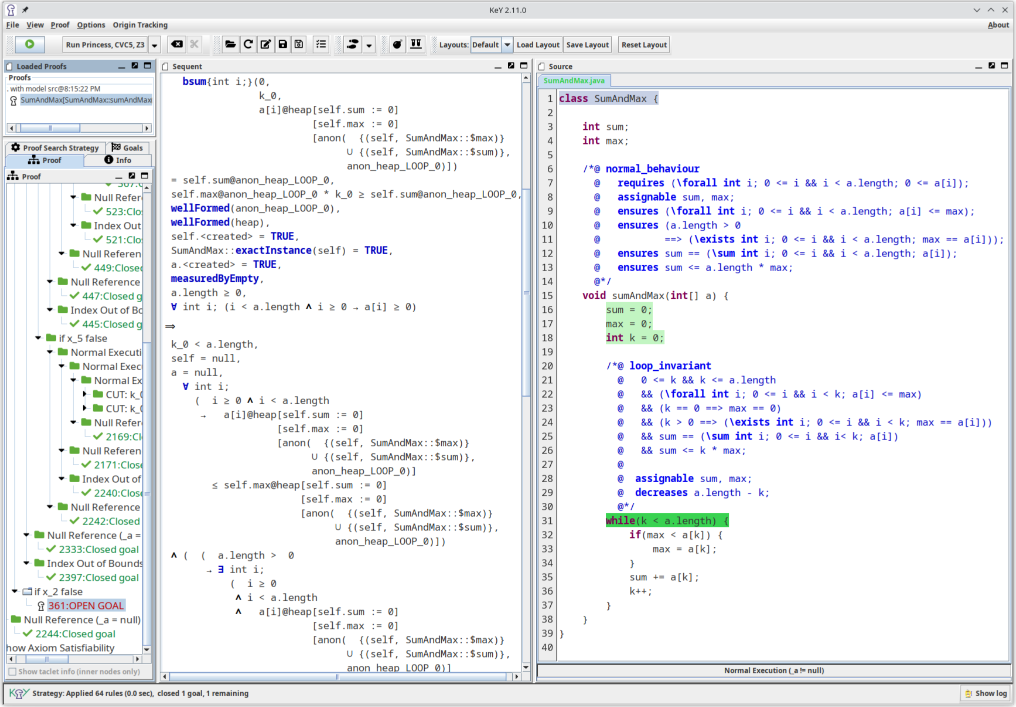
Advancements in User Interface and Usability of KeY

Wolfram Pfeifer | August 9, 2023



Proofs
. with model src@8:15:22 PM
SumAndMax[SumAndMax::sumAndMax

Proof
Null Refer
523:Clo
Index Out
521:Clo
Null Referen
449:Closed
Null Reference
447:Closed g
Index Out of Bo
445:Closed g
if x_5 false
Normal Executi
Normal Exec
Normal Ex
CUT: k_(
CUT: k_(
Null Refer
2169:Cl
Null Referen
2171:Close
Index Out of
2240:Close
Null Reference
2242:Closed
Null Reference (_a =
2333:Closed goal
Index Out of Bounds
2397:Closed goal
if x_2 false
361:OPEN GOAL
Null Reference (_a = null)
2244:Closed goal
how Axiom Satisfiability

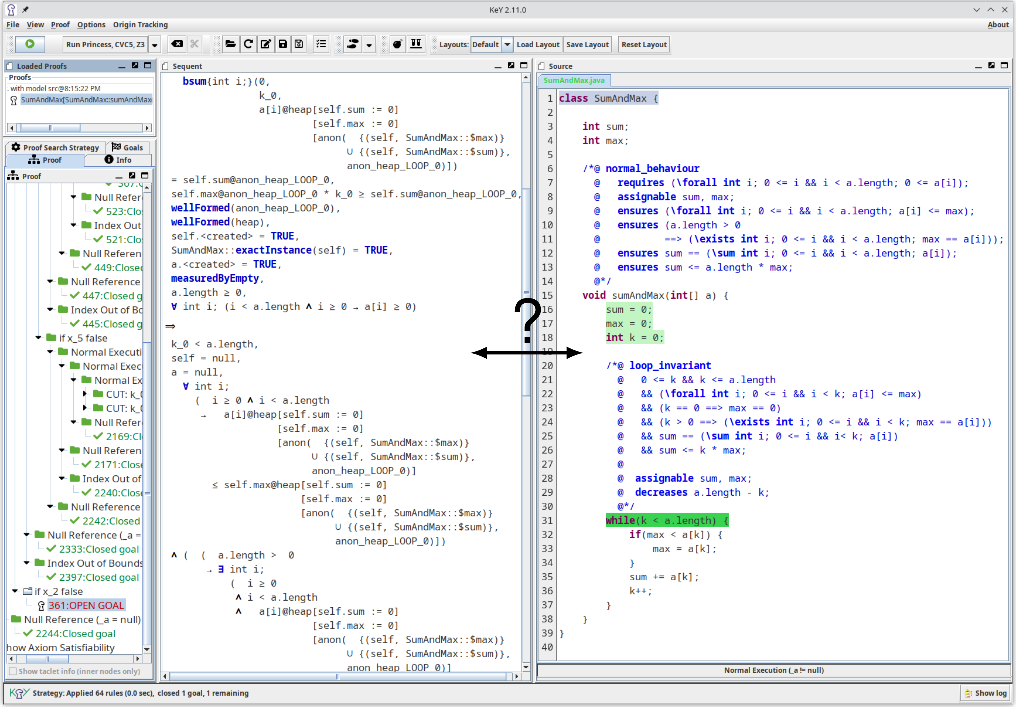
☐ Show taclet info (inner nodes only)

```
   bsum{int i;}(0,
                k_0,
                a[i]@heap[self.sum := 0]
                         [self.max := 0]
                         [anon(  {(self, SumAndMax::$max)}
                               ∪ {(self, SumAndMax::$sum)},
                               anon_heap_LOOP_0)])
 = self.sum@anon_heap_LOOP_0,
 self.max@anon_heap_LOOP_0 * k_0 ≥ self.sum@anon_heap_LOOP_0,
 wellFormed(anon_heap_LOOP_0),
 wellFormed(heap),
 self.<created> = TRUE,
 SumAndMax::exactInstance(self) = TRUE,
 a.<created> = TRUE,
 measuredByEmpty,
 a.length ≥ 0,
 ∀ int i; (i < a.length ∧ i ≥ 0 → a[i] ≥ 0)

⟹

 k_0 < a.length,
 self = null,
 a = null,
   ∀ int i;
     (  i ≥ 0 ∧ i < a.length
       →   a[i]@heap[self.sum := 0]
                    [self.max := 0]
                    [anon(  {(self, SumAndMax::$max)}
                          ∪ {(self, SumAndMax::$sum)},
                          anon_heap_LOOP_0)]
        ≤ self.max@heap[self.sum := 0]
                       [self.max := 0]
                       [anon(  {(self, SumAndMax::$max)}
                             ∪ {(self, SumAndMax::$sum)},
                             anon_heap_LOOP_0)])
 ∧ (  (  a.length >  0
        → ∃ int i;
            (  i ≥ 0
             ∧ i < a.length
             ∧    a[i]@heap[self.sum := 0]
                           [self.max := 0]
                           [anon(  {(self, SumAndMax::$max)}
                                 ∪ {(self, SumAndMax::$sum)},
                                 anon_heap_LOOP_0)]
```

```java
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a[i]);
      @   assignable sum, max;
      @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <= max);
      @   ensures (a.length > 0
      @           ==> (\exists int i; 0 <= i && i < a.length; max == a[i]));
      @   ensures sum == (\sum int i; 0 <= i && i < a.length; a[i]);
      @   ensures sum <= a.length * max;
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.length
          @   && (\forall int i; 0 <= i && i < k; a[i] <= max)
          @   && (k == 0 ==> max == 0)
          @   && (k > 0 ==> (\exists int i; 0 <= i && i < k; max == a[i]))
          @   && sum == (\sum int i; 0 <= i && i< k; a[i])
          @   && sum <= k * max;
          @
          @  assignable sum, max;
          @  decreases a.length - k;
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```

Normal Execution (_a != null)

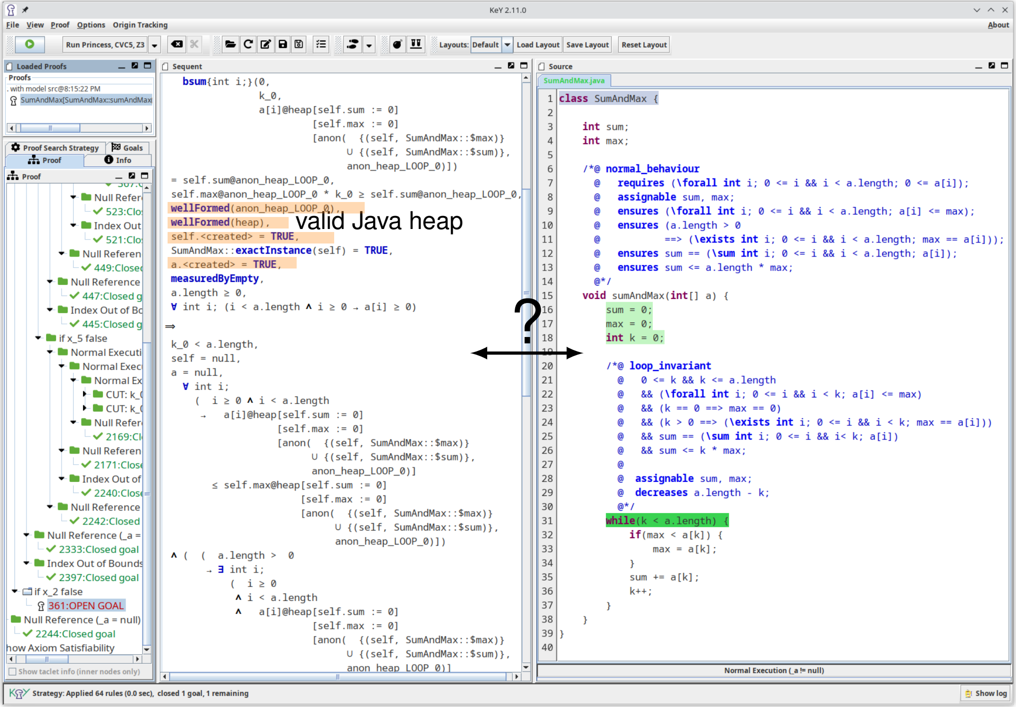
Strategy: Applied 64 rules (0.0 sec), closed 1 goal, 1 remaining



```
  bsum{int i;}(0,
               k_0,
               a[i]@heap[self.sum := 0]
                        [self.max := 0]
                        [anon(  {(self, SumAndMax::$max)}
                              ∪ {(self, SumAndMax::$sum)},
                              anon_heap_LOOP_0)])
= self.sum@anon_heap_LOOP_0,
self.max@anon_heap_LOOP_0 * k_0 ≥ self.sum@anon_heap_LOOP_0,
wellFormed(anon_heap_LOOP_0),
wellFormed(heap),
self.<created> = TRUE,
SumAndMax::exactInstance(self) = TRUE,
a.<created> = TRUE,
measuredByEmpty,
a.length ≥ 0,
∀ int i; (i < a.length ∧ i ≥ 0 → a[i] ≥ 0)

⇒

k_0 < a.length,
self = null,
a = null,
  ∀ int i;
    (  i ≥ 0 ∧ i < a.length
     →   a[i]@heap[self.sum := 0]
                  [self.max := 0]
                  [anon(  {(self, SumAndMax::$max)}
                        ∪ {(self, SumAndMax::$sum)},
                        anon_heap_LOOP_0)]
       ≤ self.max@heap[self.sum := 0]
                      [self.max := 0]
                      [anon(  {(self, SumAndMax::$max)}
                            ∪ {(self, SumAndMax::$sum)},
                            anon_heap_LOOP_0)])
∧ (  (  a.length >  0
      → ∃ int i;
          (  i ≥ 0
           ∧ i < a.length
           ∧    a[i]@heap[self.sum := 0]
                          [self.max := 0]
                          [anon(  {(self, SumAndMax::$max)}
                                ∪ {(self, SumAndMax::$sum)},
                                anon_heap_LOOP_0)]
```

```java
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a[i]);
      @   assignable sum, max;
      @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <= max);
      @   ensures (a.length > 0
      @           ==> (\exists int i; 0 <= i && i < a.length; max == a[i]));
      @   ensures sum == (\sum int i; 0 <= i && i < a.length; a[i]);
      @   ensures sum <= a.length * max;
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.length
          @   && (\forall int i; 0 <= i && i < k; a[i] <= max)
          @   && (k == 0 ==> max == 0)
          @   && (k > 0 ==> (\exists int i; 0 <= i && i < k; max == a[i]))
          @   && sum == (\sum int i; 0 <= i && i< k; a[i])
          @   && sum <= k * max;
          @
          @  assignable sum, max;
          @  decreases a.length - k;
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```

Strategy: Applied 64 rules (0.0 sec), closed 1 goal, 1 remaining

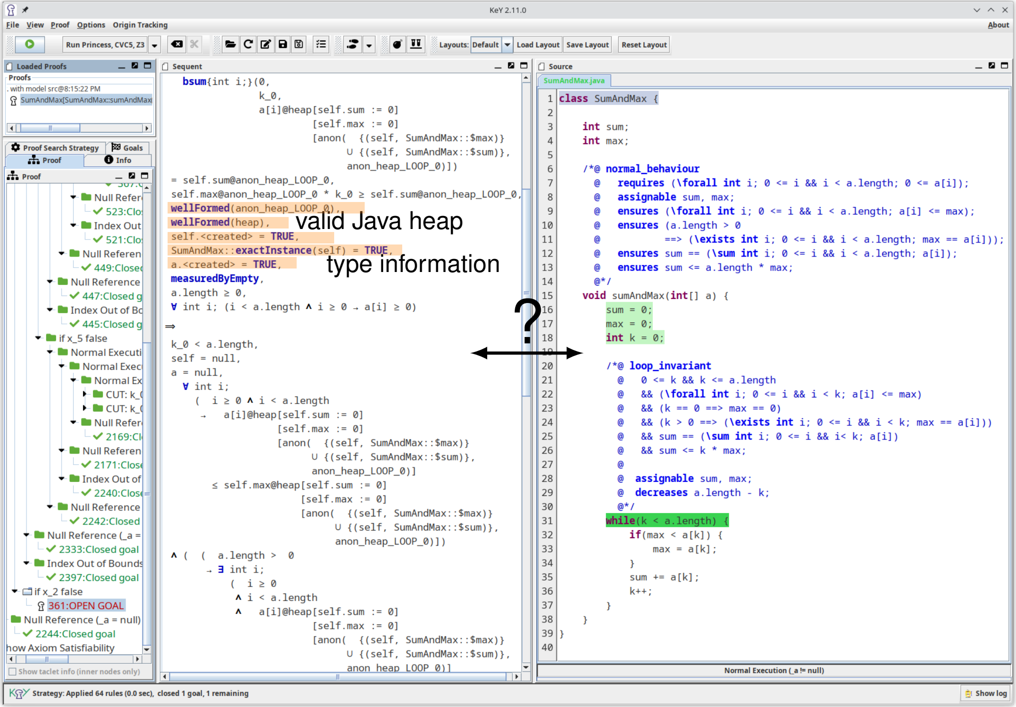


```
  bsum{int i;}(0,
               k_0,
               a[i]@heap[self.sum := 0]
                        [self.max := 0]
                        [anon(  {(self, SumAndMax::$max)}
                              ∪ {(self, SumAndMax::$sum)},
                              anon_heap_LOOP_0)])
 = self.sum@anon_heap_LOOP_0,
 self.max@anon_heap_LOOP_0 * k_0 ≥ self.sum@anon_heap_LOOP_0,
 wellFormed(anon_heap_LOOP_0),
 wellFormed(heap),
 self.<created> = TRUE,
 SumAndMax::exactInstance(self) = TRUE,
 a.<created> = TRUE,
 measuredByEmpty,
 a.length ≥ 0,
 ∀ int i; (i < a.length ∧ i ≥ 0 → a[i] ≥ 0)

⇒

 k_0 < a.length,
 self = null,
 a = null,
   ∀ int i;
     (  i ≥ 0 ∧ i < a.length
      →   a[i]@heap[self.sum := 0]
                   [self.max := 0]
                   [anon(  {(self, SumAndMax::$max)}
                         ∪ {(self, SumAndMax::$sum)},
                         anon_heap_LOOP_0)]
        ≤ self.max@heap[self.sum := 0]
                       [self.max := 0]
                       [anon(  {(self, SumAndMax::$max)}
                             ∪ {(self, SumAndMax::$sum)},
                             anon_heap_LOOP_0)])
 ∧ (  (  a.length >  0
        → ∃ int i;
            (  i ≥ 0
             ∧ i < a.length
             ∧   a[i]@heap[self.sum := 0]
                           [self.max := 0]
                           [anon(  {(self, SumAndMax::$max)}
                                 ∪ {(self, SumAndMax::$sum)},
                                 anon_heap_LOOP_0)]
```

valid Java heap

?

```
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a[i]);
      @   assignable sum, max;
      @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <= max);
      @   ensures (a.length > 0
      @           ==> (\exists int i; 0 <= i && i < a.length; max == a[i]));
      @   ensures sum == (\sum int i; 0 <= i && i < a.length; a[i]);
      @   ensures sum <= a.length * max;
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.length
          @   && (\forall int i; 0 <= i && i < k; a[i] <= max)
          @   && (k == 0 ==> max == 0)
          @   && (k > 0 ==> (\exists int i; 0 <= i && i < k; max == a[i]))
          @   && sum == (\sum int i; 0 <= i && i< k; a[i])
          @   && sum <= k * max;
          @
          @  assignable sum, max;
          @  decreases a.length - k;
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```



```
  bsum{int i;}(0,
               k_0,
               a[i]@heap[self.sum := 0]
                        [self.max := 0]
                        [anon(  {(self, SumAndMax::$max)}
                              ∪ {(self, SumAndMax::$sum)},
                              anon_heap_LOOP_0)])
= self.sum@anon_heap_LOOP_0,
self.max@anon_heap_LOOP_0 * k_0 ≥ self.sum@anon_heap_LOOP_0,
wellFormed(anon_heap_LOOP_0),
wellFormed(heap),
self.<created> = TRUE,
SumAndMax::exactInstance(self) = TRUE,
a.<created> = TRUE,
measuredByEmpty,
a.length ≥ 0,
∀ int i; (i < a.length ∧ i ≥ 0 → a[i] ≥ 0)

⟹

k_0 < a.length,
self = null,
a = null,
  ∀ int i;
    (  i ≥ 0 ∧ i < a.length
     →   a[i]@heap[self.sum := 0]
                  [self.max := 0]
                  [anon(  {(self, SumAndMax::$max)}
                        ∪ {(self, SumAndMax::$sum)},
                        anon_heap_LOOP_0)]
       ≤ self.max@heap[self.sum := 0]
                      [self.max := 0]
                      [anon(  {(self, SumAndMax::$max)}
                            ∪ {(self, SumAndMax::$sum)},
                            anon_heap_LOOP_0)])
∧ (  (  a.length >  0
      → ∃ int i;
          (  i ≥ 0
           ∧ i < a.length
           ∧    a[i]@heap[self.sum := 0]
                         [self.max := 0]
                         [anon(  {(self, SumAndMax::$max)}
                               ∪ {(self, SumAndMax::$sum)},
                               anon_heap_LOOP_0)]
```

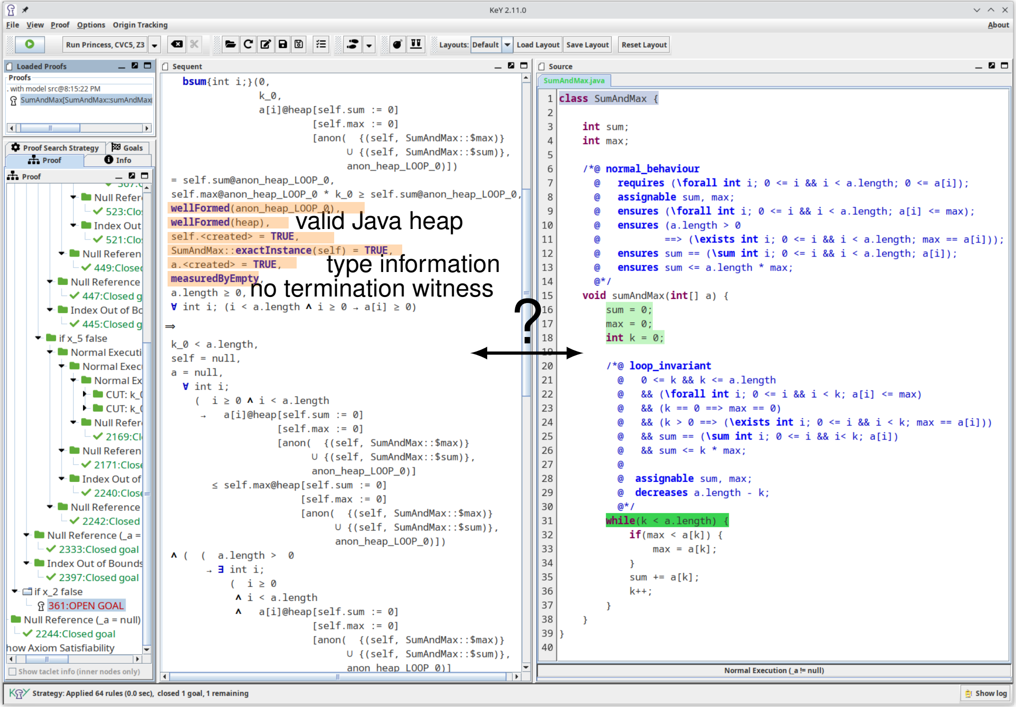
valid Java heap

type information

?

```
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a[i]);
      @   assignable sum, max;
      @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <= max);
      @   ensures (a.length > 0
      @            ==> (\exists int i; 0 <= i && i < a.length; max == a[i]));
      @   ensures sum == (\sum int i; 0 <= i && i < a.length; a[i]);
      @   ensures sum <= a.length * max;
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @    0 <= k && k <= a.length
          @    && (\forall int i; 0 <= i && i < k; a[i] <= max)
          @    && (k == 0 ==> max == 0)
          @    && (k > 0 ==> (\exists int i; 0 <= i && i < k; max == a[i]))
          @    && sum == (\sum int i; 0 <= i && i< k; a[i])
          @    && sum <= k * max;
          @
          @  assignable sum, max;
          @  decreases a.length - k;
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```

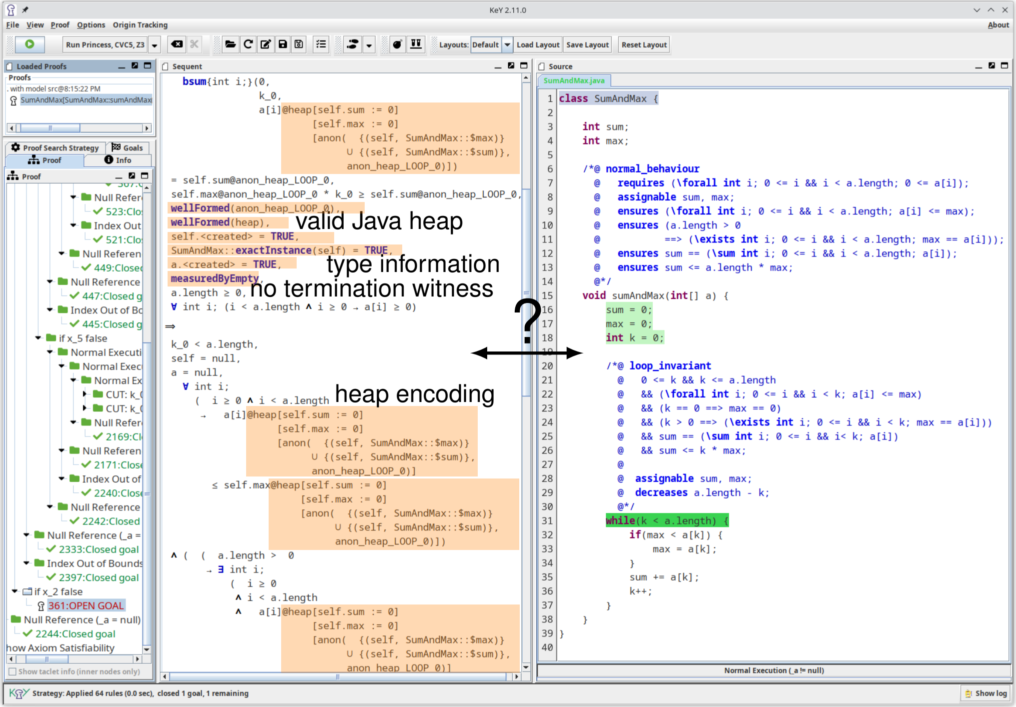
Strategy: Applied 64 rules (0.0 sec), closed 1 goal, 1 remaining



```
  bsum{int i;}(0,
               k_0,
               a[i]@heap[self.sum := 0]
                        [self.max := 0]
                        [anon(  {(self, SumAndMax::$max)}
                              ∪ {(self, SumAndMax::$sum)},
                              anon_heap_LOOP_0)])
= self.sum@anon_heap_LOOP_0,
self.max@anon_heap_LOOP_0 * k_0 ≥ self.sum@anon_heap_LOOP_0,
wellFormed(anon_heap_LOOP_0),
wellFormed(heap),
self.<created> = TRUE,
SumAndMax::exactInstance(self) = TRUE,
a.<created> = TRUE,
measuredByEmpty,
a.length ≥ 0,
∀ int i; (i < a.length ∧ i ≥ 0 → a[i] ≥ 0)

⇒

k_0 < a.length,
self = null,
a = null,
  ∀ int i;
    (  i ≥ 0 ∧ i < a.length
     →  a[i]@heap[self.sum := 0]
                 [self.max := 0]
                 [anon(  {(self, SumAndMax::$max)}
                       ∪ {(self, SumAndMax::$sum)},
                       anon_heap_LOOP_0)]
      ≤ self.max@heap[self.sum := 0]
                     [self.max := 0]
                     [anon(  {(self, SumAndMax::$max)}
                           ∪ {(self, SumAndMax::$sum)},
                           anon_heap_LOOP_0)])
∧ (  (  a.length >  0
      → ∃ int i;
          (  i ≥ 0
           ∧ i < a.length
           ∧   a[i]@heap[self.sum := 0]
                        [self.max := 0]
                        [anon(  {(self, SumAndMax::$max)}
                              ∪ {(self, SumAndMax::$sum)},
                              anon_heap_LOOP_0)]
```

```java
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a[i]);
      @   assignable sum, max;
      @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <= max);
      @   ensures (a.length > 0
      @           ==> (\exists int i; 0 <= i && i < a.length; max == a[i]));
      @   ensures sum == (\sum int i; 0 <= i && i < a.length; a[i]);
      @   ensures sum <= a.length * max;
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.length
          @   && (\forall int i; 0 <= i && i < k; a[i] <= max)
          @   && (k == 0 ==> max == 0)
          @   && (k > 0 ==> (\exists int i; 0 <= i && i < k; max == a[i]))
          @   && sum == (\sum int i; 0 <= i && i< k; a[i])
          @   && sum <= k * max;
          @
          @  assignable sum, max;
          @  decreases a.length - k;
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```

Strategy: Applied 64 rules (0.0 sec), closed 1 goal, 1 remaining



```
   bsum{int i;}(0,
                k_0,
                a[i]@heap[self.sum := 0]
                         [self.max := 0]
                         [anon(  {(self, SumAndMax::$max)}
                               ∪ {(self, SumAndMax::$sum)},
                               anon_heap_LOOP_0)])
 = self.sum@anon_heap_LOOP_0,
 self.max@anon_heap_LOOP_0 * k_0 ≥ self.sum@anon_heap_LOOP_0,
 wellFormed(anon_heap_LOOP_0),
 wellFormed(heap),
 self.<created> = TRUE,
 SumAndMax::exactInstance(self) = TRUE,
 a.<created> = TRUE,
 measuredByEmpty,
 a.length ≥ 0,
 ∀ int i; (i < a.length ∧ i ≥ 0 → a[i] ≥ 0)

⇒

 k_0 < a.length,
 self = null,
 a = null,
   ∀ int i;
     (  i ≥ 0 ∧ i < a.length
      →  a[i]@heap[self.sum := 0]
                  [self.max := 0]
                  [anon(  {(self, SumAndMax::$max)}
                        ∪ {(self, SumAndMax::$sum)},
                        anon_heap_LOOP_0)]
        ≤ self.max@heap[self.sum := 0]
                       [self.max := 0]
                       [anon(  {(self, SumAndMax::$max)}
                             ∪ {(self, SumAndMax::$sum)},
                             anon_heap_LOOP_0)])
 ∧ (  (  a.length >  0
       → ∃ int i;
           (  i ≥ 0
            ∧ i < a.length
            ∧    a[i]@heap[self.sum := 0]
                          [self.max := 0]
                          [anon(  {(self, SumAndMax::$max)}
                                ∪ {(self, SumAndMax::$sum)},
                                anon_heap_LOOP_0)]
```

valid Java heap

type information

no termination witness

?

heap encoding

```java
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a[i]);
      @   assignable sum, max;
      @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <= max);
      @   ensures (a.length > 0
      @           ==> (\exists int i; 0 <= i && i < a.length; max == a[i]));
      @   ensures sum == (\sum int i; 0 <= i && i < a.length; a[i]);
      @   ensures sum <= a.length * max;
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.length
          @   && (\forall int i; 0 <= i && i < k; a[i] <= max)
          @   && (k == 0 ==> max == 0)
          @   && (k > 0 ==> (\exists int i; 0 <= i && i < k; max == a[i]))
          @   && sum == (\sum int i; 0 <= i && i< k; a[i])
          @   && sum <= k * max;
          @
          @  assignable sum, max;
          @  decreases a.length - k;
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```

Strategy: Applied 64 rules (0.0 sec), closed 1 goal, 1 remaining

Vision

- Open goal should be presented in a view the user is familiar with.
- Interaction should be possible on the input artifact (Java/JML).
- When switching between different representations, KeY should present hints that help to connect them.
- Formulae should be presented in the way the user has written them.

Progress so far

“Integrating Source Code, Specification and Proof State into a Single Interactive View for the Deductive Verification Tool KeY” (Master’s Thesis, Mike Schwörer)

Idea: Represent a goal (sequent) of the proof as JML.

Progress so far

“Integrating Source Code, Specification and Proof State into a Single Interactive View for the Deductive Verification Tool KeY” (Master’s Thesis, Mike Schwörer)

Idea: Represent a goal (sequent) of the proof as JML.

1. Take initial PO and assign origins/categories to the terms
2. Transform correctly under rule applications
3. Render the new view:
   Input: Sequent with origin/category tags, Java/JML
   Output: Source code with additional JML assume/assert statements placed

Progress so far

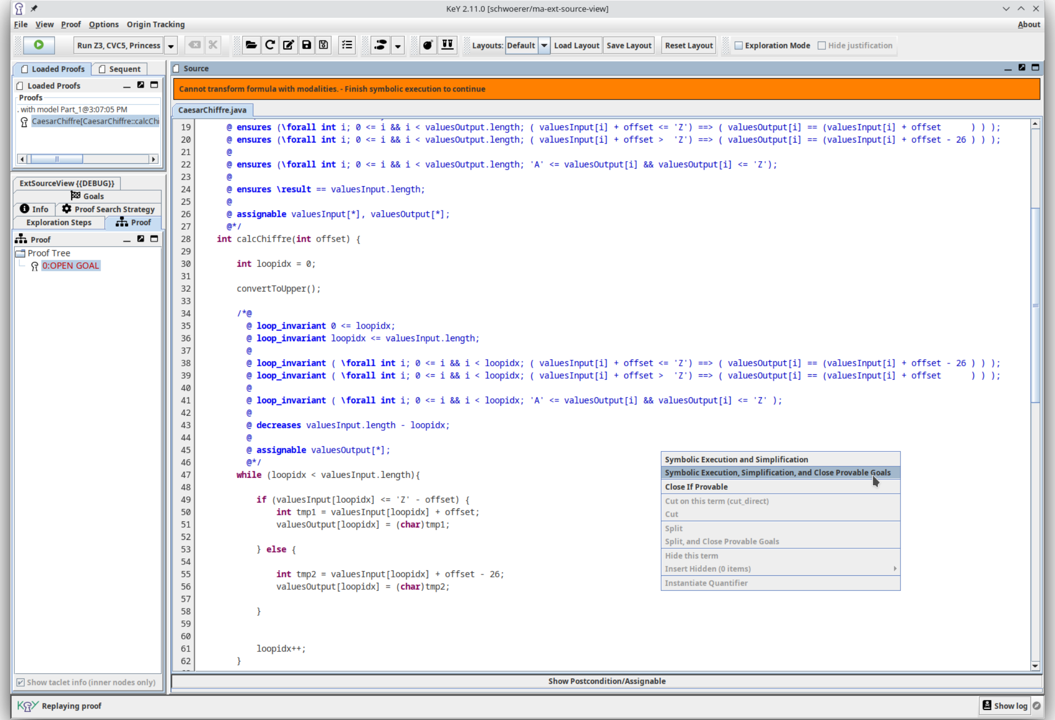
“Integrating Source Code, Specification and Proof State into a Single Interactive View for the Deductive Verification Tool KeY” (Master’s Thesis, Mike Schwörer)

Idea: Represent a goal (sequent) of the proof as JML.

1. Take initial PO and assign origins/categories to the terms
2. Transform correctly under rule applications
3. Render the new view:
   Input: Sequent with origin/category tags, Java/JML
   Output: Source code with additional JML assume/assert statements placed

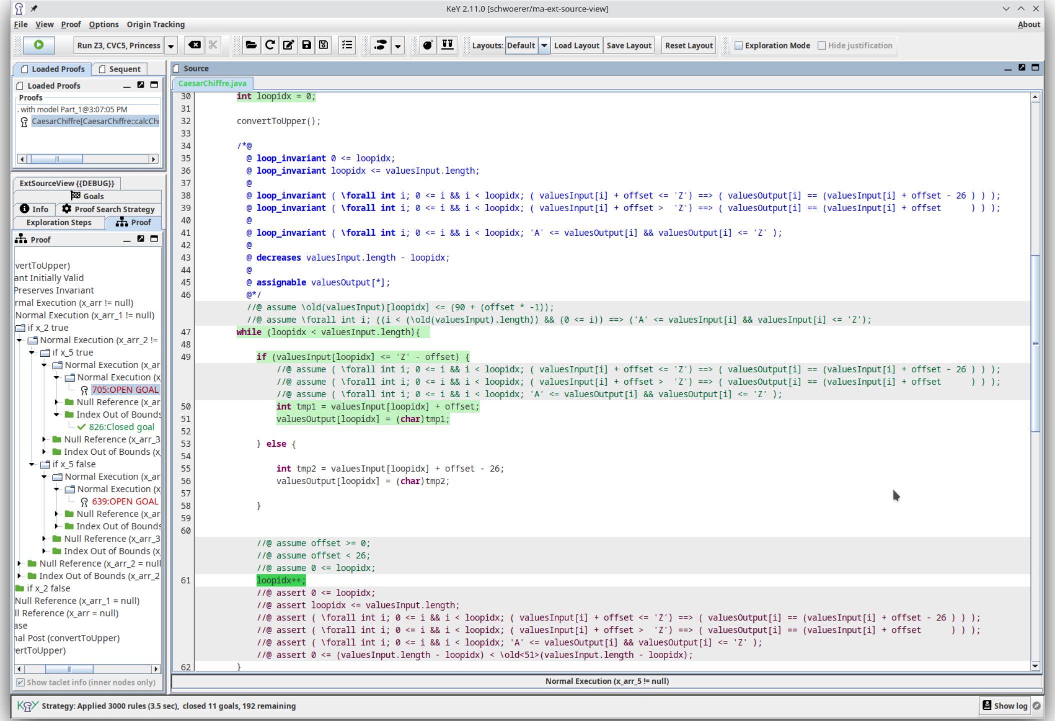
Assumptions

- Symbolic execution has finished (no modalities).
- All updates are applied.
- Restrictions to allowed programs (e.g., no `for` loops, only `return` + variable, ...).



Cannot transform formula with modalities. - Finish symbolic execution to continue

```
     @ ensures (\forall int i; 0 <= i && i < valuesOutput.length; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
     @ ensures (\forall int i; 0 <= i && i < valuesOutput.length; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
     @
     @ ensures (\forall int i; 0 <= i && i < valuesOutput.length; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z');
     @
     @ ensures \result == valuesInput.length;
     @
     @ assignable valuesInput[*], valuesOutput[*];
     @*/
    int calcChiffre(int offset) {

      int loopidx = 0;

      convertToUpper();

      /*@
       @ loop_invariant 0 <= loopidx;
       @ loop_invariant loopidx <= valuesInput.length;
       @
       @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
       @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
       @
       @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
       @
       @ decreases valuesInput.length - loopidx;
       @
       @ assignable valuesOutput[*];
       @*/
      while (loopidx < valuesInput.length){

        if (valuesInput[loopidx] <= 'Z' - offset) {
          int tmp1 = valuesInput[loopidx] + offset;
          valuesOutput[loopidx] = (char)tmp1;

        } else {

          int tmp2 = valuesInput[loopidx] + offset - 26;
          valuesOutput[loopidx] = (char)tmp2;

        }


        loopidx++;
      }
```


KeY 2.11.0 [schwoerer/ma-ext-source-view]
File View Proof Options Origin Tracking About
Run Z3, CVC5, Princess
Layouts: Default Load Layout Save Layout Reset Layout
☐ Exploration Mode ☐ Hide justification
Loaded Proofs Sequent
Source
CaesarChiffre.java
ExtSourceView {{DEBUG}}
Goals
Info
Proof Search Strategy
Exploration Steps
Proof
☑ Show taclet info (inner nodes only)
Show log

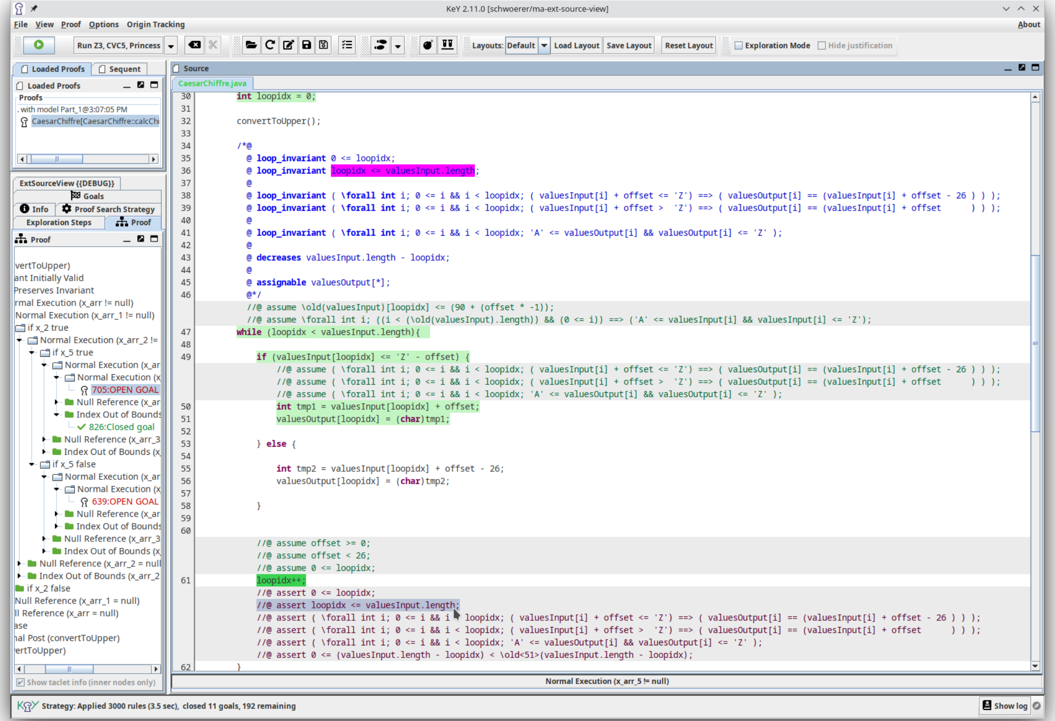
```
Loaded Proofs
Proofs
. with model Part_1@3:07:05 PM
CaesarChiffre[CaesarChiffre::calcChi
```

```
    int loopidx = 0;

    convertToUpper();

    /*@
     @ loop_invariant 0 <= loopidx;
     @ loop_invariant loopidx <= valuesInput.length;
     @
     @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
     @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
     @
     @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
     @
     @ decreases valuesInput.length - loopidx;
     @
     @ assignable valuesOutput[*];
     @*/
     //@ assume \old(valuesInput)[loopidx] <= (90 + (offset * -1));
     //@ assume \forall int i; ((i < (\old(valuesInput).length)) && (0 <= i)) ==> ('A' <= valuesInput[i] && valuesInput[i] <= 'Z');
    while (loopidx < valuesInput.length){

      if (valuesInput[loopidx] <= 'Z' - offset) {
        //@ assume ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
        //@ assume ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
        //@ assume ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
        int tmp1 = valuesInput[loopidx] + offset;
        valuesOutput[loopidx] = (char)tmp1;

      } else {

        int tmp2 = valuesInput[loopidx] + offset - 26;
        valuesOutput[loopidx] = (char)tmp2;

      }


      //@ assume offset >= 0;
      //@ assume offset < 26;
      //@ assume 0 <= loopidx;
      loopidx++;
      //@ assert 0 <= loopidx;
      //@ assert loopidx <= valuesInput.length;
      //@ assert ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
      //@ assert ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
      //@ assert ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
      //@ assert 0 <= (valuesInput.length - loopidx) < \old<51>(valuesInput.length - loopidx);
    }
```

```
Proof
vertToUpper)
ant Initially Valid
Preserves Invariant
rmal Execution (x_arr != null)
Normal Execution (x_arr_1 != null)
if x_2 true
  Normal Execution (x_arr_2 !=
    if x_5 true
      Normal Execution (x_ar
        Normal Execution (x
          705:OPEN GOAL
        Null Reference (x_ar
        Index Out of Bounds
          826:Closed goal
      Null Reference (x_arr_3
      Index Out of Bounds (x
    if x_5 false
      Normal Execution (x_ar
        Normal Execution (x
          639:OPEN GOAL
        Null Reference (x_ar
        Index Out of Bounds
      Null Reference (x_arr_3
      Index Out of Bounds (x
  Null Reference (x_arr_2 = null
  Index Out of Bounds (x_arr_2
if x_2 false
Null Reference (x_arr_1 = null)
ll Reference (x_arr = null)
ase
al Post (convertToUpper)
ertToUpper)
```

Normal Execution (x_arr_5 != null)

Strategy: Applied 3000 rules (3.5 sec), closed 11 goals, 192 remaining



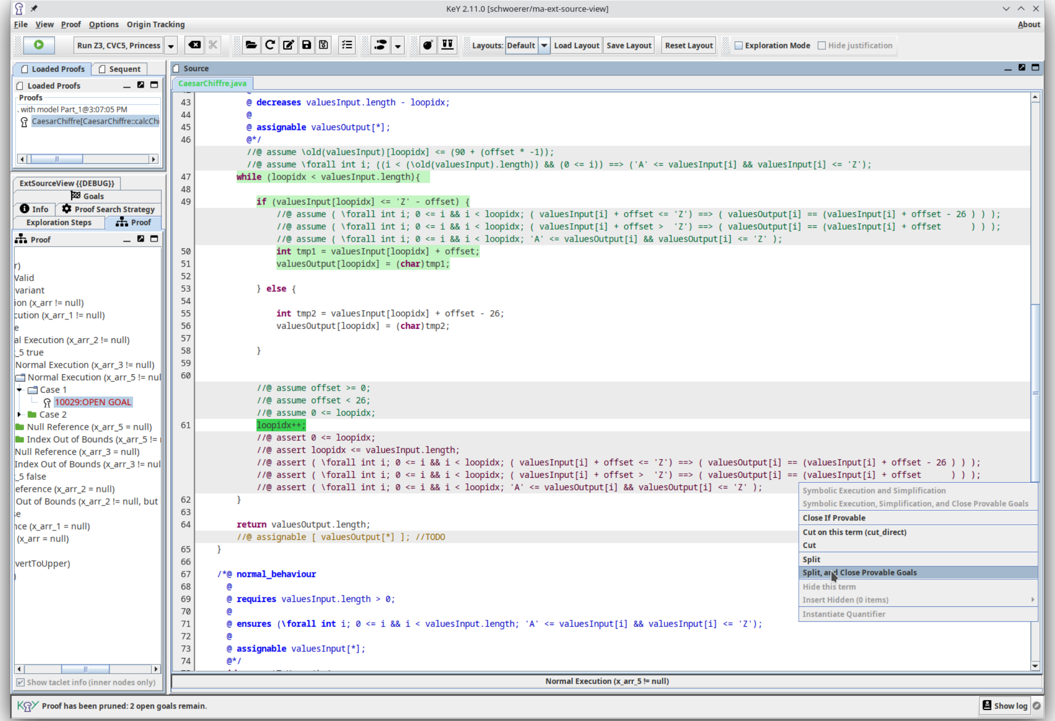
KeY 2.11.0 [schwoerer/ma-ext-source-view]
File View Proof Options Origin Tracking About
Run Z3, CVC5, Princess
Layouts: Default Load Layout Save Layout Reset Layout
☐ Exploration Mode ☐ Hide justification
Loaded Proofs Sequent
Loaded Proofs
Proofs
. with model Part_1@3:07:05 PM
CaesarChiffre[CaesarChiffre::calcChi
ExtSourceView {{DEBUG}}
Goals
Info Proof Search Strategy
Exploration Steps Proof
Proof
☑ Show taclet info (inner nodes only)
Source
CaesarChiffre.java
Show log

```
vertToUpper)
ant Initially Valid
Preserves Invariant
rmal Execution (x_arr != null)
Normal Execution (x_arr_1 != null)
if x_2 true
  Normal Execution (x_arr_2 !=
    if x_5 true
      Normal Execution (x_ar
        Normal Execution (x
          705:OPEN GOAL
        Null Reference (x_ar
        Index Out of Bounds
          ✓ 826:Closed goal
      Null Reference (x_arr_3
      Index Out of Bounds (x
    if x_5 false
      Normal Execution (x_ar
        Normal Execution (x
          639:OPEN GOAL
        Null Reference (x_ar
        Index Out of Bounds
      Null Reference (x_arr_3
      Index Out of Bounds (x
  Null Reference (x_arr_2 = null
  Index Out of Bounds (x_arr_2
if x_2 false
Null Reference (x_arr_1 = null)
ll Reference (x_arr = null)
ase
al Post (convertToUpper)
ertToUpper)
```

```
        int loopidx = 0;

        convertToUpper();

        /*@
          @ loop_invariant 0 <= loopidx;
          @ loop_invariant loopidx <= valuesInput.length;
          @
          @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
          @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
          @
          @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
          @
          @ decreases valuesInput.length - loopidx;
          @
          @ assignable valuesOutput[*];
          @*/
          //@ assume \old(valuesInput)[loopidx] <= (90 + (offset * -1));
          //@ assume \forall int i; ((i < (\old(valuesInput).length)) && (0 <= i)) ==> ('A' <= valuesInput[i] && valuesInput[i] <= 'Z');
        while (loopidx < valuesInput.length){

            if (valuesInput[loopidx] <= 'Z' - offset) {
                //@ assume ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
                //@ assume ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
                //@ assume ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
                int tmp1 = valuesInput[loopidx] + offset;
                valuesOutput[loopidx] = (char)tmp1;

            } else {

                int tmp2 = valuesInput[loopidx] + offset - 26;
                valuesOutput[loopidx] = (char)tmp2;

            }


            //@ assume offset >= 0;
            //@ assume offset < 26;
            //@ assume 0 <= loopidx;
            loopidx++;
            //@ assert 0 <= loopidx;
            //@ assert loopidx <= valuesInput.length;
            //@ assert ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
            //@ assert ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
            //@ assert ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
            //@ assert 0 <= (valuesInput.length - loopidx) < \old<51>(valuesInput.length - loopidx);
        }
```

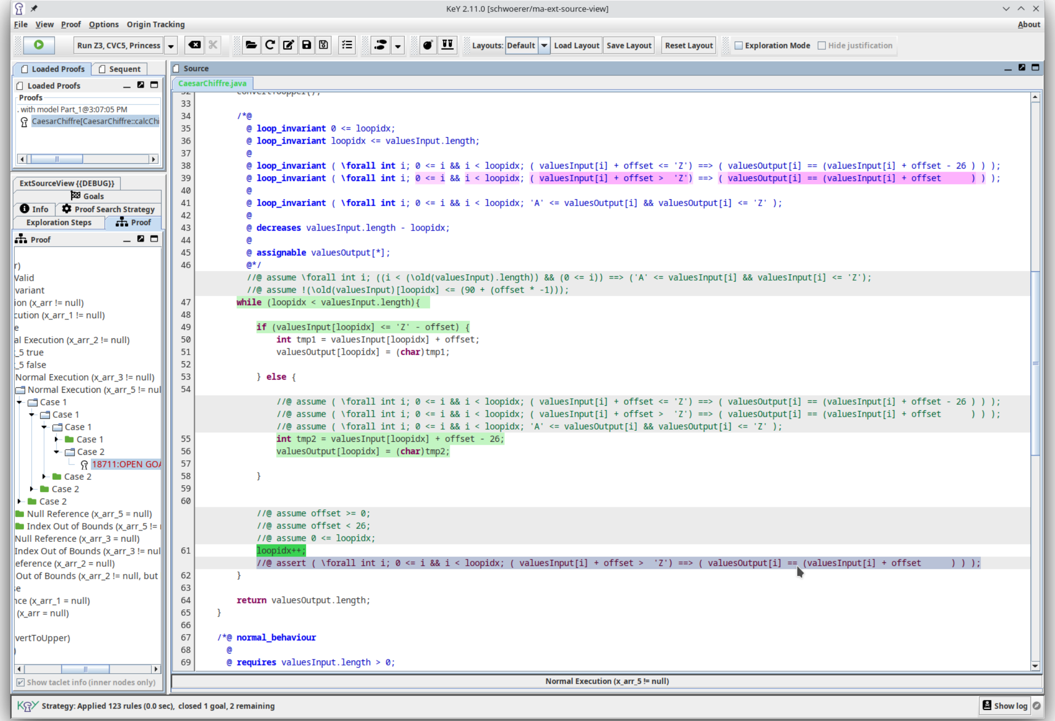
Normal Execution (x_arr_5 != null)

Strategy: Applied 3000 rules (3.5 sec), closed 11 goals, 192 remaining


KeY 2.11.0 [schwoerer/ma-ext-source-view]
File
View
Proof
Options
Origin Tracking
About
Run Z3, CVC5, Princess
Layouts: Default
Load Layout
Save Layout
Reset Layout
□ Exploration Mode
□ Hide justification
Loaded Proofs
Sequent
Proofs
. with model Part_1@3:07:05 PM
CaesarChiffre[CaesarChiffre::calcChi
ExtSourceView {{DEBUG}}
Goals
Info
Proof Search Strategy
Exploration Steps
Proof
r)
Valid
variant
ion (x_arr != null)
cution (x_arr_1 != null)
e
al Execution (x_arr_2 != null)
_5 true
Normal Execution (x_arr_3 != null)
Normal Execution (x_arr_5 != nul
Case 1
10029:OPEN GOAL
Case 2
Null Reference (x_arr_5 = null)
Index Out of Bounds (x_arr_5 != i
Null Reference (x_arr_3 = null)
Index Out of Bounds (x_arr_3 != nul
_5 false
eference (x_arr_2 = null)
Out of Bounds (x_arr_2 != null, but
e
nce (x_arr_1 = null)
(x_arr = null)
vertToUpper)
)
☑ Show taclet info (inner nodes only)
Source
CaesarChiffre.java
Symbolic Execution and Simplification
Symbolic Execution, Simplification, and Close Provable Goals
Close If Provable
Cut on this term (cut_direct)
Cut
Split
Split, and Close Provable Goals
Hide this term
Insert Hidden (0 items)
Instantiate Quantifier
Normal Execution (x_arr_5 != null)
Show log

```
        @ decreases valuesInput.length - loopidx;
        @
        @ assignable valuesOutput[*];
        @*/
        //@ assume \old(valuesInput)[loopidx] <= (90 + (offset * -1));
        //@ assume \forall int i; ((i < (\old(valuesInput).length)) && (0 <= i)) ==> ('A' <= valuesInput[i] && valuesInput[i] <= 'Z');
      while (loopidx < valuesInput.length){

        if (valuesInput[loopidx] <= 'Z' - offset) {
          //@ assume ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
          //@ assume ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
          //@ assume ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
          int tmp1 = valuesInput[loopidx] + offset;
          valuesOutput[loopidx] = (char)tmp1;

        } else {

          int tmp2 = valuesInput[loopidx] + offset - 26;
          valuesOutput[loopidx] = (char)tmp2;

        }


        //@ assume offset >= 0;
        //@ assume offset < 26;
        //@ assume 0 <= loopidx;
        loopidx++;
        //@ assert 0 <= loopidx;
        //@ assert loopidx <= valuesInput.length;
        //@ assert ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
        //@ assert ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
        //@ assert ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
      }

      return valuesOutput.length;
      //@ assignable [ valuesOutput[*] ]; //TODO
    }

    /*@ normal_behaviour
      @
      @ requires valuesInput.length > 0;
      @
      @ ensures (\forall int i; 0 <= i && i < valuesInput.length; 'A' <= valuesInput[i] && valuesInput[i] <= 'Z');
      @
      @ assignable valuesInput[*];
      @*/
```

Proof has been pruned: 2 open goals remain.


KeY 2.11.0 [schwoerer/ma-ext-source-view]
File View Proof Options Origin Tracking About
Run Z3, CVC5, Princess
Layouts: Default Load Layout Save Layout Reset Layout
Exploration Mode
Hide justification
Loaded Proofs
Sequent
Source
ExtSourceView {{DEBUG}}
Goals
Info
Proof Search Strategy
Exploration Steps
Proof
Show taclet info (inner nodes only)
Show log

Loaded Proofs

Proofs
. with model Part_1@3:07:05 PM
CaesarChiffre[CaesarChiffre::calcChi

Proof

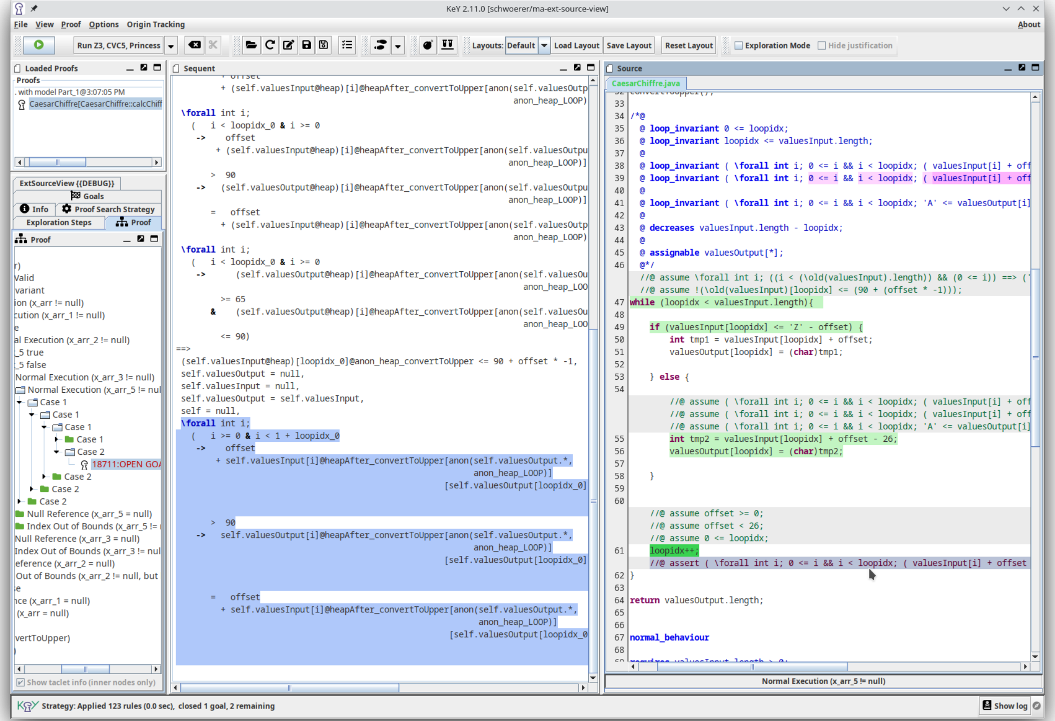
```
r)
Valid
variant
ion (x_arr != null)
cution (x_arr_1 != null)
e
al Execution (x_arr_2 != null)
_5 true
_5 false
Normal Execution (x_arr_3 != null)
Normal Execution (x_arr_5 != nul
  Case 1
    Case 1
      Case 1
        Case 1
        Case 2
          18711:OPEN GOA
      Case 2
    Case 2
  Case 2
Null Reference (x_arr_5 = null)
Index Out of Bounds (x_arr_5 != 
Null Reference (x_arr_3 = null)
Index Out of Bounds (x_arr_3 != nul
eference (x_arr_2 = null)
Out of Bounds (x_arr_2 != null, but
e
nce (x_arr_1 = null)
(x_arr = null)

vertToUpper()
)
```

CaesarChiffre.java

```
        convertToUpper();

        /*@
          @ loop_invariant 0 <= loopidx;
          @ loop_invariant loopidx <= valuesInput.length;
          @
          @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
          @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
          @
          @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
          @
          @ decreases valuesInput.length - loopidx;
          @
          @ assignable valuesOutput[*];
          @*/
          //@ assume \forall int i; ((i < (\old(valuesInput).length)) && (0 <= i)) ==> ('A' <= valuesInput[i] && valuesInput[i] <= 'Z');
          //@ assume !(\old(valuesInput)[loopidx] <= (90 + (offset * -1)));
        while (loopidx < valuesInput.length){

            if (valuesInput[loopidx] <= 'Z' - offset) {
                int tmp1 = valuesInput[loopidx] + offset;
                valuesOutput[loopidx] = (char)tmp1;

            } else {

                //@ assume ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset <= 'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset - 26 ) ) );
                //@ assume ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
                //@ assume ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i] && valuesOutput[i] <= 'Z' );
                int tmp2 = valuesInput[loopidx] + offset - 26;
                valuesOutput[loopidx] = (char)tmp2;

            }


            //@ assume offset >= 0;
            //@ assume offset < 26;
            //@ assume 0 <= loopidx;
            loopidx++;
            //@ assert ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset >  'Z') ==> ( valuesOutput[i] == (valuesInput[i] + offset      ) ) );
        }

        return valuesOutput.length;
    }

    /*@ normal_behaviour
      @
      @ requires valuesInput.length > 0;
```

Normal Execution (x_arr_5 != null)

Strategy: Applied 123 rules (0.0 sec), closed 1 goal, 2 remaining



```
. with model Part_1@3:07:05 PM
CaesarChiffre[CaesarChiffre::calcChiff
```

```
r)
Valid
variant
ion (x_arr != null)
cution (x_arr_1 != null)
e
al Execution (x_arr_2 != null)
_5 true
_5 false
Normal Execution (x_arr_3 != null)
Normal Execution (x_arr_5 != nul
  Case 1
    Case 1
      Case 1
        Case 1
        Case 2
          18711:OPEN GOA
      Case 2
    Case 2
  Case 2
Null Reference (x_arr_5 = null)
Index Out of Bounds (x_arr_5 != 
Null Reference (x_arr_3 = null)
Index Out of Bounds (x_arr_3 != nul
eference (x_arr_2 = null)
Out of Bounds (x_arr_2 != null, but
e
nce (x_arr_1 = null)
(x_arr = null)

vertToUpper)
)
```

```
       + (self.valuesInput@heap)[i]@heapAfter_convertToUpper[anon(self.valuesOutp
                                                                  anon_heap_LOOP)
 \forall int i;
   (   i < loopidx_0 & i >= 0
    ->   offset
       + (self.valuesInput@heap)[i]@heapAfter_convertToUpper[anon(self.valuesOutpu
                                                                 anon_heap_LOOP)]
       >  90
    ->  (self.valuesOutput@heap)[i]@heapAfter_convertToUpper[anon(self.valuesOutpu
                                                                 anon_heap_LOOP)]
       =   offset
       + (self.valuesInput@heap)[i]@heapAfter_convertToUpper[anon(self.valuesOutp
                                                                  anon_heap_LOOP)
 \forall int i;
   (   i < loopidx_0 & i >= 0
    ->     (self.valuesOutput@heap)[i]@heapAfter_convertToUpper[anon(self.valuesOu
                                                                    anon_heap_LOO
       >= 65
     &     (self.valuesOutput@heap)[i]@heapAfter_convertToUpper[anon(self.valuesOu
                                                                    anon_heap_LOO
       <= 90)
==>
 (self.valuesInput@heap)[loopidx_0]@anon_heap_convertToUpper <= 90 + offset * -1,
 self.valuesOutput = null,
 self.valuesInput = null,
 self.valuesOutput = self.valuesInput,
 self = null,
 \forall int i;
   (   i >= 0 & i < 1 + loopidx_0
    ->   offset
       + self.valuesInput[i]@heapAfter_convertToUpper[anon(self.valuesOutput.*,
                                                          anon_heap_LOOP)]
                                                      [self.valuesOutput[loopidx_0]


       >  90
    ->  self.valuesOutput[i]@heapAfter_convertToUpper[anon(self.valuesOutput.*,
                                                          anon_heap_LOOP)]
                                                      [self.valuesOutput[loopidx_0]


       =   offset
        + self.valuesInput[i]@heapAfter_convertToUpper[anon(self.valuesOutput.*,
                                                           anon_heap_LOOP)]
                                                       [self.valuesOutput[loopidx_0
```

```
/*@
  @ loop_invariant 0 <= loopidx;
  @ loop_invariant loopidx <= valuesInput.length;
  @
  @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + off
  @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + off
  @
  @ loop_invariant ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i]
  @
  @ decreases valuesInput.length - loopidx;
  @
  @ assignable valuesOutput[*];
  @*/
  //@ assume \forall int i; ((i < (\old(valuesInput).length)) && (0 <= i)) ==> ('
  //@ assume !(\old(valuesInput)[loopidx] <= (90 + (offset * -1)));
while (loopidx < valuesInput.length){

    if (valuesInput[loopidx] <= 'Z' - offset) {
        int tmp1 = valuesInput[loopidx] + offset;
        valuesOutput[loopidx] = (char)tmp1;

    } else {

        //@ assume ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + off
        //@ assume ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + off
        //@ assume ( \forall int i; 0 <= i && i < loopidx; 'A' <= valuesOutput[i]
        int tmp2 = valuesInput[loopidx] + offset - 26;
        valuesOutput[loopidx] = (char)tmp2;

    }


    //@ assume offset >= 0;
    //@ assume offset < 26;
    //@ assume 0 <= loopidx;
    loopidx++;
    //@ assert ( \forall int i; 0 <= i && i < loopidx; ( valuesInput[i] + offset
}

return valuesOutput.length;


normal_behaviour

```

Normal Execution (x_arr_5 != null)

Strategy: Applied 123 rules (0.0 sec), closed 1 goal, 2 remaining

Evaluation

- Evaluated with 6 KeY experts
- The work shows:
  - Identifying bugs can be faster, even for KeY experts.
  - Origin/category tracking of formulas is really important.
  - For making more interactions available, we need a better parser.

Ongoing work

“Embedding Proof Scripts into Java/JML Source Code” (Master’s thesis)
Idea: Write the interactions in form of a script into the source code.

Ongoing work

“Embedding Proof Scripts into Java/JML Source Code” (Master’s thesis)
Idea: Write the interactions in form of a script into the source code.

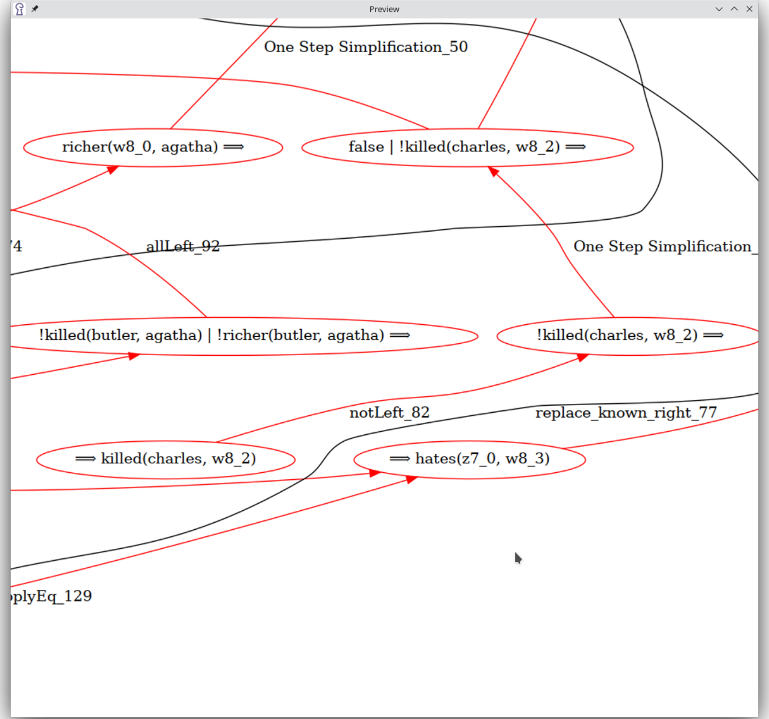
```
//@ requires req1: (\exists int x; (\forall int y; p(x,y)));
//@ ensures ens1: (\forall int v; (\exists int u; p(u,v)));
void m(int param) {
    if (param > 7) {
        //@ pragma [StrategyProperty]QUERYAXIOM_OPTIONS_KEY=QUERYAXIOM_ON;
        /*@ assert phi \by {
          @   var sk1 = req1.skolemize();
          @   var cutTerm = (\forall int z; pred(z));
          @   assert cutTerm \by {
          @     ...
          @   };
          @ ens1.instantiate(v=sk1.x);
          @ ...
          @ };                                                      @*/
    }
}
```

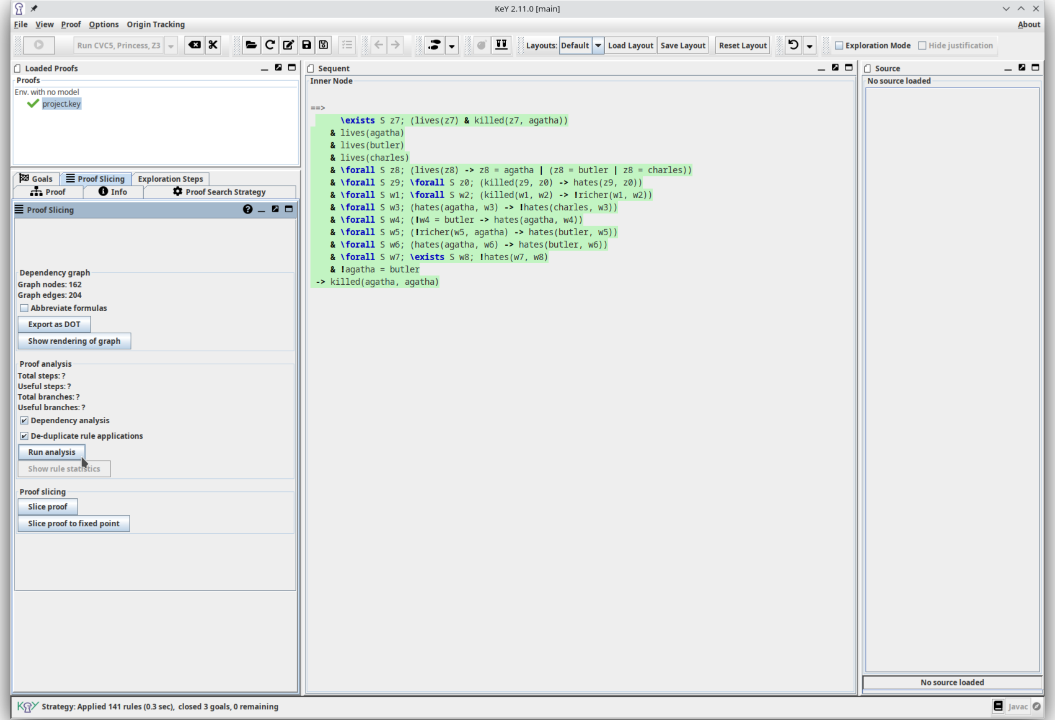
Further Usability Improvements

- Proof Slicing
- Navigation History
- Undoing Interactions
- Automatically run JavaC first (Alexander Weigl)
- Background SMT (ongoing)
- Proof Caching (ongoing)

Further Usability Improvements

- Proof Slicing ◀
- Navigation History
- Undoing Interactions
- Automatically run JavaC first (Alexander Weigl)
- Background SMT (ongoing)
- Proof Caching (ongoing)


Preview
One Step Simplification_50
richer(w8_0, agatha) ⟹
false | !killed(charles, w8_2) ⟹
'4
allLeft_92
One Step Simplification_
!killed(butler, agatha) | !richer(butler, agatha) ⟹
!killed(charles, w8_2) ⟹
notLeft_82
replace_known_right_77
⟹ killed(charles, w8_2)
⟹ hates(z7_0, w8_3)
plyEq_129



Env. with no model

✓ project.key

Inner Node

```
==>
    \exists S z7; (lives(z7) & killed(z7, agatha))
  & lives(agatha)
  & lives(butler)
  & lives(charles)
  & \forall S z8; (lives(z8) -> z8 = agatha | (z8 = butler | z8 = charles))
  & \forall S z9; \forall S z0; (killed(z9, z0) -> hates(z9, z0))
  & \forall S w1; \forall S w2; (killed(w1, w2) -> !richer(w1, w2))
  & \forall S w3; (hates(agatha, w3) -> !hates(charles, w3))
  & \forall S w4; (!w4 = butler -> hates(agatha, w4))
  & \forall S w5; (!richer(w5, agatha) -> hates(butler, w5))
  & \forall S w6; (hates(agatha, w6) -> hates(butler, w6))
  & \forall S w7; \exists S w8; !hates(w7, w8)
  & !agatha = butler
-> killed(agatha, agatha)
```

No source loaded

Dependency graph

Graph nodes: 162

Graph edges: 204

□ Abbreviate formulas

Proof analysis

Total steps: ?

Useful steps: ?

Total branches: ?

Useful branches: ?

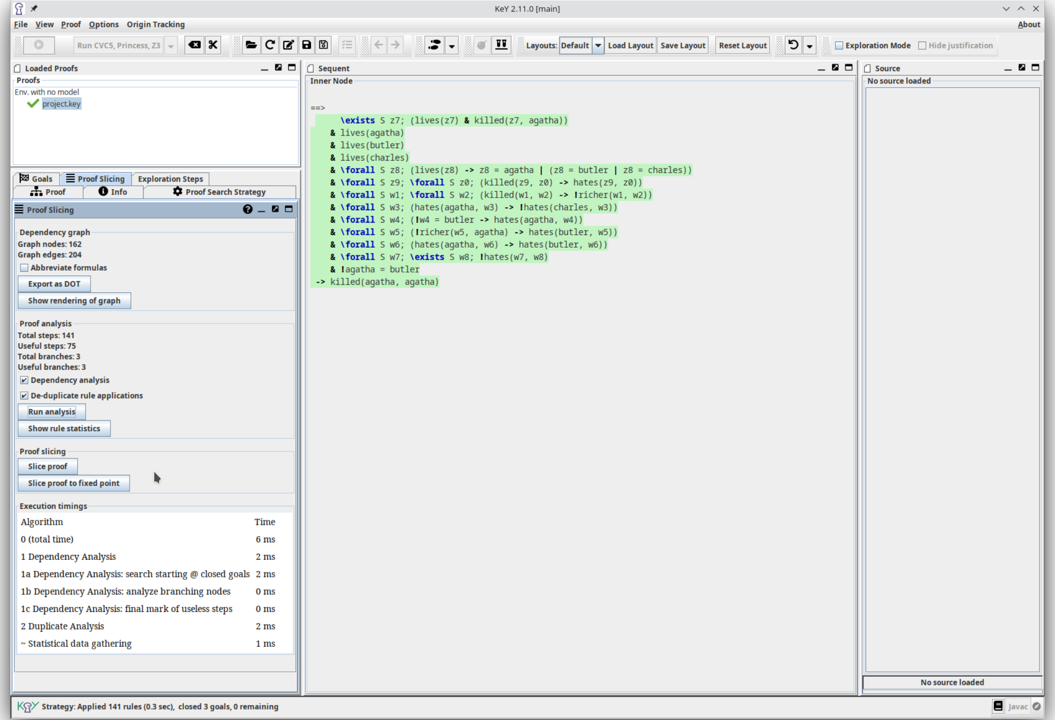
☑ Dependency analysis

☑ De-duplicate rule applications

Proof slicing

No source loaded

Strategy: Applied 141 rules (0.3 sec), closed 3 goals, 0 remaining



Dependency graph

Graph nodes: 162

Graph edges: 204

□ Abbreviate formulas

Proof analysis

Total steps: 141

Useful steps: 75

Total branches: 3

Useful branches: 3

☑ Dependency analysis

☑ De-duplicate rule applications

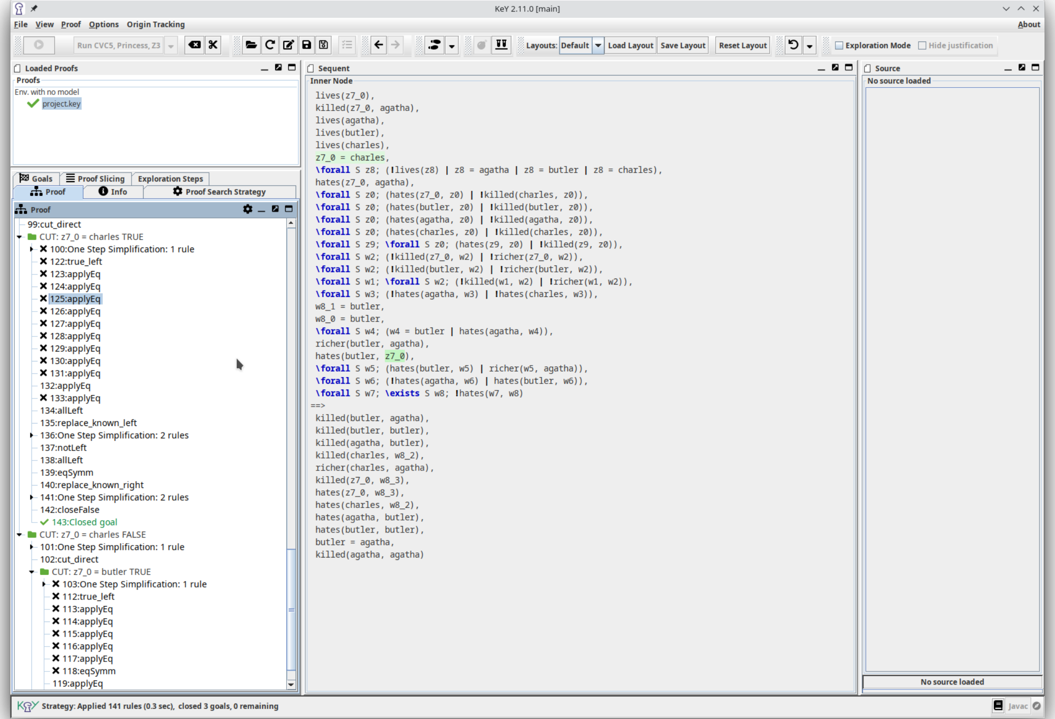
Proof slicing

Execution timings

Algorithm	Time
0 (total time)	6 ms
1 Dependency Analysis	2 ms
1a Dependency Analysis: search starting @ closed goals	2 ms
1b Dependency Analysis: analyze branching nodes	0 ms
1c Dependency Analysis: final mark of useless steps	0 ms
2 Duplicate Analysis	2 ms
~ Statistical data gathering	1 ms

```
==>
    \exists S z7; (lives(z7) & killed(z7, agatha))
  & lives(agatha)
  & lives(butler)
  & lives(charles)
  & \forall S z8; (lives(z8) -> z8 = agatha | (z8 = butler | z8 = charles))
  & \forall S z9; \forall S z0; (killed(z9, z0) -> hates(z9, z0))
  & \forall S w1; \forall S w2; (killed(w1, w2) -> !richer(w1, w2))
  & \forall S w3; (hates(agatha, w3) -> !hates(charles, w3))
  & \forall S w4; (!w4 = butler -> hates(agatha, w4))
  & \forall S w5; (!richer(w5, agatha) -> hates(butler, w5))
  & \forall S w6; (hates(agatha, w6) -> hates(butler, w6))
  & \forall S w7; \exists S w8; !hates(w7, w8)
  & !agatha = butler
 -> killed(agatha, agatha)
```

Strategy: Applied 141 rules (0.3 sec), closed 3 goals, 0 remaining

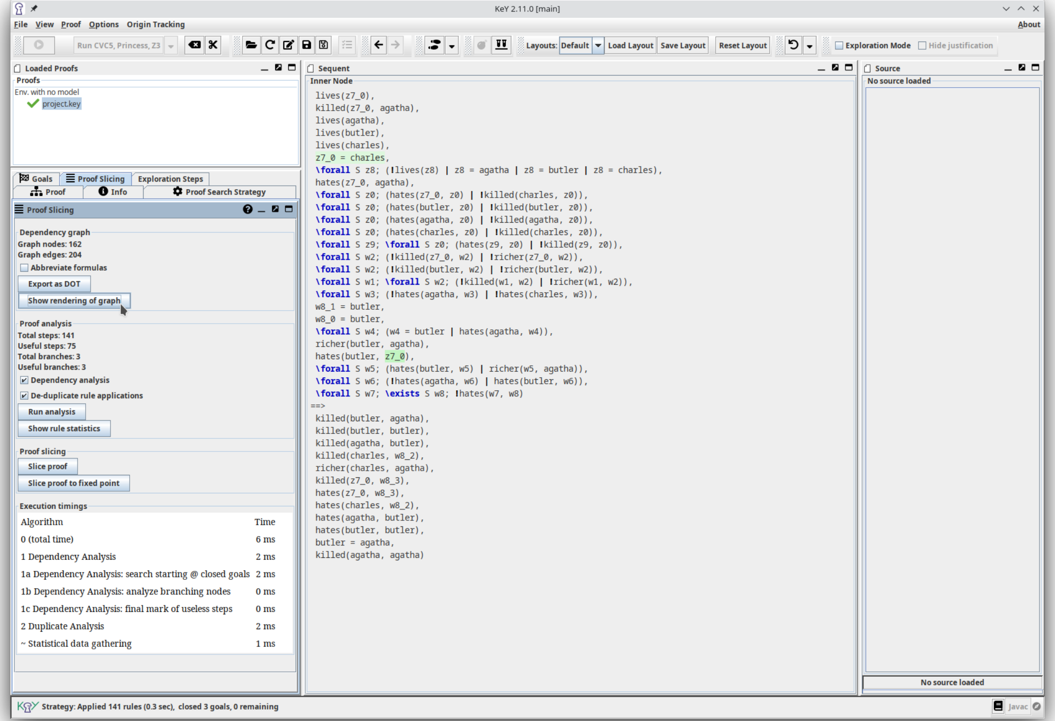


KeY 2.11.0 [main]
File View Proof Options Origin Tracking About
Run CVC5, Princess, Z3
Layouts: Default Load Layout Save Layout Reset Layout
☐ Exploration Mode ☐ Hide justification
Loaded Proofs
Proofs
Env. with no model
✓ project.key
Goals Proof Slicing Exploration Steps
Proof Info Proof Search Strategy
Proof
Sequent
Inner Node
Source
No source loaded
No source loaded
Javac

```
99:cut_direct
CUT: z7_0 = charles TRUE
  ✗ 100:One Step Simplification: 1 rule
  ✗ 122:true_left
  ✗ 123:applyEq
  ✗ 124:applyEq
  ✗ 125:applyEq
  ✗ 126:applyEq
  ✗ 127:applyEq
  ✗ 128:applyEq
  ✗ 129:applyEq
  ✗ 130:applyEq
  ✗ 131:applyEq
  132:applyEq
  ✗ 133:applyEq
  134:allLeft
  135:replace_known_left
  136:One Step Simplification: 2 rules
  137:notLeft
  138:allLeft
  139:eqSymm
  140:replace_known_right
  141:One Step Simplification: 2 rules
  142:closeFalse
  ✓ 143:Closed goal
CUT: z7_0 = charles FALSE
  101:One Step Simplification: 1 rule
  102:cut_direct
  CUT: z7_0 = butler TRUE
    ✗ 103:One Step Simplification: 1 rule
    ✗ 112:true_left
    ✗ 113:applyEq
    ✗ 114:applyEq
    ✗ 115:applyEq
    ✗ 116:applyEq
    ✗ 117:applyEq
    ✗ 118:eqSymm
    119:applyEq
```

```
 lives(z7_0),
 killed(z7_0, agatha),
 lives(agatha),
 lives(butler),
 lives(charles),
 z7_0 = charles,
 \forall S z8; (!lives(z8) | z8 = agatha | z8 = butler | z8 = charles),
 hates(z7_0, agatha),
 \forall S z0; (hates(z7_0, z0) | !killed(charles, z0)),
 \forall S z0; (hates(butler, z0) | !killed(butler, z0)),
 \forall S z0; (hates(agatha, z0) | !killed(agatha, z0)),
 \forall S z0; (hates(charles, z0) | !killed(charles, z0)),
 \forall S z9; \forall S z0; (hates(z9, z0) | !killed(z9, z0)),
 \forall S w2; (!killed(z7_0, w2) | !richer(z7_0, w2)),
 \forall S w2; (!killed(butler, w2) | !richer(butler, w2)),
 \forall S w1; \forall S w2; (!killed(w1, w2) | !richer(w1, w2)),
 \forall S w3; (!hates(agatha, w3) | !hates(charles, w3)),
 w8_1 = butler,
 w8_0 = butler,
 \forall S w4; (w4 = butler | hates(agatha, w4)),
 richer(butler, agatha),
 hates(butler, z7_0),
 \forall S w5; (hates(butler, w5) | richer(w5, agatha)),
 \forall S w6; (!hates(agatha, w6) | hates(butler, w6)),
 \forall S w7; \exists S w8; !hates(w7, w8)
==>
 killed(butler, agatha),
 killed(butler, butler),
 killed(agatha, butler),
 killed(charles, w8_2),
 richer(charles, agatha),
 killed(z7_0, w8_3),
 hates(z7_0, w8_3),
 hates(charles, w8_2),
 hates(agatha, butler),
 hates(butler, butler),
 butler = agatha,
 killed(agatha, agatha)
```

Strategy: Applied 141 rules (0.3 sec), closed 3 goals, 0 remaining



Dependency graph

Graph nodes: 162

Graph edges: 204

□ Abbreviate formulas

Export as DOT

Show rendering of graph

Proof analysis

Total steps: 141

Useful steps: 75

Total branches: 3

Useful branches: 3

☑ Dependency analysis

☑ De-duplicate rule applications

Run analysis

Show rule statistics

Proof slicing

Slice proof

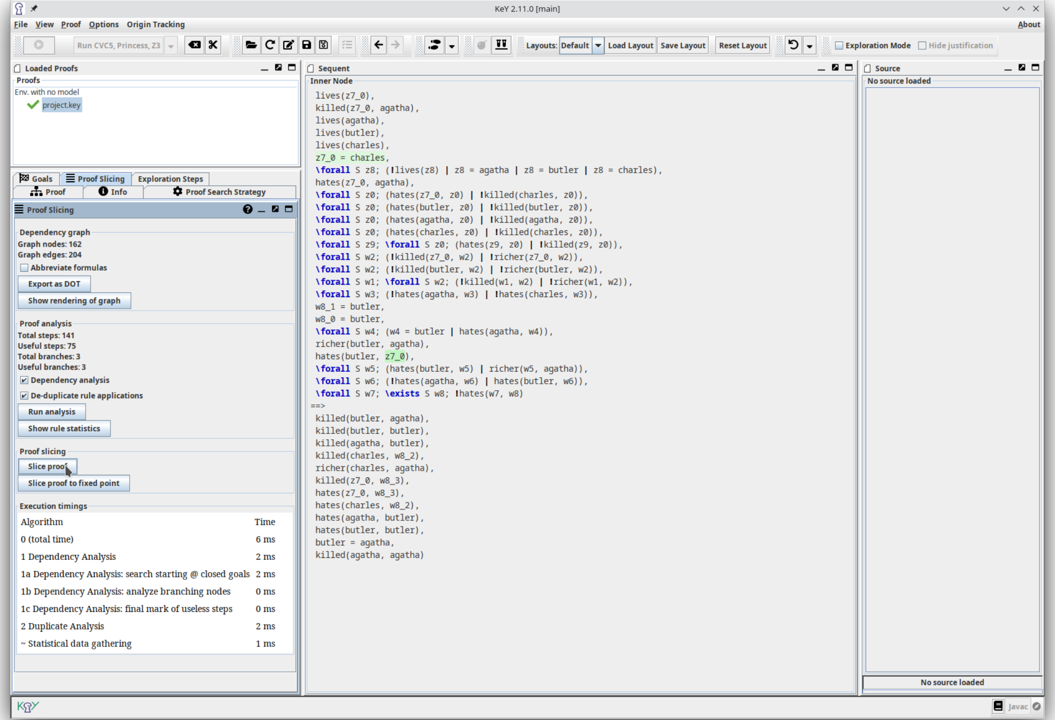
Slice proof to fixed point

Execution timings

Algorithm	Time
0 (total time)	6 ms
1 Dependency Analysis	2 ms
1a Dependency Analysis: search starting @ closed goals	2 ms
1b Dependency Analysis: analyze branching nodes	0 ms
1c Dependency Analysis: final mark of useless steps	0 ms
2 Duplicate Analysis	2 ms
~ Statistical data gathering	1 ms

```
 lives(z7_0),
 killed(z7_0, agatha),
 lives(agatha),
 lives(butler),
 lives(charles),
 z7_0 = charles,
 \forall S z8; (!lives(z8) | z8 = agatha | z8 = butler | z8 = charles),
 hates(z7_0, agatha),
 \forall S z0; (hates(z7_0, z0) | !killed(charles, z0)),
 \forall S z0; (hates(butler, z0) | !killed(butler, z0)),
 \forall S z0; (hates(agatha, z0) | !killed(agatha, z0)),
 \forall S z0; (hates(charles, z0) | !killed(charles, z0)),
 \forall S z9; \forall S z0; (hates(z9, z0) | !killed(z9, z0)),
 \forall S w2; (!killed(z7_0, w2) | !richer(z7_0, w2)),
 \forall S w2; (!killed(butler, w2) | !richer(butler, w2)),
 \forall S w1; \forall S w2; (!killed(w1, w2) | !richer(w1, w2)),
 \forall S w3; (!hates(agatha, w3) | !hates(charles, w3)),
 w8_1 = butler,
 w8_0 = butler,
 \forall S w4; (w4 = butler | hates(agatha, w4)),
 richer(butler, agatha),
 hates(butler, z7_0),
 \forall S w5; (hates(butler, w5) | richer(w5, agatha)),
 \forall S w6; (!hates(agatha, w6) | hates(butler, w6)),
 \forall S w7; \exists S w8; !hates(w7, w8)
==>
 killed(butler, agatha),
 killed(butler, butler),
 killed(agatha, butler),
 killed(charles, w8_2),
 richer(charles, agatha),
 killed(z7_0, w8_3),
 hates(z7_0, w8_3),
 hates(charles, w8_2),
 hates(agatha, butler),
 hates(butler, butler),
 butler = agatha,
 killed(agatha, agatha)
```

Strategy: Applied 141 rules (0.3 sec), closed 3 goals, 0 remaining



Proof Slicing

Dependency graph

Graph nodes: 162

Graph edges: 204

□ Abbreviate formulas

Proof analysis

Total steps: 141

Useful steps: 75

Total branches: 3

Useful branches: 3

☑ Dependency analysis

☑ De-duplicate rule applications

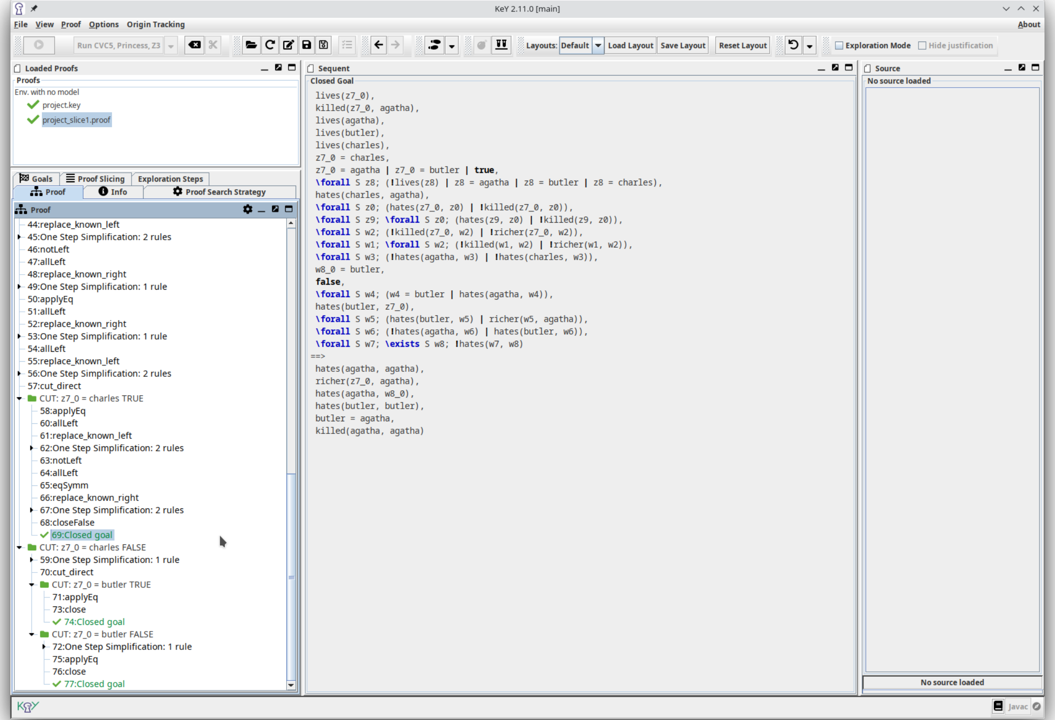
Proof slicing

Execution timings

Algorithm	Time
0 (total time)	6 ms
1 Dependency Analysis	2 ms
1a Dependency Analysis: search starting @ closed goals	2 ms
1b Dependency Analysis: analyze branching nodes	0 ms
1c Dependency Analysis: final mark of useless steps	0 ms
2 Duplicate Analysis	2 ms
~ Statistical data gathering	1 ms

Inner Node

```
 lives(z7_0),
 killed(z7_0, agatha),
 lives(agatha),
 lives(butler),
 lives(charles),
 z7_0 = charles,
 \forall S z8; (!lives(z8) | z8 = agatha | z8 = butler | z8 = charles),
 hates(z7_0, agatha),
 \forall S z0; (hates(z7_0, z0) | !killed(charles, z0)),
 \forall S z0; (hates(butler, z0) | !killed(butler, z0)),
 \forall S z0; (hates(agatha, z0) | !killed(agatha, z0)),
 \forall S z0; (hates(charles, z0) | !killed(charles, z0)),
 \forall S z9; \forall S z0; (hates(z9, z0) | !killed(z9, z0)),
 \forall S w2; (!killed(z7_0, w2) | !richer(z7_0, w2)),
 \forall S w2; (!killed(butler, w2) | !richer(butler, w2)),
 \forall S w1; \forall S w2; (!killed(w1, w2) | !richer(w1, w2)),
 \forall S w3; (!hates(agatha, w3) | !hates(charles, w3)),
 w8_1 = butler,
 w8_0 = butler,
 \forall S w4; (w4 = butler | hates(agatha, w4)),
 richer(butler, agatha),
 hates(butler, z7_0),
 \forall S w5; (hates(butler, w5) | richer(w5, agatha)),
 \forall S w6; (!hates(agatha, w6) | hates(butler, w6)),
 \forall S w7; \exists S w8; !hates(w7, w8)
==>
 killed(butler, agatha),
 killed(butler, butler),
 killed(agatha, butler),
 killed(charles, w8_2),
 richer(charles, agatha),
 killed(z7_0, w8_3),
 hates(z7_0, w8_3),
 hates(charles, w8_2),
 hates(agatha, butler),
 hates(butler, butler),
 butler = agatha,
 killed(agatha, agatha)
```



Proofs

Env. with no model

✓ project.key

✓ project_slice1.proof

Proof

```
44:replace_known_left
45:One Step Simplification: 2 rules
46:notLeft
47:allLeft
48:replace_known_right
49:One Step Simplification: 1 rule
50:applyEq
51:allLeft
52:replace_known_right
53:One Step Simplification: 1 rule
54:allLeft
55:replace_known_left
56:One Step Simplification: 2 rules
57:cut_direct
CUT: z7_0 = charles TRUE
    58:applyEq
    60:allLeft
    61:replace_known_left
    62:One Step Simplification: 2 rules
    63:notLeft
    64:allLeft
    65:eqSymm
    66:replace_known_right
    67:One Step Simplification: 2 rules
    68:closeFalse
    ✓ 69:Closed goal
CUT: z7_0 = charles FALSE
    59:One Step Simplification: 1 rule
    70:cut_direct
    CUT: z7_0 = butler TRUE
        71:applyEq
        73:close
        ✓ 74:Closed goal
    CUT: z7_0 = butler FALSE
        72:One Step Simplification: 1 rule
        75:applyEq
        76:close
        ✓ 77:Closed goal
```

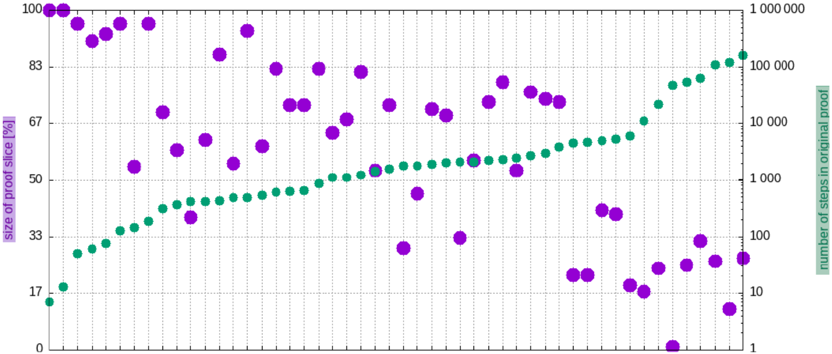
Closed Goal

```
lives(z7_0),
killed(z7_0, agatha),
lives(agatha),
lives(butler),
lives(charles),
z7_0 = charles,
z7_0 = agatha | z7_0 = butler | true,
\forall S z8; (!lives(z8) | z8 = agatha | z8 = butler | z8 = charles),
hates(charles, agatha),
\forall S z0; (hates(z7_0, z0) | !killed(z7_0, z0)),
\forall S z9; \forall S z0; (hates(z9, z0) | !killed(z9, z0)),
\forall S w2; (!killed(z7_0, w2) | !richer(z7_0, w2)),
\forall S w1; \forall S w2; (!killed(w1, w2) | !richer(w1, w2)),
\forall S w3; (!hates(agatha, w3) | !hates(charles, w3)),
w8_0 = butler,
false,
\forall S w4; (w4 = butler | hates(agatha, w4)),
hates(butler, z7_0),
\forall S w5; (hates(butler, w5) | richer(w5, agatha)),
\forall S w6; (!hates(agatha, w6) | hates(butler, w6)),
\forall S w7; \exists S w8; !hates(w7, w8)
==>
hates(agatha, agatha),
richer(z7_0, agatha),
hates(agatha, w8_0),
hates(butler, butler),
butler = agatha,
killed(agatha, agatha)
```

No source loaded

No source loaded

Evaluation



- Often, very large parts of proofs could be removed.
- Trend: The larger the proof, the larger the percentage.
- Most of the removed steps are normalizations of formulas which are never used later on.

Further Applications of the Dependency Analysis

- Rule de-duplication (implemented):
  - If the same rule is applied to the same formula(s) in two branches, it possibly can be moved in front of the branching rule.

Further Applications of the Dependency Analysis

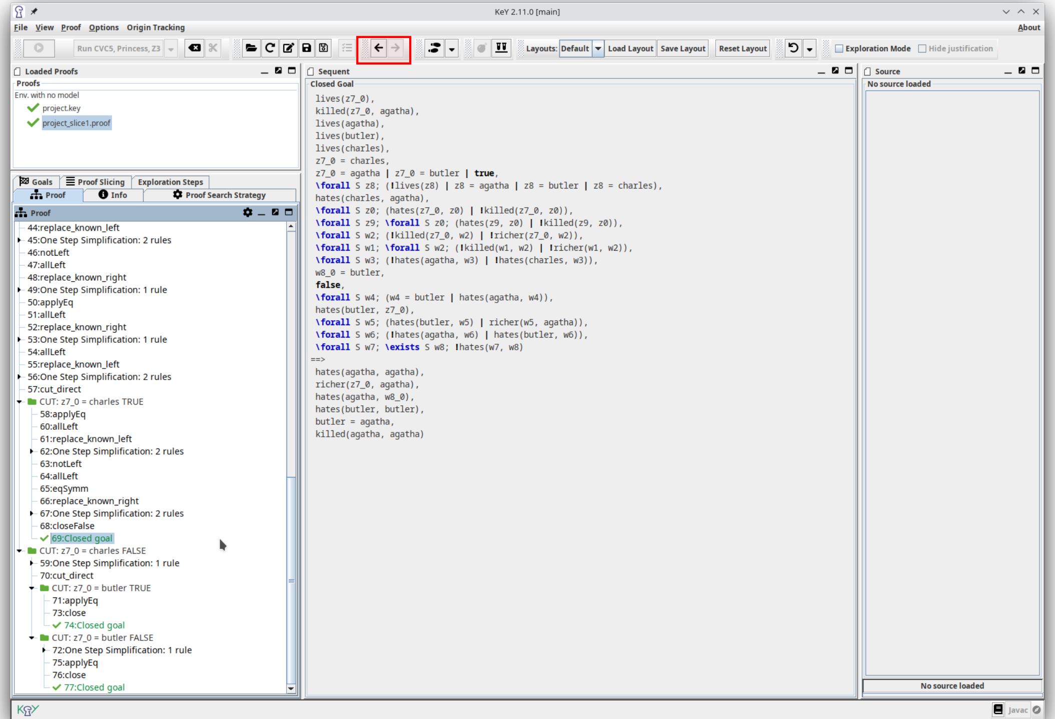
- Rule de-duplication (implemented):
  - If the same rule is applied to the same formula(s) in two branches, it possibly can be moved in front of the branching rule.
- Slice w.r.t. a selected formula (implemented):
  - “Which steps produced this formula?”
  - “Which input formulas are needed to derive this formula?”

Further Applications of the Dependency Analysis

- Rule de-duplication (implemented):
  - If the same rule is applied to the same formula(s) in two branches, it possibly can be moved in front of the branching rule.
- Slice w.r.t. a selected formula (implemented):
  - “Which steps produced this formula?”
  - “Which input formulas are needed to derive this formula?”
- Proof Reordering (ongoing work):
  - Group certain rule applications (similar to One-Step-Simplification)
  - Possible categories: NNF, Polynomial simplification, ...

Further Usability Improvements

- Proof Slicing ✓
- Navigation History ◀
- Undoing Interactions ◀
- Automatically run JavaC first (Alexander Weigl)
- Background SMT (ongoing)
- Proof Caching (ongoing)



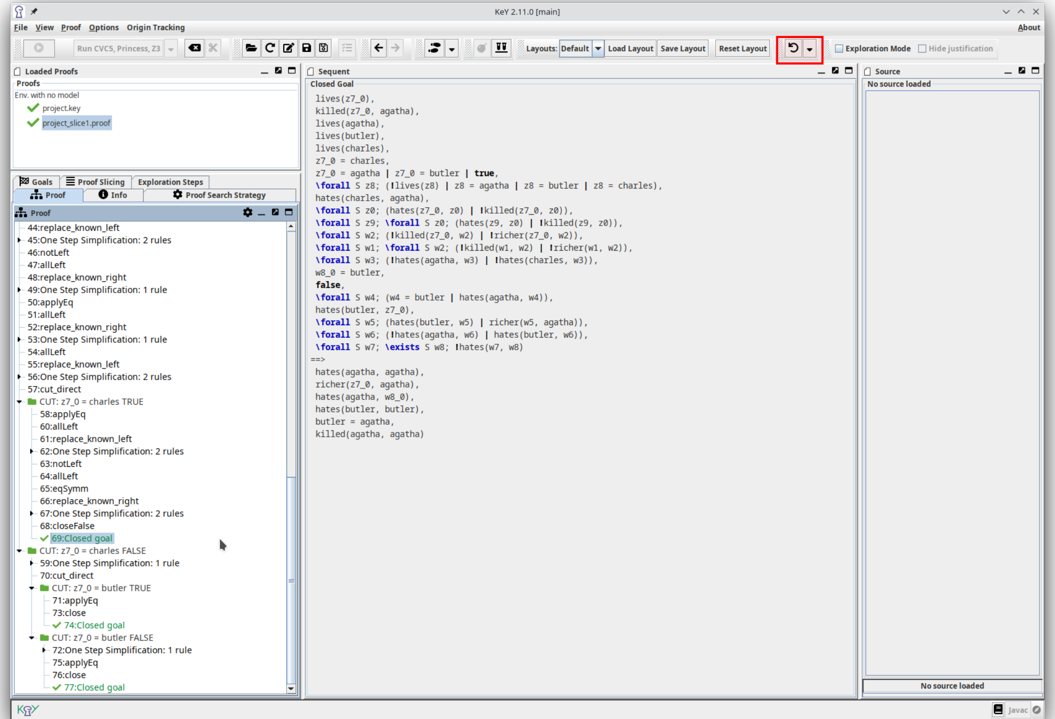
Proofs
Env. with no model
✓ project.key
✓ project_slice1.proof

Proof

```
44:replace_known_left
45:One Step Simplification: 2 rules
46:notLeft
47:allLeft
48:replace_known_right
49:One Step Simplification: 1 rule
50:applyEq
51:allLeft
52:replace_known_right
53:One Step Simplification: 1 rule
54:allLeft
55:replace_known_left
56:One Step Simplification: 2 rules
57:cut_direct
CUT: z7_0 = charles TRUE
  58:applyEq
  60:allLeft
  61:replace_known_left
  62:One Step Simplification: 2 rules
  63:notLeft
  64:allLeft
  65:eqSymm
  66:replace_known_right
  67:One Step Simplification: 2 rules
  68:closeFalse
  ✓ 69:Closed goal
CUT: z7_0 = charles FALSE
  59:One Step Simplification: 1 rule
  70:cut_direct
  CUT: z7_0 = butler TRUE
    71:applyEq
    73:close
    ✓ 74:Closed goal
  CUT: z7_0 = butler FALSE
    72:One Step Simplification: 1 rule
    75:applyEq
    76:close
    ✓ 77:Closed goal
```

Closed Goal

```
 lives(z7_0),
 killed(z7_0, agatha),
 lives(agatha),
 lives(butler),
 lives(charles),
 z7_0 = charles,
 z7_0 = agatha | z7_0 = butler | true,
 \forall S z8; (!lives(z8) | z8 = agatha | z8 = butler | z8 = charles),
 hates(charles, agatha),
 \forall S z0; (hates(z7_0, z0) | !killed(z7_0, z0)),
 \forall S z9; \forall S z0; (hates(z9, z0) | !killed(z9, z0)),
 \forall S w2; (!killed(z7_0, w2) | !richer(z7_0, w2)),
 \forall S w1; \forall S w2; (!killed(w1, w2) | !richer(w1, w2)),
 \forall S w3; (!hates(agatha, w3) | !hates(charles, w3)),
 w8_0 = butler,
 false,
 \forall S w4; (w4 = butler | hates(agatha, w4)),
 hates(butler, z7_0),
 \forall S w5; (hates(butler, w5) | richer(w5, agatha)),
 \forall S w6; (!hates(agatha, w6) | hates(butler, w6)),
 \forall S w7; \exists S w8; !hates(w7, w8)
==>
 hates(agatha, agatha),
 richer(z7_0, agatha),
 hates(agatha, w8_0),
 hates(butler, butler),
 butler = agatha,
 killed(agatha, agatha)
```



Proofs
Env. with no model
✓ project.key
✓ project_slice1.proof

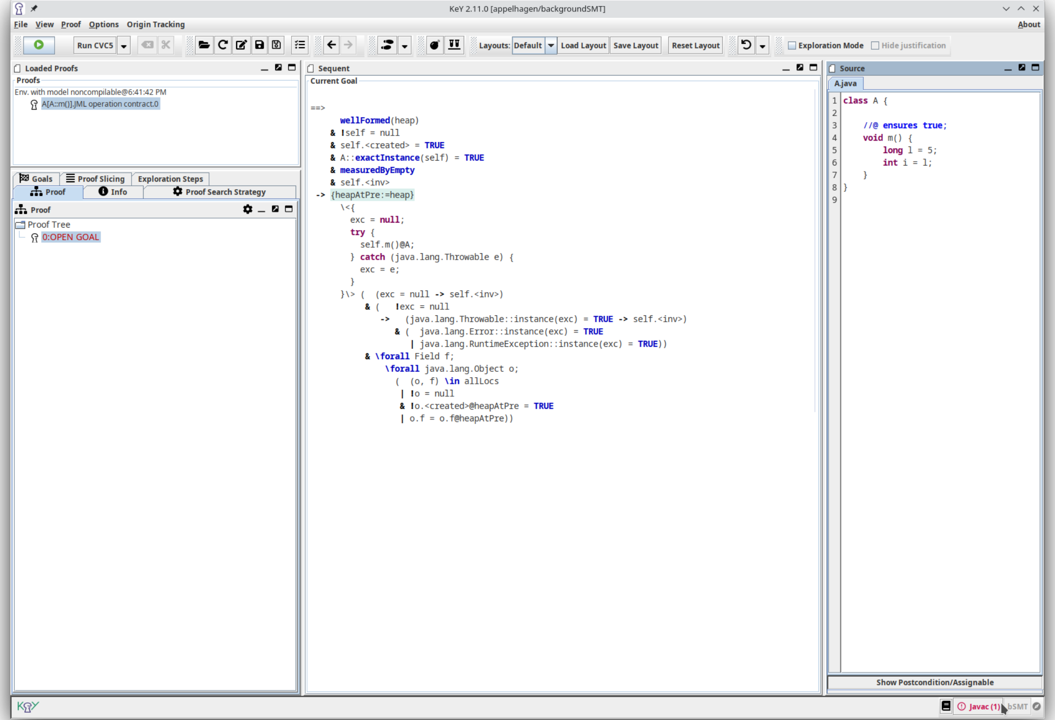
Closed Goal

```
 lives(z7_0),
 killed(z7_0, agatha),
 lives(agatha),
 lives(butler),
 lives(charles),
 z7_0 = charles,
 z7_0 = agatha | z7_0 = butler | true,
 \forall S z8; (!lives(z8) | z8 = agatha | z8 = butler | z8 = charles),
 hates(charles, agatha),
 \forall S z0; (hates(z7_0, z0) | !killed(z7_0, z0)),
 \forall S z9; \forall S z0; (hates(z9, z0) | !killed(z9, z0)),
 \forall S w2; (!killed(z7_0, w2) | !richer(z7_0, w2)),
 \forall S w1; \forall S w2; (!killed(w1, w2) | !richer(w1, w2)),
 \forall S w3; (!hates(agatha, w3) | !hates(charles, w3)),
 w8_0 = butler,
 false,
 \forall S w4; (w4 = butler | hates(agatha, w4)),
 hates(butler, z7_0),
 \forall S w5; (hates(butler, w5) | richer(w5, agatha)),
 \forall S w6; (!hates(agatha, w6) | hates(butler, w6)),
 \forall S w7; \exists S w8; !hates(w7, w8)
==>
 hates(agatha, agatha),
 richer(z7_0, agatha),
 hates(agatha, w8_0),
 hates(butler, butler),
 butler = agatha,
 killed(agatha, agatha)
```

```
44:replace_known_left
45:One Step Simplification: 2 rules
46:notLeft
47:allLeft
48:replace_known_right
49:One Step Simplification: 1 rule
50:applyEq
51:allLeft
52:replace_known_right
53:One Step Simplification: 1 rule
54:allLeft
55:replace_known_left
56:One Step Simplification: 2 rules
57:cut_direct
CUT: z7_0 = charles TRUE
    58:applyEq
    60:allLeft
    61:replace_known_left
    62:One Step Simplification: 2 rules
    63:notLeft
    64:allLeft
    65:eqSymm
    66:replace_known_right
    67:One Step Simplification: 2 rules
    68:closeFalse
    ✓ 69:Closed goal
CUT: z7_0 = charles FALSE
    59:One Step Simplification: 1 rule
    70:cut_direct
    CUT: z7_0 = butler TRUE
        71:applyEq
        73:close
        ✓ 74:Closed goal
    CUT: z7_0 = butler FALSE
        72:One Step Simplification: 1 rule
        75:applyEq
        76:close
        ✓ 77:Closed goal
```

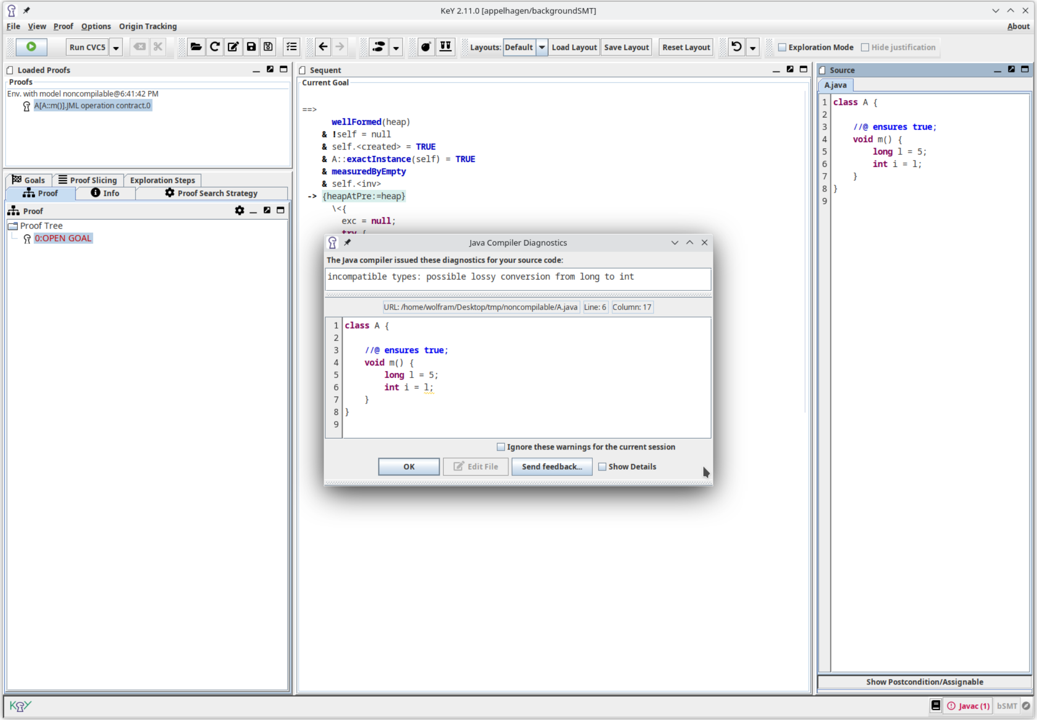
Further Usability Improvements

- Proof Slicing ✓
- Navigation History ✓
- Undoing Interactions ✓
- Automatically run JavaC first (Alexander Weigl) ◀
- Background SMT (ongoing)
- Proof Caching (ongoing)



```
==>
     wellFormed(heap)
   & !self = null
   & self.<created> = TRUE
   & A::exactInstance(self) = TRUE
   & measuredByEmpty
   & self.<inv>
 -> {heapAtPre:=heap}
      \<{
        exc = null;
        try {
          self.m()@A;
        } catch (java.lang.Throwable e) {
          exc = e;
        }
      }\> (  (exc = null -> self.<inv>)
           & (   !exc = null
              ->   (java.lang.Throwable::instance(exc) = TRUE -> self.<inv>)
                 & (  java.lang.Error::instance(exc) = TRUE
                    | java.lang.RuntimeException::instance(exc) = TRUE))
           & \forall Field f;
               \forall java.lang.Object o;
                 (  (o, f) \in allLocs
                  | !o = null
                  & !o.<created>@heapAtPre = TRUE
                  | o.f = o.f@heapAtPre))
```

```java
class A {

    //@ ensures true;
    void m() {
        long l = 5;
        int i = l;
    }
}
```



Env. with model noncompilable@6:41:42 PM

A[A::m()].JML operation contract.0

0:OPEN GOAL

```
==>
   wellFormed(heap)
 & !self = null
 & self.<created> = TRUE
 & A::exactInstance(self) = TRUE
 & measuredByEmpty
 & self.<inv>
-> {heapAtPre:=heap}
    \<{
     exc = null;
     try {
```

```
class A {

    //@ ensures true;
    void m() {
        long l = 5;
        int i = l;
    }
}
```

Java Compiler Diagnostics

The Java compiler issued these diagnostics for your source code:

```
incompatible types: possible lossy conversion from long to int
```

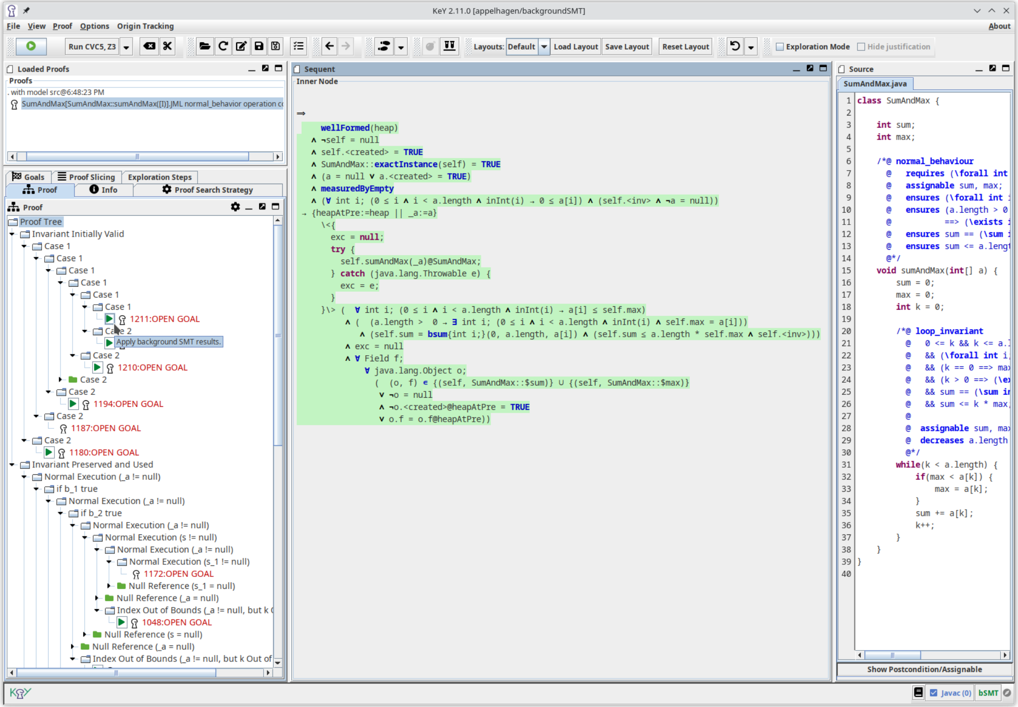
URL: /home/wolfram/Desktop/tmp/noncompilable/A.java Line: 6 Column: 17

```
class A {

    //@ ensures true;
    void m() {
        long l = 5;
        int i = l;
    }
}
```

□ Ignore these warnings for the current session

Further Usability Improvements

- Proof Slicing ✓
- Navigation History ✓
- Undoing Interactions ✓
- Automatically run JavaC first (Alexander Weigl) ✓
- Background SMT (ongoing) ◀
- Proof Caching (ongoing)



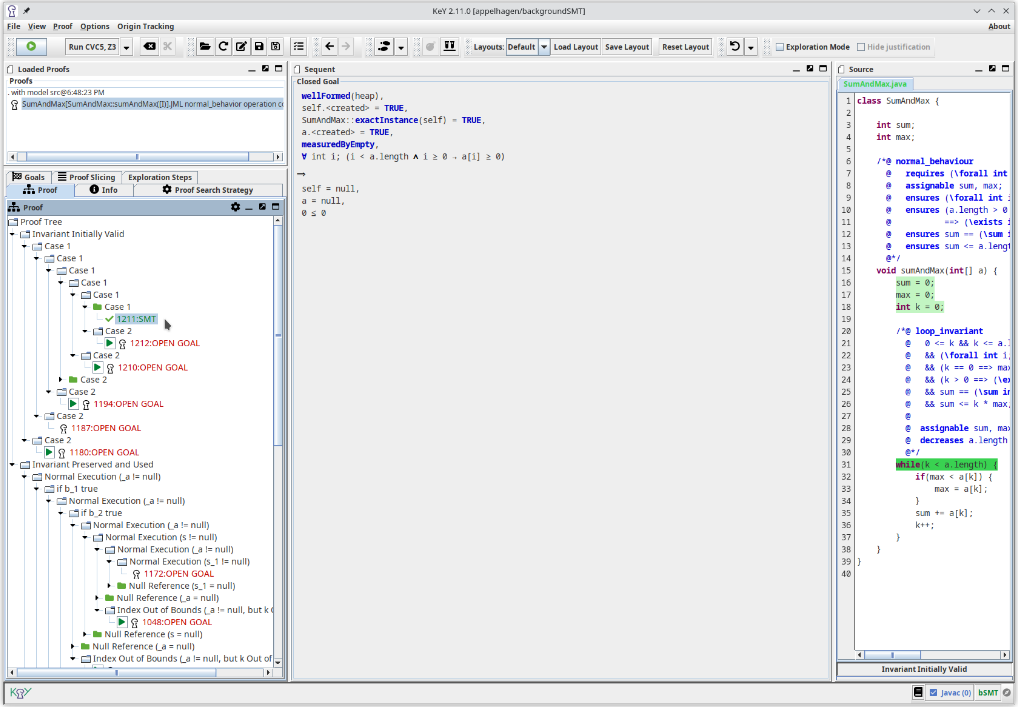
. with model src@6:48:23 PM
SumAndMax[SumAndMax::sumAndMax([I)].JML normal_behavior operation co

```
Proof Tree
  Invariant Initially Valid
    Case 1
      Case 1
        Case 1
          Case 1
            Case 1
              Case 1
                1211:OPEN GOAL
              Case 2
                Apply background SMT results.
            Case 2
              1210:OPEN GOAL
          Case 2
        Case 2
          1194:OPEN GOAL
      Case 2
        1187:OPEN GOAL
    Case 2
      1180:OPEN GOAL
  Invariant Preserved and Used
    Normal Execution (_a != null)
      if b_1 true
        Normal Execution (_a != null)
          if b_2 true
            Normal Execution (_a != null)
              Normal Execution (s != null)
                Normal Execution (_a != null)
                  Normal Execution (s_1 != null)
                    1172:OPEN GOAL
                  Null Reference (s_1 = null)
                Null Reference (_a = null)
                Index Out of Bounds (_a != null, but k (
                  1048:OPEN GOAL
              Null Reference (s = null)
            Null Reference (_a = null)
            Index Out of Bounds (_a != null, but k Out of
```

Inner Node

```
⟹
    wellFormed(heap)
  ∧ ¬self = null
  ∧ self.<created> = TRUE
  ∧ SumAndMax::exactInstance(self) = TRUE
  ∧ (a = null ∨ a.<created> = TRUE)
  ∧ measuredByEmpty
  ∧ (∀ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) → 0 ≤ a[i]) ∧ (self.<inv> ∧ ¬a = null))
→ {heapAtPre:=heap || _a:=a}
    \<{
      exc = null;
      try {
        self.sumAndMax(_a)@SumAndMax;
      } catch (java.lang.Throwable e) {
        exc = e;
      }
    }\> (  ∀ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) → a[i] ≤ self.max)
         ∧ (  (a.length >  0 → ∃ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) ∧ self.max = a[i]))
            ∧ (self.sum = bsum{int i;}(0, a.length, a[i]) ∧ (self.sum ≤ a.length * self.max ∧ self.<inv>)))
         ∧ exc = null
         ∧ ∀ Field f;
             ∀ java.lang.Object o;
               (  (o, f) ∈ {(self, SumAndMax::$sum)} ∪ {(self, SumAndMax::$max)}
                ∨ ¬o = null
                ∧ ¬o.<created>@heapAtPre = TRUE
                ∨ o.f = o.f@heapAtPre))
```

```java
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int
      @   assignable sum, max;
      @   ensures (\forall int i
      @   ensures (a.length > 0
      @           ==> (\exists i
      @   ensures sum == (\sum i
      @   ensures sum <= a.lengt
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.l
          @   && (\forall int i,
          @   && (k == 0 ==> max
          @   && (k > 0 ==> (\ex
          @   && sum == (\sum in
          @   && sum <= k * max;
          @
          @  assignable sum, max
          @  decreases a.length
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```



KeY 2.11.0 [appelhagen/backgroundSMT]
File View Proof Options Origin Tracking About
Run CVC5, Z3
Layouts: Default Load Layout Save Layout Reset Layout
Exploration Mode
Hide justification
Loaded Proofs
Proofs
Goals Proof Slicing Exploration Steps
Proof Info Proof Search Strategy
Sequent
Source
SumAndMax.java
Javac (0) bSMT

. with model src@6:48:23 PM
SumAndMax[SumAndMax::sumAndMax([I)].JML normal_behavior operation c

Proof

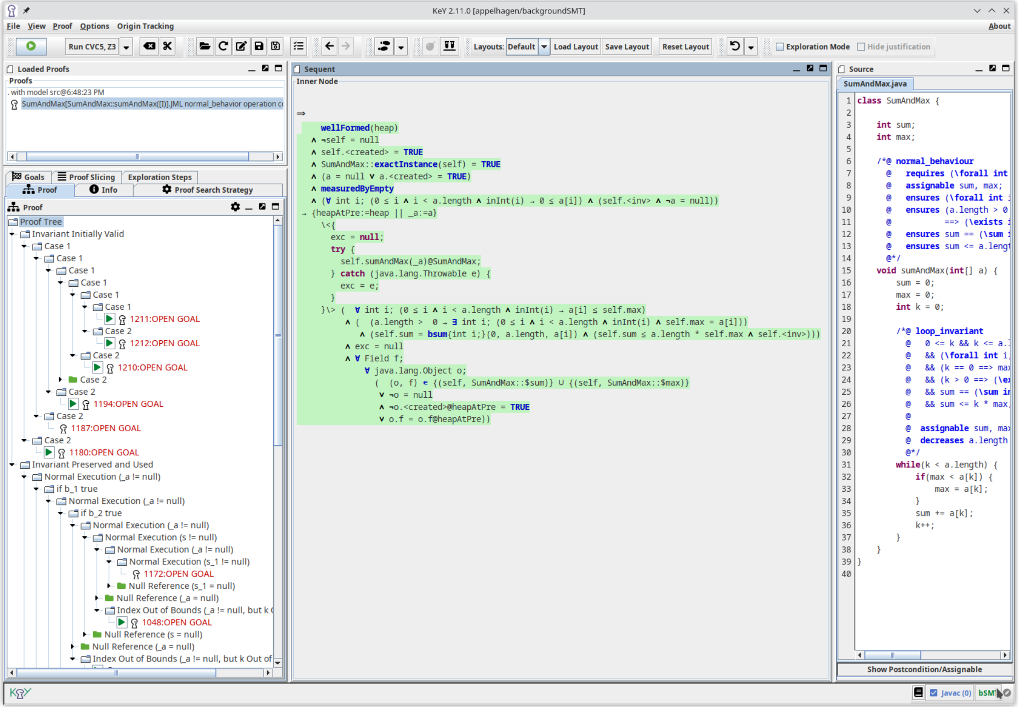
```
Proof Tree
  Invariant Initially Valid
    Case 1
      Case 1
        Case 1
          Case 1
            Case 1
              Case 1
                1211:SMT
              Case 2
                1212:OPEN GOAL
            Case 2
              1210:OPEN GOAL
          Case 2
        Case 2
          1194:OPEN GOAL
      Case 2
        1187:OPEN GOAL
    Case 2
      1180:OPEN GOAL
  Invariant Preserved and Used
    Normal Execution (_a != null)
      if b_1 true
        Normal Execution (_a != null)
          if b_2 true
            Normal Execution (_a != null)
              Normal Execution (s != null)
                Normal Execution (_a != null)
                  Normal Execution (s_1 != null)
                    1172:OPEN GOAL
                  Null Reference (s_1 = null)
                Null Reference (_a = null)
                Index Out of Bounds (_a != null, but k (
                  1048:OPEN GOAL
              Null Reference (s = null)
            Null Reference (_a = null)
            Index Out of Bounds (_a != null, but k Out of
```

Closed Goal

```
wellFormed(heap),
self.<created> = TRUE,
SumAndMax::exactInstance(self) = TRUE,
a.<created> = TRUE,
measuredByEmpty,
∀ int i; (i < a.length ∧ i ≥ 0 → a[i] ≥ 0)

⟹

self = null,
a = null,
0 ≤ 0
```

```
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int
      @   assignable sum, max;
      @   ensures (\forall int i
      @   ensures (a.length > 0
      @           ==> (\exists i
      @   ensures sum == (\sum i
      @   ensures sum <= a.lengt
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.l
          @   && (\forall int i,
          @   && (k == 0 ==> max
          @   && (k > 0 ==> (\ex
          @   && sum == (\sum in
          @   && sum <= k * max;
          @
          @  assignable sum, max
          @  decreases a.length
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```

Invariant Initially Valid

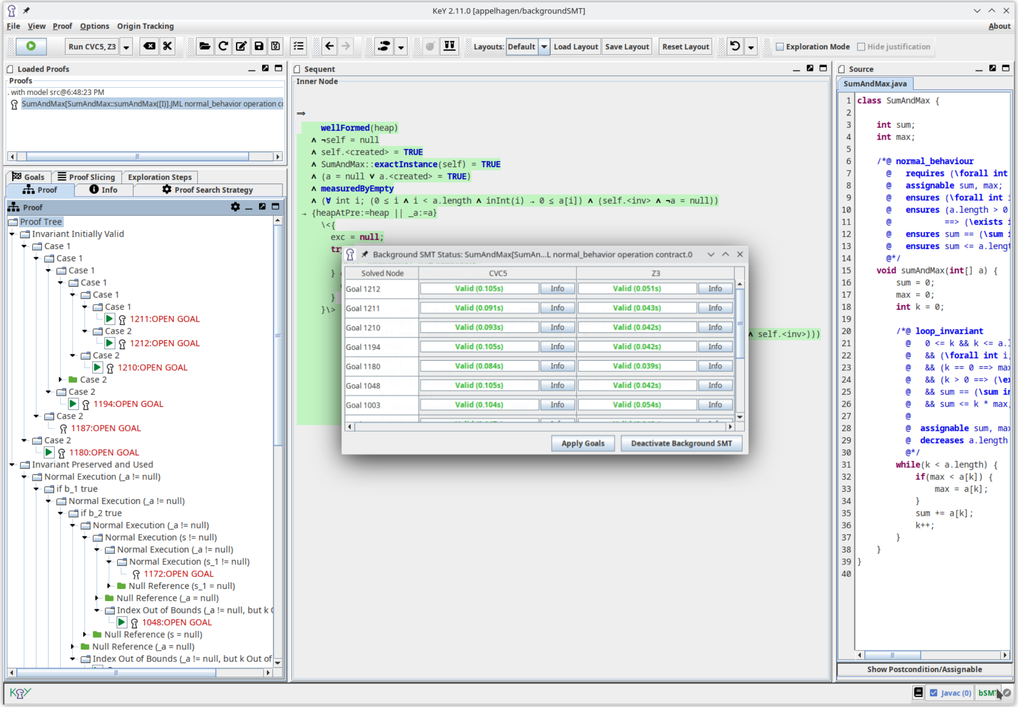


. with model src@6:48:23 PM
SumAndMax[SumAndMax::sumAndMax([I)].JML normal_behavior operation c

```
Proof Tree
  Invariant Initially Valid
    Case 1
      Case 1
        Case 1
          Case 1
            Case 1
              Case 1
                1211:OPEN GOAL
              Case 2
                1212:OPEN GOAL
            Case 2
              1210:OPEN GOAL
          Case 2
        Case 2
          1194:OPEN GOAL
      Case 2
        1187:OPEN GOAL
    Case 2
      1180:OPEN GOAL
  Invariant Preserved and Used
    Normal Execution (_a != null)
      if b_1 true
        Normal Execution (_a != null)
          if b_2 true
            Normal Execution (_a != null)
              Normal Execution (s != null)
                Normal Execution (_a != null)
                  Normal Execution (s_1 != null)
                    1172:OPEN GOAL
                  Null Reference (s_1 = null)
                Null Reference (_a = null)
                Index Out of Bounds (_a != null, but k (
                  1048:OPEN GOAL
              Null Reference (s = null)
            Null Reference (_a = null)
            Index Out of Bounds (_a != null, but k Out of
```

```
⟹
    wellFormed(heap)
  ∧ ¬self = null
  ∧ self.<created> = TRUE
  ∧ SumAndMax::exactInstance(self) = TRUE
  ∧ (a = null ∨ a.<created> = TRUE)
  ∧ measuredByEmpty
  ∧ (∀ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) → 0 ≤ a[i]) ∧ (self.<inv> ∧ ¬a = null))
→ {heapAtPre:=heap || _a:=a}
    \<{
      exc = null;
      try {
        self.sumAndMax(_a)@SumAndMax;
      } catch (java.lang.Throwable e) {
        exc = e;
      }
    }\> (  ∀ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) → a[i] ≤ self.max)
         ∧ (  (a.length >  0 → ∃ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) ∧ self.max = a[i]))
            ∧ (self.sum = bsum{int i;}(0, a.length, a[i]) ∧ (self.sum ≤ a.length * self.max ∧ self.<inv>)))
         ∧ exc = null
         ∧ ∀ Field f;
             ∀ java.lang.Object o;
               (  (o, f) ∈ {(self, SumAndMax::$sum)} ∪ {(self, SumAndMax::$max)}
                ∨ ¬o = null
                ∧ ¬o.<created>@heapAtPre = TRUE
                ∨ o.f = o.f@heapAtPre))
```

```
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int
      @   assignable sum, max;
      @   ensures (\forall int i
      @   ensures (a.length > 0
      @             ==> (\exists i
      @   ensures sum == (\sum i
      @   ensures sum <= a.lengt
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.l
          @   && (\forall int i,
          @   && (k == 0 ==> max
          @   && (k > 0 ==> (\ex
          @   && sum == (\sum in
          @   && sum <= k * max,
          @
          @  assignable sum, max
          @  decreases a.length
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}

```



Proof

```
Proof Tree
  Invariant Initially Valid
    Case 1
      Case 1
        Case 1
          Case 1
            Case 1
              Case 1
                1211:OPEN GOAL
              Case 2
                1212:OPEN GOAL
            Case 2
              1210:OPEN GOAL
          Case 2
        Case 2
          1194:OPEN GOAL
      Case 2
        1187:OPEN GOAL
    Case 2
      1180:OPEN GOAL
  Invariant Preserved and Used
    Normal Execution (_a != null)
      if b_1 true
        Normal Execution (_a != null)
          if b_2 true
            Normal Execution (_a != null)
              Normal Execution (s != null)
                Normal Execution (_a != null)
                  Normal Execution (s_1 != null)
                    1172:OPEN GOAL
                  Null Reference (s_1 = null)
                Null Reference (_a = null)
                Index Out of Bounds (_a != null, but k
                  1048:OPEN GOAL
              Null Reference (s = null)
            Null Reference (_a = null)
            Index Out of Bounds (_a != null, but k Out of
```

Inner Node

```
⟹

    wellFormed(heap)
  ∧ ¬self = null
  ∧ self.<created> = TRUE
  ∧ SumAndMax::exactInstance(self) = TRUE
  ∧ (a = null ∨ a.<created> = TRUE)
  ∧ measuredByEmpty
  ∧ (∀ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) → 0 ≤ a[i]) ∧ (self.<inv> ∧ ¬a = null))
→ {heapAtPre:=heap || _a:=a}
    \<{
      exc = null;
      tr
      }
      }
    }\>                                                                    ∧ self.<inv>)))
```

Background SMT Status: SumAndMax[SumAn...L normal_behavior operation contract.0

Solved Node	CVC5		Z3	
Goal 1212	Valid (0.105s)	Info	Valid (0.051s)	Info
Goal 1211	Valid (0.091s)	Info	Valid (0.043s)	Info
Goal 1210	Valid (0.093s)	Info	Valid (0.042s)	Info
Goal 1194	Valid (0.105s)	Info	Valid (0.042s)	Info
Goal 1180	Valid (0.084s)	Info	Valid (0.039s)	Info
Goal 1048	Valid (0.105s)	Info	Valid (0.042s)	Info
Goal 1003	Valid (0.104s)	Info	Valid (0.054s)	Info

Apply Goals
Deactivate Background SMT

```java
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int
      @   assignable sum, max;
      @   ensures (\forall int i
      @   ensures (a.length > 0
      @             ==> (\exists i
      @   ensures sum == (\sum i
      @   ensures sum <= a.lengt
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.l
          @   && (\forall int i,
          @   && (k == 0 ==> max
          @   && (k > 0 ==> (\ex
          @   && sum == (\sum in
          @   && sum <= k * max,
          @
          @  assignable sum, max
          @  decreases a.length
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```

Further Usability Improvements

- Proof Slicing ✓
- Navigation History ✓
- Undoing Interactions ✓
- Automatically run JavaC first (Alexander Weigl) ✓
- Background SMT (ongoing) ✓
- Proof Caching (ongoing) ◀

Proof Caching

Motivation: Finding the correct and provable specification is often an iterative process.

Proof Caching

Motivation: Finding the correct and provable specification is often an iterative process.

Observation: If $\Gamma \vdash \Delta$ is valid, then $\Gamma, E \vdash \Delta, Z$ is also valid (*).

Proof Caching

Motivation: Finding the correct and provable specification is often an iterative process.

Observation: If $\Gamma \vdash \Delta$ is valid, then $\Gamma, E \vdash \Delta, Z$ is also valid (*).

(*) Under some restrictions:

- The extended sequent must not have modalities or queries (Java code could differ).
- Both must use the same taclet options.
- The same added rules must be present.

Proof Caching

Motivation: Finding the correct and provable specification is often an iterative process.

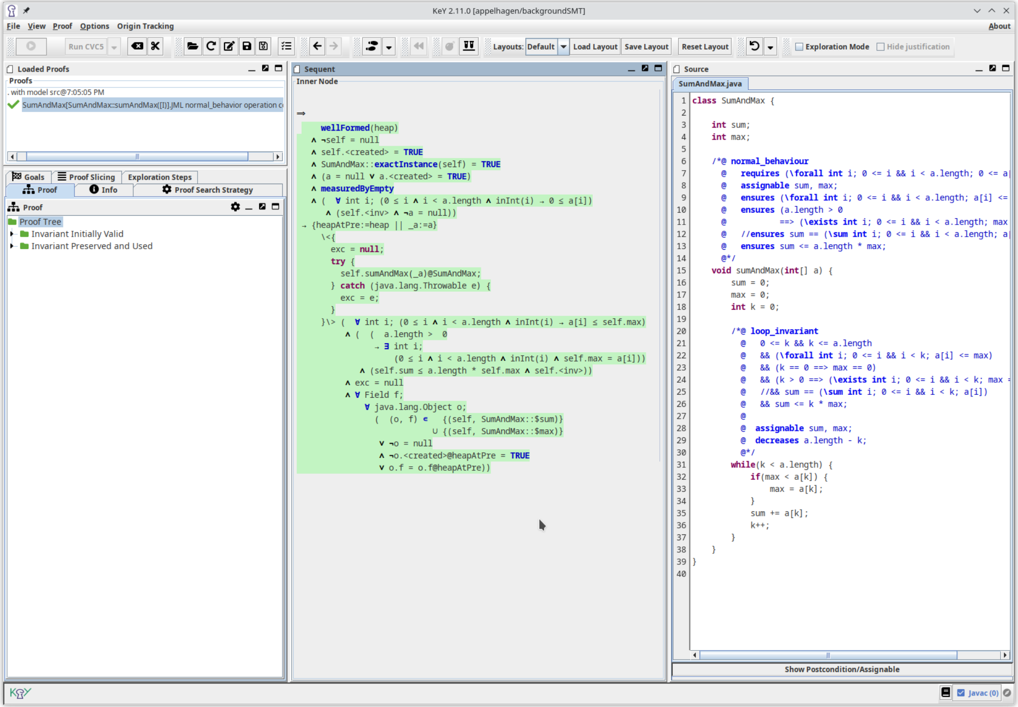
Observation: If $\Gamma \vdash \Delta$ is valid, then $\Gamma, E \vdash \Delta, Z$ is also valid (*).

(*) Under some restrictions:

- The extended sequent must not have modalities or queries (Java code could differ).
- Both must use the same taclet options.
- The same added rules must be present.

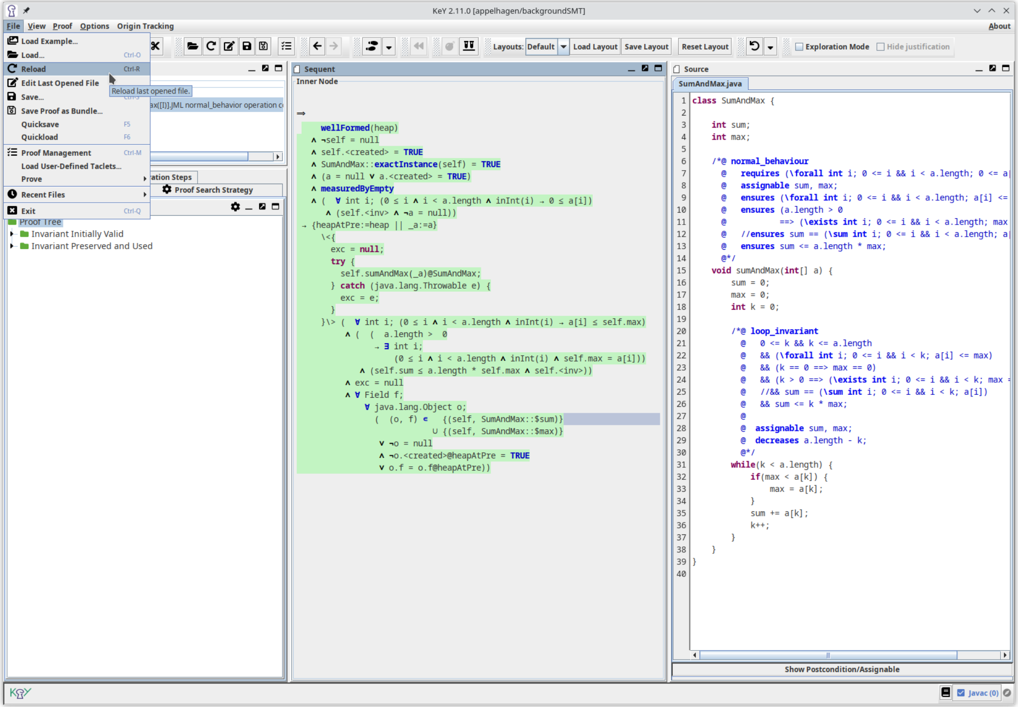
Ongoing work:

- Which sequents should be in the cache?
- Extend the caching beyond a single run of KeY (ongoing).
- Relax the above conditions.


KeY 2.11.0 [appelhagen/backgroundSMT]
File View Proof Options Origin Tracking
About
Run CVC5
Layouts: Default
Load Layout
Save Layout
Reset Layout
□ Exploration Mode
□ Hide justification
Loaded Proofs
Proofs
. with model src@7:05:05 PM
✓ SumAndMax[SumAndMax::sumAndMax([I)].JML normal_behavior operation c
Goals
Proof Slicing
Exploration Steps
Proof
Info
Proof Search Strategy
Proof
Proof Tree
Invariant Initially Valid
Invariant Preserved and Used
Sequent
Inner Node
Source
SumAndMax.java
Show Postcondition/Assignable
☑ Javac (0)

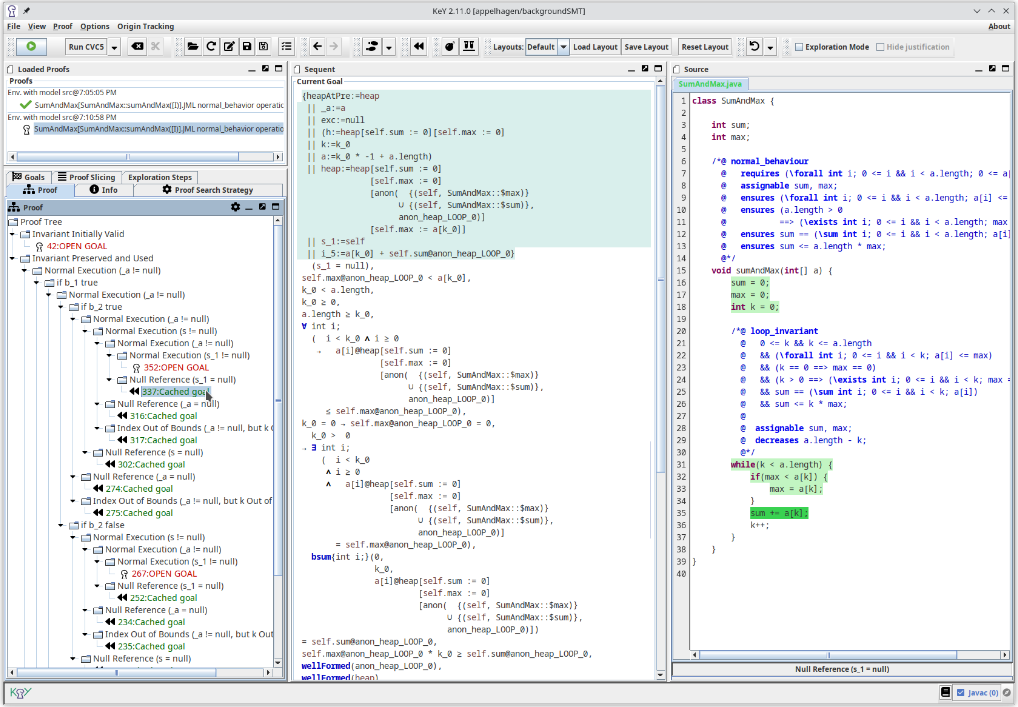
```
⟹
    wellFormed(heap)
  ∧ ¬self = null
  ∧ self.<created> = TRUE
  ∧ SumAndMax::exactInstance(self) = TRUE
  ∧ (a = null ∨ a.<created> = TRUE)
  ∧ measuredByEmpty
  ∧ (  ∀ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) → 0 ≤ a[i])
     ∧ (self.<inv> ∧ ¬a = null))
→ {heapAtPre:=heap || _a:=a}
    \<{
      exc = null;
      try {
        self.sumAndMax(_a)@SumAndMax;
      } catch (java.lang.Throwable e) {
        exc = e;
      }
    }\> (  ∀ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) → a[i] ≤ self.max)
         ∧ (  (  a.length >  0
                → ∃ int i;
                    (0 ≤ i ∧ i < a.length ∧ inInt(i) ∧ self.max = a[i]))
            ∧ (self.sum ≤ a.length * self.max ∧ self.<inv>))
         ∧ exc = null
         ∧ ∀ Field f;
             ∀ java.lang.Object o;
               (  (o, f) ∈   {(self, SumAndMax::$sum)}
                           ∪ {(self, SumAndMax::$max)}
                ∨ ¬o = null
                ∧ ¬o.<created>@heapAtPre = TRUE
                ∨ o.f = o.f@heapAtPre))
```

```java
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a
      @   assignable sum, max;
      @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <=
      @   ensures (a.length > 0
      @           ==> (\exists int i; 0 <= i && i < a.length; max
      @   //ensures sum == (\sum int i; 0 <= i && i < a.length; a
      @   ensures sum <= a.length * max;
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.length
          @   && (\forall int i; 0 <= i && i < k; a[i] <= max)
          @   && (k == 0 ==> max == 0)
          @   && (k > 0 ==> (\exists int i; 0 <= i && i < k; max =
          @   //&& sum == (\sum int i; 0 <= i && i < k; a[i])
          @   && sum <= k * max;
          @
          @  assignable sum, max;
          @  decreases a.length - k;
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}

```



```
⟹
    wellFormed(heap)
  ∧ ¬self = null
  ∧ self.<created> = TRUE
  ∧ SumAndMax::exactInstance(self) = TRUE
  ∧ (a = null ∨ a.<created> = TRUE)
  ∧ measuredByEmpty
  ∧ (  ∀ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) → 0 ≤ a[i])
     ∧ (self.<inv> ∧ ¬a = null))
→ {heapAtPre:=heap || _a:=a}
    \<{
      exc = null;
      try {
        self.sumAndMax(_a)@SumAndMax;
      } catch (java.lang.Throwable e) {
        exc = e;
      }
    }\> (  ∀ int i; (0 ≤ i ∧ i < a.length ∧ inInt(i) → a[i] ≤ self.max)
         ∧ (  (  a.length >  0
               → ∃ int i;
                   (0 ≤ i ∧ i < a.length ∧ inInt(i) ∧ self.max = a[i]))
            ∧ (self.sum ≤ a.length * self.max ∧ self.<inv>))
         ∧ exc = null
         ∧ ∀ Field f;
             ∀ java.lang.Object o;
               (  (o, f) ∊   {(self, SumAndMax::$sum)}
                           ∪ {(self, SumAndMax::$max)}
                ∨ ¬o = null
                ∧ ¬o.<created>@heapAtPre = TRUE
                ∨ o.f = o.f@heapAtPre))
```

```
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a
      @   assignable sum, max;
      @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <=
      @   ensures (a.length > 0
      @           ==> (\exists int i; 0 <= i && i < a.length; max
      @   //ensures sum == (\sum int i; 0 <= i && i < a.length; a
      @   ensures sum <= a.length * max;
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.length
          @   && (\forall int i; 0 <= i && i < k; a[i] <= max)
          @   && (k == 0 ==> max == 0)
          @   && (k > 0 ==> (\exists int i; 0 <= i && i < k; max =
          @   //&& sum == (\sum int i; 0 <= i && i < k; a[i])
          @   && sum <= k * max;
          @
          @  assignable sum, max;
          @  decreases a.length - k;
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```



Proofs

Env. with model src@7:05:05 PM
✓ SumAndMax[SumAndMax::sumAndMax([I)].JML normal_behavior operatic
Env. with model src@7:10:58 PM
SumAndMax[SumAndMax::sumAndMax([I)].JML normal_behavior operatic

Proof

```
Proof Tree
  Invariant Initially Valid
    42:OPEN GOAL
  Invariant Preserved and Used
    Normal Execution (_a != null)
      if b_1 true
        Normal Execution (_a != null)
          if b_2 true
            Normal Execution (_a != null)
              Normal Execution (s != null)
                Normal Execution (_a != null)
                  Normal Execution (s_1 != null)
                    352:OPEN GOAL
                  Null Reference (s_1 = null)
                    337:Cached goal
                Null Reference (_a = null)
                  316:Cached goal
                Index Out of Bounds (_a != null, but k (
                  317:Cached goal
              Null Reference (s = null)
                302:Cached goal
            Null Reference (_a = null)
              274:Cached goal
            Index Out of Bounds (_a != null, but k Out of
              275:Cached goal
          if b_2 false
            Normal Execution (s != null)
              Normal Execution (_a != null)
                Normal Execution (s_1 != null)
                  267:OPEN GOAL
                Null Reference (s_1 = null)
                  252:Cached goal
              Null Reference (_a = null)
                234:Cached goal
              Index Out of Bounds (_a != null, but k Out
                235:Cached goal
            Null Reference (s = null)
```

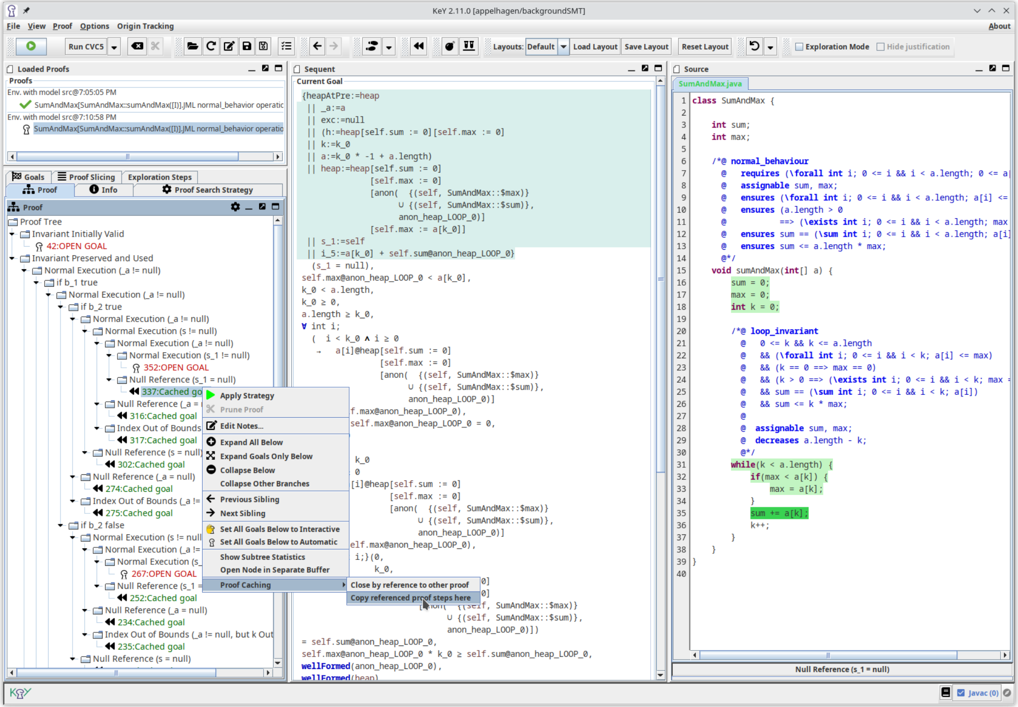
Current Goal

```
{heapAtPre:=heap
 || _a:=a
 || exc:=null
 || (h:=heap[self.sum := 0][self.max := 0]
 || k:=k_0
 || a:=k_0 * -1 + a.length)
 || heap:=heap[self.sum := 0]
             [self.max := 0]
             [anon(  {(self, SumAndMax::$max)}
                   ∪ {(self, SumAndMax::$sum)},
                   anon_heap_LOOP_0)]
             [self.max := a[k_0]]
 || s_1:=self
 || i_5:=a[k_0] + self.sum@anon_heap_LOOP_0}
  (s_1 = null),
self.max@anon_heap_LOOP_0 < a[k_0],
k_0 < a.length,
k_0 ≥ 0,
a.length ≥ k_0,
∀ int i;
  (  i < k_0 ∧ i ≥ 0
   →   a[i]@heap[self.sum := 0]
                [self.max := 0]
                [anon(  {(self, SumAndMax::$max)}
                      ∪ {(self, SumAndMax::$sum)},
                      anon_heap_LOOP_0)]
     ≤ self.max@anon_heap_LOOP_0),
k_0 = 0 → self.max@anon_heap_LOOP_0 = 0,
  k_0 >  0
→ ∃ int i;
    (  i < k_0
     ∧ i ≥ 0
     ∧   a[i]@heap[self.sum := 0]
                  [self.max := 0]
                  [anon(  {(self, SumAndMax::$max)}
                        ∪ {(self, SumAndMax::$sum)},
                        anon_heap_LOOP_0)]
       = self.max@anon_heap_LOOP_0),
  bsum{int i;}(0,
               k_0,
               a[i]@heap[self.sum := 0]
                        [self.max := 0]
                        [anon(  {(self, SumAndMax::$max)}
                              ∪ {(self, SumAndMax::$sum)},
                              anon_heap_LOOP_0)])
= self.sum@anon_heap_LOOP_0,
self.max@anon_heap_LOOP_0 * k_0 ≥ self.sum@anon_heap_LOOP_0,
wellFormed(anon_heap_LOOP_0),
wellFormed(heap)
```

SumAndMax.java

```
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a
      @   assignable sum, max;
      @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <=
      @   ensures (a.length > 0
      @           ==> (\exists int i; 0 <= i && i < a.length; max
      @   ensures sum == (\sum int i; 0 <= i && i < a.length; a[i]
      @   ensures sum <= a.length * max;
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.length
          @   && (\forall int i; 0 <= i && i < k; a[i] <= max)
          @   && (k == 0 ==> max == 0)
          @   && (k > 0 ==> (\exists int i; 0 <= i && i < k; max =
          @   && sum == (\sum int i; 0 <= i && i < k; a[i])
          @   && sum <= k * max;
          @
          @  assignable sum, max;
          @  decreases a.length - k;
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}

```

Null Reference (s_1 = null)



KeY 2.11.0 [appelhagen/backgroundSMT]

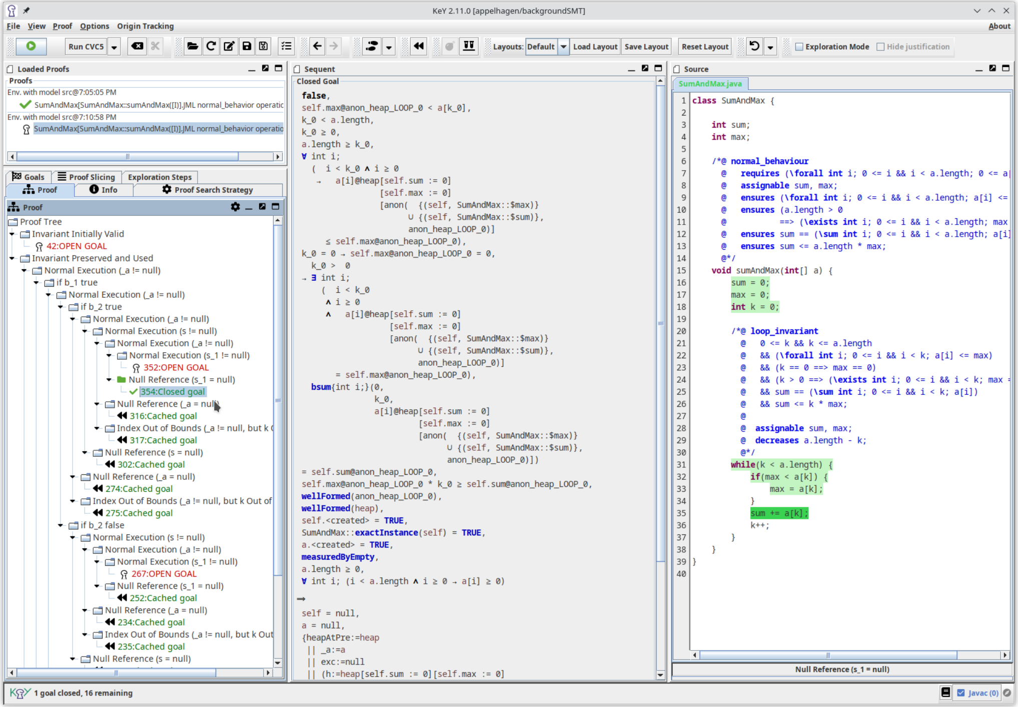
Env. with model src@7:05:05 PM
✓ SumAndMax[SumAndMax::sumAndMax([I)].JML normal_behavior operatic
Env. with model src@7:10:58 PM
SumAndMax[SumAndMax::sumAndMax([I)].JML normal_behavior operatic

```
Proof Tree
  Invariant Initially Valid
    42:OPEN GOAL
  Invariant Preserved and Used
    Normal Execution (_a != null)
      if b_1 true
        Normal Execution (_a != null)
          if b_2 true
            Normal Execution (_a != null)
              Normal Execution (s != null)
                Normal Execution (_a != null)
                  Normal Execution (s_1 != null)
                    352:OPEN GOAL
                  Null Reference (s_1 = null)
                    337:Cached go
                Null Reference (_a =
                  316:Cached goal
                Index Out of Bounds
                  317:Cached goal
              Null Reference (s = null)
                302:Cached goal
            Null Reference (_a = null)
              274:Cached goal
            Index Out of Bounds (_a !=
              275:Cached goal
          if b_2 false
            Normal Execution (s != null
              Normal Execution (_a !=
                Normal Execution (s_
                  267:OPEN GOAL
                Null Reference (s_1 =
                  252:Cached goal
              Null Reference (_a = null)
                234:Cached goal
              Index Out of Bounds (_a != null, but k Out
                235:Cached goal
            Null Reference (s = null)
```

```
{heapAtPre:=heap
 || _a:=a
 || exc:=null
 || (h:=heap[self.sum := 0][self.max := 0]
 || k:=k_0
 || a:=k_0 * -1 + a.length)
 || heap:=heap[self.sum := 0]
              [self.max := 0]
              [anon(  {(self, SumAndMax::$max)}
                    ∪ {(self, SumAndMax::$sum)},
                    anon_heap_LOOP_0)]
              [self.max := a[k_0]]
 || s_1:=self
 || i_5:=a[k_0] + self.sum@anon_heap_LOOP_0}
  (s_1 = null),
self.max@anon_heap_LOOP_0 < a[k_0],
k_0 < a.length,
k_0 ≥ 0,
a.length ≥ k_0,
∀ int i;
  (  i < k_0 ∧ i ≥ 0
   →  a[i]@heap[self.sum := 0]
               [self.max := 0]
               [anon(  {(self, SumAndMax::$max)}
                     ∪ {(self, SumAndMax::$sum)},
                     anon_heap_LOOP_0)]
f.max@anon_heap_LOOP_0),
self.max@anon_heap_LOOP_0 = 0,

 k_0
 0
[i]@heap[self.sum := 0]
        [self.max := 0]
        [anon(  {(self, SumAndMax::$max)}
              ∪ {(self, SumAndMax::$sum)},
              anon_heap_LOOP_0)]
elf.max@anon_heap_LOOP_0),
 i;}(0,
   k_0,
0]
0]
[anon(  {(self, SumAndMax::$max)}
      ∪ {(self, SumAndMax::$sum)},
      anon_heap_LOOP_0)])
= self.sum@anon_heap_LOOP_0,
self.max@anon_heap_LOOP_0 * k_0 ≥ self.sum@anon_heap_LOOP_0,
wellFormed(anon_heap_LOOP_0),
wellFormed(heap)
```

```java
class SumAndMax {

   int sum;
   int max;

   /*@ normal_behaviour
     @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a
     @   assignable sum, max;
     @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <=
     @   ensures (a.length > 0
     @           ==> (\exists int i; 0 <= i && i < a.length; max
     @   ensures sum == (\sum int i; 0 <= i && i < a.length; a[i]
     @   ensures sum <= a.length * max;
     @*/
   void sumAndMax(int[] a) {
      sum = 0;
      max = 0;
      int k = 0;

      /*@ loop_invariant
        @   0 <= k && k <= a.length
        @   && (\forall int i; 0 <= i && i < k; a[i] <= max)
        @   && (k == 0 ==> max == 0)
        @   && (k > 0 ==> (\exists int i; 0 <= i && i < k; max =
        @   && sum == (\sum int i; 0 <= i && i < k; a[i])
        @   && sum <= k * max;
        @
        @  assignable sum, max;
        @  decreases a.length - k;
        @*/
      while(k < a.length) {
          if(max < a[k]) {
              max = a[k];
          }
          sum += a[k];
          k++;
      }
   }
}
```

Null Reference (s_1 = null)



Proofs

Env. with model src@7:05:05 PM

✓ SumAndMax[SumAndMax::sumAndMax([I)].JML normal_behavior operatic

Env. with model src@7:10:58 PM

SumAndMax[SumAndMax::sumAndMax([I)].JML normal_behavior operatic

```
Proof Tree
  Invariant Initially Valid
    42:OPEN GOAL
  Invariant Preserved and Used
    Normal Execution (_a != null)
      if b_1 true
        Normal Execution (_a != null)
          if b_2 true
            Normal Execution (_a != null)
              Normal Execution (s != null)
                Normal Execution (_a != null)
                  Normal Execution (s_1 != null)
                    352:OPEN GOAL
                  Null Reference (s_1 = null)
                    ✓ 354:Closed goal
                Null Reference (_a = null)
                  316:Cached goal
                Index Out of Bounds (_a != null, but k (
                  317:Cached goal
              Null Reference (s = null)
                302:Cached goal
            Null Reference (_a = null)
              274:Cached goal
            Index Out of Bounds (_a != null, but k Out of
              275:Cached goal
          if b_2 false
            Normal Execution (s != null)
              Normal Execution (_a != null)
                Normal Execution (s_1 != null)
                  267:OPEN GOAL
                Null Reference (s_1 = null)
                  252:Cached goal
              Null Reference (_a = null)
                234:Cached goal
              Index Out of Bounds (_a != null, but k Out
                235:Cached goal
            Null Reference (s = null)
```

Closed Goal

```
false,
self.max@anon_heap_LOOP_0 < a[k_0],
k_0 < a.length,
k_0 ≥ 0,
a.length ≥ k_0,
∀ int i;
  (  i < k_0 ∧ i ≥ 0
   →   a[i]@heap[self.sum := 0]
                [self.max := 0]
                [anon(  {(self, SumAndMax::$max)}
                      ∪ {(self, SumAndMax::$sum)},
                      anon_heap_LOOP_0)]
     ≤ self.max@anon_heap_LOOP_0),
k_0 = 0 → self.max@anon_heap_LOOP_0 = 0,
  k_0 >  0
→ ∃ int i;
    (  i < k_0
     ∧ i ≥ 0
     ∧   a[i]@heap[self.sum := 0]
                  [self.max := 0]
                  [anon(  {(self, SumAndMax::$max)}
                        ∪ {(self, SumAndMax::$sum)},
                        anon_heap_LOOP_0)]
       = self.max@anon_heap_LOOP_0),
  bsum{int i;}(0,
               k_0,
               a[i]@heap[self.sum := 0]
                        [self.max := 0]
                        [anon(  {(self, SumAndMax::$max)}
                              ∪ {(self, SumAndMax::$sum)},
                              anon_heap_LOOP_0)])
= self.sum@anon_heap_LOOP_0,
self.max@anon_heap_LOOP_0 * k_0 ≥ self.sum@anon_heap_LOOP_0,
wellFormed(anon_heap_LOOP_0),
wellFormed(heap),
self.<created> = TRUE,
SumAndMax::exactInstance(self) = TRUE,
a.<created> = TRUE,
measuredByEmpty,
a.length ≥ 0,
∀ int i; (i < a.length ∧ i ≥ 0 → a[i] ≥ 0)

⟹

self = null,
a = null,
{heapAtPre:=heap
 || _a:=a
 || exc:=null
 || (h:=heap[self.sum := 0][self.max := 0]
```

```java
class SumAndMax {

    int sum;
    int max;

    /*@ normal_behaviour
      @   requires (\forall int i; 0 <= i && i < a.length; 0 <= a
      @   assignable sum, max;
      @   ensures (\forall int i; 0 <= i && i < a.length; a[i] <=
      @   ensures (a.length > 0
      @           ==> (\exists int i; 0 <= i && i < a.length; max
      @   ensures sum == (\sum int i; 0 <= i && i < a.length; a[i]
      @   ensures sum <= a.length * max;
      @*/
    void sumAndMax(int[] a) {
        sum = 0;
        max = 0;
        int k = 0;

        /*@ loop_invariant
          @   0 <= k && k <= a.length
          @   && (\forall int i; 0 <= i && i < k; a[i] <= max)
          @   && (k == 0 ==> max == 0)
          @   && (k > 0 ==> (\exists int i; 0 <= i && i < k; max =
          @   && sum == (\sum int i; 0 <= i && i < k; a[i])
          @   && sum <= k * max;
          @
          @   assignable sum, max;
          @   decreases a.length - k;
          @*/
        while(k < a.length) {
            if(max < a[k]) {
                max = a[k];
            }
            sum += a[k];
            k++;
        }
    }
}
```

Null Reference (s_1 = null)

1 goal closed, 16 remaining

Conclusion

What we have seen

- A novel way to represent (certain) proof goals as JML.
- Multiple new UI features (some already in the new 2.12 release).

- Proof Slicing ✓
- Navigation History ✓
- Undoing Interactions ✓
- Automatically run JavaC first (Alexander Weigl) ✓
- Background SMT (ongoing) ✓
- Proof Caching (ongoing) ✓