Software Modeling and Analysis

10. Dynamic Analysis

Nevena Milojković

Based on the slides of Jorge Ressia
Roadmap

> Motivation
> What is dynamic analysis?
> Why dynamic analysis?
> Sources of run-time data
> Dynamic analysis techniques
> Usage of dynamic analysis
Roadmap

> **Motivation**
> What is dynamic analysis?
> Why dynamic analysis?
> Sources of run-time data
> Dynamic analysis techniques
> Usage of dynamic analysis
“Programs must be written for people to read, and only incidentally for machines to execute.”

Harold Abelson, “Structure and Interpretation of Computer Programs”
The idea behind programming is to be able to clearly explain ideas that are otherwise difficult to formulate; to explain an algorithm without ambiguity.
Motivation — so, how difficult it is to understand software?

```java
public class UndoCommand extends AbstractCommand {
    public void execute() {
        ...
        Undoable lastUndoable = um.popUndo();
        // Execute undo
        boolean hasBeenUndone = lastUndoable.undo();
        ...  
    }
}
```
Let us imagine a developer who needs to fix a bug, and has to understand control-flow at call site “undo()”. She needs to know exact which exact method “undo()” is invoked at the marked call site. In order to make it cheaper, and avoid running the code, let us start with static analysis.

So, the first hint is that static type of the receiver “lastUndoable” is “Undoable”. Let us dive into class hierarchy of this class, and check where method “undo()” is implemented.
Motivation — so, how difficult it is to understand software?

```java
public interface Undoable

public class UndoableAdapter
    public static class UndoActivity
    public static class UndoActivity
    public static class UndoActivity
    public static class UndoActivity
    public static class UndoActivity
    public static class UndoActivity
    public static class UndoActivity
    ...

    public static class UndoActivity

public class UndoRedoActivity
```
Awkwardly enough “Undoable” is an interface, with 31 classes implementing it, and 27 of them implementing method “undo()”.
Roadmap

- Motivation
- **What is dynamic analysis?**
- Why dynamic analysis?
- Sources of run-time data
- Dynamic analysis techniques
- Usage of dynamic analysis
What is dynamic analysis?

“Dynamic analysis is the analysis of the properties of a running program.”

Thomas Ball, “The Concept of Dynamic Analysis”, 1999
What is dynamic analysis?

Properties of a software system are represented by the system behaviour.

System behaviour is established by method implementations.
Definition

Software

Instrument

Execute and collect data
Why Dynamic Analysis?

Gap between run-time structure and code structure in OO programs

Trying to understand one [structure] from the other is like trying to understand the dynamism of living ecosystems from the static taxonomy of plants and animals, and vice-versa.

— Erich Gamma et al., Design Patterns
Software can be hard to understand solely from source code. Code is static and it does not explicitly reflect the dynamic behaviour. The code structure consists of classes in inheritance relationships, while we need to understand relations and collaborations among objects during run time. These collaborations are spread throughout the code, and may not be clearly visible from the source code.

Data polymorphism may hide the actual receiver’s type, thus introducing one more level of difficulty.
Roadmap

> Motivation
> What is dynamic analysis?
> **Why dynamic analysis?**
> Sources of run-time data
> Dynamic analysis techniques
> Usage of dynamic analysis
Static analysis is not enough

CHA (Class Hierarchy Analysis)
RTA (Rapid Type Analysis)
CTA
MTA
FTA
XTA
k-CFA (Control Flow Analysis)
The algorithm we have used in the motivation example is known under the name Class Hierarchy Analysis (CHA). For further reading, please visit [http://web.cs.ucla.edu/~palsberg/tba/papers/dean-grove-chambers-ecoop95.pdf](http://web.cs.ucla.edu/~palsberg/tba/papers/dean-grove-chambers-ecoop95.pdf).

One improvement of CHA is Rapid Type Analysis ([http://www.cs.cornell.edu/courses/cs711/2005fa/papers/bs-oopsla96.pdf](http://www.cs.cornell.edu/courses/cs711/2005fa/papers/bs-oopsla96.pdf)), which also depends on the class hierarchy, but collects a set of instantiated classes along the way of call graph construction, these being the only available classes for variable’s type.

CTA, MTA, FTA and XTA algorithms are refinements of the idea behind RTA, creating sets of instantiated classes per class, method and/or field. They are explained in [http://web.cs.ucla.edu/~palsberg/paper/oopsla00.pdf](http://web.cs.ucla.edu/~palsberg/paper/oopsla00.pdf).
Control-flow analysis tends to answer the question of “which method will this call invoke?”, while data-flow analysis tends to answer the question of “which data (value, object) will hold variable x at run time?”. It is obvious that these two problems are interleaved, and heavily dependent on each other.

The CFA acronym for control-flow analysis algorithms is misleading, since this family of analyses is focused on data-flow analysis.

CFA analysis performs an abstract interpretation of a program by assigning abstract values to variables, e.g., if a variable would have a value of 5, an abstract interpreter would say it’s a positive integer.

Problem of control and data flow is NP-hard problem.

Static analysis is not precise enough

Problem: over approximation
Roadmap

> Motivation
> What is dynamic analysis?
> Why dynamic analysis?
> Sources of run-time data
> Dynamic analysis techniques
> Usage of dynamic analysis
> Tracing method execution
> Tracing values of variables
> Tracing memory usage
Two ways of getting the information

> External
  — execute program and collect the information from outside

> Internal
  — instrument program, and get the information from inside
External View

> System.out.println

> Examine logs

> Analyse used resources
  > CPU and memory usage
Developers need effective ways to inspect the run-time information of a program in question. Developers often use “print” statement to get the information from a running system. But, they only can print the static output and this doesn’t allow the developer to explore them further.
Loggers - low tech debugging

“...debugging statements stay with the program; debugging sessions are transient. “

Kerningham and Pike

```java
public class Main {

    public static void main(String[] args) {
        Clingon anAlien = new Clingon();
        System.out.println("in main");
        anAlien.spendLife();
    }
}
```
Inserting log statements into the code is a low-tech method for debugging. It may also be the only way because debuggers are not always available or applicable. This is often the case for distributed applications.

DoodleDebug is a graphical drop-in replacement for System.out.println(), Eclipse plugin, which integrates into the IDE and replaces the Console. The output is inspectable.

Many different tools are based on tracing, like execution profilers and test coverage analysers.
Execution Tracing

How to capture full program execution?

Trace entry and exit of each method.

Additional information to record:
  receiver and arguments
  return values
  fields assigning
  class instantiations
Pros and challenges:

Pros:
- allows concise analysis
- depends on the input

Challenges:
- concurrency, “Heisenbugs”
- instrumentation influences the behaviour of the execution
- overhead: increased execution time
- large amount of data, scalability
- incomplete
Pros:

Precision: with runtime analysis, one can log just the data of interest, a method call at specific call site, without constructing the whole call graph.

Input-dependent: with runtime analysis one can relate input with the output, so it makes it easy to capture the reflection of the input changes to the running system.
Challenges:

Concurrency: many threads interacting can produce confusion if logging data in one file.

Instrumentation introduces overhead in running system, so if the system execution is time sensitive, it will cause problems.

If logging a lot of information is needed, it will produce a large amount of data, which can be difficult to understand.
Dynamic analysis is not precise enough

Problem: under approximation
Important

One should introduce as little overhead as possible when performing dynamic analysis.
Roadmap

- Motivation
- What is dynamic analysis?
- Why dynamic analysis?
- Sources of run-time data
- **Dynamic Analysis Techniques**
- Usage of dynamic analysis
Bytecode Instrumentation
Bytecode Instrumentation

Smalltalk
Reflectivity

- Reflectivity is a tool to annotate AST nodes with metalinks.

- A metalink is a message sent to an arbitrary object.

- A metalink can be executed before a node, instead a node or after a node.

http://smalltalkhub.com/#!/~RMoD/Reflectivity
Example: Logging

> Goal: logging message send with parameters
> First way: directly edit the text.

Object subclass: #Point
  instanceVariableNames: 'x y'

Point>>movedX: anIntX dy: anIntY
  x := x + anIntX.
  y := y + anIntY.

Point>>movedX: anIntX dy: anIntY
  Transcript show: anIntX asString, ' ', anIntY asString; cr.
  x := x + dx.
  y := y + dy.
Logging with Reflectivity

> Goal: Change the method without changing program text
> Example:

```
argumentsLink := MetaLink new
    metaObject: PrintClass;
    selector: #show:;
    control: #before;
    arguments: #(arguments)

Object subclass: #PrintClass
    instanceVariableNames: ' '

PointClass class>>show: aCollection
    aCollection do: [:each | Transcript show: each asString; cr].

(Point>>movedX:dy:) methodNode link: argumentsLink.
argumentsLink uninstall.
```
Reflectivity works with metavariables.
In our example let’s us assume that we want to print actual arguments of each message send of method “Point>>movedX:dy:”. We need to install a metalink (an instance of class MetaLink) on the method node representing this method, and we want it to log arguments #before method execution (hence control: #before during metalink creation). This metalink will send message “show:” to class PrintClass, and this message will actually log data to Transcript.

Installing link is done in one line, by sending a message “link:” to a method node.
By sending message “uninstall” to the link, it will uninstall the link from all nodes on which link was installed.
Reflectivity

> Metavariabes:
  > #node (for each program node)
  > #selector (for message or method selector)
  > #sender (for message send)
  > #variable (for variable node)
  > #value (for value node)

> Control:
  > #before
  > #instead
  > #after
Bytecode Instrumentation

Java
Bytecode Manipulation

— Javassist
  - high-level API

— ASM
  - working on low-level
class Point {

    int x, y;

    void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
}

Javassist - example
ClassPool pool = ClassPool.getDefault();

CtClass cc = pool.get("Point");

CtMethod m = cc.getDeclaredMethod("move");

m.insertBefore("{ System.out.println($1);
    System.out.println($2); }");

cc.writeFile();
The program first obtains a ClassPool object, which controls bytecode modification with Javassist. The ClassPool object is a container of CtClass objects representing class files. It reads a class file on demand for constructing a CtClass object and records the constructed object to be able to respond on later accesses.

To modify the definition of a class, the user must first obtain from a ClassPool object the reference to the CtClass object representing the class under revision. It is done by invoking method get(String) on ClassPool object, whose argument is the string representing the name of the class.

Next, by invoking method getDeclaredMethod(String) on the CtClass object, we are able to get the CtMethod object representing the method we want to modify (the name of the method is the argument of the getDeclaredMethod(String) method).

By invoking insertBefore(String) method on the CtMethod object, we are inserting byte code as the first statement in the method body. The argument must represent valid Java code. Similar with insertAfter(String); insertAt(int, String).

Javassist provides metavariables to access some objects within the method body:

- $0, $1, $2, ... this and actual parameters
- $args An array of parameters. The type of $args is Object[].
- $$ All actual parameters. //for example, m($$) is equivalent to m($1,$2,...)
$sig An array of java.lang.Class objects representing the formal parameter types.
$_ The resulting value
$r The result type. It is used in a cast expression.
$type A java.lang.Class object representing the formal result type.
$w The wrapper type. It is used in a cast expression (from a primitive type to its corresponding wrapper type)
$class A java.lang.Class object representing the class currently edited.

Accessing local variables declared in the method is not allowed, but declaring a new one is allowed.

In the last line, invocation of writeFile() method translates the CtClass object into a class file and writes it on a local disk. Without this call, all changes will be lost.
class Point {
    int x, y;
    void move(int dx, int dy) {
        { System.out.println(dx);
           System.out.println(dy); }
        x += dx; y += dy;
    }
}
Roadmap

> Motivation
> What is dynamic analysis?
> Why dynamic analysis?
> Sources of run-time data
> Dynamic Analysis Techniques
> **Usage of dynamic analysis**
Usage of dynamic analysis

- Reverse engineering
- Debugging, as execution traces
- Program comprehension
What you should know!

- How are represented properties of a software?
- What are the limitations of static analysis?
- What is dynamic analysis?
- How can dynamic analysis be performed?
- What are sources of run-time data?
- What are pros of dynamic analysis?
- What are the challenges?
- What is bytecode instrumentation?
- Why is dynamic analysis useful?
Can you answer these questions?

> How do control and data flow depend on each other?
> How to detect dead code in software?
> Can static and dynamic analysis be combined and how?
> How can dynamic analysis change software behaviour?
Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

You are free to:

**Share** — copy and redistribute the material in any medium or format
**Adapt** — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

**Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

**ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

[http://creativecommons.org/licenses/by-sa/4.0/](http://creativecommons.org/licenses/by-sa/4.0/)