Formal Correctness Proofs of Refactorings

Volker Stolz $^1$

Ole Jørgen Abusdal $^1$  Eduard Kamburjan $^2$  Violet Ka I Pun $^1$

$^1$Western Norway University of Applied Sciences  $^2$University of Oslo

← Our KeY updates — for the paper, see ISoLA (2)!
Refactoring and relational verification

Refactoring:
Improve structure of code, preserve behavior of executions

\[
\text{if}(E) \{ S1; \} \text{ else } \{ S2; \} \text{ return; } \sim \text{ if}(!E) \{ \text{S2; return;} \} \text{ S1; return;}
\]

placeholders

Relational verification:
Relate pairs of executions, given initial state satisfy $\Phi$ then final state satisfy $\Psi$

\[
\text{original } \sim \text{ refactored} : \Phi \implies \Psi
\]
Applications to Security

- KeY framework: information flow, non-interference for Java programs
- REFINITY/abstract execution: proofs on code-fragments (as specifications)
A fundamental relational property

Program equivalence
Two programs are equivalent iff they produce the same output when executed on the same input.

Here: let’s look at Java fragments that we consider equivalent.

- How far can current tool support take us?
- Other definitions of equivalence?
Relational verification in practice

REFINITY

• Built on top of the KeY automated theorem prover
• Enables relational verification of “Java” with placeholders
• Placeholders are subject to Abstract Execution
• Has been sufficiently powerful to verify statement level refactorings\(^1\)

\(^1\)See Dominic Steinhöfel’s PhD thesis: https://tuprints.ulb.tu-darmstadt.de/8540/
Example - slide stmt. abs. proved
Synchronize Scrolling

**Abstract Program Fragments**

1. `x = new C();`
2. `y = new D();`

1. `y = new D();`
2. `x = new C();`
Intro

REFINITY

Object Creation

Future Challenges

Equivalence in REFINITY

⊿

10
Equivalence

REFINITY checks that the following are identical by default:

• return values

• exceptions

• objects in the so-called relevant location set
Equivalent?

Refactoring tools often get this wrong:

```
x.n();
x.n();
```
(a) Before

```
X temp = x;
temp.n();
temp.n(); //change?
```
(b) After

REFINITY won’t close the proof unless you can show the required side-conditions on method $n()$. 

Example - condition on $n()$ in REFINITY
Different output, but equivalent?
Exception origin moved, no additional capture in `h()`

```
  o.f().g();
  \to o.h(); \text{ with } h() \{ \text{ this.f().g(); } \}
```

Example - in REFINITY (postcondition)
Intro

REFINITY

Object Creation

Future Challenges

Challenges

Try to use tooltips if feeling unsure about the functionality of an element.
Challenges in complex refactorings

Succesfully verified variants of *Extract Local Variable* and *Hide Delegate* and investigated how to approach others.

We discuss
Simplifying postcondition specifications

Unresolved
Making the proofs useful artefacts: what about *instantitations*?
Object equality

REFINITY lacked rules for object equality over multiple modalities:

- can verify \textit{Slide Statement} with abstract statements
Object equality

REFINITY lacked rules for object equality over multiple modalities:

• can verify Slide Statement with abstract statements

• can’t verify Slide Statement with statements involving concrete objects
Intro

REFINITY

Future Challenges

Object Creation

Why?
REFINITY Internals

Core issue:
- Objects are placed in a symbolic heap during SE
- Before and After program executed in same proof

Not sufficient for two new objects to be equal:
- the allocation must, additionally, be deterministic

Schematic sequent rules in KeY are specified as taclets:
- we add rules to make objects indistinguishable under certain conditions
New taclet for object creation

\[ \Gamma, \{U\}(v \neq \text{null} \land v \vdash C :: \text{allocate(heap)} \land C :: \text{exactInstance}(v) \vdash \text{TRUE}) \]
\[ \Rightarrow \{U\}\{\text{heap := create(heap, v)}\}[s]\phi, \Delta \]
\[ \Gamma \Rightarrow \{U\}[v = \text{C.allocate()}; s]\phi, \Delta \]

Additionally, we give two simplification rules for heaps within any allocate function application. Let \( \sqsubseteq \) be the subtype relation and \( T(t) \) the type of a term.

\begin{align*}
C :: \text{allocate(store(h, o, f, v))} & \rightsquigarrow C :: \text{allocate(h)} \quad \text{if } f \neq \text{<allocated>} \\
C :: \text{allocate(create(h, o))} & \rightsquigarrow C :: \text{allocate(h)} \quad \text{if } C \not\subseteq T(o)
\end{align*}
Postcondition simplification

In **Hide Delegate** exception objects now equivalent

- we need no special postcondition to handle exceptions...

- ...although we should because in practice exceptions capture state! (Not *our* problem, though 😞)
Future challenges
Future Challenge
Trace based notions of equivalence

\begin{align*}
\theta & ::= \left[ \phi \right] \\
& \quad | \text{call}(m) \\
& \quad | \text{finite} \\
& \quad | \theta \ast\ast\theta
\end{align*}

(a) Before

\begin{verbatim}
File f = new File();
String s = "";
/*@ ensures finite ** call(f.open) ** finite; */
\abstract_statement A;
s = f.read();
f.write(s);
/*@ ensures finite ** call(f.close) ** finite; */
\abstract_statement B;
\end{verbatim}

(b) After

\begin{verbatim}
File f = new File();
String s = "";
/*@ ensures finite ** call(f.open) ** finite; */
\abstract_statement A;
f.write(s);
s = f.read();
/*@ ensures finite ** call(f.close) ** finite; */
\abstract_statement B;
\end{verbatim}
Summary

• REFINITY/KeY excellent foundation for reasoning about OO in general

• abstract code + side conditions

• initial application area: checking refactorings via symbolic execution

• next: application-specific?
SILM Workshop

Welcome to the 6th edition of our workshop on the Security of Software/Hardware Interfaces. SILM 2024 will take place on Friday, July 12 2024, in Vienna (Austria), co-located with the 9th IEEE European Symposium on Security and Privacy (EuroS&amp;P 2024)

SUBMISSIONS

Submission deadline is March 29, 2024 -- 11:59pm AoE (was: March 15, 2024---11:59pm AoE); check our Call for Papers for details.