EXPLORER: Query- and Demand-Driven Exploration of Interprocedural Control Flow Properties

Yu Feng, Xinyu Wang, Isil Dillig and Calvin Lin

UT Austin
Motivation

Many problems require answers to queries about control flow properties
Motivation

Many problems require answers to queries about control flow properties

Can foo() transitively invoke bar()?
Motivation

Many problems require answers to queries about control flow properties

Can foo() transitively invoke bar()?
Motivation

Many problems require answers to queries about control flow properties
Motivation

Many problems require answers to queries about control flow properties

Can foo() transitively invoke bar() without calling goo() in the middle?
Motivation

Many problems require answers to queries about control flow properties.

Can foo() transitively invoke bar() without calling goo() in the middle?
Steps to answer control flow queries
Steps to answer control flow queries

Step 1: Build a calligraph
Steps to answer control flow queries

Step 1: Build a calligraph

Diagram:
- **main**
- **foo** connected to **bar** and **goo**

Steps to answer control flow queries

Step 1: Build a calligraph

Step 2: Customize analysis
Steps to answer control flow queries

Step 1: Build a calligraph

Step 2: Customize analysis
Why is this suboptimal?
Why is this suboptimal?

- User must figure out what callgraph to use
Why is this suboptimal?

- User must figure out what callgraph to use
  - CHA and RTA are fast but imprecise
Why is this suboptimal?

- User must figure out what callgraph to use
  - CHA and RTA are fast but imprecise
  - Using pointer analysis (kcfa, kobj) are precise but not scalable
Why is this suboptimal?

- User must figure out what callgraph to use
  - CHA and RTA are fast but imprecise
  - Using pointer analysis (kcfa, kobj) are precise but not scalable
- User must write different analyses for answering each kind of queries
Our insights
Our insights

- General framework to answer many different kinds of queries
Our insights

- General framework to answer many different kinds of queries
  - Users do not need to write custom analyses
Our insights

- General framework to answer many different kinds of queries
  - Users do not need to write custom analyses
- To give a precise answer for a query, don’t need callgraph that is precise everywhere
Our insights

- General framework to answer many different kinds of queries
  - Users do not need to write custom analyses
- To give a precise answer for a query, don’t need callgraph that is precise everywhere
  - Only need to focus on parts of callgraph that are relevant to the query
Contributions
Contributions

- General query language for describing queries on control flow properties
Contributions

- General query language for describing queries on control flow properties
- Refinement-based algorithm for answering control flow queries
Contributions

- General query language for describing queries on control flow properties
- Refinement-based algorithm for answering control flow queries
- Only refine parts of callgraph relevant to the query
Overview
Key idea 1
Key idea 1

- Our query language: regular expression
Key idea 1

- Our query language: regular expression
- Examples written in our query language
Key idea 1

- Our query language: regular expression
- Examples written in our query language

main → foo
Key idea 1

- Our query language: regular expression
- Examples written in our query language

\[
\text{main} \rightarrow \text{foo}
\]

\[
\text{main} \rightarrow \ast \rightarrow \text{foo}
\]
Key idea 1

- Our query language: regular expression
- Examples written in our query language

```
main → foo
main → .* → foo
.*(!foo)* → bar
```
Key idea 2

- Focus on graph that is relevant to the query
Key idea 2

- Focus on graph that is relevant to the query

foo ➔ .* ➔ bar
Key idea 2

- Focus on graph that is relevant to the query

foo $\rightarrow$ .* $\rightarrow$ bar
Key idea 2

- Focus on graph that is relevant to the query

foo ➔ .* ➔ bar

callgraph
Key idea 2

- Focus on graph that is relevant to the query

foo \rightarrow .* \rightarrow bar

Relevant part!

callgraph
Solution
Solution

Callgraph
Automaton
Solution

Callgraph Automaton

Query Automaton
Solution
Solution

Callgraph Automaton

Product Automaton

Empty

NO
Refine callgraph by issuing pts-to query
Solution

Callgraph Automaton

Query Automaton

Product Automaton

Empty

NO
Solution

Minimize the number of queries
Minimize the number of queries
Minimize the number of queries
Example

```java
void main(...) {
    A x;   A y;
    if (...) x = new A();
    y = new B();
    else x = new B();
    y = new C();
    x.foo();
}
```
Example - Step 1

- Convert Query to Query Automaton

Query: Can A:foo transitively call C:bar?

\[ \bullet \quad (A:foo) \quad \bullet \quad (C:bar) \]
Example - Step 1

- Convert Query to Query Automaton

Query: Can A:foo transitively call C:bar?

\[.* \rightarrow (A:foo) \rightarrow .* (C:bar)\]
Example - Step 2

- Convert callgraph to callgraph automaton
Example - Step 2

- Convert callgraph to callgraph automaton
Example - Step 3

- Compute product automaton by intersecting query and callgraph automata
Example - Step 3

- Compute product automaton by intersecting query and callgraph automata

Won’t scale if we construct it naively
Example - Step 3

- Compute product automaton by intersecting query and callgraph automata

Won’t scale if we construct it naively
Example - Step 3

- Compute product automaton by intersecting query and callgraph automata
How to refine effectively
How to refine effectively

- If intersection is non-empty: callgraph may have spurious edges and we need to refine
How to refine effectively

- If intersection is non-empty: callgraph may have spurious edges and we need to refine
- Performing min-cut on product automaton to minimize # points-to queries
How to refine effectively

- If intersection is non-empty: callgraph may have spurious edges and we need to refine
- Performing min-cut on product automaton to minimize \# points-to queries

Weight on the edges are either 1 or infinite.
Example - Step 4

- Refine Product automaton via min-cut and demand-driven pointer analysis

```c
void main(...) {
    A x;
    A y;
    if (...) x = new A();
    y = new B();
    else x = new B();
    y = new C();
    x.foo();
}
```
Example - Step 4

- Refine Product automaton via min-cut and demand-driven pointer analysis

```c
void main(...) {
    A x;
    A y;
    if (...) x = new A();
    y = new B();
    else x = new B();
    y = new C();
    x.foo();
}
```
Example - Step 4

- Refine Product automaton via min-cut and demand-driven pointer analysis

```c
void main(...) {
    A x;
    A y;
    if (...) x = new A();
    y = new B();
    else x = new B();
    y = new C();
    x.foo();
}
```
Example - Step 4

- Refine Product automaton via min-cut and demand-driven pointer analysis

```c
void main(...) {
    A x;
    A y;
    if (...) x = new A();
    y = new B();
    else x = new B();
    y = new C();
    x.foo();
}
```
Example - Step 4

- Refine Product automaton via min-cut and demand-driven pointer analysis

```c
void main(...) {
    A x;
    A y;
    if (...) x = new A();
    y = new B();
    else x = new B();
    y = new C();
    x.foo();
}
```
Example - Step 4

- Refine Product automaton via min-cut and demand-driven pointer analysis

```c
void main(...) {
    A x;
    A y;
    if (...) x = new A();
    y = new B();
    else x = new B();
    y = new C();
    x.foo();
}
```
Example - Step 4

- Refine Product automaton via min-cut and demand-driven pointer analysis

void main(...) {
    A x;
    A y;
    if (...) x = new A();
    y = new B();
    else x = new B();
    y = new C();
    x.foo();
}
Example - Step 4

- Refine Product automaton via min-cut and demand-driven pointer analysis

```c
void main(...) {
    A x;
    A y;
    if (...) x = new A();
    y = new B();
    else x = new B();
    y = new C();
    x.foo();
}
```

Answer: NO!
Evaluation
Evaluation

- Analysis of the observer design pattern in Java programs
Evaluation

- Analysis of the observer design pattern in Java programs
- Identification of performance bugs caused by GUI lagging in Android applications
Evaluation

- Analysis of the observer design pattern in Java programs
- Identification of performance bugs caused by GUI lagging in Android applications
- Analysis of inter-component communication in Android
Evaluation

- Analysis of the observer design pattern in Java programs
- Identification of performance bugs caused by GUI lagging in Android applications
- Analysis of inter-component communication in Android
Evaluation

- Analysis of the observer design pattern in Java programs
- Identification of performance bugs caused by GUI lagging in Android applications
- Analysis of inter-component communication in Android
Evaluation

- Analysis of the observer design pattern in Java programs
- Identification of performance bugs caused by GUI lagging in Android applications
- Analysis of inter-component communication in Android
class MainActivity {
    onClick() {
        Cursor c = sqliteDatabase.query(...);
    }
}

class MainActivity {
    onClick() {
        new LongOperation().execute(""");
    }
}

class LongOperation extends AsyncTask {
    doInBackground(...) {
        Cursor c = sqliteDatabase.query(...);
    }
}
class MainActivity {
    onClick() {
        Cursor c = sqLiteDatabase.query(...);
    }
}

class LongOperation extends AsyncTask {
    doInBackground(...) {
        Cursor c = sqLiteDatabase.query(...);
    }
}
Evaluation

class MainActivity {
    onClick() {
        new LongOperation().execute(""");
    }
}

class LongOperation extends AsyncTask {
    doInBackground(...) {
        Cursor c = sqLiteDatabase.query(...);
    }
}
Evaluation

class MainActivity {
    onClick() {
        Cursor c = sqLiteDatabase.query(...);
    }
}

class MainActivity {
    onClick() {
        new LongOperation().execute(""");
    }
}

class LongOperation extends AsyncTask {
    doInBackground(...) {
        Cursor c = sqLiteDatabase.query(...);
    }
}
Evaluation

Running time for detecting performance bugs (Lower is better)

[Liu et al. ICSE’ 2014]
Evaluation

Running time for detecting performance bugs (Lower is better)

[Liu et al. ICSE’ 2014]
Evaluation

Running time for detecting performance bugs (Lower is better)

[Liu et al. ICSE’ 2014]
Evaluation

Running time for detecting performance bugs (Lower is better)

[Li et al. ICSE’ 2014]
Evaluation

Running time for detecting performance bugs (Lower is better)

[Li et al. ICSE' 2014]
Evaluation

Number of warnings (Lower is better)
Evaluation

Number of warnings (Lower is better)
Evaluation

Number of warnings (Lower is better)
Evaluation

Number of warnings (Lower is better)

- Ushahidi
- c:geo
- Geohash
- FireFox
- APG
- BitcoinWallet
- MyTrack

- N/A
Evaluation

Number of warnings (Lower is better)

- Ushahidi
- c:geo
- Geohash
- FireFox
- APG
- BitcoinWallet
- MyTrack

- CHA
- KOBJ
- EXPLORER

N/A
Conclusion

- EXPLORER is …

http://fredfeng.github.io/explorer/
Conclusion

- EXPLORER is …

http://fredfeng.github.io/explorer/
Conclusion

- EXPLORER is …

Query Driven  Demand Driven

http://fredfeng.github.io/explorer/
Conclusion

- EXPLORER is …

Query Driven

Demand Driven

Practical

http://fredfeng.github.io/explorer/
Thank you!


Sridharan et al. "Refinement-based context-sensitive points-to analysis for Java." PLDI’ 06.