Advanced Static Analysis of Atomicity in Concurrent Programs through Facebook Infer

Master's Thesis

Dominik Harmim

Supervisor: prof. Ing. Tomáš Vojnar, Ph.D.

xharmi00@stud.fit.vutbr.cz

Brno University of Technology, Faculty of Information Technology

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

24th June 2021

Motivation

The goal: improve detection of atomicity violations within Facebook Infer.

- Detecting and checking desired atomicity of function call sequences.
  - Often required in concurrent programs.
  - Violation may cause nasty errors.

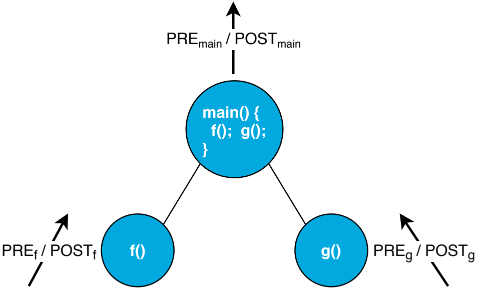
```
void invoke(char *method) {
  ...
  if (server.is_registered(method)) {
    server.invoke(method);
  }
  ...
}
```

The sequence of **`is_registered`** and **`invoke`** should be executed atomically.

If not locked, **`method`** can be unregistered by a concurrent thread.

Facebook Infer

- Open-source static analysis framework for interprocedural analyses.
    - Based on abstract interpretation.
- Highly scalable.
    - Follows principles of compositionality.
    - Computes function summaries bottom-up on call-trees.
- Supports C, C++, Java, Obj-C, C#.



Atomer: Atomicity Violations Analyser

- Facebook Infer plugin created within the author's BSc thesis:

  Harmim, D. *Static Analysis Using Facebook Infer to Find Atomicity Violations*. Brno, 2019. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Vojnar, T.

- **Assumption**: call sequences executed atomically once should (probably) be executed always atomically.

- Implemented for C programs that use PThread locks.

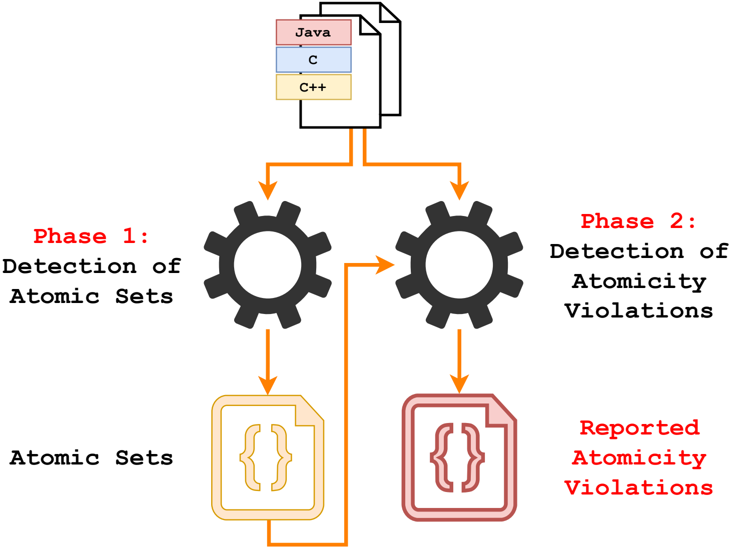
- Limited scalability on large codebases.

- Reports many false alarms when analysing real-life code.

Proposed Atomer's Enhancements

- Approximating sequences of calls by sets of calls (described later on).
- Support for C++ and Java.
  - Working with advanced locks: re-entrant locks, monitors, lock guards, etc.
- Distinguishing different lock instances.
  - Approximating lock objects using syntactic access paths — a representation of heap locations via the paths used to access them.
- Analysis's parametrisation:
  - ignoring generic functions versus concentrating on critical functions;
  - limiting the number of calls or the depth of nested calls in critical sections.

High-Level Analysis Process (Approximated with Sets)



Java

C

C++

Phase 1:
Detection of
Atomic Sets

Phase 2:
Detection of
Atomicity
Violations

Atomic Sets

Reported
Atomicity
Violations

Phases of the Analysis (Approximated with Sets)

1. Detection of atomic call sets.
   - Approximates sequences by sets.
   - **Summary**: $\chi \in 2^{2^{\Sigma}}$ (set of atomic call sets)

```
void f() {
  lock(L);
  x(); y(); z(); // x.y.z -> {x,y,z}
  unlock(L);
  a();
  lock(L);
  z(); y(); x(); // z.y.x -> {x,y,z}
  unlock(L);
}
```

$$\chi_{\mathtt{f}} = \{\{\mathtt{x}, \mathtt{y}, \mathtt{z}\}\}$$

2. Detection of atomicity violations.
   - Derives "atomic pairs" from the first phase: $\Omega \in 2^{\Sigma \times \Sigma}$.
   - Looks for non-atomic pairs of calls assumed to run atomically.
   - **Summary**: $\chi \in 2^{\Sigma \times \Sigma}$ (set of atomicity violations)

```
void g() {
  a(); x(); y(); b();
}
```

$$\Omega = \{(\mathtt{x}, \mathtt{y}), (\mathtt{x}, \mathtt{z}), (\mathtt{y}, \mathtt{x}), (\mathtt{y}, \mathtt{z}), (\mathtt{z}, \mathtt{x}), (\mathtt{z}, \mathtt{y})\}$$
$$(\mathtt{x}, \mathtt{y}) \in \Omega \implies \chi_{\mathtt{g}} = \{(\mathtt{x}, \mathtt{y})\}$$

Phases of the Analysis (Approximated with Sets)

1. Detection of atomic call sets.
   - Approximates sequences by sets.
   - **Summary**: $\chi \in 2^{2^{\Sigma}}$ (set of atomic call sets)

```
void f() {
  lock(L);
  x(); y(); z(); // x.y.z -> {x,y,z}
  unlock(L);
  a();
  lock(L);
  z(); y(); x(); // z.y.x -> {x,y,z}
  unlock(L);
}
```

$$\chi_{\mathtt{f}} = \{\{\mathtt{x}, \mathtt{y}, \mathtt{z}\}\}$$

2. Detection of atomicity violations.
   - Derives "atomic pairs" from the first phase: $\Omega \in 2^{\Sigma \times \Sigma}$.
   - Looks for non-atomic pairs of calls assumed to run atomically.
   - **Summary**: $\chi \in 2^{\Sigma \times \Sigma}$ (set of atomicity violations)

```
void g() {
  a(); x(); y(); b();
}
```

$$\Omega = \{(\mathtt{x}, \mathtt{y}), (\mathtt{x}, \mathtt{z}), (\mathtt{y}, \mathtt{x}), (\mathtt{y}, \mathtt{z}), (\mathtt{z}, \mathtt{x}), (\mathtt{z}, \mathtt{y})\}$$

$$(\mathtt{x}, \mathtt{y}) \in \Omega \Longrightarrow \chi_{\mathtt{g}} = \{(\mathtt{x}, \mathtt{y})\}$$

Experimental Evaluation

- Scalability evaluated on 54 real-life complex C programs.
    - 806,431 LOC in total.
- On average, twice faster.

	v1.0.0		**v2.0.0**	
	Phs. 1	**Phs. 2**	**Phs. 1**	**Phs. 2**
Avg. Time (s)	70.98	109.11	37.96	50.93
Total Time (s)	4,117	5,892	2,164	2,750

- Experiments with Apache Cassandra and Apache Tomcat (both ∼250 KLOC).
    - Successfully rediscovered already fixed reported real bugs.
    - The number of reported bugs was significantly reduced (∼4×).
    - Still hard to say which of the bugs are real — the accuracy needs to be further improved.

Summary[1]

- Proposed and implemented extensions for Atomer:
  - approximation with sets, support for C++ and Java, distinguishing different lock instances, parametrisation of the analysis.
- Successfully tested and experimentally evaluated.
  - Both scalability and accuracy were significantly increased.
- Experiments with real-life programs.

Future goals

- Further increase accuracy/reduce the number of false alarms.
  - Combining with dynamic analysis.
  - Statistic ranking of atomic functions/reported errors.
  - Considering formal parameters of functions.
  - Machine learning of analysis' parameter values.

[1] The preliminary results of this work were presented at the Excel@FIT'21 (won two awards). It is supported by the H2020 ECSEL project VALU3S.

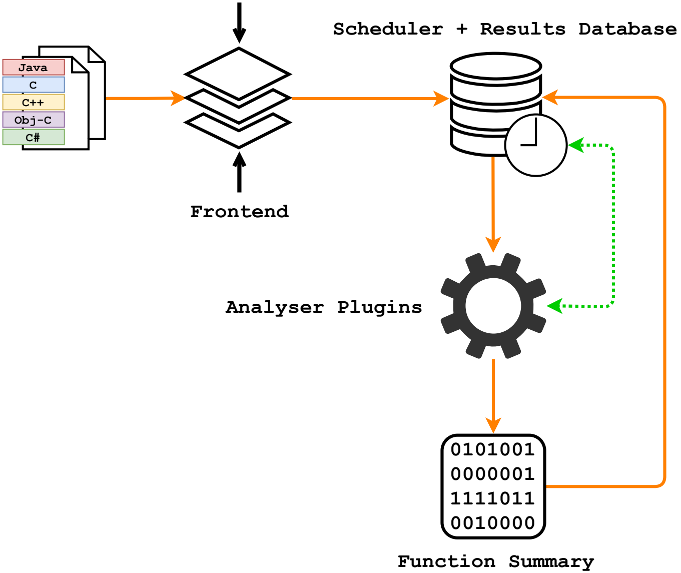
Otázky oponenta

1. **Plánujete podniknout další kroky pro zařazení Atomeru do hlavní větve frameworku Facebook Infer?**
   - Ano, určitě bychom se rádi o zařazení v budoucnu pokusili.
   - Repositář Atomeru je pravidelně aktualizován na nejnovější verzi frameworku.
   - Atomer už byl dříve (úspěšně) presentován a konsultován s vývojáři Inferu.
     - Presentace na Infer Practitioners Workshop v rámci konference PLDI 2020.

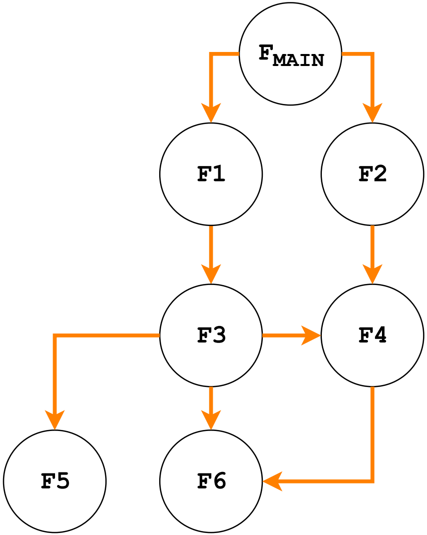
Advanced Manipulation with Locks

- Access path used for a lock's identification: $\pi \in \Pi ::= \mathit{Var} \times \mathit{Field}^*$,
  - *Var* is a set of all variables,
  - *Field* is a set of field names.
- Identification of a critical section: $(\pi, l) \in \Pi \times \mathbb{N}^\top$,
  - π is an access path that identifies a lock object that locks the section,
  - l is the number of locks of the lock object identified by π,
  - $\mathbb{N}^\top$ denotes $\mathbb{N} \cup \{\top\}$,
    - $\top$ represents a number larger than some upper bound $t \in \mathbb{N}$.
- Representation of a lock guard: $(\pi_g, L) \in \Pi \times 2^\Pi$,
  - π_g is an access path that identifies the lock guard,
  - L is a set of access paths that identify lock objects associated with the guard.

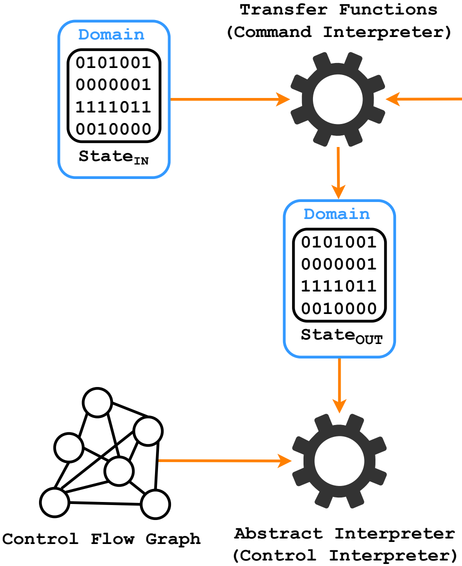
Facebook Infer's Architecture



Demonstration of Facebook Infer's Analysis



Abstract Interpretation in Facebook Infer



Rediscovered Bug in Apache Tomcat

Real-life bug in a package **`org.apache.catalina.core.StandardContext`**

```
public void addParameter(String name, String value) {
  ...
  if (parameters.get(name) != null)
    throw new IllegalArgumentException
      (sm.getString("standardContext.parameter.duplicate", name));

  // Add this parameter to our defined set
  synchronized (parameters) {
    parameters.put(name, value);
  }
  fireContainerEvent("addParameter", name);
}
```