

Static Control-Flow Analysis for Reverse Engineering of UML Sequence Diagrams

Atanas Rountev; Olga Volgin and Miriam Reddoch

Presented by: Hui Shen
03/11/2007

Outline

- What is Reverse Engineering
- UML sequence diagram & Static analysis
- Analysis of algorithm
- Related work--RED
- Conclusion

What is Reverse Engineering

- **Reverse engineering (RE)** is the process of analyzing a subject system to create representations of the system at a higher level of abstraction. (From Wikipedia)
- Reverse engineering of sequence diagram allows the automatic extraction of such diagrams from existing code.
 - During *iterative development*.

What is Reverse Engineering

- Software Understanding
- Software Maintenance
- Software Testing
- Documentation

What is Reverse Engineering

- How should the intraprocedural flow of control in the code be represented in the reverse-engineered sequence diagrams?

UML sequence diagram & Static Analysis

UML sequence diagram

- Sequence diagram is the most common kind of Interaction Diagram.
- A sequence diagram shows a set of interaction objects and the sequence of message exchanged among them.

Analysis of Algorithm

- UML interaction fragments
 - opt, alt, loop and break
 - An opt/loop/break fragment encloses an ordered sequence of other fragments.
 - Generalized break

Analysis of Algorithm

- Control flow analysis
 - Post-dominates
 - Branches and branch successors
 - alt fragments
 - “stopping” point
 - Loops and loop successors

Analysis of Algorithm

- Multiple CFG exits
 - Alt fragments can be used to represent multiple CFG exits
 - It produces deeply nested fragments
 - Define *return fragment* which is similar to a break fragment

Analysis of Algorithm

- Exceptions
 - In Java, the exceptions maybe synchronous and asynchronous
 - Implicitly-thrown exceptions are not considered
 - All catch clauses are ignored
 - Throw statements are represented as return statements

Analysis of Algorithm

- Precision & replication
 - Either use standard UML control-flow primitives, or the flow of control in the code can be represented precisely
 - But not both
- For program understanding
 - Avoid replication together
- For test coverage measurements
 - Precision is more important than others

Analysis of Algorithm

- Example: CFG node 5 is replicated
 - Full precise choice

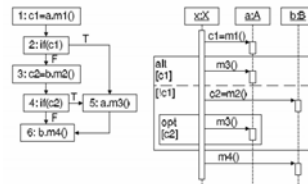


Figure 2: Message m3 is replicated.

Analysis of Algorithm

- No-replication mappings
 - Not precise, no replication

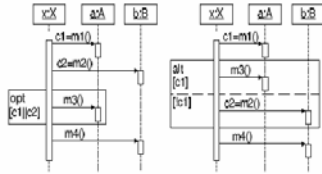


Figure 3: Possible imprecise mappings.

Related Work

- RED stands for reverse engineering of UML sequence diagrams.
- It is developed by PRESTO research group of Ohio state university
- The goal of this project is to build a state-of-the-art tool for extracting sequence diagrams from Java code. The tool is in the form of a public Eclipse plug-in.

Related Work

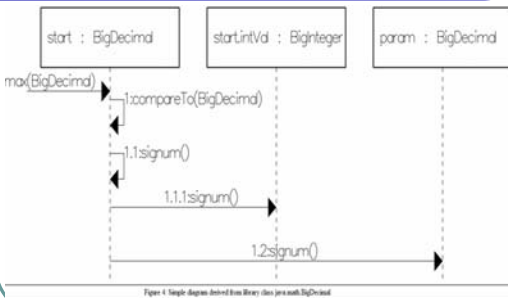


Figure 4: Sample diagram derived from Java class java.math.BigDecimal

Related Work

- Object naming analysis
 - *Given some call site in the analyzed code, how should the receiver object(s) at this site be represented in the diagram?*
- Call chain analysis
 - *How should infeasible call chains be eliminated from the diagrams?*

Conclusion

- A novel algorithm for mapping reducible exception-free control-flow graphs to UML interaction fragments
- Balance precision and practicality

Reference

- A. Rountev and B.H. Connell. Object naming analysis for reverse-engineered sequence diagram. In ICSE 2005
- A. Rountev, S. Kagan, and M. Gibas. Static and dynamic analysis of call chains in Java. In ISSTA 2004
- OMG. UML 2.0 Infrastructure Specification. Object Management Group www.omg.org 2003
- PRESTO research group of Ohio state university Webpage of RED project presto.cse.ohio-stat.edu/red

END

Question?

Thanks.