KIT
Karlsruher Institut für Technologie

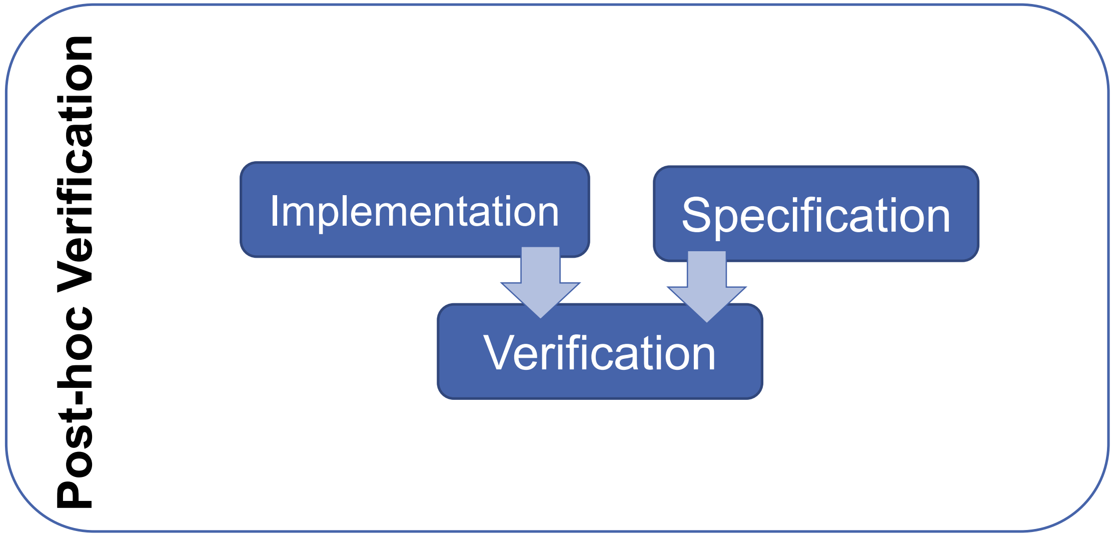
Improved Correctness-by-Construction Engineering through Successive Levels of Correctness Guarantees

KeY Symposium 2023

8 August 2023, Bergen

Tabea Bordis, Tobias Runge, Fynn Demmler, and Ina Schaefer

Motivation

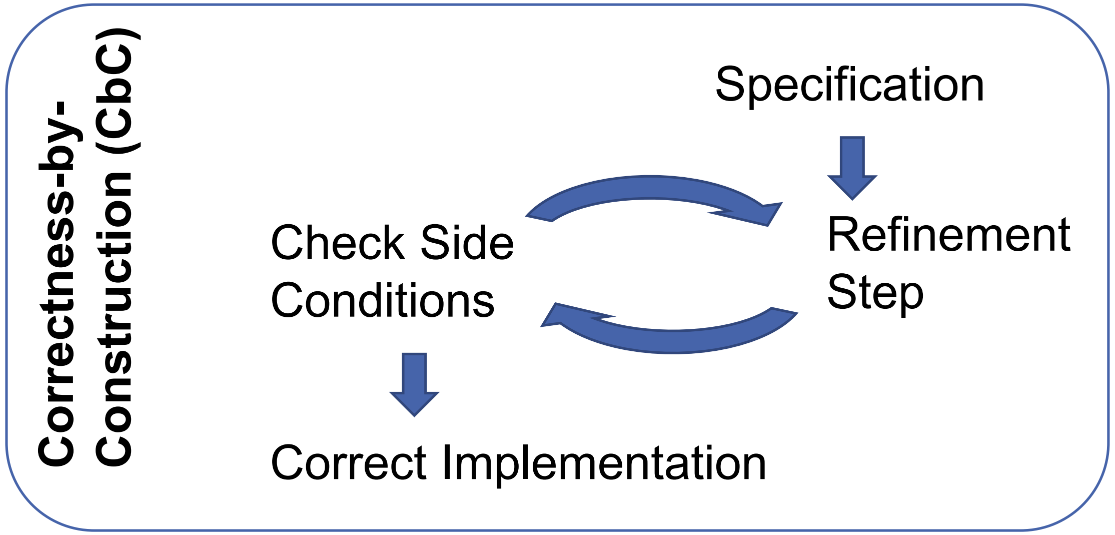


Post-hoc Verification

Implementation

Specification

Verification



Correctness-by-Construction (CbC)

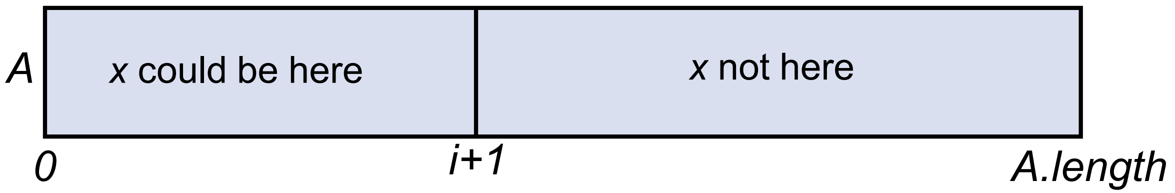
Specification

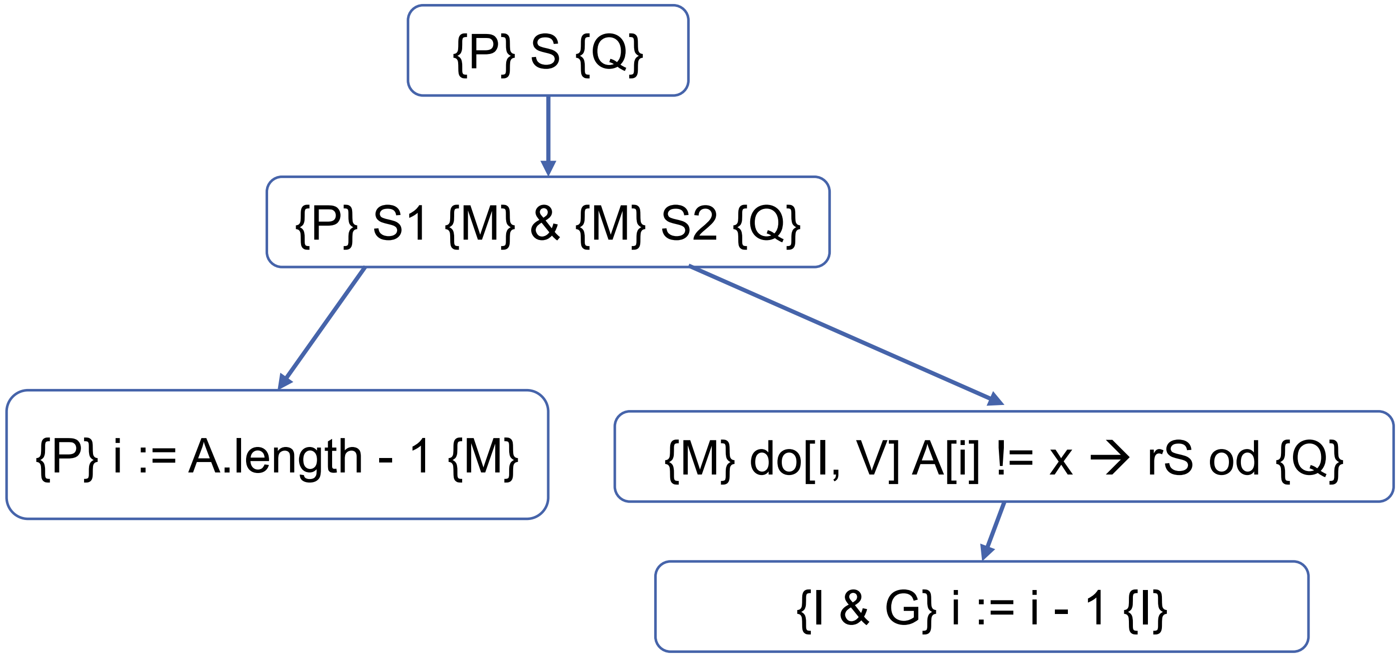
Refinement Step

Check Side Conditions

Correct Implementation

Correctness-by-Construction – Linear Search





P := A != null
Q := i >= 0 → A[i] = x
M := !app(A, x, i+1, A.length)

Refinement Rules

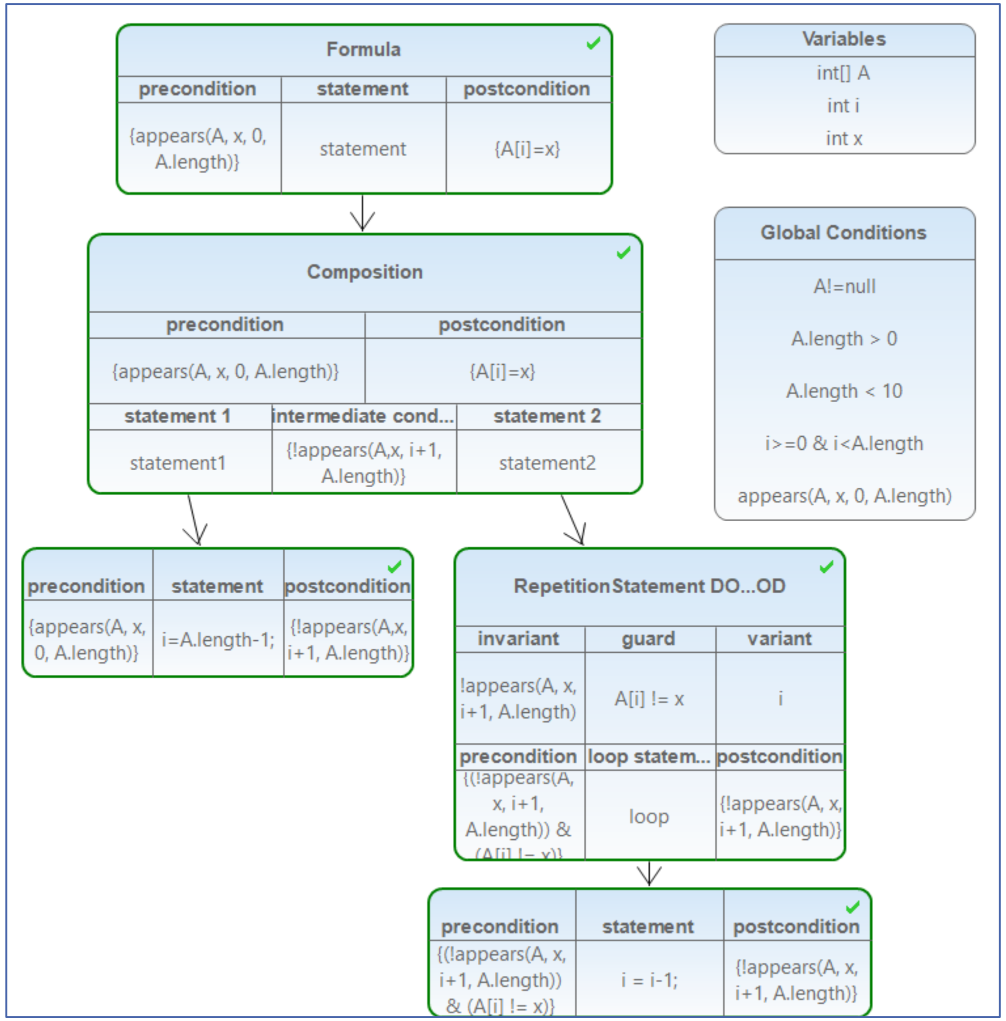
- Assignment
- Composition
- Repetition
- Selection
- Method call
- …

Composition

{P} S {Q} *can be refined to* {P} S1 ; S2 {Q} *iff there is an intermediate condition* M *such that* {P} S1 {M} *and* {M} S2 {Q}

CorC – Tool Support for CbC

- IDE for Correctness-by-Construction
- Textual and graphical editor
  - Meta-model with EMF
  - Interchangable
- KeY* used to verify the refinements
- Available at https://github.com/KIT-TVA/CorC



Formula ✓

precondition	statement	postcondition
{appears(A, x, 0, A.length)}	statement	{A[i]=x}

Variables

int[] A
int i
int x

Composition ✓

precondition	postcondition
{appears(A, x, 0, A.length)}	{A[i]=x}

statement 1	intermediate cond...	statement 2
statement1	{!appears(A,x, i+1, A.length)}	statement2

Global Conditions

A!=null
A.length > 0
A.length < 10
i>=0 & i<A.length
appears(A, x, 0, A.length)

✓

precondition	statement	postcondition
{appears(A, x, 0, A.length)}	i=A.length-1;	{!appears(A,x, i+1, A.length)}

RepetitionStatement DO...OD ✓

invariant	guard	variant
!appears(A, x, i+1, A.length)	A[i] != x	i

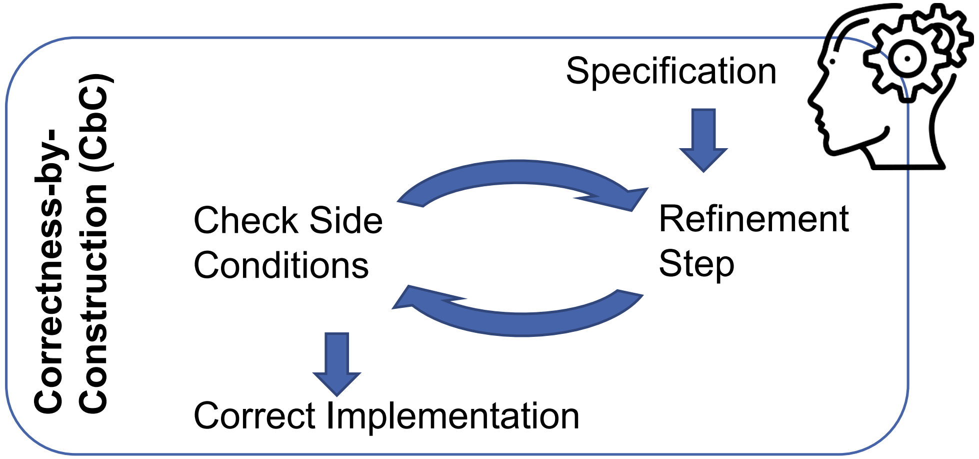
precondition	loop statem...	postcondition
{(!appears(A, x, i+1, A.length)) & (A[i] != x)}	loop	{!appears(A, x, i+1, A.length)}

✓

precondition	statement	postcondition
{(!appears(A, x, i+1, A.length)) & (A[i] != x)}	i = i-1;	{!appears(A, x, i+1, A.length)}

Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M.: Deductive Software Verification - The KeY Book: From Theory to Practice. Springer (2016)

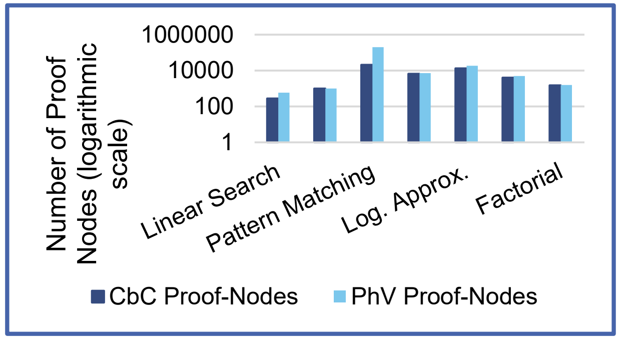
Why CbC?



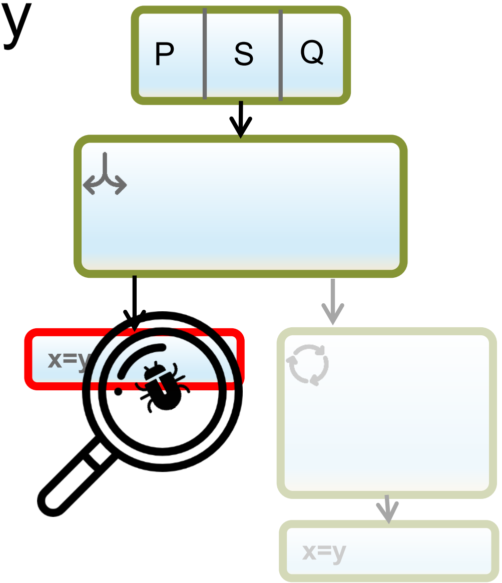
Think first rather than hacking things into correctness

Errors detected earlier

Reduced proof complexity

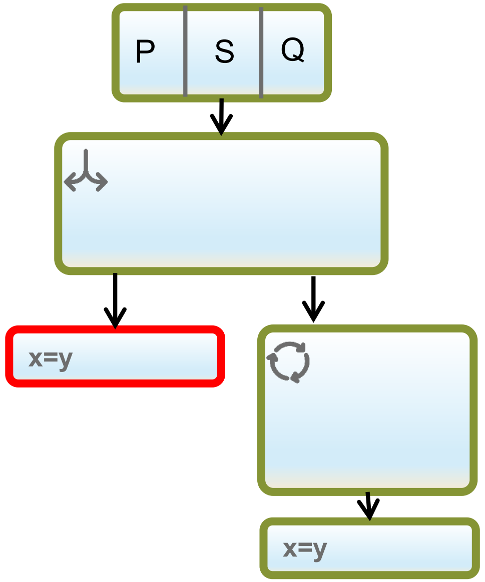


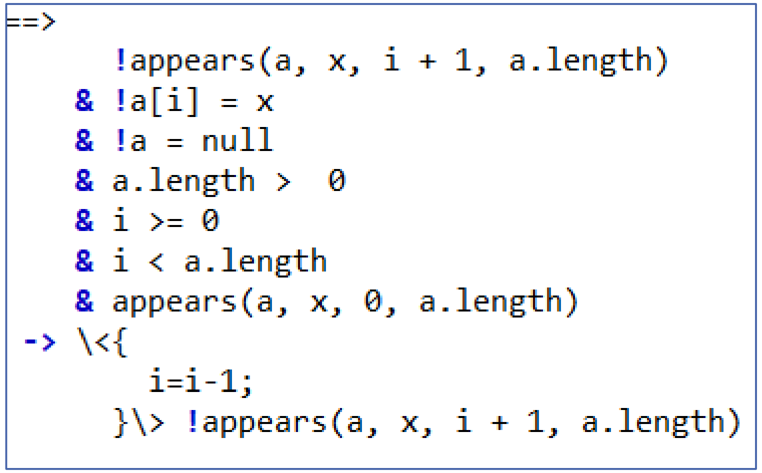
[Runge et al. 2019]



[Runge et al., 2019] T. Runge, I. Schaefer, L. Cleophas, T. Thüm, D. G. Kourie, B. W. Watson: Tool Support for Correctness-by-Construction, Proc. of the International Conference on Fundamental Approaches to Software Engineering (FASE), Springer, 2019.

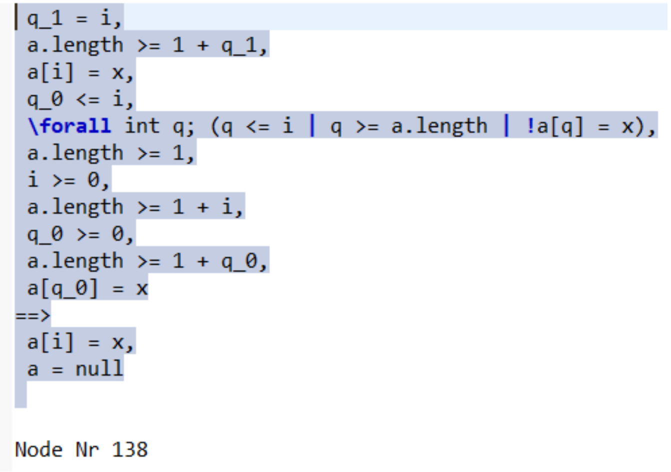
Difficulty of Verification





```
==>
      !appears(a, x, i + 1, a.length)
    & !a[i] = x
    & !a = null
    & a.length >  0
    & i >= 0
    & i < a.length
    & appears(a, x, 0, a.length)
 -> \<{
        i=i-1;
      }\> !appears(a, x, i + 1, a.length)
```

Problem: Finding the defect if the proof is not closable



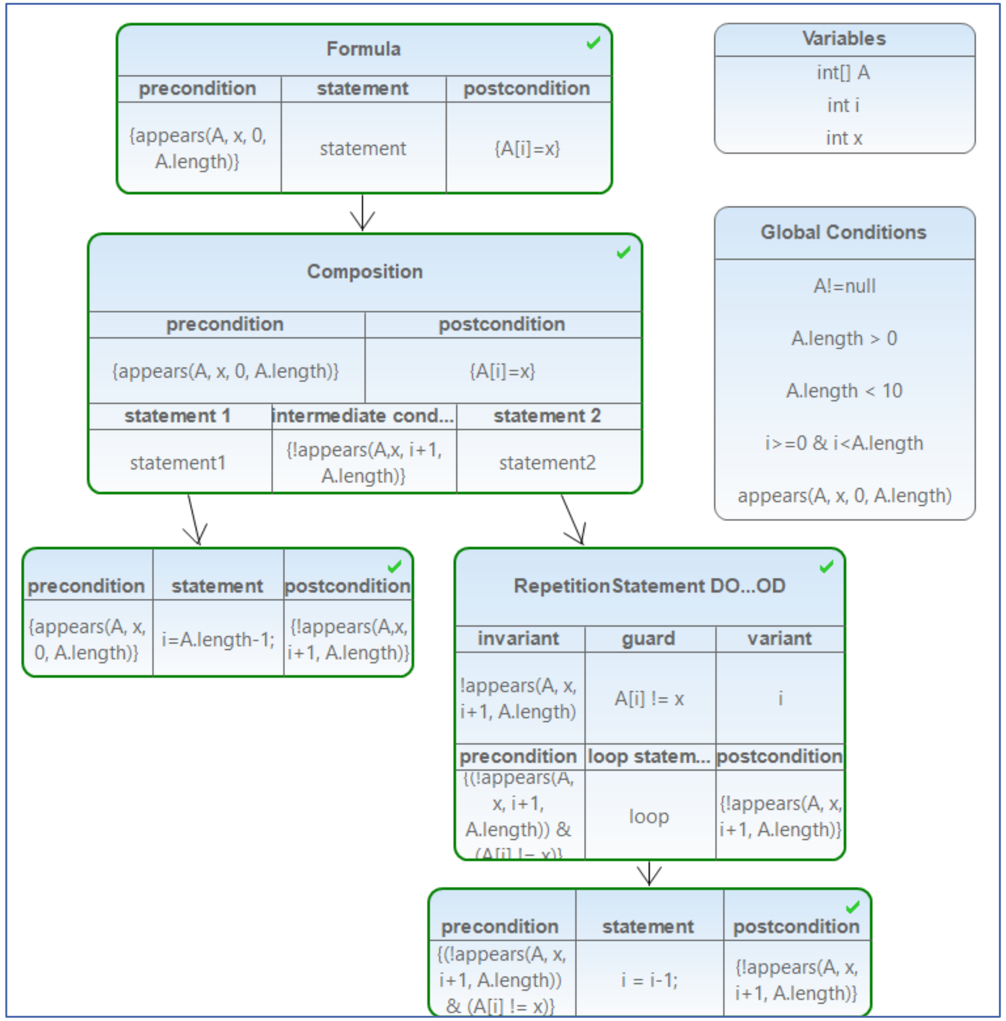
```
 q_1 = i,
 a.length >= 1 + q_1,
 a[i] = x,
 q_0 <= i,
 \forall int q; (q <= i | q >= a.length | !a[q] = x),
 a.length >= 1,
 i >= 0,
 a.length >= 1 + i,
 q_0 >= 0,
 a.length >= 1 + q_0,
 a[q_0] = x
==>
 a[i] = x,
 a = null


Node Nr 138
```

Vision

- Tool support for CbC with any knowledge in formal verification
  - Use 3 concepts from software engineering
    1. Better error messages (KeY exception handling)
    2. Generation of test cases
    3. Counter examples

- 3 stages of guarantees
  1. Specified
  2. Tested
  3. Verified

First Level: Specified



Formula ✓

precondition	statement	postcondition
{appears(A, x, 0, A.length)}	statement	{A[i]=x}

Variables

int[] A
int i
int x

Global Conditions

A!=null
A.length > 0
A.length < 10
i>=0 & i<A.length
appears(A, x, 0, A.length)

Composition ✓

precondition	postcondition
{appears(A, x, 0, A.length)}	{A[i]=x}

statement 1	intermediate cond...	statement 2
statement1	{!appears(A,x, i+1, A.length)}	statement2

✓

precondition	statement	postcondition
{appears(A, x, 0, A.length)}	i=A.length-1;	{!appears(A,x, i+1, A.length)}

RepetitionStatement DO...OD ✓

invariant	guard	variant
!appears(A, x, i+1, A.length)	A[i] != x	i

precondition	loop statem...	postcondition
{(!appears(A, x, i+1, A.length)) & (A[i] != x)}	loop	{!appears(A, x, i+1, A.length)}

✓

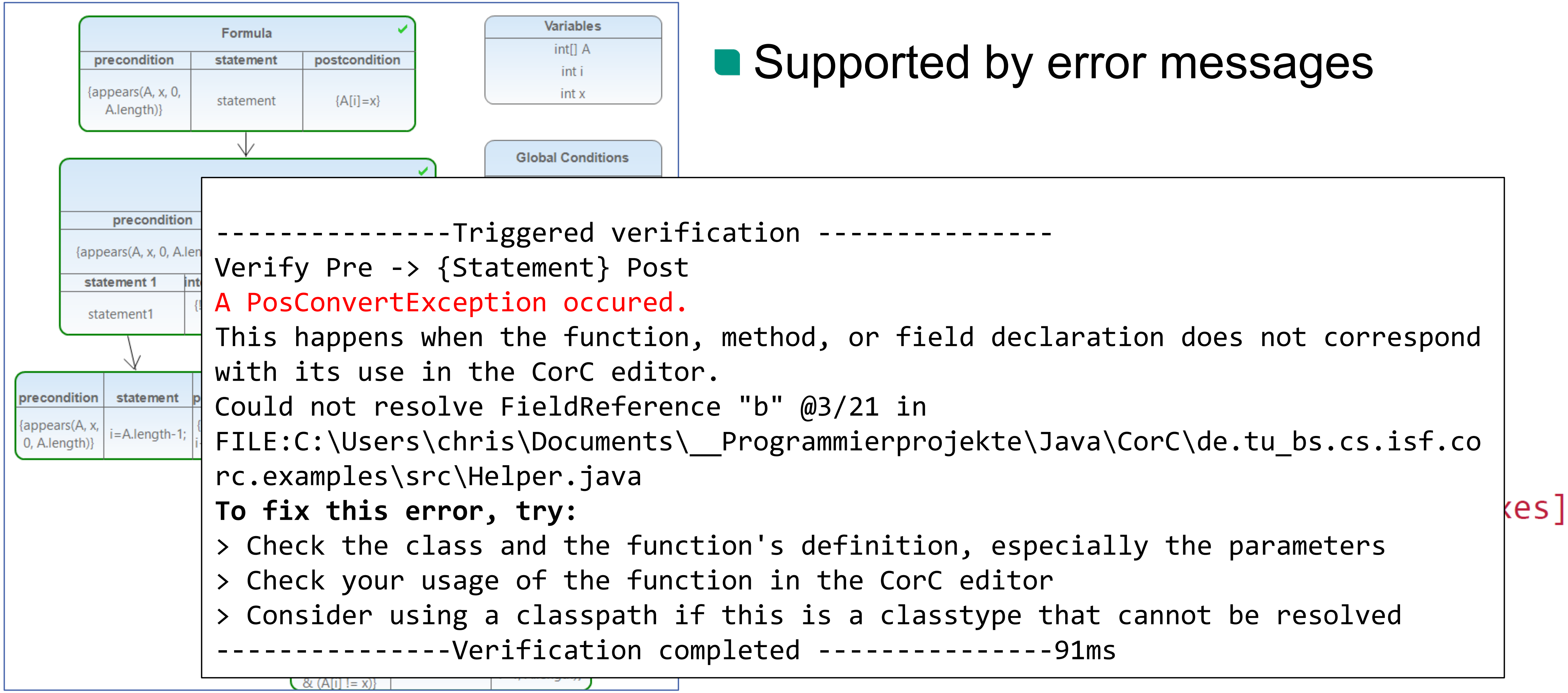
precondition	statement	postcondition
{(!appears(A, x, i+1, A.length)) & (A[i] != x)}	i = i-1;	{!appears(A, x, i+1, A.length)}

- Supported by error messages

```
An [ErrorName] Exception occurred.
This happens when [Reason for error].

[Additional Information if available]

To fix this error, try:
  > [Bullet points of potential error fixes]
  > ...
```

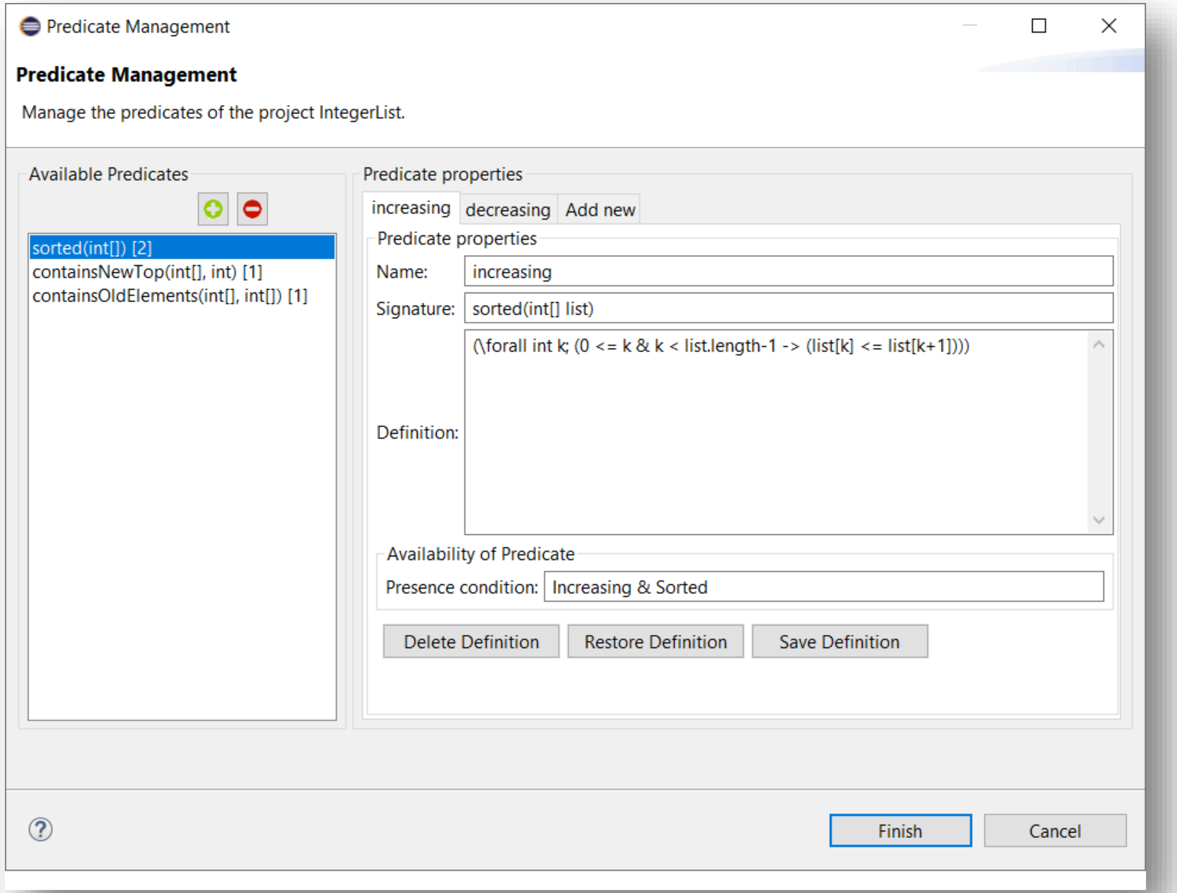
First Level: Specified

- Supported by error messages



```
---------------Triggered verification ---------------
Verify Pre -> {Statement} Post
A PosConvertException occured.
This happens when the function, method, or field declaration does not correspond
with its use in the CorC editor.
Could not resolve FieldReference "b" @3/21 in
FILE:C:\Users\chris\Documents\__Programmierprojekte\Java\CorC\de.tu_bs.cs.isf.co
rc.examples\src\Helper.java
To fix this error, try:
> Check the class and the function's definition, especially the parameters
> Check your usage of the function in the CorC editor
> Consider using a classpath if this is a classtype that cannot be resolved
---------------Verification completed ---------------91ms
```

FOL Predicate Manager



Predicate Management

Manage the predicates of the project IntegerList.

Available Predicates:

- sorted(int[]) [2]
- containsNewTop(int[], int) [1]
- containsOldElements(int[], int[]) [1]

Predicate properties

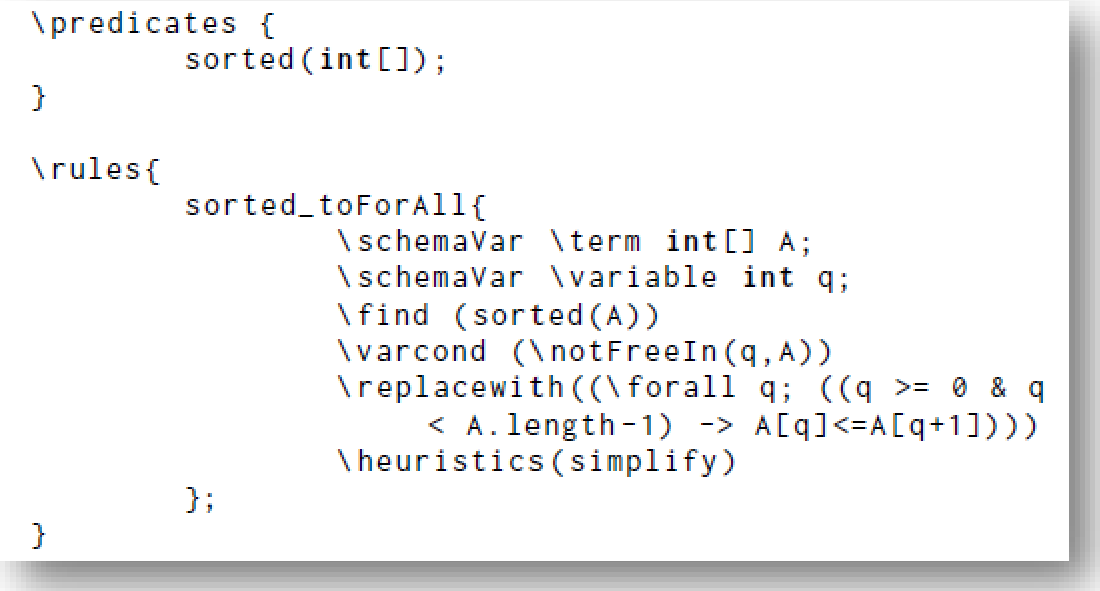
Name: increasing

Signature: sorted(int[] list)

Definition: (\forall int k; (0 <= k & k < list.length-1 -> (list[k] <= list[k+1])))

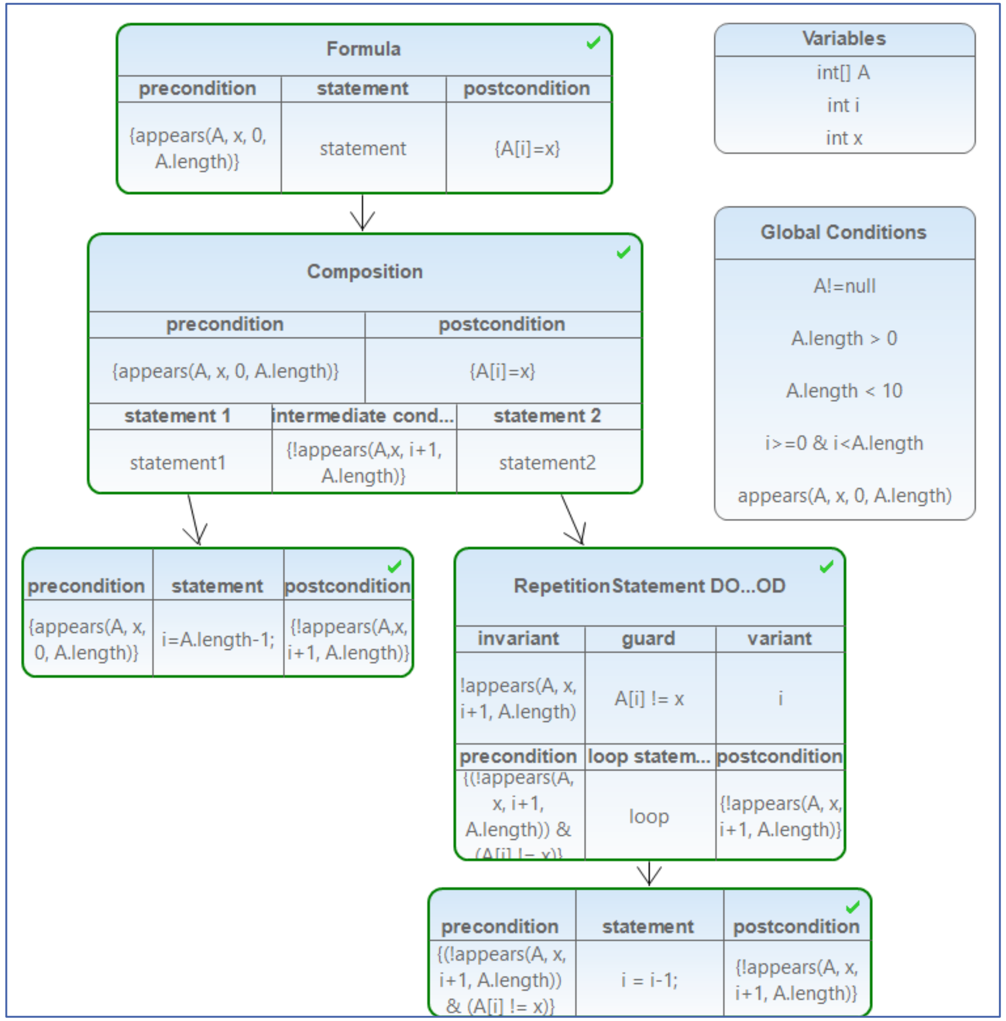
Availability of Predicate

Presence condition: Increasing & Sorted



```
\predicates {
        sorted(int[]);
}

\rules{
        sorted_toForAll{
                \schemaVar \term int[] A;
                \schemaVar \variable int q;
                \find (sorted(A))
                \varcond (\notFreeIn(q,A))
                \replacewith((\forall q; ((q >= 0 & q
                     < A.length-1) -> A[q]<=A[q+1])))
                \heuristics(simplify)
        };
}
```

Second Level: Tested



Formula ✓

precondition	statement	postcondition
{appears(A, x, 0, A.length)}	statement	{A[i]=x}

Variables

int[] A

int i

int x

Global Conditions

A!=null

A.length > 0

A.length < 10

i>=0 & i<A.length

appears(A, x, 0, A.length)

Composition ✓

precondition	postcondition
{appears(A, x, 0, A.length)}	{A[i]=x}

statement 1	intermediate cond...	statement 2
statement1	{!appears(A,x, i+1, A.length)}	statement2

✓

precondition	statement	postcondition
{appears(A, x, 0, A.length)}	i=A.length-1;	{!appears(A,x, i+1, A.length)}

RepetitionStatement DO...OD ✓

invariant	guard	variant
!appears(A, x, i+1, A.length)	A[i] != x	i

precondition	loop statem...	postcondition
{(!appears(A, x, i+1, A.length)) & (A[i] != x)}	loop	{!appears(A, x, i+1, A.length)}

✓

precondition	statement	postcondition
{(!appears(A, x, i+1, A.length)) & (A[i] != x)}	i = i-1;	{!appears(A, x, i+1, A.length)}

- Precondition for test input
- Postcondition for assertions
- Generate and execute testcase

Test Input and Assertions

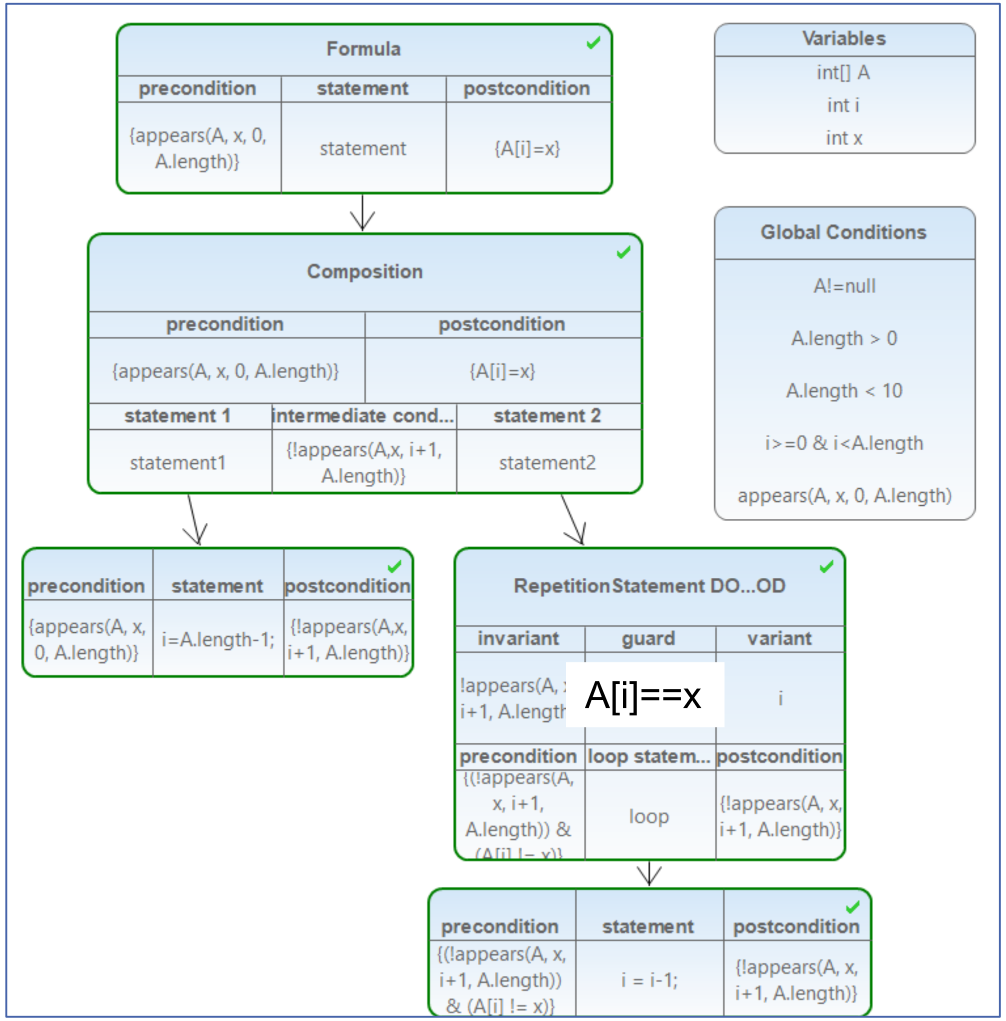
- Generate test with AAA-principle (Arrange, Act, Assert)

- Without precondition
  - Default values for primitive types
- With precondition
  - Using SMT-Solver to find an assignment that fulfills the precondition

- Postcondition is used as test oracle

```
@Test
Public void exampleTest(ITestContext context)
{
  //Arrange
  generatedClass = new GeneratedClass();
  int x = 0;

  //Act
  int result = generatedClass.example(x);

  //Assert
  Assert.assertTrue(x%2 == result);
}
```

Example of a Test Report



Formula ✓

precondition	statement	postcondition
{appears(A, x, 0, A.length)}	statement	{A[i]=x}

Variables

int[] A
int i
int x

Composition ✓

precondition	postcondition
{appears(A, x, 0, A.length)}	{A[i]=x}

statement 1	intermediate cond...	statement 2
statement1	{!appears(A,x, i+1, A.length)}	statement2

Global Conditions

A!=null
A.length > 0
A.length < 10
i>=0 & i<A.length
appears(A, x, 0, A.length)

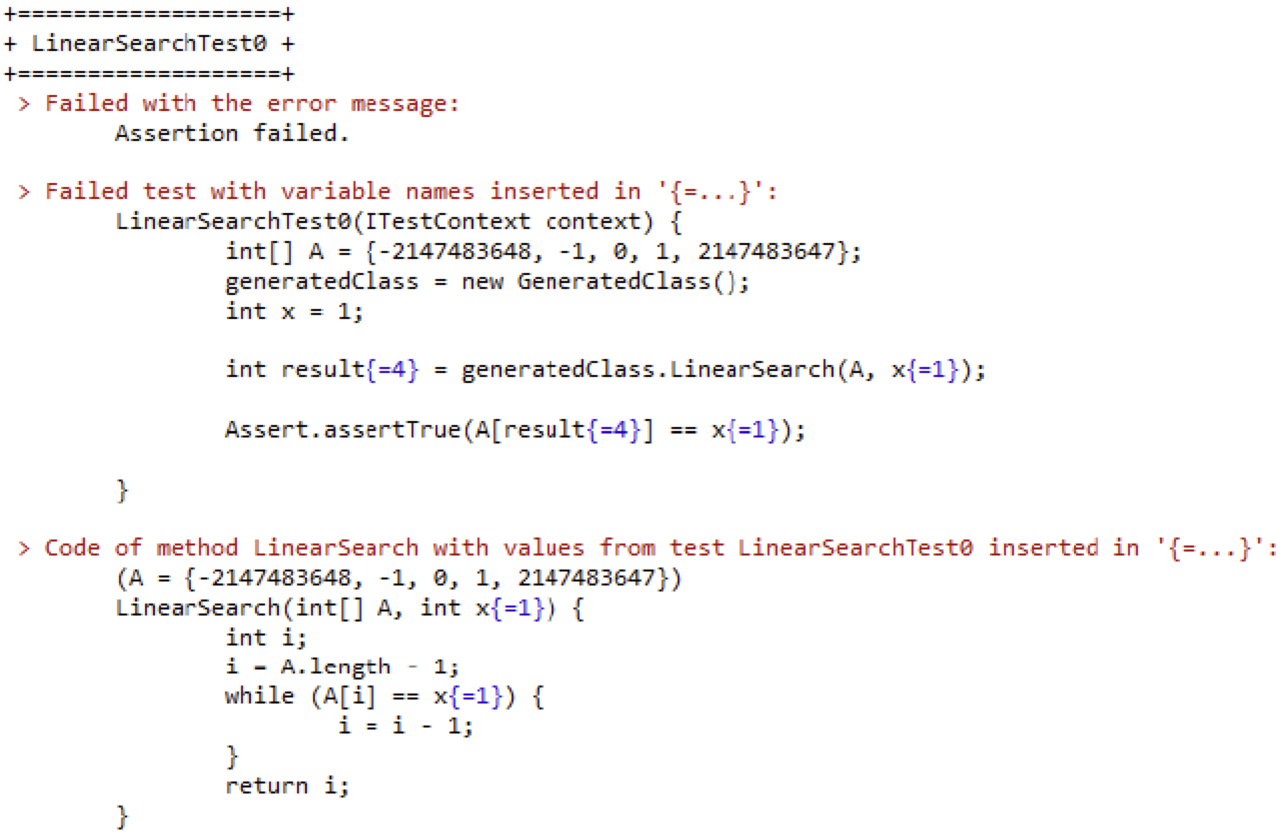
precondition	statement	postcondition
{appears(A, x, 0, A.length)}	i=A.length-1;	{!appears(A,x, i+1, A.length)}

RepetitionStatement DO...OD ✓

invariant	guard	variant
!appears(A, x, i+1, A.length	A[i]==x	i

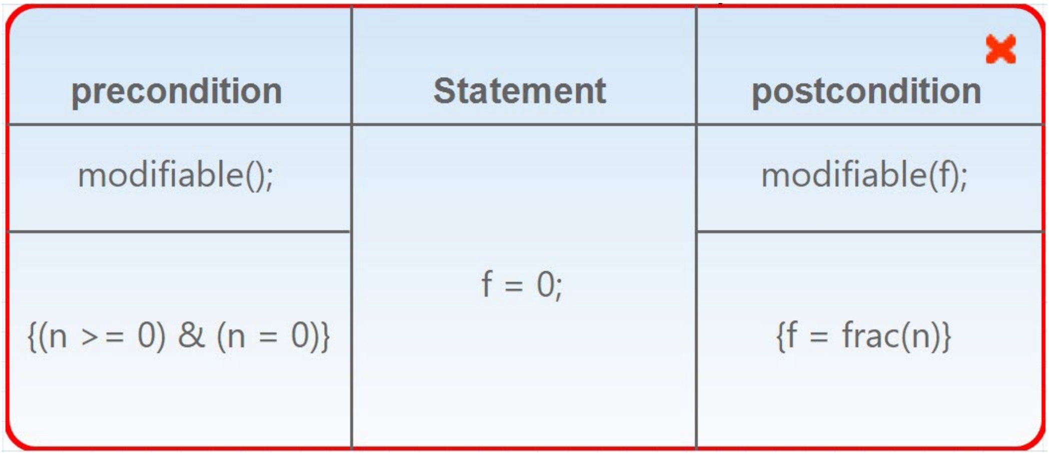
precondition	loop statem...	postcondition
{(!appears(A, x, i+1, A.length)) & (A[i] != x)}	loop	{!appears(A, x, i+1, A.length)}

precondition	statement	postcondition
{(!appears(A, x, i+1, A.length)) & (A[i] != x)}	i = i-1;	{!appears(A, x, i+1, A.length)}



```
+===================+
+ LinearSearchTest0 +
+===================+
 > Failed with the error message:
        Assertion failed.

 > Failed test with variable names inserted in '{=...}':
        LinearSearchTest0(ITestContext context) {
                int[] A = {-2147483648, -1, 0, 1, 2147483647};
                generatedClass = new GeneratedClass();
                int x = 1;

                int result{=4} = generatedClass.LinearSearch(A, x{=1});

                Assert.assertTrue(A[result{=4}] == x{=1});

        }

 > Code of method LinearSearch with values from test LinearSearchTest0 inserted in '{=...}':
        (A = {-2147483648, -1, 0, 1, 2147483647})
        LinearSearch(int[] A, int x{=1}) {
                int i;
                i = A.length - 1;
                while (A[i] == x{=1}) {
                        i = i - 1;
                }
                return i;
        }
```

Third Level: Verified



precondition	Statement	postcondition ✖
modifiable();	f = 0;	modifiable(f);
{(n >= 0) & (n = 0)}		{f = frac(n)}

Verification with KeY

And supported by counter example generation

Proof goal:
$P \rightarrow \{S\}Q$

P:= n==0
S:= f=0;

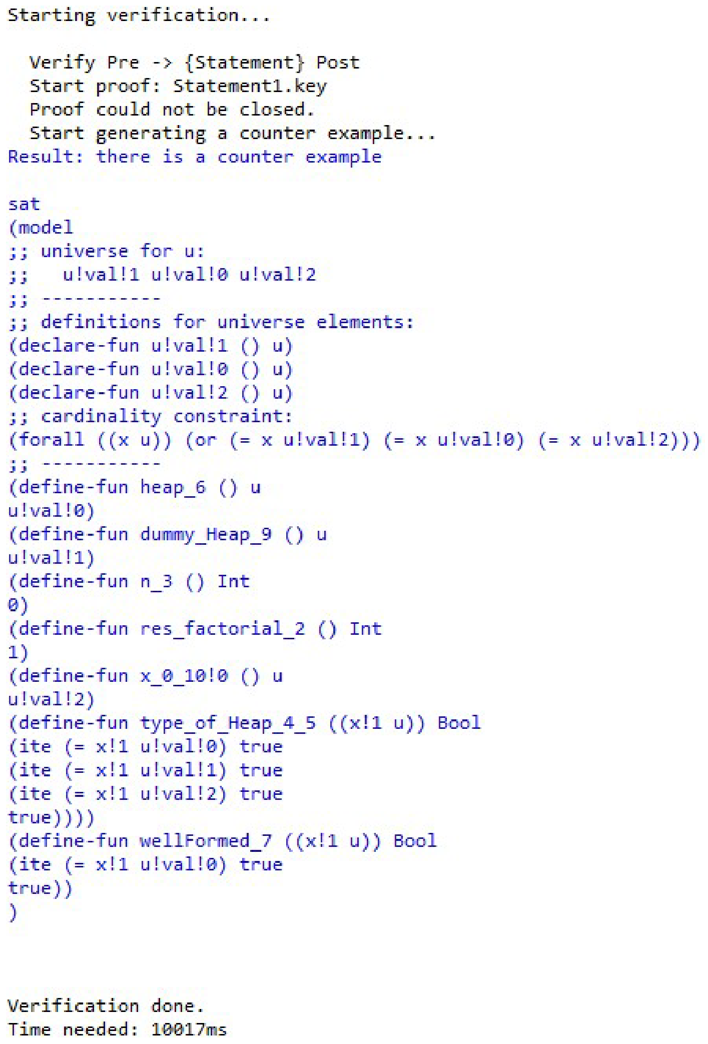
Values:
n:=0
f:=0

Is satisfiable?
Q := f==frac(n)

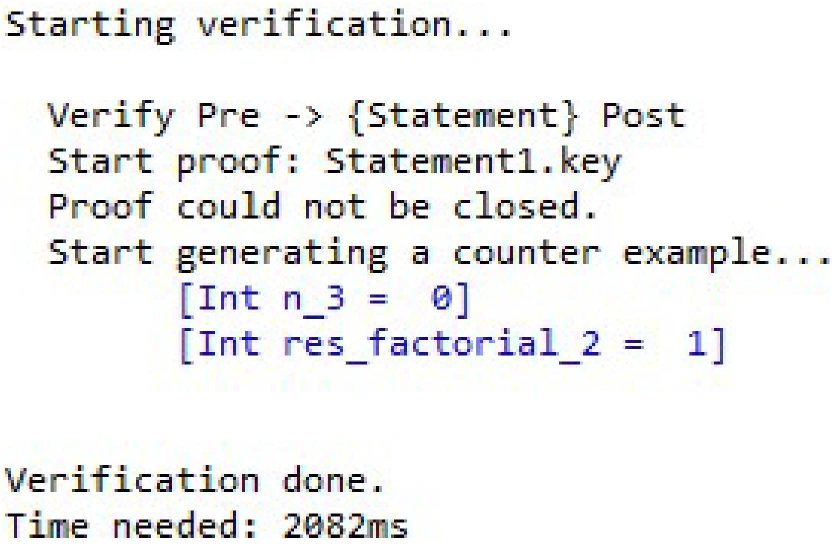
Inserted:
0==1

Conclusion:
f must be 1

Update of the Counter Example Output



```
Starting verification...

  Verify Pre -> {Statement} Post
  Start proof: Statement1.key
  Proof could not be closed.
  Start generating a counter example...
Result: there is a counter example

sat
(model
;; universe for u:
;;   u!val!1 u!val!0 u!val!2
;; -----------
;; definitions for universe elements:
(declare-fun u!val!1 () u)
(declare-fun u!val!0 () u)
(declare-fun u!val!2 () u)
;; cardinality constraint:
(forall ((x u)) (or (= x u!val!1) (= x u!val!0) (= x u!val!2)))
;; -----------
(define-fun heap_6 () u
u!val!0)
(define-fun dummy_Heap_9 () u
u!val!1)
(define-fun n_3 () Int
0)
(define-fun res_factorial_2 () Int
1)
(define-fun x_0_10!0 () u
u!val!2)
(define-fun type_of_Heap_4_5 ((x!1 u)) Bool
(ite (= x!1 u!val!0) true
(ite (= x!1 u!val!1) true
(ite (= x!1 u!val!2) true
true))))
(define-fun wellFormed_7 ((x!1 u)) Bool
(ite (= x!1 u!val!0) true
true))
)


Verification done.
Time needed: 10017ms
```



```
Starting verification...

  Verify Pre -> {Statement} Post
  Start proof: Statement1.key
  Proof could not be closed.
  Start generating a counter example...
        [Int n_3 =  0]
        [Int res_factorial_2 =  1]


Verification done.
Time needed: 2082ms
```

Expert Study

- Research question:
  - Do the participants perceive the new features as a useful extension to CorC?

- Two user studies with five experts each
  - Debug CorC programs with and without the usabiliy features
  - Interview regarding the benefits of the new features

Summary of the Study Results

- Error messages
  - All participants found it useful
  - Especially the rust-like component of giving troubleshooting tips

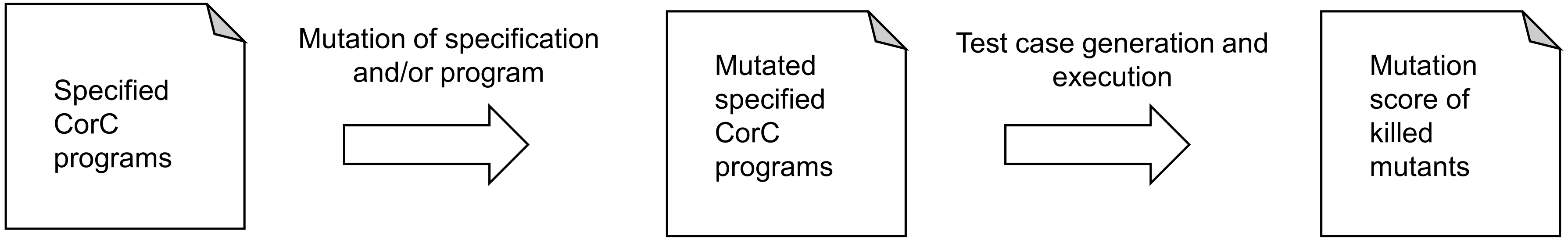
- Test case generation
  - All participants found it useful
  - Final test report is readable
  - “Concrete values facilitate error detection”

Summary of the Study Results

- Counter examples
  - Most agree that the counterexample generation is a useful addition
  - “The counter example syntax is hard-to-read and hinders comprehension”

ToDo: Mutation-based Evaluation

- How many bugs can we find with test cases?



Specified CorC programs

Mutation of specification and/or program

Mutated specified CorC programs

Test case generation and execution

Mutation score of killed mutants

Conclusion

- CbC is a good way to create correct programs
  - But has entry threshold

- Easier entry and better user experience through
  - Error messages
  - Test cases
  - Counter examples