Finding Concurrency Bugs Using Graph-Based Anomaly Detection in Big Code

Andrew Habib
andrew.a.habib@gmail.com
Department of Computer Science,
TU Darmstadt, Germany

Adviser: Michael Pradel
michael@binaervarianz.de

TECHNISCHE UNIVERSITÄT DARMSTADT

SOLA SoftwareLab

Abstract

Concurrency bugs are very subtle and difficult to discover, reproduce, and fix. Many techniques have been devised to find these bugs. However, most of the effective techniques tend to focus on a subset of the various concurrency bugs types. We propose a new generic concurrency bug detection technique that leverages "Big Code": millions of lines of code freely available on code repositories. Our approach learns a probabilistic graphical model of synchronization patterns from hundreds of concurrent programs. Then the approach predicts synchronization pattern for a given program by posing a structured prediction query to the trained model. We evaluate the approach using a corpus of Java programs and our results show that we correctly predict up to 85% of synchronization properties.

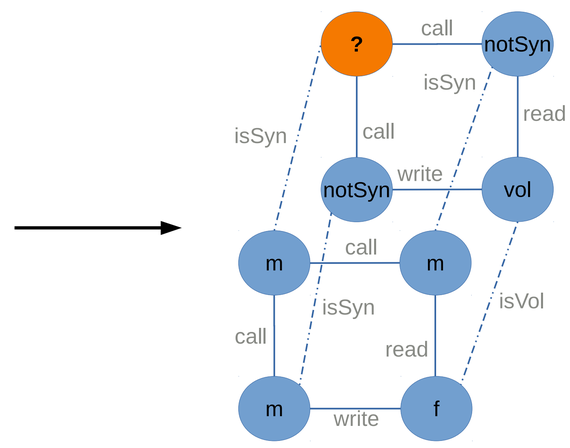
Motivation

- Concurrency bugs are subtle and difficult to find and fix.
- Bug detection tools are even more difficult to write.
- Bug finding tools manually hard-code bug patterns.
- Bug detection focuses on subset of bug types.
- Availability of high-quality source code.
- Powerful machine learning techniques.

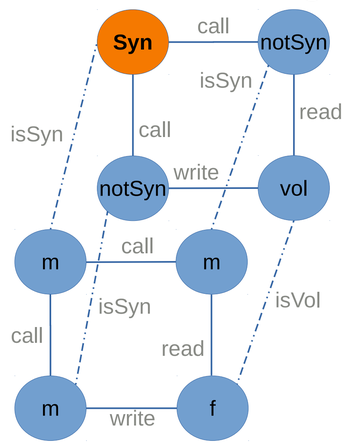
Example

- `class Person` is supposed to be thread-safe.
- Synchronization on method writeDefault() is missing.

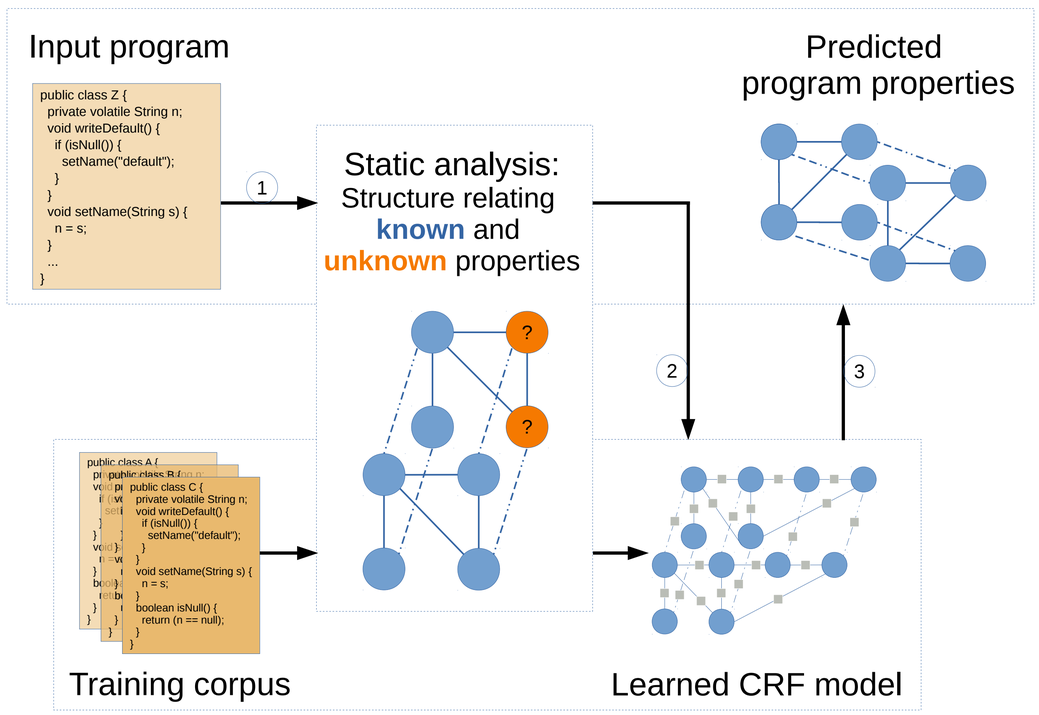
```
public class Person {
  private volatile String n;
  public void writeDefault() {
    if (isNull()) {
      setName("default");
    }
  }
  void setName(String s) {
    n = s;
  }
  boolean isNull() {
    return (n == null);
  }
}
```



A trained CRF model predicts the missing synchronization:



Approach



Lightweight static analysis: using the Soot framework [1]

- Extract program structure and properties: reads/writes to shared fields, volatile fields, synchronized methods and blocks, etc.

Learning a statistical model: using the Nice2Predict framework [2]

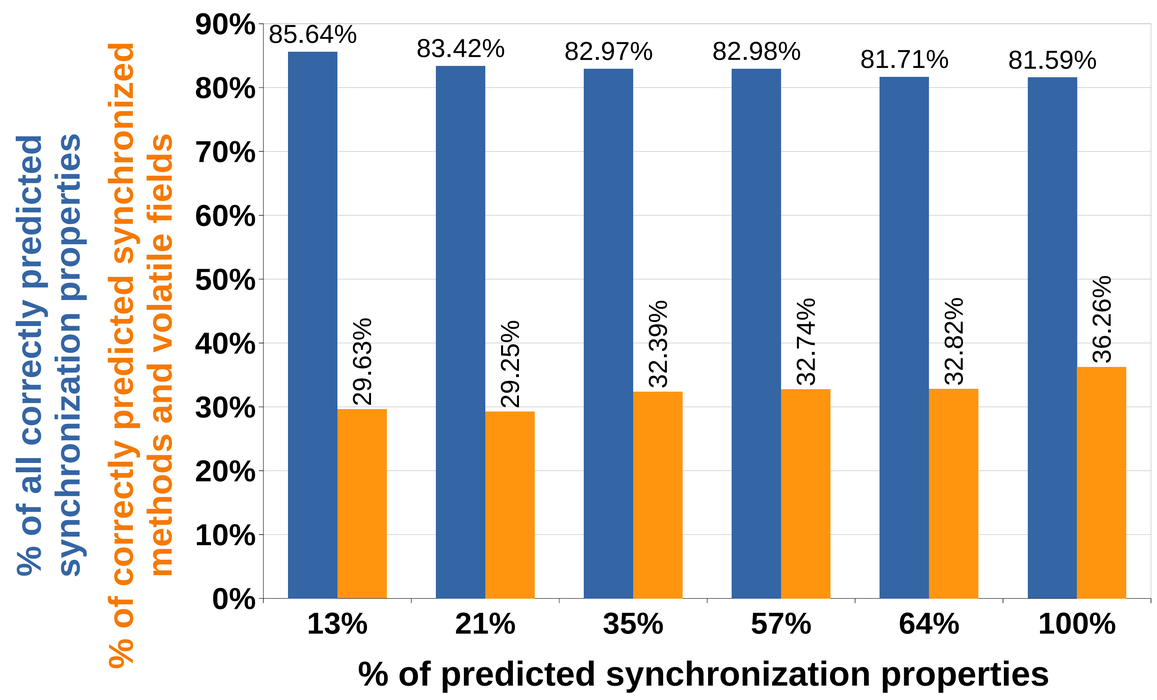
- Learn a probabilistic model: conditional random field (CRF).

Predicting Synchronization

1) Extract program structure with known and unknown properties.
2) Formulate Maximum a Posterior (MAP) query.
   - Joint prediction over all unknown properties.
   - Query the trained CRF model using the MAP query.
3) Returned result indicates which methods and blocks should be synchronized, which shared fields should be volatile, etc.

Evaluation

- Using the Qualitas Corpus [3]
- **Training CRF model:** a subset with 1767 classes
- **Evaluation:** a different subset with 252 classes
  - Remove synchronization properties from the evaluation set.
  - Vary the percentage the removed synchronization properties.
  - Use the trained CRF model to predict missing synchronization.



Ongoing Work

- Finer-grained static analysis (precise synchronization information)
- Manual inspection and evaluation of reported potential bugs
- Higher quality and more training data

[1] https://sable.github.io/soot/
[2] http://nice2predict.org/
[3] http://www.qualitascorpus.com/