# Advanced Static Analysis of Atomicity in Concurrent Programs through Facebook Infer

Master's Thesis

Bc. Dominik Harmim

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

xharmi00@stud.fit.vutbr.cz

Brno University of Technology, Faculty of Information Technology


14th June 2021

# Motivation

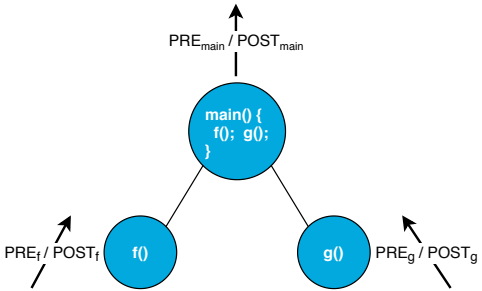
- Detecting and checking desired atomicity of function call sequences.
  - Often required in concurrent programs.
  - Violation may cause nasty errors.

```
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, the method can be unregistered by a concurrent thread.

- 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.

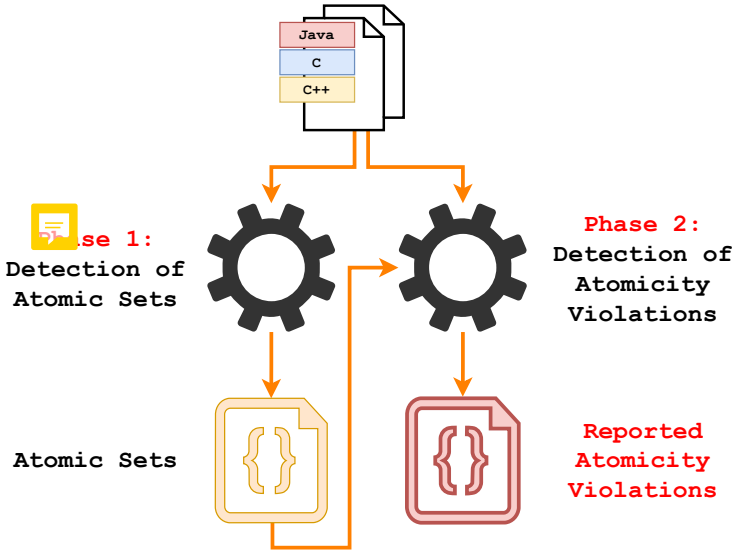
- **Assumption**: call sequences executed atomically once should (probably) be executed always atomically.

- Implemented for C programs that use PThread locks.

- Limited scalability on extensive codebases.

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

# High-Level Analysis Process



# Phases of the Analysis (Approximation with Sets)

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

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

$\chi'_{\mathtt{f}} = (\mathtt{x} \cdot \mathtt{y} \cdot \mathtt{z} \cdot \mathtt{a}, \{\mathtt{x} \cdot \mathtt{y} \cdot \mathtt{z}, \mathtt{z} \cdot \mathtt{y} \cdot \mathtt{x}\})$

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();
}
```

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

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

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

# Further Atomer's Enhancements

- 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.

# Experimental Evaluation

- Scalability evaluated on 54 real-life complex C programs.
  - 806,431 LOC in total.
- ~~Double acceleration in~~ average.

| | 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 (it won two awards). It is supported by the H2020 ECSEL project VALU3S.

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) prezentován a konzultován s vývojáři Inferu.