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

```
if(E) {S1;} else {S2;} return;    ~    if(!E) {S2;return;} S1; return;
```

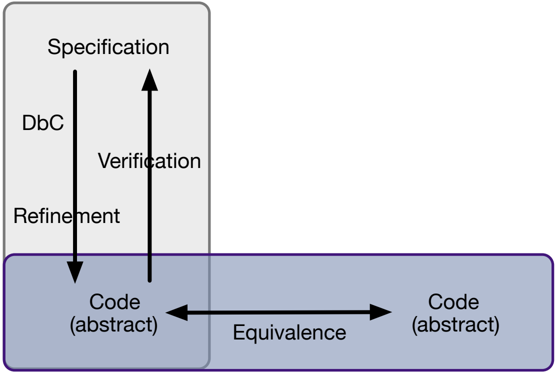
placeholders

Relational verification:

Relate pairs of executions, given initial state satisfy Φ then final state satisfy Ψ

$$\text{original} \sim \text{refactored} : \Phi \implies \Psi$$

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.

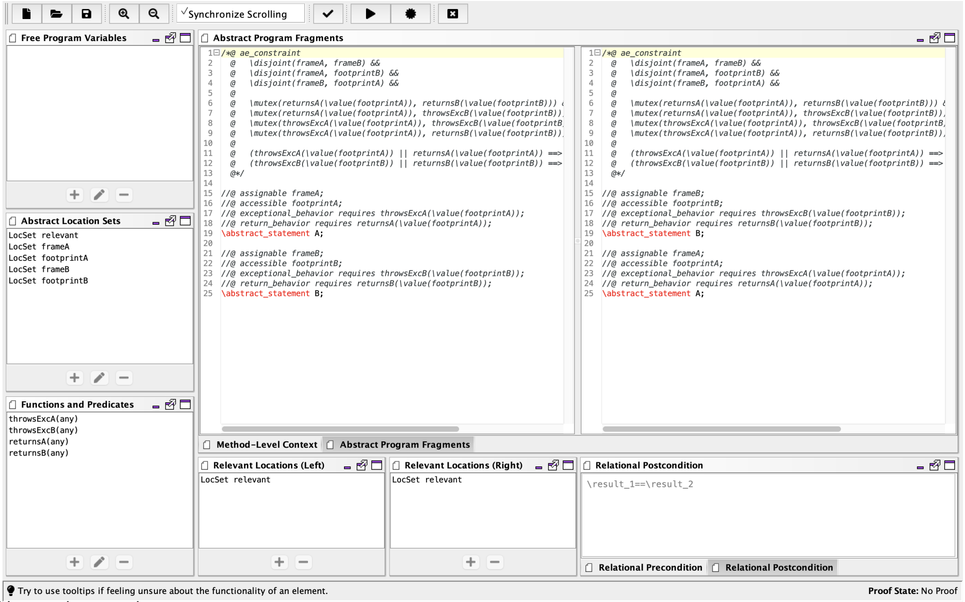
- How far can current tool support take us?
- Other definitions of equivalence?

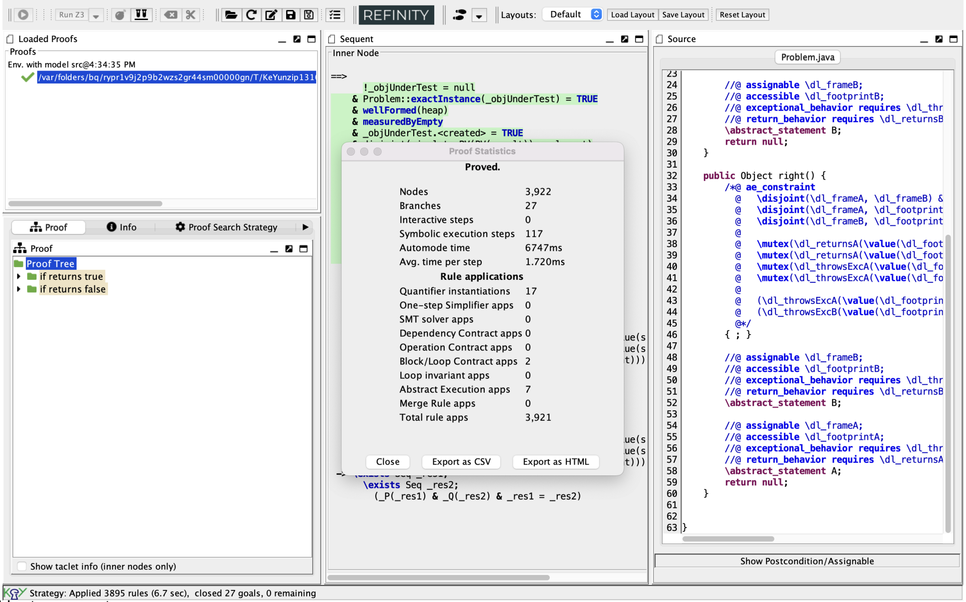
Relational verification in practice

REFINITY

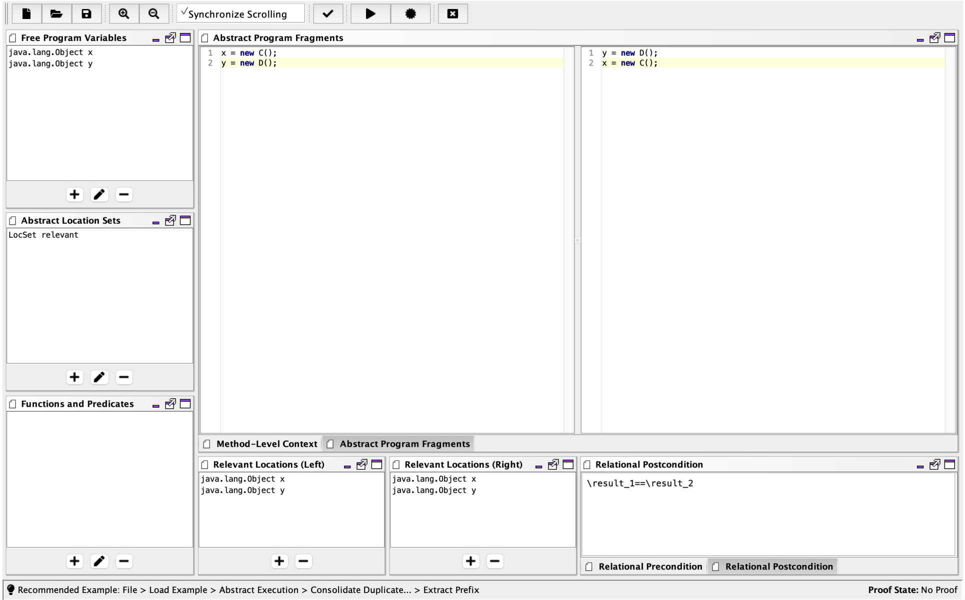
- Built on top of the KeY automated theorem prover
- Enables relational verification of “Java” with placeholders
- Placeholders are subject to Abstract Execution
- Has been sufficiently powerful to verify statement level refactorings[1]

[1] See Dominic Steinhöfel’s PhD thesis: https://tuprints.ulb.tu-darmstadt.de/8540/

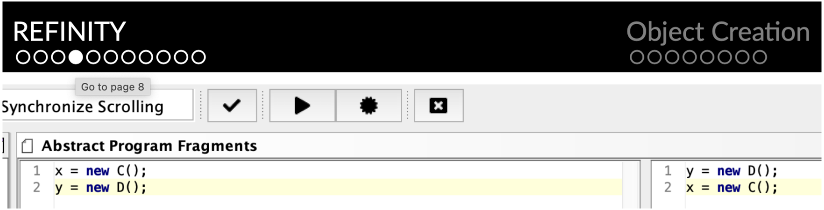

Synchronize Scrolling
Free Program Variables
Abstract Program Fragments
/*@ ae_constraint
@ \disjoint(frameA, frameB) &&
@ \disjoint(frameA, footprintB) &&
@ \disjoint(frameB, footprintA) &&
@
@ \mutex(returnsA(\value(footprintA)), returnsB(\value(footprintB))) &
@ \mutex(returnsA(\value(footprintA)), throwsExcB(\value(footprintB)),
@ \mutex(throwsExcA(\value(footprintA)), throwsExcB(\value(footprintB
@ \mutex(throwsExcA(\value(footprintA)), returnsB(\value(footprintB)),
@
@ (throwsExcA(\value(footprintA)) || returnsA(\value(footprintA)) ==>
@ (throwsExcB(\value(footprintB)) || returnsB(\value(footprintB)) ==>
@*/
//@ assignable frameA;
//@ accessible footprintA;
//@ exceptional_behavior requires throwsExcA(\value(footprintA));
//@ return_behavior requires returnsA(\value(footprintA));
\abstract_statement A;
//@ assignable frameB;
//@ accessible footprintB;
//@ exceptional_behavior requires throwsExcB(\value(footprintB));
//@ return_behavior requires returnsB(\value(footprintB));
\abstract_statement B;
/*@ ae_constraint
@ \disjoint(frameA, frameB) &&
@ \disjoint(frameA, footprintB) &&
@ \disjoint(frameB, footprintA) &&
@
@ \mutex(returnsA(\value(footprintA)), returnsB(\value(footprintB))) &
@ \mutex(returnsA(\value(footprintA)), throwsExcB(\value(footprintB)),
@ \mutex(throwsExcA(\value(footprintA)), throwsExcB(\value(footprintB
@ \mutex(throwsExcA(\value(footprintA)), returnsB(\value(footprintB)),
@
@ (throwsExcA(\value(footprintA)) || returnsA(\value(footprintA)) ==>
@ (throwsExcB(\value(footprintB)) || returnsB(\value(footprintB)) ==>
@*/
//@ assignable frameB;
//@ accessible footprintB;
//@ exceptional_behavior requires throwsExcB(\value(footprintB));
//@ return_behavior requires returnsB(\value(footprintB));
\abstract_statement B;
//@ assignable frameA;
//@ accessible footprintA;
//@ exceptional_behavior requires throwsExcA(\value(footprintA));
//@ return_behavior requires returnsA(\value(footprintA));
\abstract_statement A;
Abstract Location Sets
LocSet relevant
LocSet frameA
LocSet footprintA
LocSet frameB
LocSet footprintB
Functions and Predicates
throwsExcA(any)
throwsExcB(any)
returnsA(any)
returnsB(any)
Method-Level Context
Abstract Program Fragments
Relevant Locations (Left)
LocSet relevant
Relevant Locations (Right)
LocSet relevant
Relational Postcondition
\result_1==\result_2
Relational Precondition
Relational Postcondition
Try to use tooltips if feeling unsure about the functionality of an element.
Proof State: No Proof



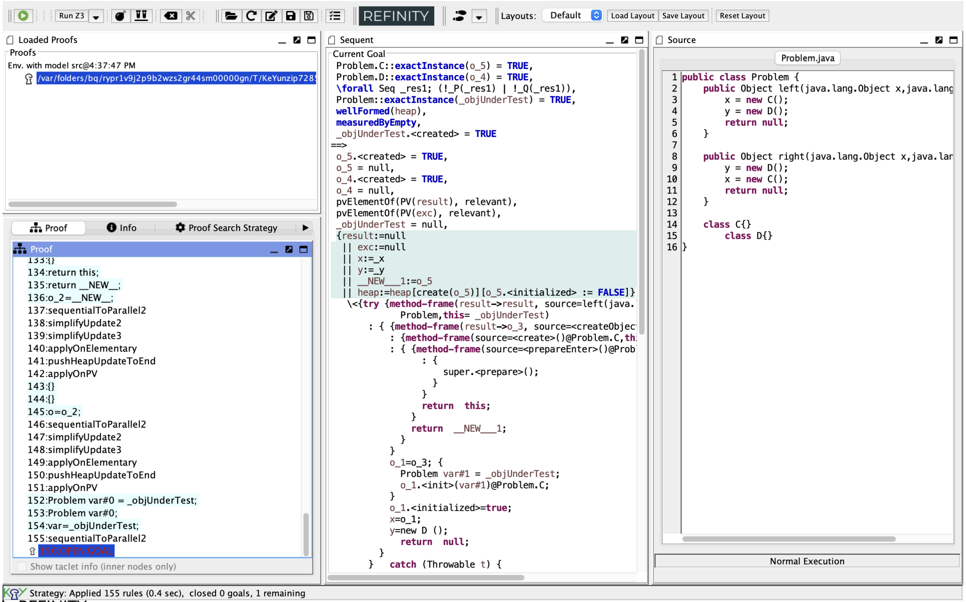
Example - slide stm. concrete ▷



Example - slide stm. conc. open goal ▷



Example - slide stm. conc. open goal ▷



Run Z3
REFINITY
Layouts: Default
Load Layout
Save Layout
Reset Layout
Loaded Proofs
Proofs
Env. with model src@4:37:47 PM
/var/folders/bq/rypr1v9j2p9b2wzs2gr44sm00000gn/T/KeYunzip728
Proof
Info
Proof Search Strategy
134:return this;
135:return __NEW__;
136:o_2=__NEW__;
137:sequentialToParallel2
138:simplifyUpdate2
139:simplifyUpdate3
140:applyOnElementary
141:pushHeapUpdateToEnd
142:applyOnPV
143:{}
144:{}
145:o=o_2;
146:sequentialToParallel2
147:simplifyUpdate2
148:simplifyUpdate3
149:applyOnElementary
150:pushHeapUpdateToEnd
151:applyOnPV
152:Problem var#0 = _objUnderTest;
153:Problem var#0;
154:var=_objUnderTest;
155:sequentialToParallel2
156:OPEN GOAL
Show taclet info (inner nodes only)
Sequent
Current Goal
Problem.C::exactInstance(o_5) = TRUE,
Problem.D::exactInstance(o_4) = TRUE,
\forall Seq _res1; (!_P(_res1) | !_Q(_res1)),
Problem::exactInstance(_objUnderTest) = TRUE,
wellFormed(heap),
measuredByEmpty,
_objUnderTest.<created> = TRUE
==>
o_5.<created> = TRUE,
o_5 = null,
o_4.<created> = TRUE,
o_4 = null,
pvElementOf(PV(result), relevant),
pvElementOf(PV(exc), relevant),
_objUnderTest = null,
{result:=null
|| exc:=null
|| x:=_x
|| y:=_y
|| __NEW___1:=o_5
|| heap:=heap[create(o_5)][o_5.<initialized> := FALSE]}
\<{try {method-frame(result->result, source=left(java.
Problem,this= _objUnderTest)
: { {method-frame(result->o_3, source=<createObjec
: {method-frame(source=<create>()@Problem.C,th
: { {method-frame(source=<prepareEnter>()@Prob
: {
super.<prepare>();
}
}
return this;
}
return __NEW___1;
}
}
o_1=o_3; {
Problem var#1 = _objUnderTest;
o_1.<init>(var#1)@Problem.C;
}
o_1.<initialized>=true;
x=o_1;
y=new D ();
return null;
}
} catch (Throwable t) {
Source
Problem.java
public class Problem {
public Object left(java.lang.Object x,java.lang
x = new C();
y = new D();
return null;
}
public Object right(java.lang.Object x,java.lar
y = new D();
x = new C();
return null;
}
class C{}
class D{}
}
Normal Execution
Strategy: Applied 155 rules (0.4 sec), closed 0 goals, 1 remaining

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:

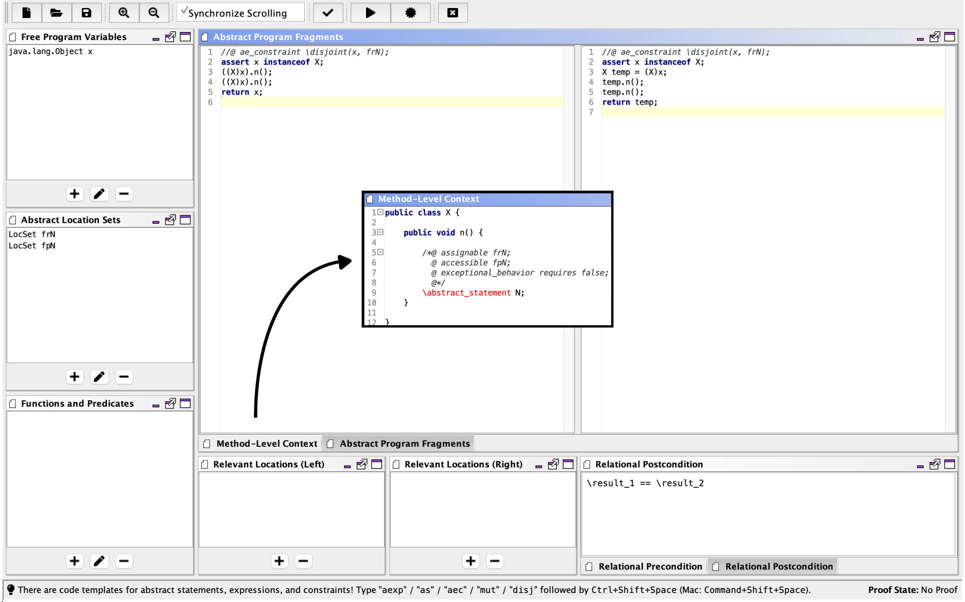
```
x.n();
x.n();
```

(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()`.

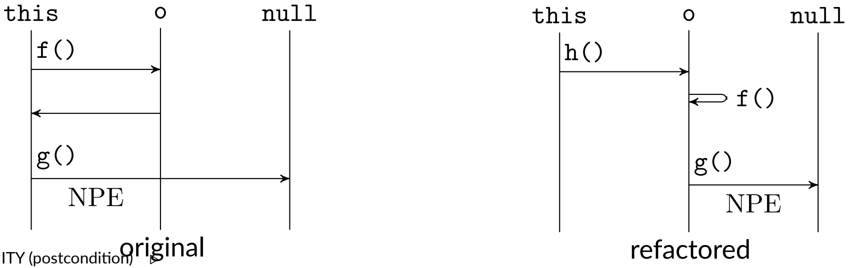

Synchronize Scrolling
Free Program Variables
java.lang.Object x
Abstract Location Sets
LocSet frN
LocSet fpN
Functions and Predicates
Abstract Program Fragments
//@ ae_constraint \disjoint(x, frN);
assert x instanceof X;
((X)x).n();
((X)x).n();
return x;
//@ ae_constraint \disjoint(x, frN);
assert x instanceof X;
X temp = (X)x;
temp.n();
temp.n();
return temp;
Method-Level Context
public class X {
public void n() {
/*@ assignable frN;
@ accessible fpN;
@ exceptional_behavior requires false;
@*/
\abstract_statement N;
}
}
Method-Level Context
Abstract Program Fragments
Relevant Locations (Left)
Relevant Locations (Right)
Relational Postcondition
\result_1 == \result_2
Relational Precondition
Relational Postcondition
There are code templates for abstract statements, expressions, and constraints! Type "aexp" / "as" / "aec" / "mut" / "disj" followed by Ctrl+Shift+Space (Mac: Command+Shift+Space).
Proof State: No Proof

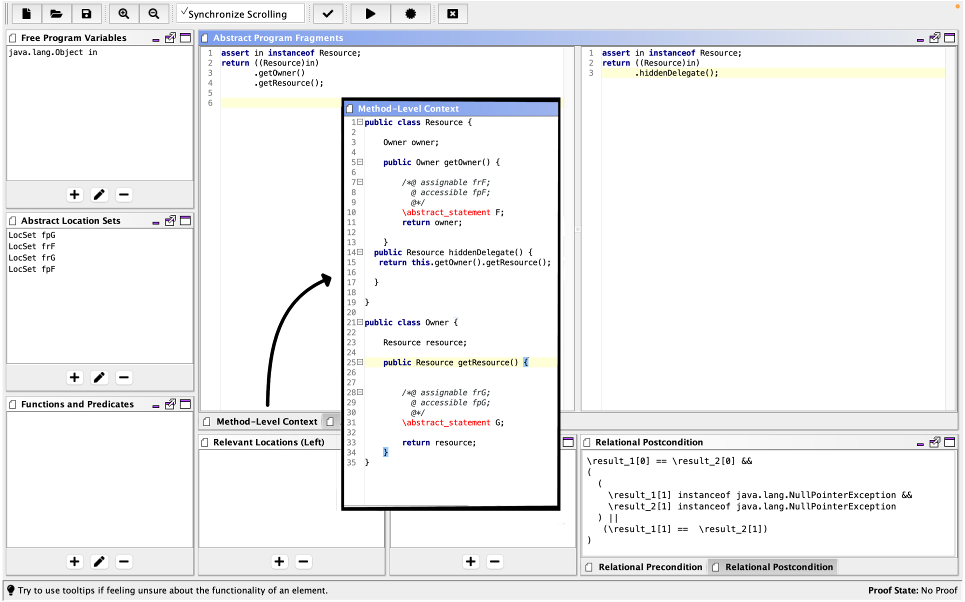
Different output, but equivalent?

Exception origin moved, no additional capture in `h()`

$$\begin{array}{ll} & \texttt{o.f().g();} \\ \rightarrow & \texttt{o.h();} \end{array}$$

with `h(){ this.f().g();}`




Synchronize Scrolling
Free Program Variables
java.lang.Object in
Abstract Location Sets
LocSet fpG
LocSet frF
LocSet frG
LocSet fpF
Functions and Predicates
Abstract Program Fragments
assert in instanceof Resource;
return ((Resource)in)
.getOwner()
.getResource();
assert in instanceof Resource;
return ((Resource)in)
.hiddenDelegate();
Method-Level Context
public class Resource {
Owner owner;
public Owner getOwner() {
/*@ assignable frF;
@ accessible fpF;
@*/
\abstract_statement F;
return owner;
}
public Resource hiddenDelegate() {
return this.getOwner().getResource();
}
}
public class Owner {
Resource resource;
public Resource getResource() {
/*@ assignable frG;
@ accessible fpG;
@*/
\abstract_statement G;
return resource;
}
}
Method-Level Context
Relevant Locations (Left)
Relational Postcondition
\result_1[0] == \result_2[0] &&
(
(
\result_1[1] instanceof java.lang.NullPointerException &&
\result_2[1] instanceof java.lang.NullPointerException
) ||
(\result_1[1] == \result_2[1])
)
Relational Precondition
Relational Postcondition
Try to use tooltips if feeling unsure about the functionality of an element.
Proof State: No Proof

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

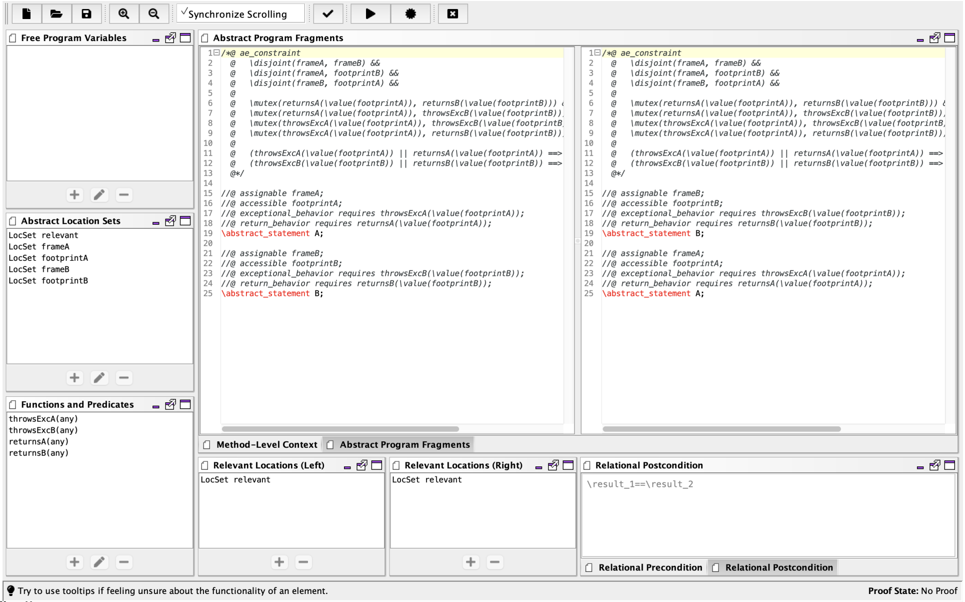
Unresolved

Making the proofs useful artefacts:
what about *instantitations*?

Object equality

REFINITY lacked rules for object equality over multiple modalities:

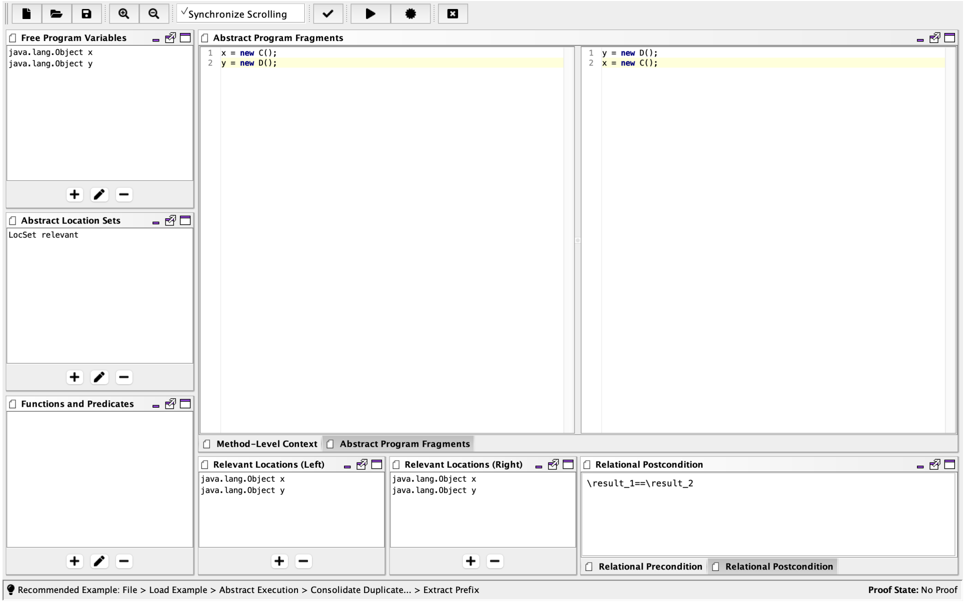
- can verify SLIDE STATEMENT with abstract statements


Synchronize Scrolling
Free Program Variables
Abstract Location Sets
LocSet relevant
LocSet frameA
LocSet footprintA
LocSet frameB
LocSet footprintB
Functions and Predicates
throwsExcA(any)
throwsExcB(any)
returnsA(any)
returnsB(any)
Abstract Program Fragments
/*@ ae_constraint
@ \disjoint(frameA, frameB) &&
@ \disjoint(frameA, footprintB) &&
@ \disjoint(frameB, footprintA) &&
@
@ \mutex(returnsA(\value(footprintA)), returnsB(\value(footprintB))) &
@ \mutex(returnsA(\value(footprintA)), throwsExcB(\value(footprintB)),
@ \mutex(throwsExcA(\value(footprintA)), throwsExcB(\value(footprintB,
@ \mutex(throwsExcA(\value(footprintA)), returnsB(\value(footprintB)),
@
@ (throwsExcA(\value(footprintA)) || returnsA(\value(footprintA)) ==>
@ (throwsExcB(\value(footprintB)) || returnsB(\value(footprintB)) ==>
@*/
//@ assignable frameA;
//@ accessible footprintA;
//@ exceptional_behavior requires throwsExcA(\value(footprintA));
//@ return_behavior requires returnsA(\value(footprintA));
\abstract_statement A;
//@ assignable frameB;
//@ accessible footprintB;
//@ exceptional_behavior requires throwsExcB(\value(footprintB));
//@ return_behavior requires returnsB(\value(footprintB));
\abstract_statement B;
/*@ ae_constraint
@ \disjoint(frameA, frameB) &&
@ \disjoint(frameA, footprintB) &&
@ \disjoint(frameB, footprintA) &&
@
@ \mutex(returnsA(\value(footprintA)), returnsB(\value(footprintB))) &
@ \mutex(returnsA(\value(footprintA)), throwsExcB(\value(footprintB)),
@ \mutex(throwsExcA(\value(footprintA)), throwsExcB(\value(footprintB,
@ \mutex(throwsExcA(\value(footprintA)), returnsB(\value(footprintB)),
@
@ (throwsExcA(\value(footprintA)) || returnsA(\value(footprintA)) ==>
@ (throwsExcB(\value(footprintB)) || returnsB(\value(footprintB)) ==>
@*/
//@ assignable frameB;
//@ accessible footprintB;
//@ exceptional_behavior requires throwsExcB(\value(footprintB));
//@ return_behavior requires returnsB(\value(footprintB));
\abstract_statement B;
//@ assignable frameA;
//@ accessible footprintA;
//@ exceptional_behavior requires throwsExcA(\value(footprintA));
//@ return_behavior requires returnsA(\value(footprintA));
\abstract_statement A;
Method-Level Context
Abstract Program Fragments
Relevant Locations (Left)
LocSet relevant
Relevant Locations (Right)
LocSet relevant
Relational Postcondition
\result_1==\result_2
Relational Precondition
Relational Postcondition
Try to use tooltips if feeling unsure about the functionality of an element.
Proof State: No Proof

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


Synchronize Scrolling
Free Program Variables
java.lang.Object x
java.lang.Object y
Abstract Location Sets
LocSet relevant
Functions and Predicates
Abstract Program Fragments
1 x = new C();
2 y = new D();
1 y = new D();
2 x = new C();
Method-Level Context
Abstract Program Fragments
Relevant Locations (Left)
java.lang.Object x
java.lang.Object y
Relevant Locations (Right)
java.lang.Object x
java.lang.Object y
Relational Postcondition
\result_1==\result_2
Relational Precondition
Relational Postcondition
Recommended Example: File > Load Example > Abstract Execution > Consolidate Duplicate... > Extract Prefix
Proof State: No Proof

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 under certain conditions

New *taclet* for object creation

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

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.*

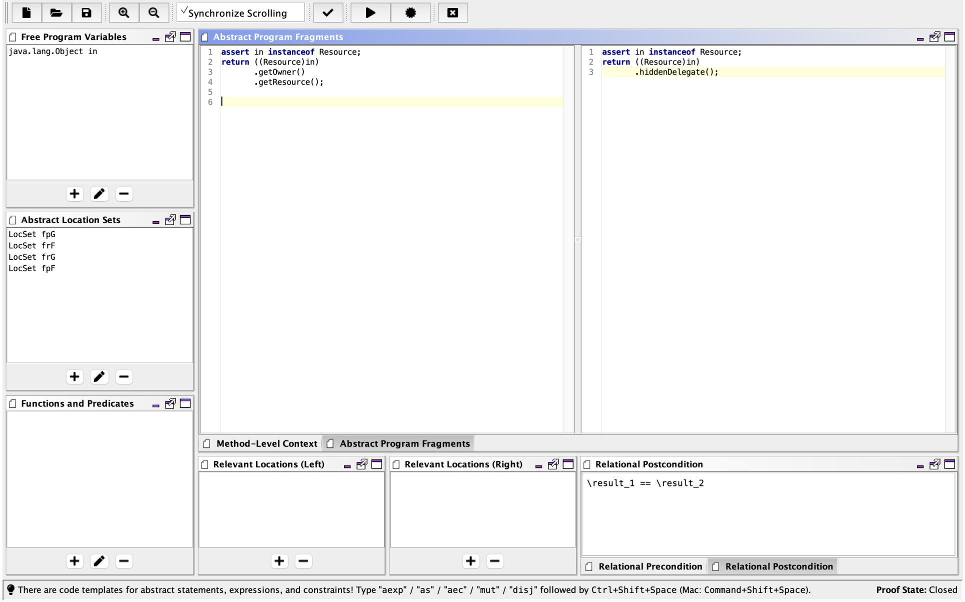
$$\mathtt{C} :: \mathtt{allocate}(\mathtt{store}(\mathtt{h}, \mathtt{o}, \mathtt{f}, \mathtt{v})) \rightsquigarrow \mathtt{C} :: \mathtt{allocate}(\mathtt{h}) \quad \textit{if } f \neq \texttt{<allocated>}$$

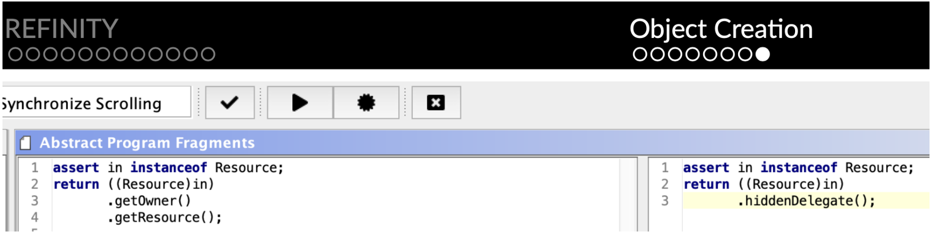
$$\mathtt{C} :: \mathtt{allocate}(\mathtt{create}(\mathtt{h}, \mathtt{o})) \rightsquigarrow \mathtt{C} :: \mathtt{allocate}(\mathtt{h}) \quad \textit{if } C \not\sqsubseteq T(o)$$

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 🙃)


Synchronize Scrolling
Free Program Variables
java.lang.Object in
Abstract Location Sets
LocSet fpG
LocSet frF
LocSet frG
LocSet fpF
Functions and Predicates
Abstract Program Fragments
assert in instanceof Resource;
return ((Resource)in)
.getOwner()
.getResource();
assert in instanceof Resource;
return ((Resource)in)
.hiddenDelegate();
Method-Level Context
Abstract Program Fragments
Relevant Locations (Left)
Relevant Locations (Right)
Relational Postcondition
\result_1 == \result_2
Relational Precondition
Relational Postcondition
There are code templates for abstract statements, expressions, and constraints! Type "aexp" / "as" / "aec" / "mut" / "disj" followed by Ctrl+Shift+Space (Mac: Command+Shift+Space).
Proof State: Closed


REFINITY
Object Creation
Synchronize Scrolling
Abstract Program Fragments
assert in instanceof Resource;
return ((Resource)in)
.getOwner()
.getResource();
assert in instanceof Resource;
return ((Resource)in)
.hiddenDelegate();

Future Challenge

Trace based notions of equivalence

```
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;
```

(a) Before

```
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;
```

(b) After

$$\theta ::= \lceil \phi \rceil$$

$$\mid \mathsf{call}(\mathtt{m})$$

$$\mid \mathbf{finite}$$

$$\mid \theta \ast\ast \theta$$

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

SILM Workshop

Dates | Call for Papers | Submission | Program | Registration | Past Events

SILM 2024

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&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.