

- Given two clones, the number of unique identifiers within them are different
- E.g., 6 vs. 7

```
if ( l_stride != NULL )
    mpz_cdiv_q ( X1, X1, l_stride->value.integer );
```

```
if ( l_stride != NULL )
    mpz_cdiv_q ( X2, X2, r_stride->value.integer );
```

Outline

- Introduction and samples of cloning errors
- Definitions of inconsistencies
- Inconsistency filters for bug detection
- Empirical evaluation
- Conclusion and future work

Inconsistency Filters

- Not all inconsistencies are created equal
- Many may not be bug indicators
  - Spatially close clones
  - Intentional (w/ big differences) clones

Inconsistency Filters - Examples 1/5

Contextual nodes are `loop` vs. `function-def`

for (x = A; x <= B; x++) **for (y = C; y<= D; y++)** **…more similar code…**	**for (y = C; y<= D; y++)** **…more similar code…**

Inconsistency Filters – Examples 2/5

Contextual nodes are `loop` vs. `if-statement`

for (x = A; x <= B; x++) **…similar code…**	if (y != D) **…similar code…**

Inconsistency Filters – Examples 3/5

Contextual conditions are significantly different

if (x <= B) **…similar code…**	if (y+z != C*D) **…similar code…**

Inconsistency Filters - Examples 4/5

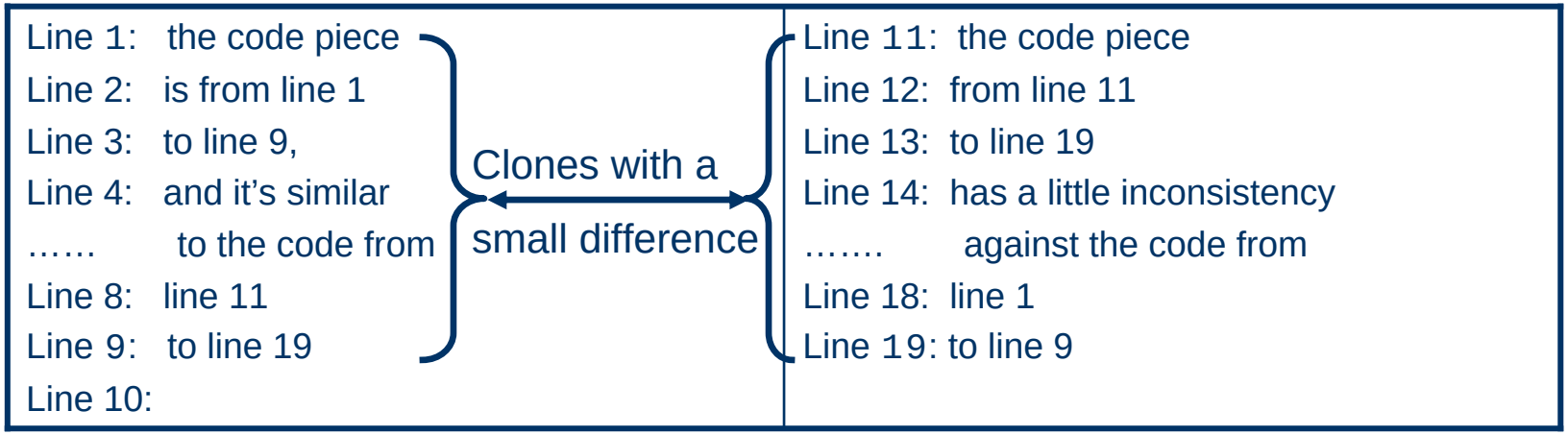
Identifiers within clones are significantly different

```
if ( data != NULL )
{
   data->in[ix] =
      read8(data, VT_REG1211);
   data->in_min[ix] =
      read8(data, VT_REG1211);
   data->in_max[ix] =
      read8(data, VT_REG1211);
   ……
}
```

```
If ( data != NULL )
{
   data->in[i] =
     read_data(client, ADM1025_REG);
   data->in_min[i] =
     read_data(client, ADM1025_MIN);
   data->in_max[i] =
     read_data(client, ADM1025_MAX);
   ……
}
```

Inconsistency Filters – Examples 5/5

Clones are spatially very close



Outline

- Introduction and samples of cloning errors
- Definitions of inconsistencies
- Inconsistency filters for bug detection
- Empirical evaluation
- Conclusion and future work

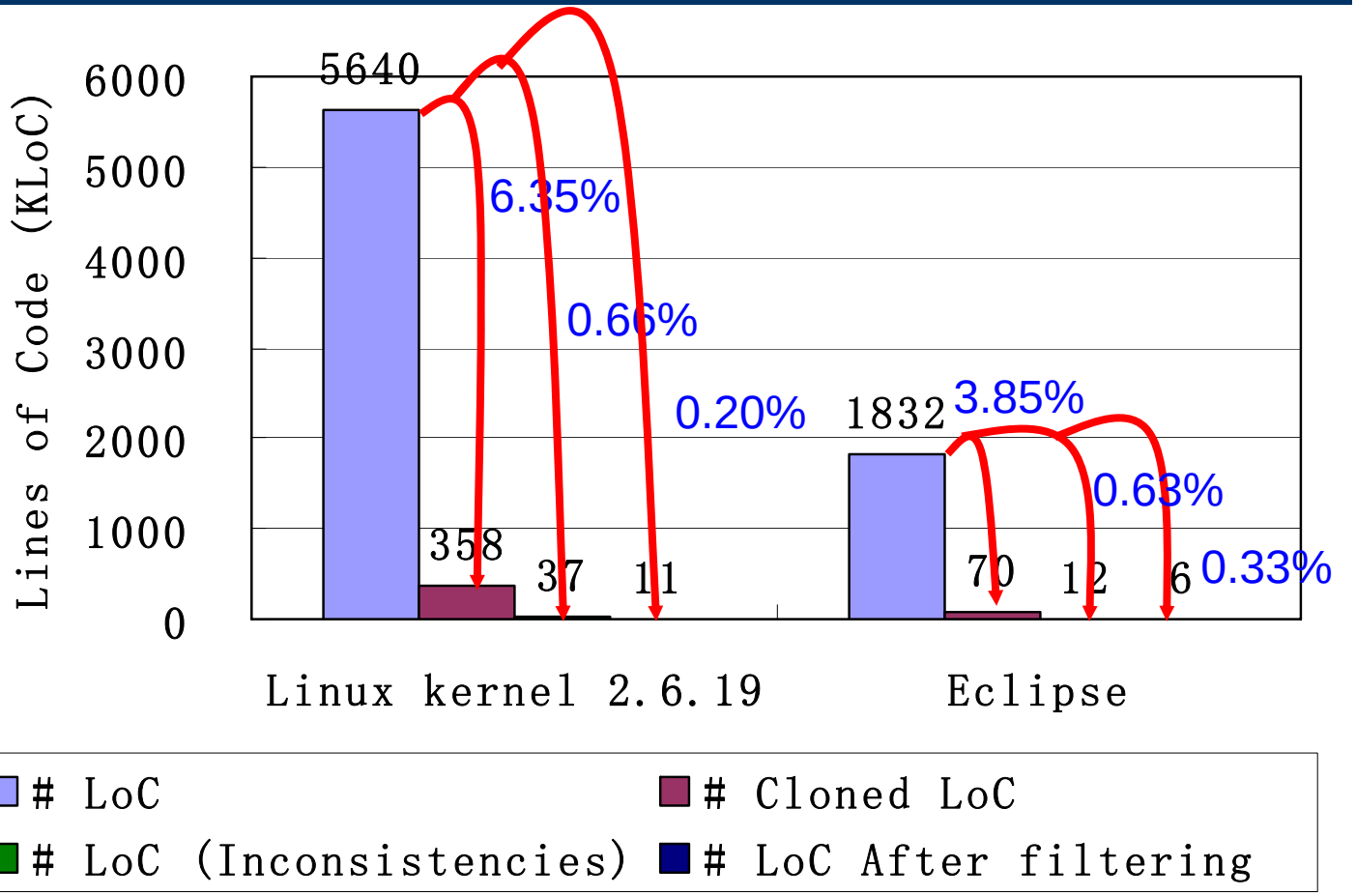
Empirical Evaluation

Application	# Files	# LoC	# Cloned LoC
Linux kernel 2.6.19 (C)	8733	5639833	358331
Eclipse (Java)	8320	1832332	70455

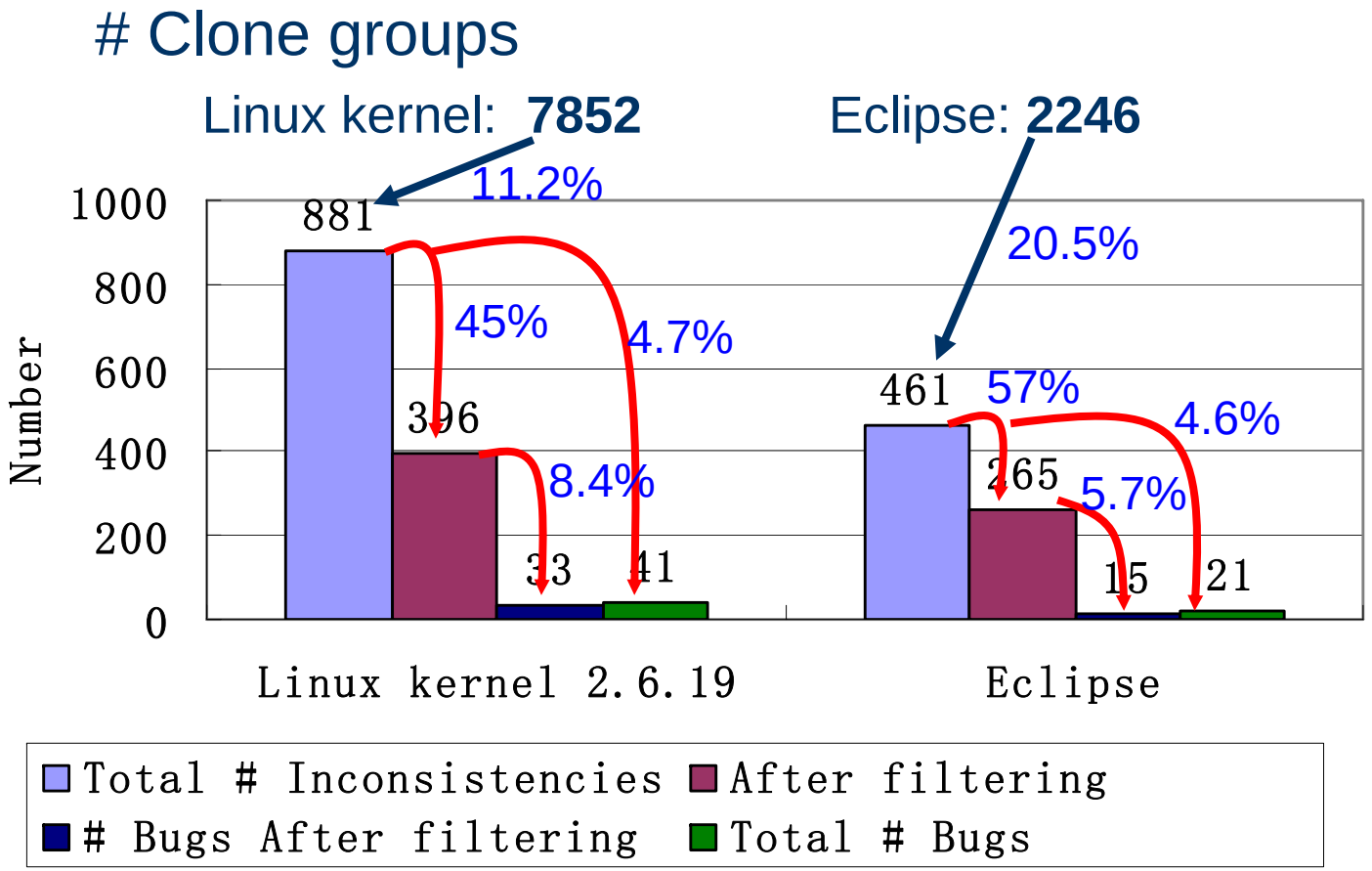
Based on DECKARD

- A tree-based, accurate, scalable clone detection tool
- Parameters for clone detection:
  - Similarity
  - Minimum token number
  - Stride

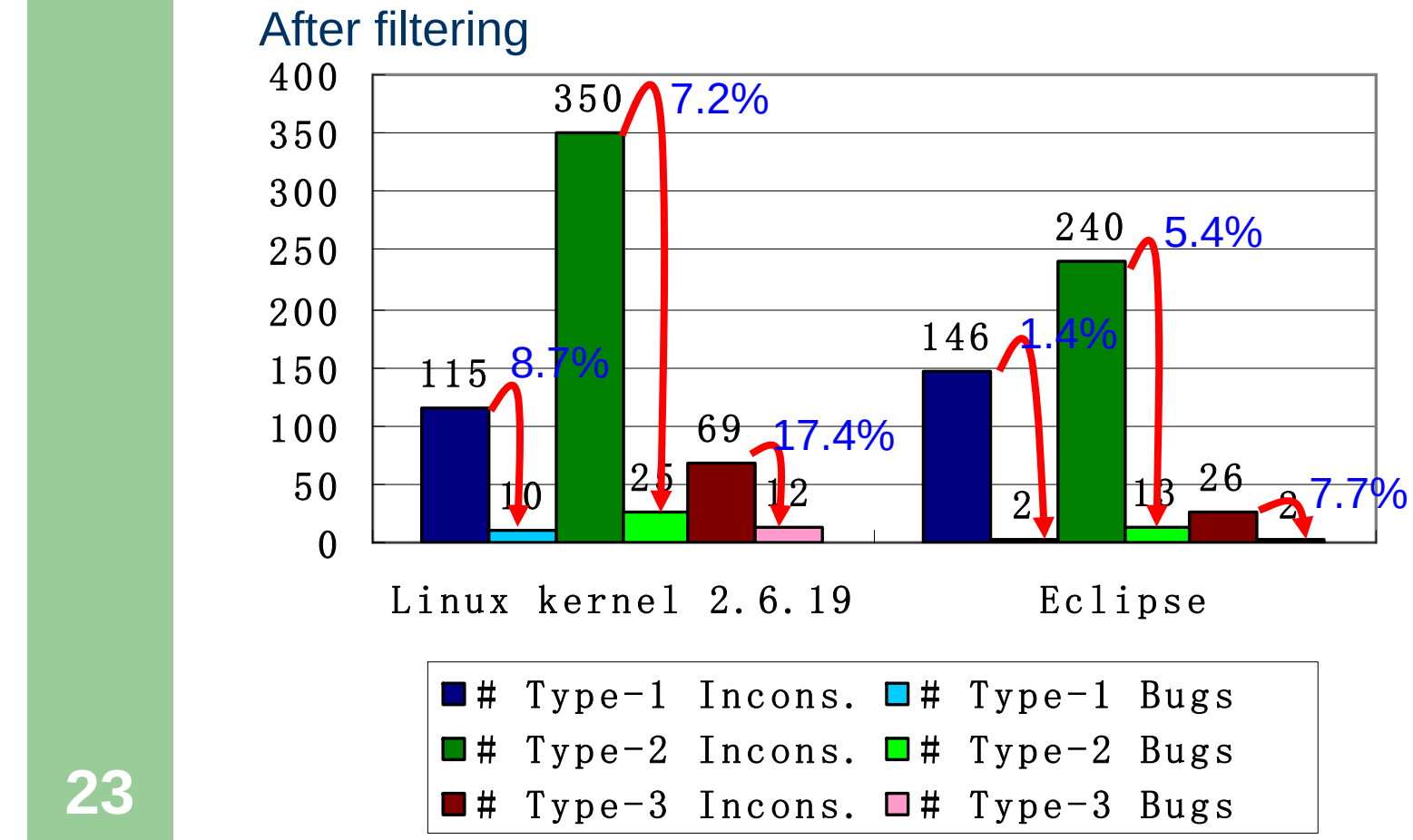
Code Reduction for Inspection



Bug Detection Rates



Inconsistency Types vs. Bugs



False positives

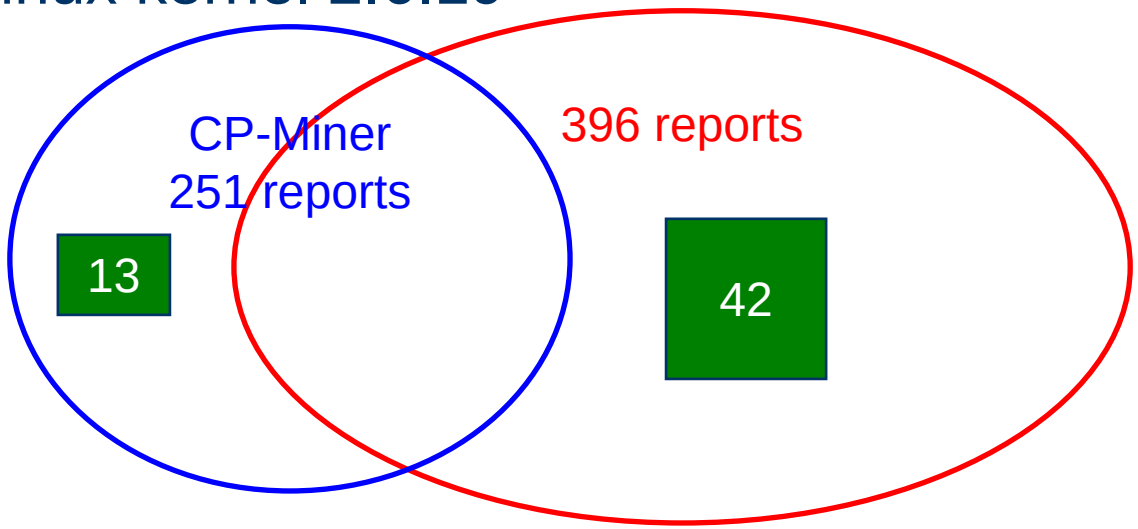
- Syntactic nature
  - Deckard
  - Inconsistencies
  - Filters
- Hypothesized for further pruning
  - Semantic-based clone detection
  - Semantic-aware inconsistencies

Breakdown of Discovered Bugs

No.	Category	# Bugs (Linux)	# Bugs (Eclipse)
0	Total	41	21
1	Missed conditional checks	9	8
2	Negated conditions	1	0
3	Inappropriate conditions	1	3
4	Off-by-one	2	1
5	Inappropriate scoping	2	0
6	Missed or inappropriate qualifiers	2	0
7	Wrong variables	3	4
8	Missed or inappropriate locks	4	0
9	Inappropriate logic for corner cases	3	2
10	Unhandled cases or exceptions	2	3
11	Wrong function calls	3	0
12	Wrong data fields	5	0
13	Wrong macros	4	0

Additional Evaluation: Comparison with CP-Miner

- CP-Miner
  - Token-based clone detection
  - Identifier mapping-based bug detection
- On Linux kernel 2.6.19



Outline

- Introduction and samples of cloning errors
- Definitions of inconsistencies
- Inconsistency filters for bug detection
- Empirical evaluation
- Conclusion and future work

Conclusion and Future work

- Proposed a general notion of inconsistencies among code clones that can help reveal program errors
- Complement program analyses
  - Reduce the amount of code for analysis
  - Provide hints for possible causes of defects
- Extend to semantic-based contextual inconsistencies
  - Semantic-based (PDG-based) clones
  - Semantic inconsistency inference
    - E.g., Dillig, Dillig, Aiken, PLDI 2007

Thank you!

Questions?

jiangl@cs.ucdavis.edu