Lightning Talks

What happened in KeY since the last Symposium?

1. KeY on GitHub (Alexander Weigl)
2. Assume and Assert (Florian Lanzinger)
3. Free Invariants and Assignables (Florian Lanzinger)
4. Math modes (Mattias Ulbrich)
5. Proof Management (Wolfram Pfeifer)
6. Support of Final Fields (Mattias Ulbrich)
7. Redux Maker Tool (Lukas Grätz)
8. Support for Java 21 (Alexander Weigl)
9. Towards Language Independent Formulas (Daniel Drodt)
10. SolidiKeY (Wolfgang Ahrendt)

KeY on Github

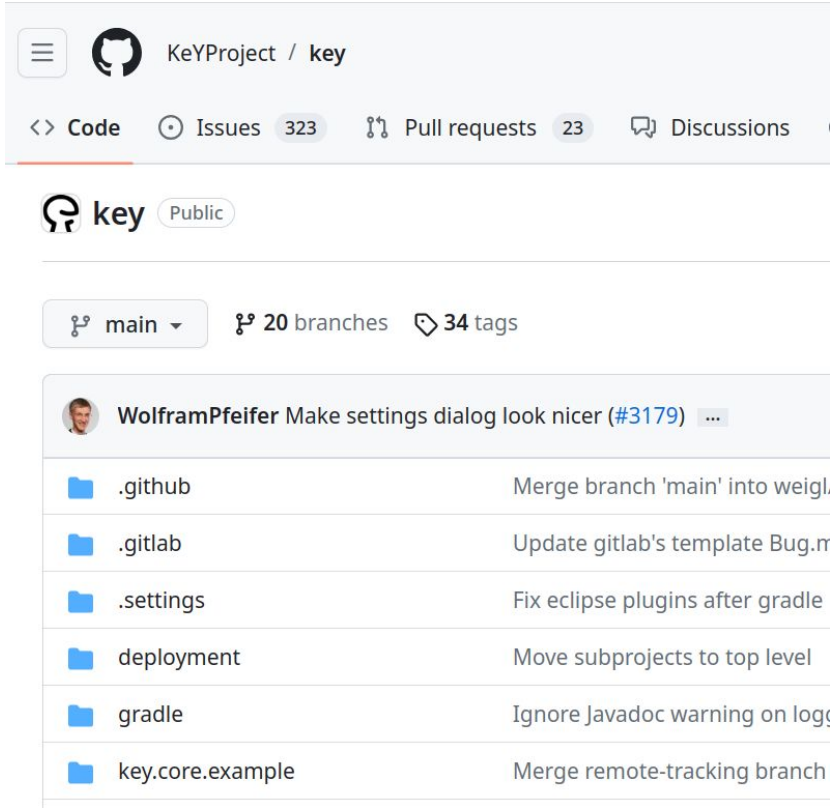
Alexander Weigl


- We moved to Github on Feb, 01.
- Main development and sources are now under: https://github.com/**keyproject/key**

KeY on GitHub

- Feel free to collaborate
- Rights for merging can be granted

- Gitlab instance remains



Home for Case Studies

Gathering of Case Studies

Archive, Reuse, Knowledge Base,
Proof Mining, …

⛏️ + 🌳 = 💎

TimSort Public

Resources of the TimmSort Case Study

Java 0 0 1 0 Updated on Jun 13

VerifyingIdentityHashMap Public

Forked from m4ndeb2r/IM9906-2-VerifyingIdentityHashMap

KeY verification case study in which we verify Java's IdentityHashMap with JML and KeY.

Java 1 0 0 0 Updated on Mar 25, 2022

DualPivotQuickSort Public

Java 0 0 0 0 Updated on Jun 14

Please contribute your
KeY Case Studies

Assume and Assert

Florian Lanzinger

Assume and assert statements for JML

```
1  public class AssertAssumeDemo {
2
3      public static int field;
4
5      /*@ public normal_behavior
6        @ ensures field == 42;
7        @*/
8      public static void foo() {
9          //@ assume field == 40;
10         ++field;
11         //@ assert field == 41;
12         ++field;
13     }
14 }
```

Proof tree:

- JML assume: Usage (add `field == 40` to antecedent)
- JML assert
  - Validity (add `field == 41` to succedent)
  - Usage (add `field == 41` to antecedent)

Free Invariants and Assignables

Florian Lanzinger

Free specifications in JML

- Specifications that can be used without being proven are called “free”
- Unsound if used incorrectly
- But useful to encode information from previously done proofs / other tools / etc.
- So far: `requires_free`, `ensures_free`, `assume`

```
1  public class FreeSpecifications {
2      public static int field;
3
4      /*@ public normal_behavior
5        @ requires_free field == 41;
6        @ ensures field == 42;
7        @*/
8      public static void foo() {
9          ++field;
10     }
11     /*@ public normal_behavior
12       @ ensures field == 42;
13       @*/
14     public static void bar() { foo(); }
15 }
```

Free specifications in JML

- So far: `requires_free`, `ensures_free`, `assume`
- New: `assignable_free`

```
1  public class FreeSpecifications {
2      public static int field;
3
4      /*@ public normal_behavior
5        @ ensures true;
6        @ assignable \nothing;
7        @ assignable_free field; */
8      void bar() { field = 42; }
9
10     /*@ public normal_behavior
11       @ ensures field == 21; */
12     void foo() {
13         field = 21;
14         bar();
15     }
16 }
```

Free specifications in JML

- So far: `requires_free`, `ensures_free`, `assume`
- New: `assignable_free`
- New: `invariant_free`

```
1  public class FreeSpecifications {
2      int f; int g;
3      //@ invariant f > 0;
4      //@ invariant_free g > 0;
5
6      /*@ normal_behavior
7        @ ensures true; */
8      FreeSpecifications() {
9          f = 1; g = 0;
10     }
11
12     /*@ normal_behavior
13       @ ensures \result > 0;
14       @*/
15     static int foo() {
16         return new FreeSpecifications().g;
17     }
```

Free specifications in JML

- So far: `requires_free`, `ensures_free`, `assume`
- New: `assignable_free`
- New: `invariant_free`

```
1  public class FreeSpecifications {
2      int f; int g;
3      //@ invariant f > 0;
4      //@ invariant_free g > 0;
5
6      /*@ normal_behavior
7        @ ensures true; */
8      FreeSpecifications() {
9          f = 1; g = 0;
10     }
11
12     /*@ normal_behavior
13       @ requires \invariant_free_for(a);
14       @ ensures \result > 0;
15       @*/
16     static int bar(FreeSpecifications a) {
17         return a.g;
18     }
```

Math Mode

Mattias Ulbrich

Math Semantics in Specifications

```
//@ ensures \result == x+1;
int m(int x) { return x+1; }
```

violates spec! (acc. to JML)

Use \spec_bigint_math (default) for \bigint
\spec_safe_math for int w/o overflow
\spec_java_math for int w/ overflow

on contracts or classes.

(In expressions use \java_math(...) etc.)

Proof Management

Wolfram Pfeifer

Problems with verification projects (> 1 contract)

- Source code, specification, proofs consistent?
- Unproven (but used) contracts?
- Proof settings compatible?
- Dependency cycles?

KeY has a built-in proof management. However, it

- only considers proofs loaded at the same time in the GUI.
- only considers proofs in the same environment.
- does not check settings.
- does not check all dependencies (model methods, dependency contracts, …).

Proof Bundles

- Directory or zip file
- “File”→“Save Proof as Bundle” (single proof)
- Can be merged via `./pm merge b1 b2 … output`

```
bundle.zproof
- src           // java classes (.java files only)
  - A.java
  - mypackage
    - B.java
    - C.java
- classpath     // optional: may contain .jar files and
                //  directories with .java and/or .class files
- bootclasspath // optional: system classes from the Java class
                //  library, replace the files shipped with KeY
  // .key and .proof files (top level)
- rules.key
- project.key
- A[A::m1(int)].JML operation contract.0.proof
- A[A::m2(int)].JML operation contract.0.proof
- B[B::m1(int)].JML operation contract.0.proof
- C[C::m1(int)].JML operation contract.0.proof
```

Proof Management Tool

Post-hoc checks (CLI or GUI extension)

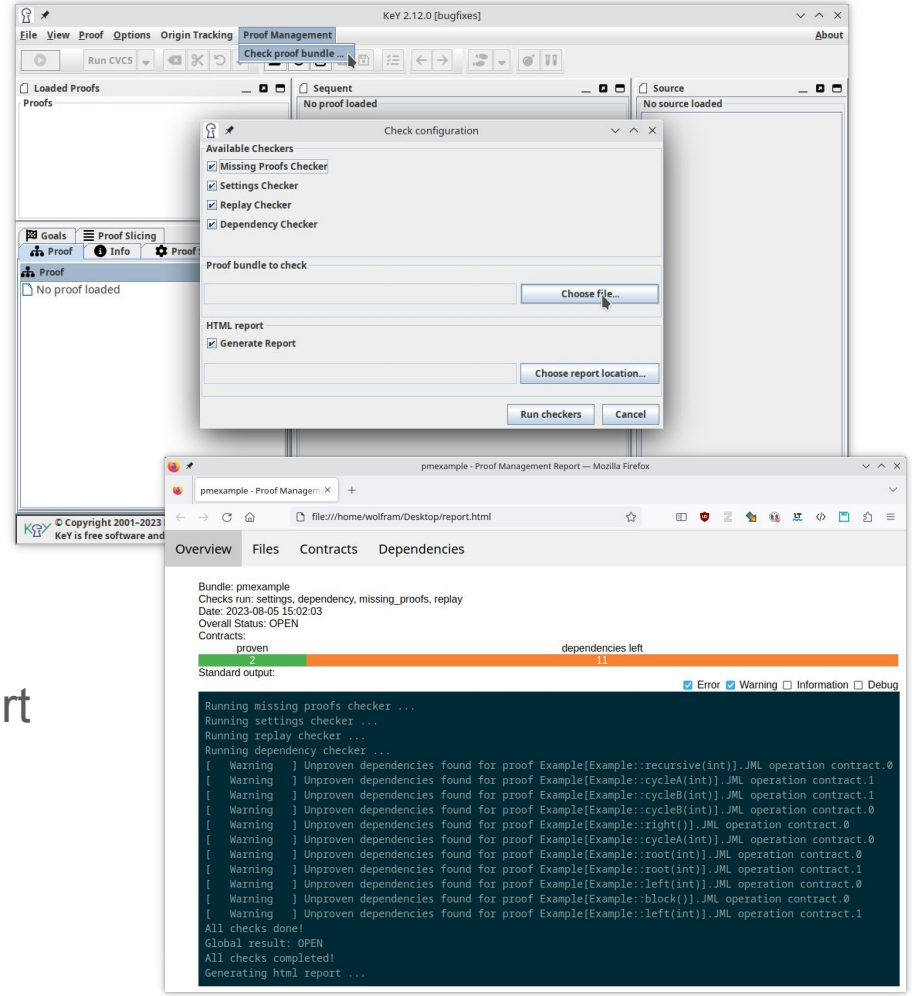
./pm check --settings --replay bundle

- MissingProofsChecker
- SettingsChecker
- ReplayChecker
- DependencyChecker

→ outputs a command line or HTML report

Documentation: Readme in repo

Outlook: We can do better than post-hoc checks → talk by Daniel Drodt



Support of Final Fields

Mattias Ulbrich

Expensive heaps

Reading fields from heaps can be **expensive during reasoning** (for the calculus)

Consider: `class F { final int f; }`,
then by design of Java, `f` cannot change its value.

Yet, difficult to prove in KeY:

```
    obj.f@heap[other.x := 42][anon(other.fp, h2)][create(obj2)]
  = obj.f
```

Solution: Final fields are not on the heap

Heap:

```
T T::select(Heap, Object, Field)
Heap store(Heap, Object, Field, Any)
…
```

Finals:

`T T::final(Object, Field)` … do not care about the heap

Challenge:

Finals may be changed in constructors (--> do not use this then)

Outlook: Heaps and Dependencies

Framing Properties are still a challenge in KeY

Dynamic Frames are a very flexible solution,
yet require a lot of reasoning not needed in 90% of the cases

Working on:

- Final fields (as shown)
- Exploit patterns in dynamic frame specifications to simplify obligations
- Combining ownership with dynamic frames

Redux MakerTool

Lukas Grätz
(Master Thesis by Fabian Bauer)

Status Quo in KeY

- Referenced classes must be loaded
  - Otherwise KeY can’t *start*
  - Loading all sources takes too long
- Old StubMaker
  - Eclipse plugin
  - Input: Java sources ≤ 1.6
  - Output: Stubs
- Hand-written contracts
  - Not exhaustive
  - Some wrong/inconsistent

Goal

- Stand alone tool
- Support for modern Java
- Automatic contract generation

Result: ReduxMaker

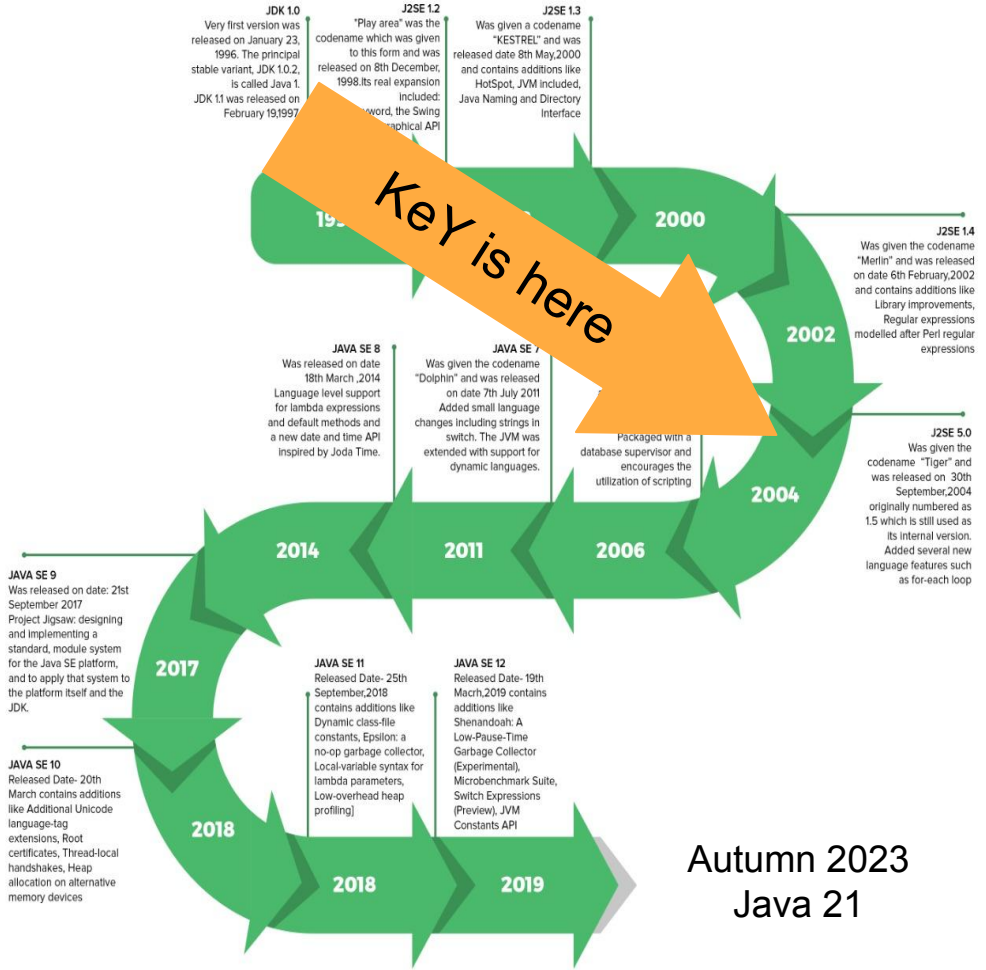
- Implemented
  - Easy to use, stand-alone tool
  - Contract templates
- Input
  - Java source or bytecode
- Output
  - Stubs
  - Assignable clauses
  - Termination behavior (diverges or not)
  - Contracts from the templates
- TODO
  - Publish
  - Explain in KeY’s wiki

Support for Java 21

Alexander Weigl

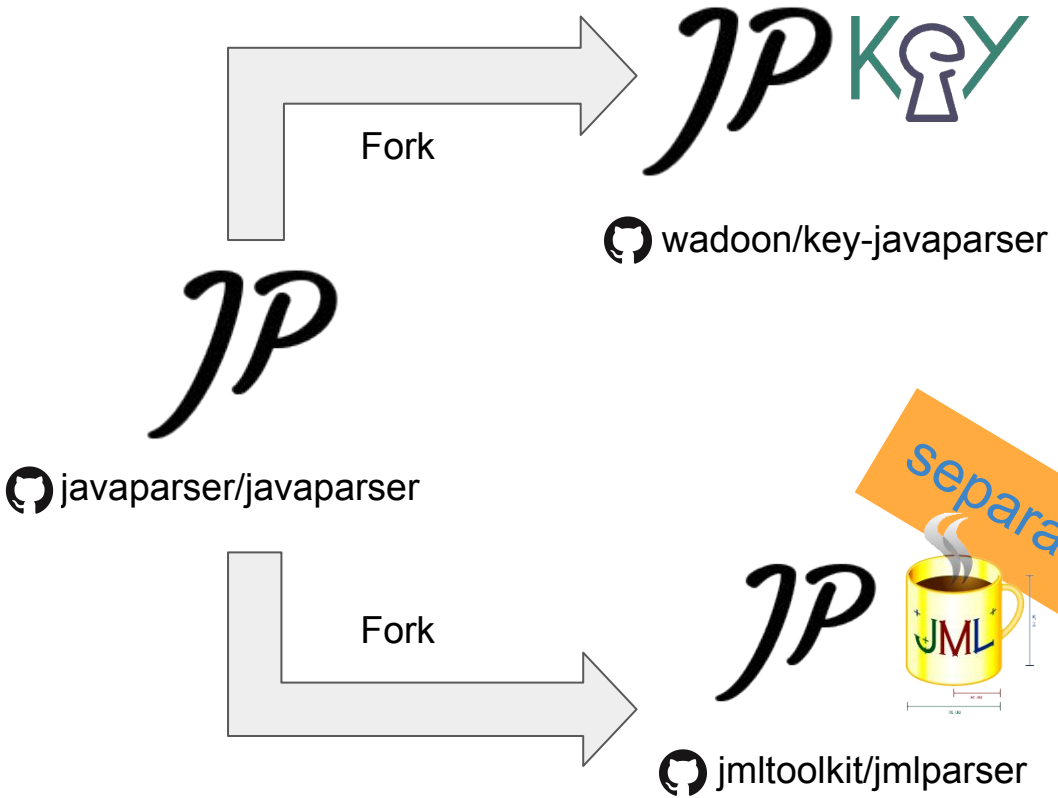
State of Java-Support

- KeY supports **Java 1.4*ish***
- Main Blocker:
  Recoder Framework
  for parsing and Java



Autumn 2023
Java 21

JavaParser

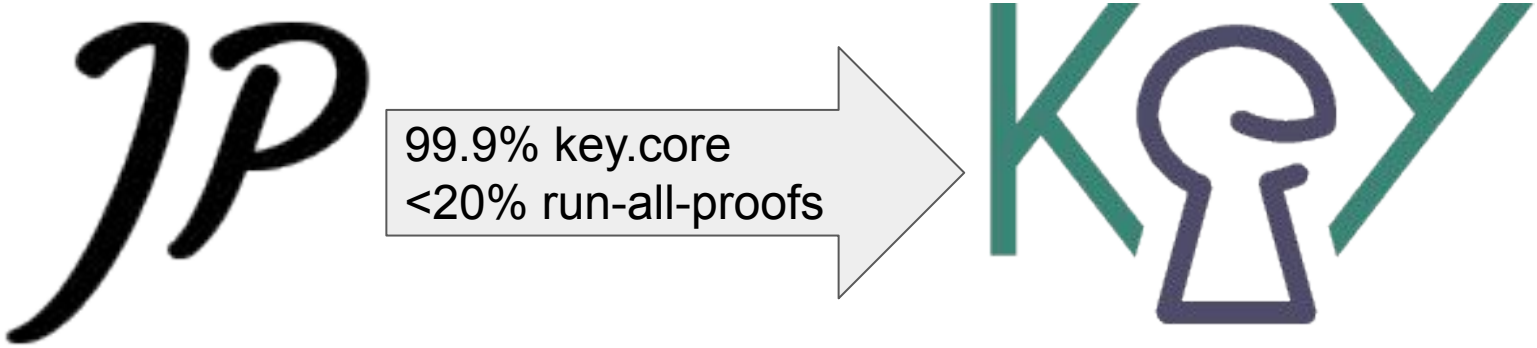


Parser, AST and Name Resolution for

ProofJava and **SchemaJava**

Support for: method frame, loop scopes, schema variables, transaction commands, meta-constructs, #typeof, …

State of integration



Towards Language Independent Formulas

Daniel Drodt

Making KeY Language Independent

- KeY is designed for Java
- We aim to make it usable for many languages
- First task: Terms
  - Current term structure is designed for Java
  - Interfaces and matching logic is not language-independent
  - We want to redesign terms, formulas and language elements

```
36  public interface Term
37      extends SVSubstitute,
38              Sorted,
39              EqualsModProofIrrelevancy
40  {
41    /**
42     * The Java block at top level.
43     */
44    JavaBlock javaBlock();
45  }
46
```

If you have comments/input, please reach out!

SolidiKeY

Wolfgang Ahrendt

KeY for Smart Contracts: Logic and Calculus

- Datatypes:
  - Fixed sized arrays
  - Dynamic arrays
  - Structs
  - Maps, e.g.: `mapping (address => uint) public balances`
- Datatypes can be nested, but not recursive
- Fixed depth: no null, defaults instead
- “memory” (local variables): references, aliases, basically a heap
- “storage” (persistent memory): value semantics, no references, no aliasing

KeY for Smart Contracts: Logic and Calculus

- Datatypes:
  - fixed sized arrays
  - dynamic arrays
  - structs
  - maps, e.g.: `mapping (address => uint) public balances`
- Datatypes can be nested, but not recursive
- Fixed depth: no null, defaults instead
- “memory” (local variables): references, aliases, basically a heap
- “storage” (persistent memory): value semantics, no references, no aliasing

Quest: we need back the flexible functions (from old KeY)

Why: explicit heap is in the way for modelling storage and maps