CONSTRUCTOR INSTITUTE

Chair of Software Engineering

Roaming the proof-test border

Bertrand Meyer, Li Huang

Constructor Institute, Schaffhausen (CH)

(Work with Manuel Oriol and Ilgiz Mustafin)

WG2.3, Trento, 12 *October* 2023

Overall idea

Using a modern SMT-based program prover to derive

counter-examples

for both correct and incorrect programs, hence:

- (1) **Failing tests**
- (2) **Better counter-examples**
- (3) **Full-coverage test suites**
- (4) **Automatically generated program fixes**

Roaming the proof-test border

Bertrand Meyer, Li Huang

Constructor Institute, Schaffhausen (CH)

(Work with Manuel Oriol and Ilgiz Mustafin)

WG 2.3, Trento, 12 *October* 2023

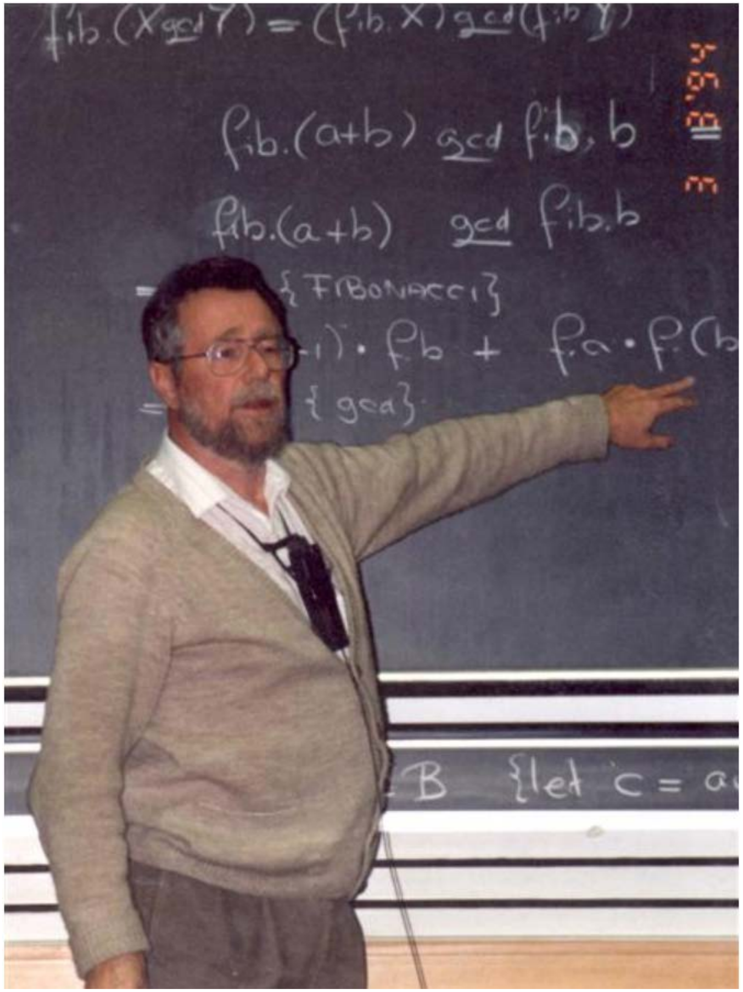
Part 1 (presented by Bertrand Meyer)

Tests and proofs?

The Battle of San Romano (1432) by Paolo Uccello

Edsger Dijkstra (1970)

Program testing can be used to show the presence of bugs, but never to show their absence!



A key role of tests: the regression test suite

Consider a correct program

We shouldn't need to test it any more

But: we do want a test suite for future evolution, to spot possible **regressions**

Tests and proofs?

The Battle of San Romano (1432) by Paolo Uccello

Tests and proofs

Rubens: Allegory of the Blessings of Peace (of Westphalia)

Tests and proofs: duality

	Proof	Test
Success		
Failure		

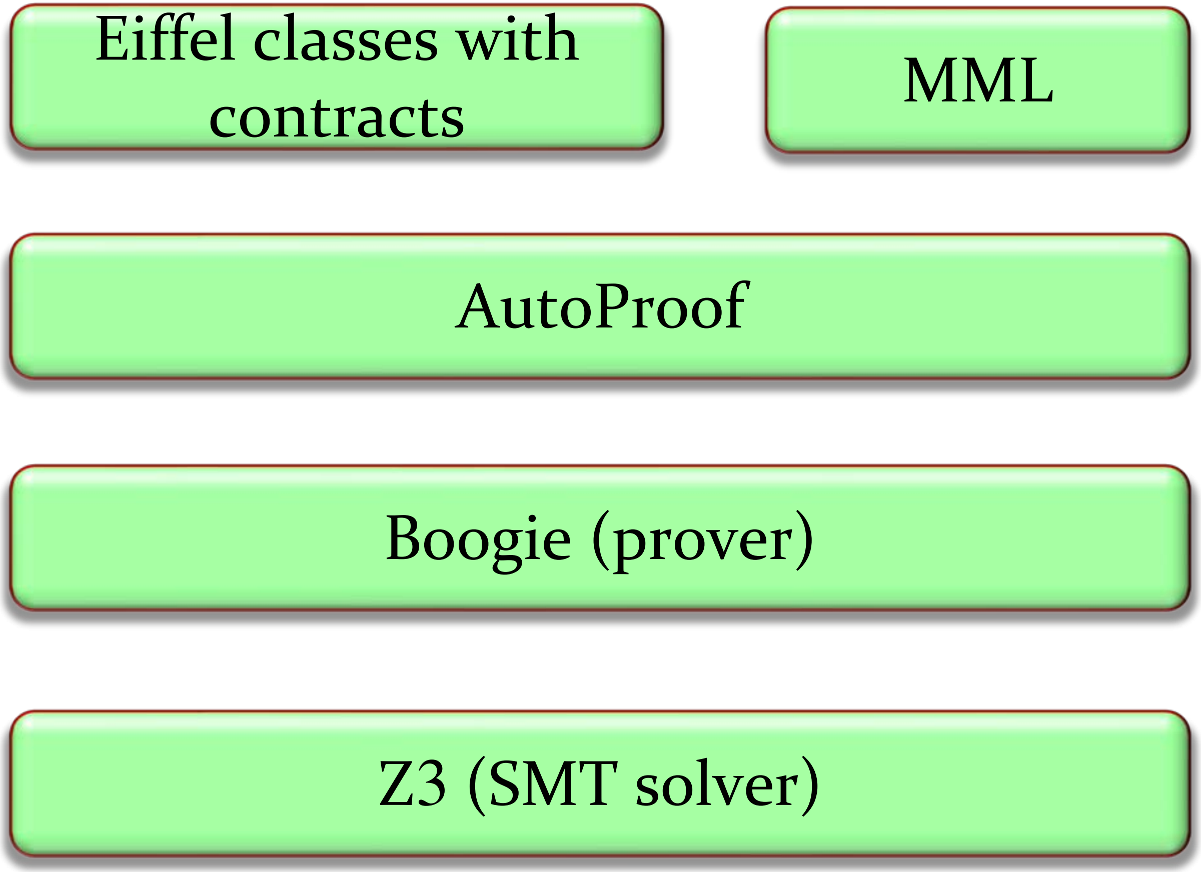
Counterexamples

Some modern provers use an SMT solver:

- Attempt to prove program correct by trying to find a counterexample
- Normally, we hope to find none, and then declare victory

- If the proof attempt **fails**, it yields a counterexample
- This counterexample is a **test** for the corresponding path

AutoProof technology stack



Eiffel classes with contracts

MML

AutoProof

Boogie (prover)

Z3 (SMT solver)

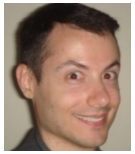
Bernd Schoeller

Rustan Leino

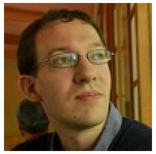
Leonardo de Moura

Nikolaj Bjørner

Nadia Polikarpova



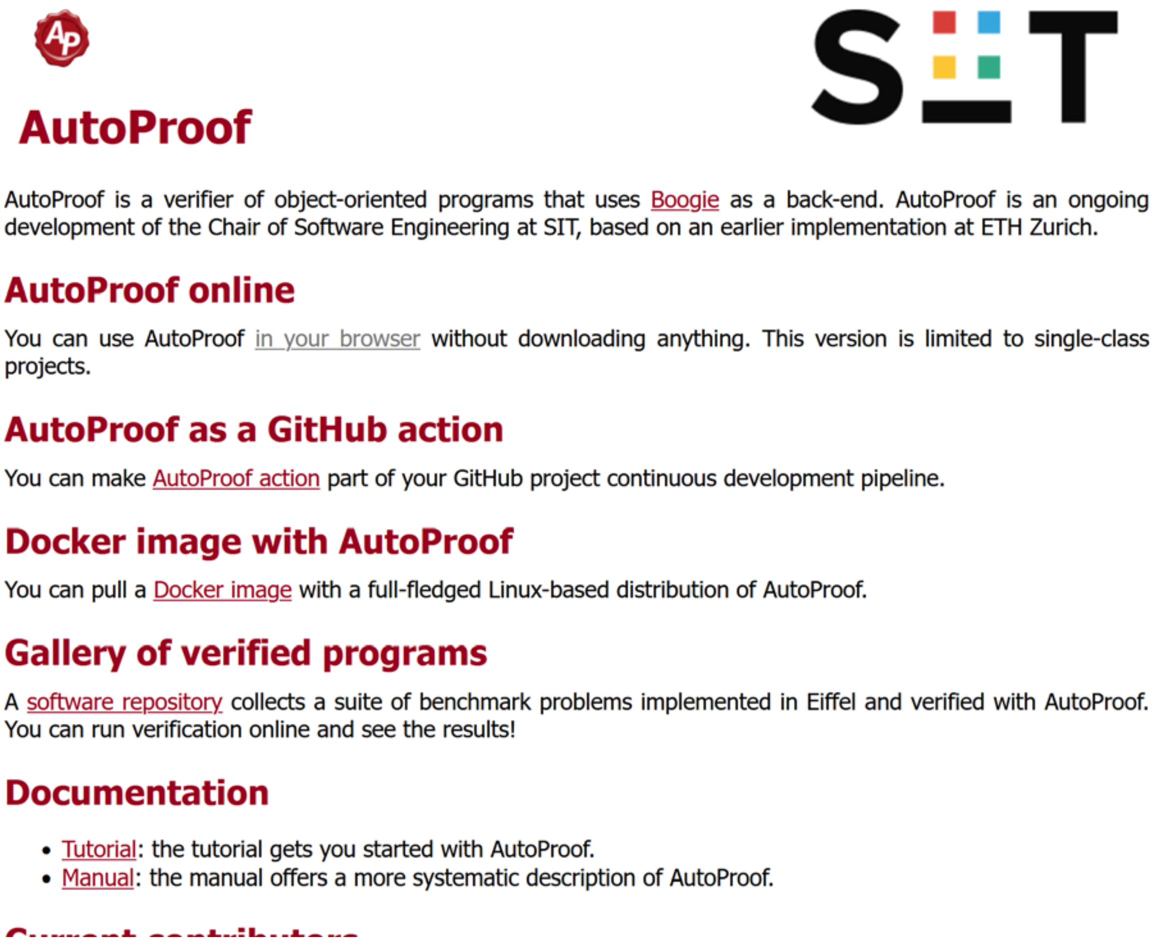
Carlo Furia



Julian Tschannen

Try AutoProof

http://autoproof.sit.org



AutoProof

AutoProof is a verifier of object-oriented programs that uses Boogie as a back-end. AutoProof is an ongoing development of the Chair of Software Engineering at SIT, based on an earlier implementation at ETH Zurich.

AutoProof online

You can use AutoProof in your browser without downloading anything. This version is limited to single-class projects.

AutoProof as a GitHub action

You can make AutoProof action part of your GitHub project continuous development pipeline.

Docker image with AutoProof

You can pull a Docker image with a full-fledged Linux-based distribution of AutoProof.

Gallery of verified programs

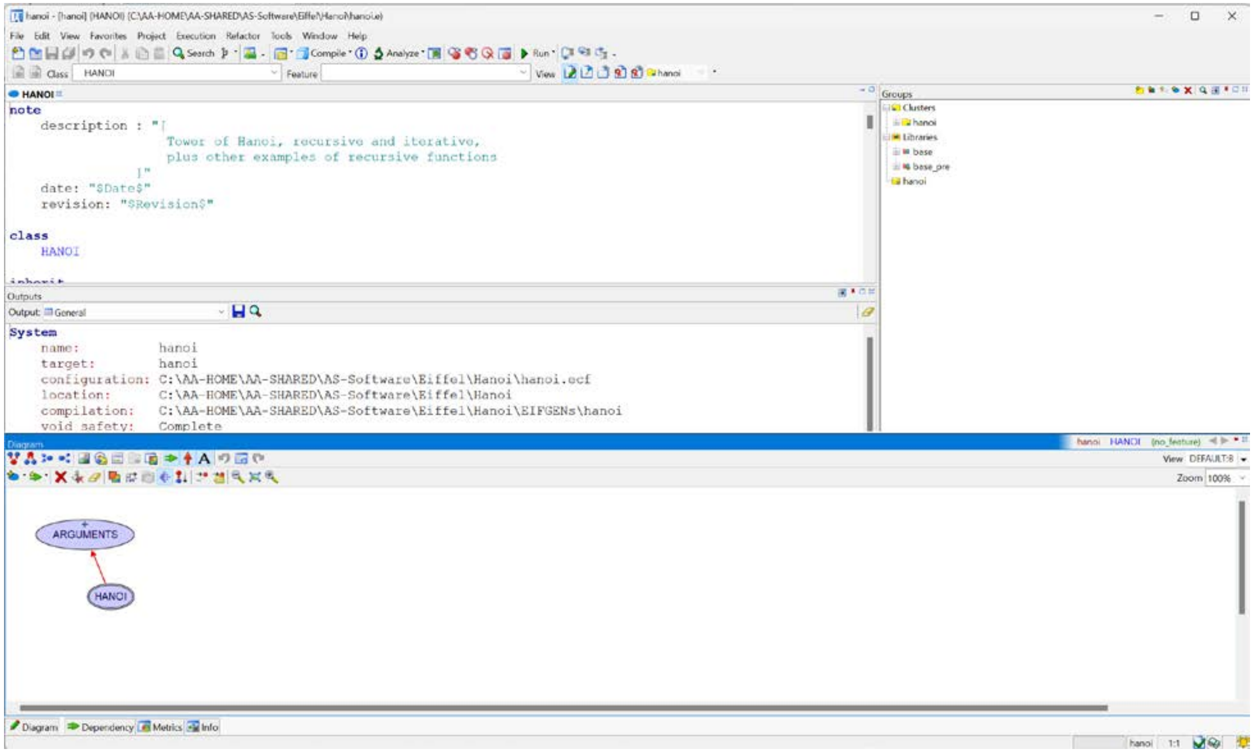
A software repository collects a suite of benchmark problems implemented in Eiffel and verified with AutoProof. You can run verification online and see the results!

Documentation

- Tutorial: the tutorial gets you started with AutoProof.
- Manual: the manual offers a more systematic description of AutoProof.

Current contributors

Reminder: Eiffel technology



```
note
    description : "[
                Tower of Hanoi, recursive and iterative,
                plus other examples of recursive functions
            ]"
    date: "$Date$"
    revision: "$Revision$"

class
    HANOI

inherit
```

```
System
    name:          hanoi
    target:        hanoi
    configuration: C:\AA-HOME\AA-SHARED\AS-Software\Eiffel\Hanoi\hanoi.ecf
    location:      C:\AA-HOME\AA-SHARED\AS-Software\Eiffel\Hanoi
    compilation:   C:\AA-HOME\AA-SHARED\AS-Software\Eiffel\Hanoi\EIFGENs\hanoi
    void safety:   Complete
```

Software development approach based on methodology, language and tools

Encompasses entire lifecycle

Built around principles: Design by Contract™, Open-Closed, Command-Query Separation, Single-Choice…

Full and uncompromising application of object technology

Supporting environment: EiffelStudio – open-source and commercial versions

A failed proof

	Proof	Test
Success		
Failure		

A failed pr

eyer, Oriol,
3

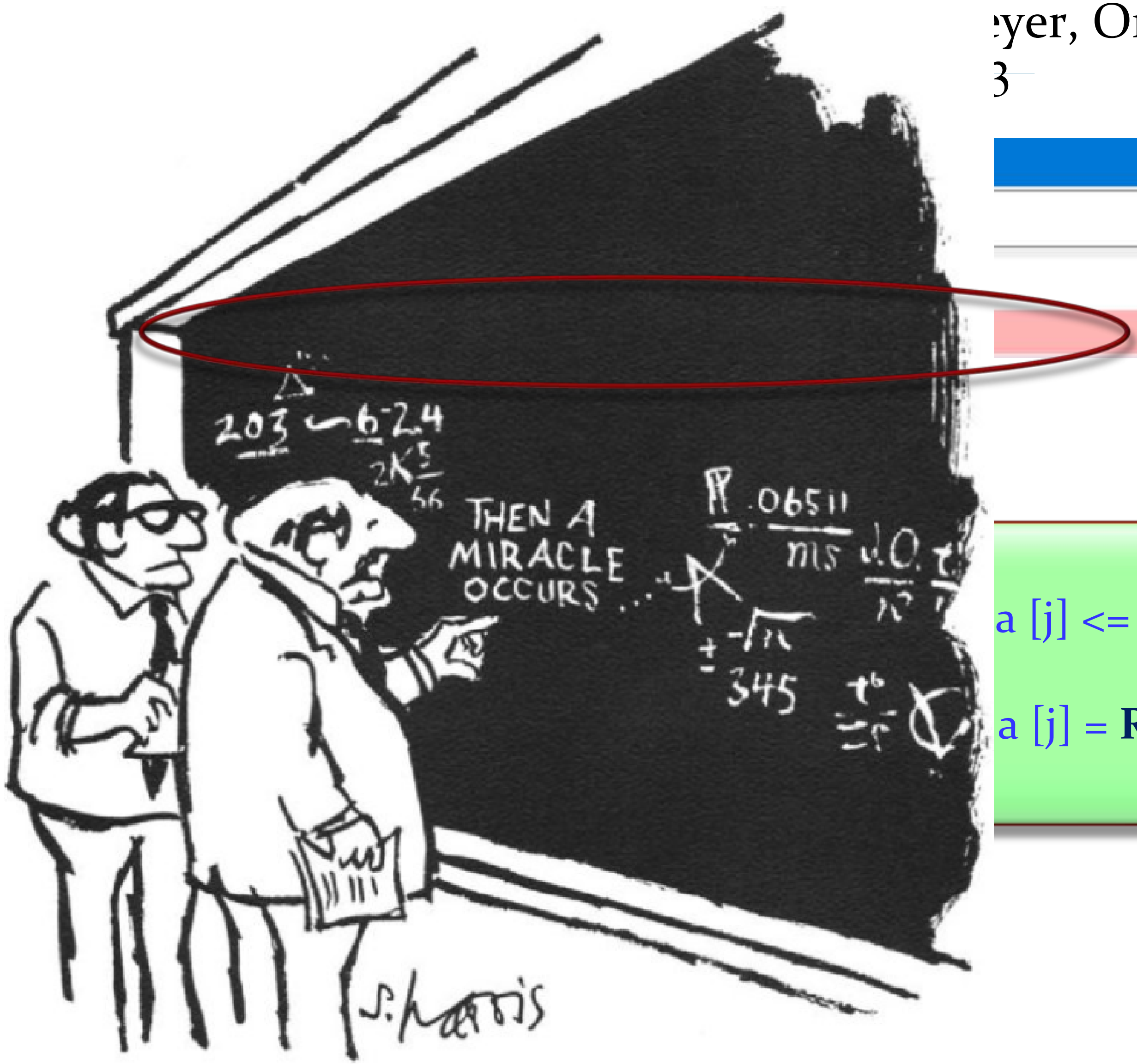
AutoProof

Verify 2 Suc

Class

MAX

```
max (a: ARRAY
    require a.c
    local i: INT
    do
        from Res
            2 ≤ i a
            ∀ : 1 |.
            ∃ j: 1 |
        until i =a.
            if a
        end
        i := i + 1
        variant a.
end
```



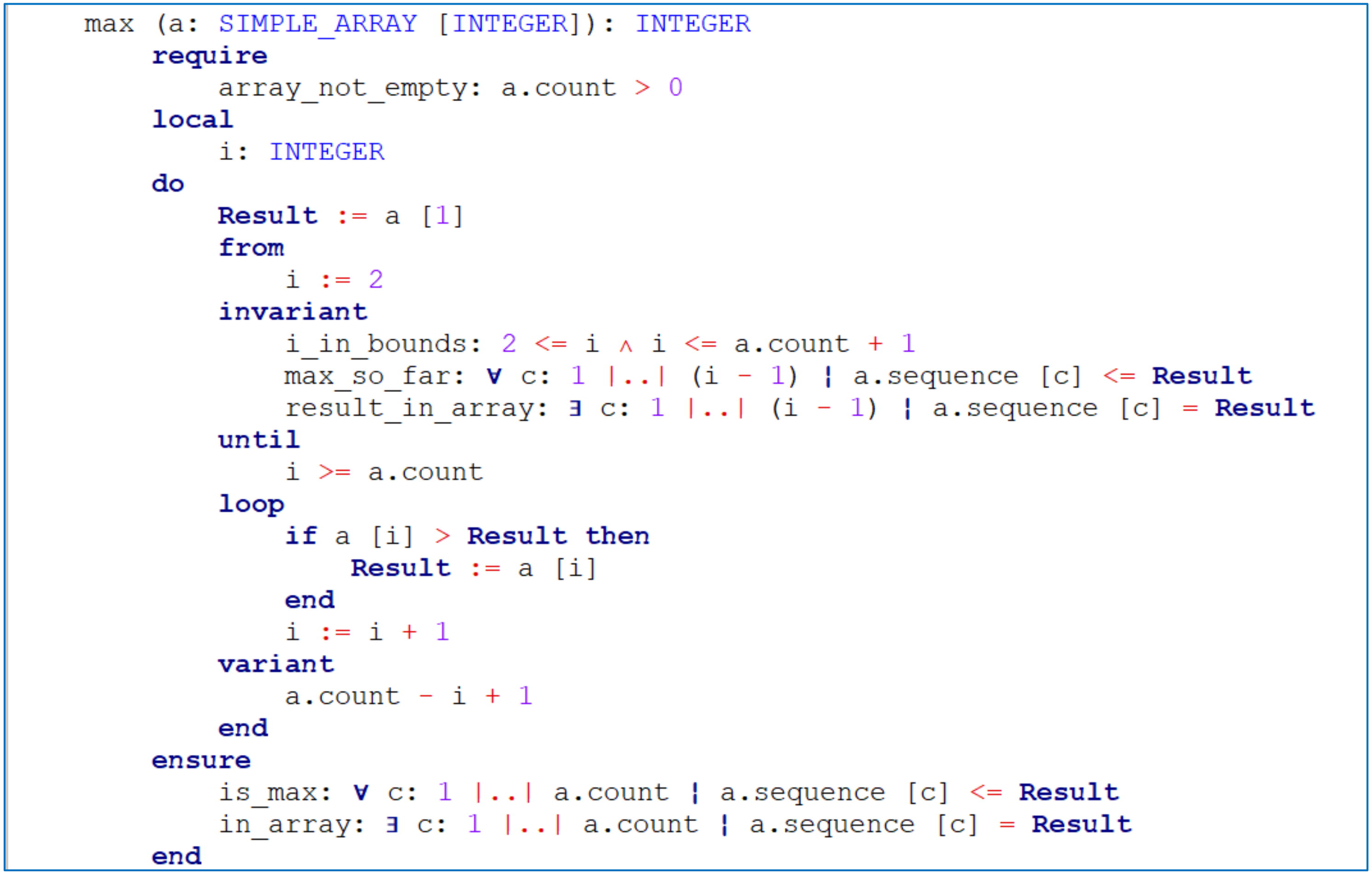
"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

a [j] <= Result

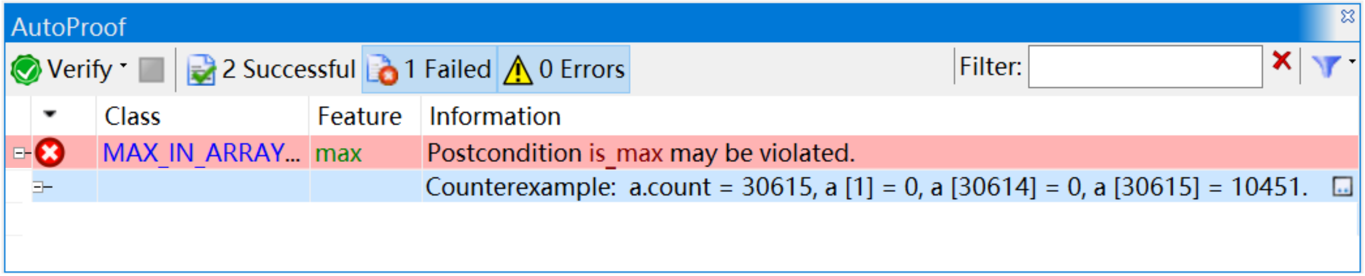
a [j] = Result

Part 2 (presented by Li Huang)

Counterexample generation

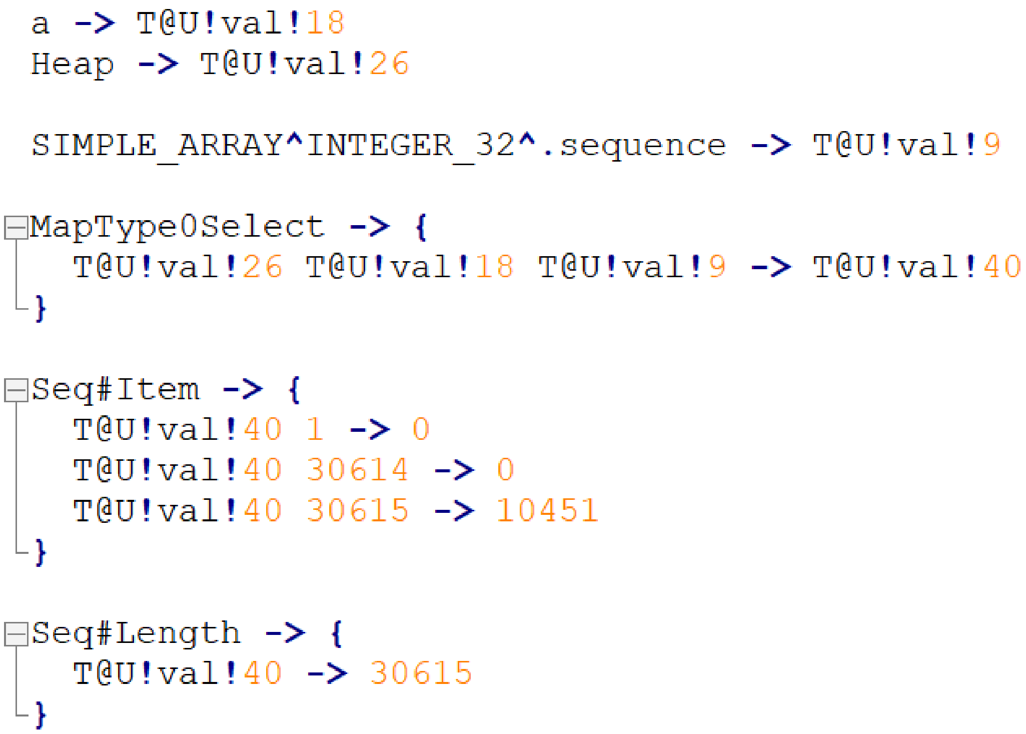


```
    max (a: SIMPLE_ARRAY [INTEGER]): INTEGER
        require
            array_not_empty: a.count > 0
        local
            i: INTEGER
        do
            Result := a [1]
            from
                i := 2
            invariant
                i_in_bounds: 2 <= i ∧ i <= a.count + 1
                max_so_far: ∀ c: 1 |..| (i - 1) ¦ a.sequence [c] <= Result
                result_in_array: ∃ c: 1 |..| (i - 1) ¦ a.sequence [c] = Result
            until
                i >= a.count
            loop
                if a [i] > Result then
                    Result := a [i]
                end
                i := i + 1
            variant
                a.count - i + 1
            end
        ensure
            is_max: ∀ c: 1 |..| a.count ¦ a.sequence [c] <= Result
            in_array: ∃ c: 1 |..| a.count ¦ a.sequence [c] = Result
        end
```



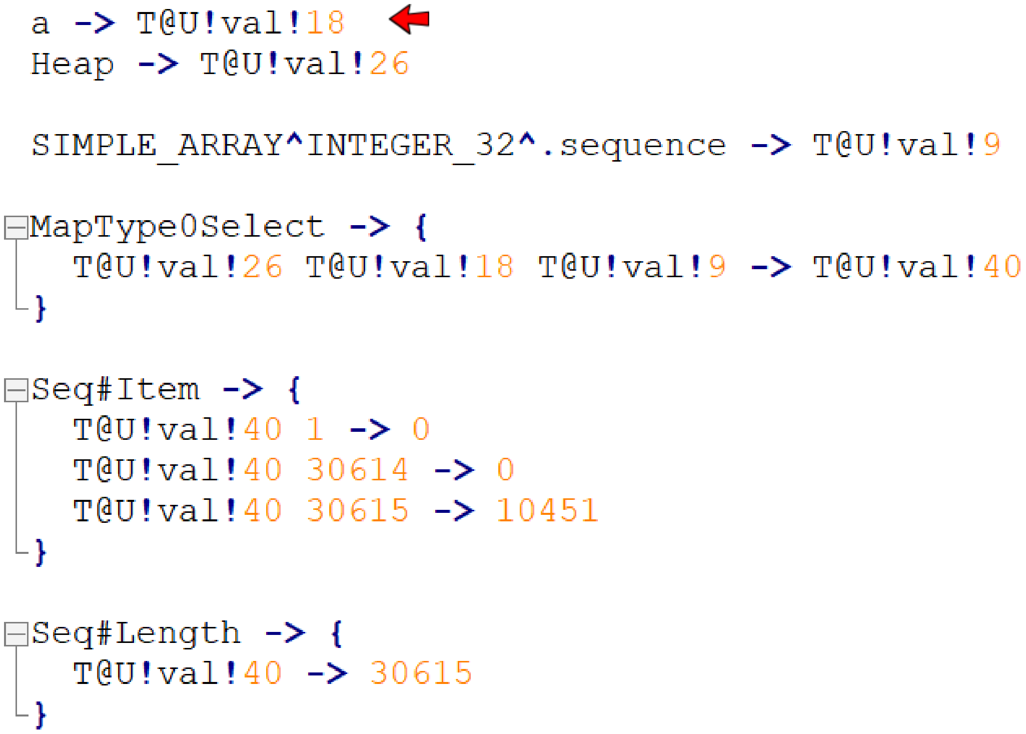
Class	Feature	Information
MAX_IN_ARRAY...	max	Postcondition is_max may be violated.
		Counterexample: a.count = 30615, a [1] = 0, a [30614] = 0, a [30615] = 10451.

The decipherment of an SMT model



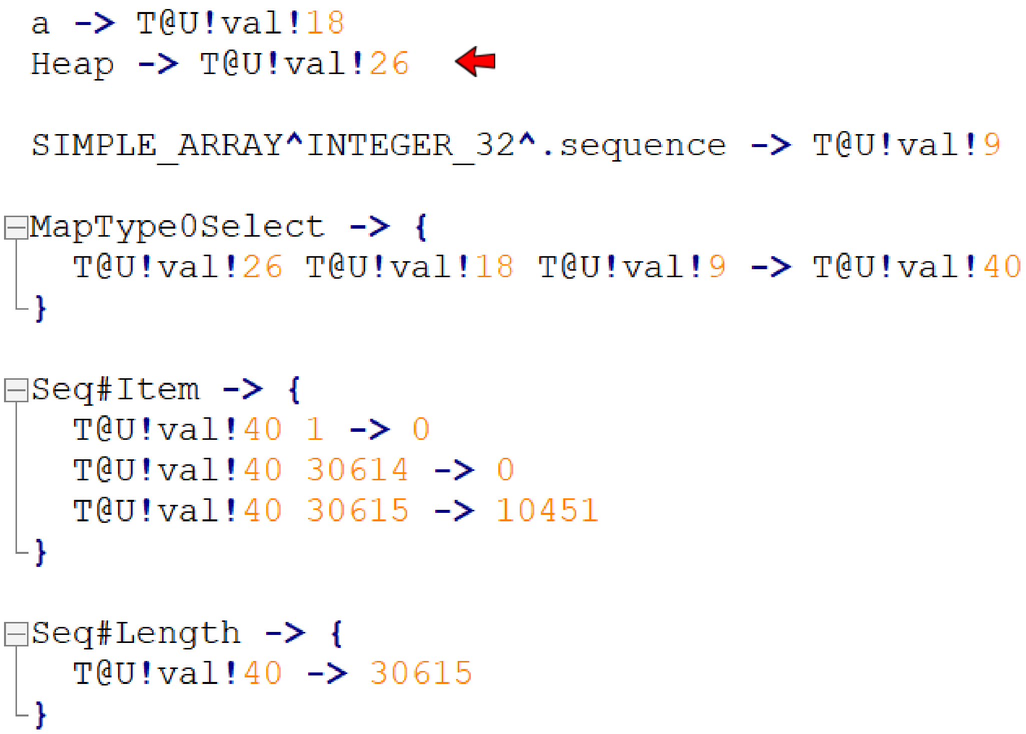
```
  a -> T@U!val!18
  Heap -> T@U!val!26

  SIMPLE_ARRAY^INTEGER_32^.sequence -> T@U!val!9

MapType0Select -> {
    T@U!val!26 T@U!val!18 T@U!val!9 -> T@U!val!40
}

Seq#Item -> {
    T@U!val!40 1 -> 0
    T@U!val!40 30614 -> 0
    T@U!val!40 30615 -> 10451
}

Seq#Length -> {
    T@U!val!40 -> 30615
}
```

The decipherment of an SMT model



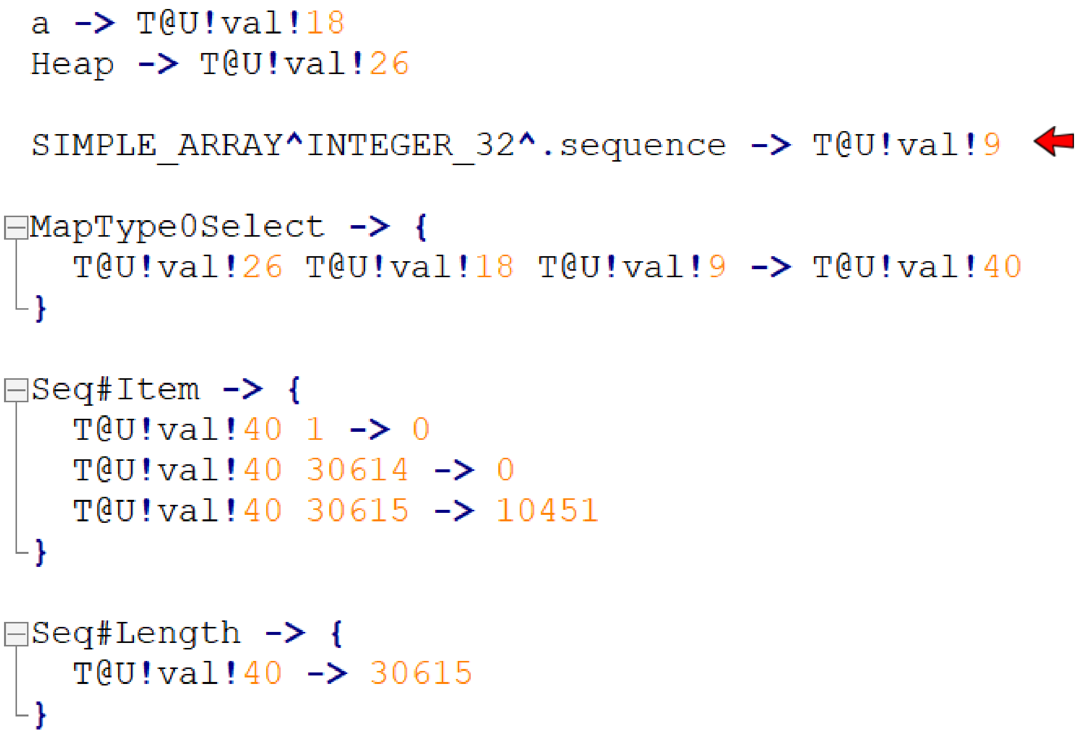
```
a -> T@U!val!18
Heap -> T@U!val!26

SIMPLE_ARRAY^INTEGER_32^.sequence -> T@U!val!9

MapType0Select -> {
  T@U!val!26 T@U!val!18 T@U!val!9 -> T@U!val!40
}

Seq#Item -> {
  T@U!val!40 1 -> 0
  T@U!val!40 30614 -> 0
  T@U!val!40 30615 -> 10451
}

Seq#Length -> {
  T@U!val!40 -> 30615
}
```

The decipherment of an SMT model



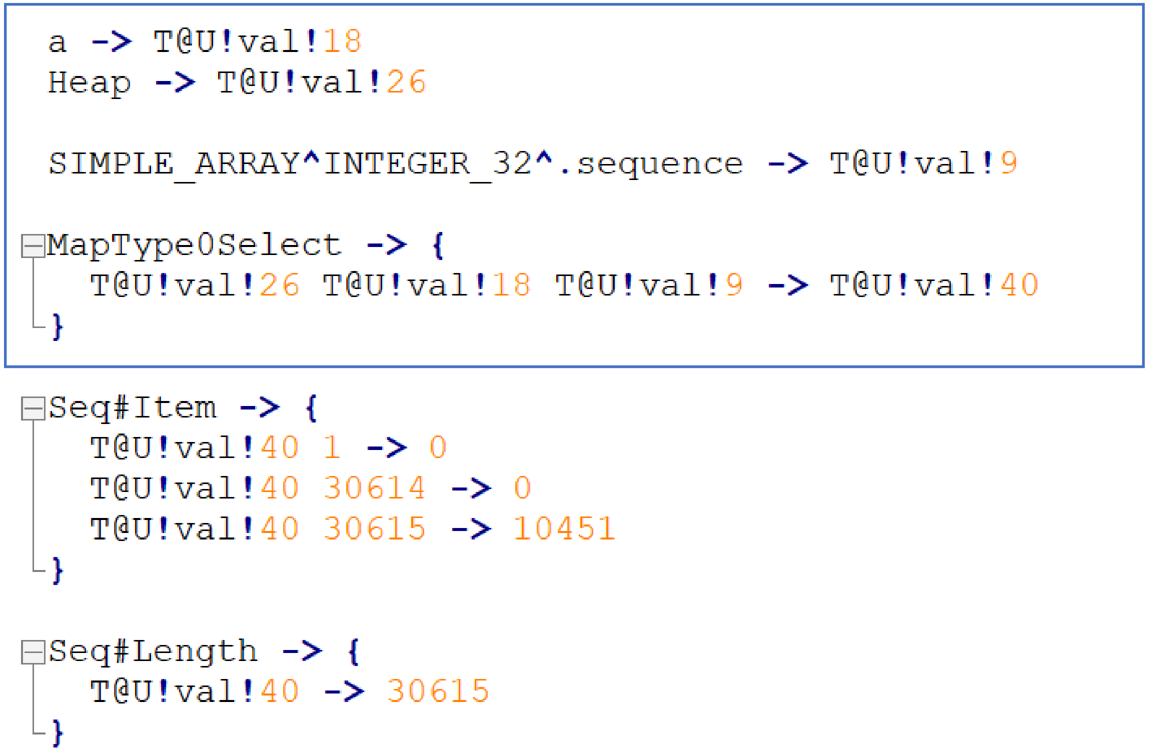
```
a -> T@U!val!18
Heap -> T@U!val!26

SIMPLE_ARRAY^INTEGER_32^.sequence -> T@U!val!9

MapType0Select -> {
   T@U!val!26 T@U!val!18 T@U!val!9 -> T@U!val!40
}

Seq#Item -> {
   T@U!val!40 1 -> 0
   T@U!val!40 30614 -> 0
   T@U!val!40 30615 -> 10451
}

Seq#Length -> {
   T@U!val!40 -> 30615
}
```

The decipherment of an SMT model

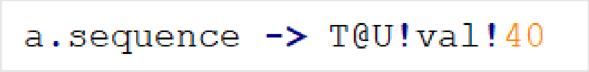


```
a -> T@U!val!18
Heap -> T@U!val!26

SIMPLE_ARRAY^INTEGER_32^.sequence -> T@U!val!9

MapType0Select -> {
   T@U!val!26 T@U!val!18 T@U!val!9 -> T@U!val!40
}

Seq#Item -> {
   T@U!val!40 1 -> 0
   T@U!val!40 30614 -> 0
   T@U!val!40 30615 -> 10451
}

Seq#Length -> {
   T@U!val!40 -> 30615
}
```

The decipherment of an SMT model

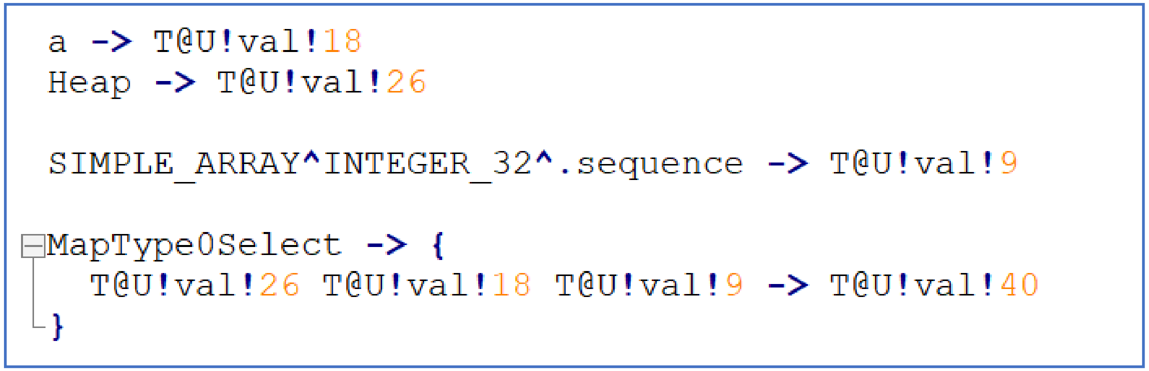


```
  a -> T@U!val!18
  Heap -> T@U!val!26

  SIMPLE_ARRAY^INTEGER_32^.sequence -> T@U!val!9

 MapType0Select -> {
    T@U!val!26 T@U!val!18 T@U!val!9 -> T@U!val!40
  }

 Seq#Item -> {
    T@U!val!40 1 -> 0
    T@U!val!40 30614 -> 0
    T@U!val!40 30615 -> 10451
  }

 Seq#Length -> {
    T@U!val!40 -> 30615
  }
```

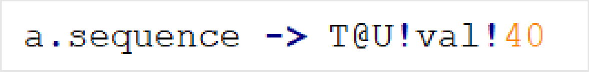


```
a.sequence -> T@U!val!40
```

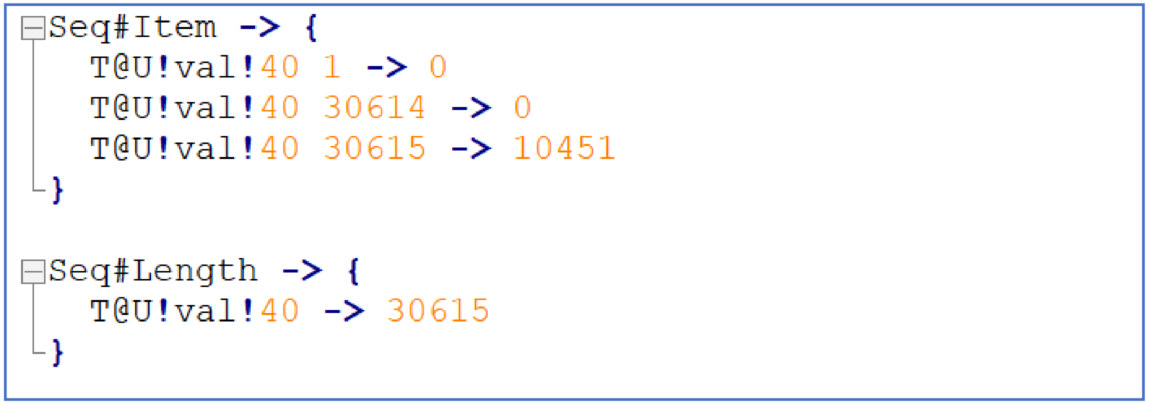
The decipherment of an SMT model



```
a -> T@U!val!18
Heap -> T@U!val!26

SIMPLE_ARRAY^INTEGER_32^.sequence -> T@U!val!9

MapType0Select -> {
  T@U!val!26 T@U!val!18 T@U!val!9 -> T@U!val!40
}
```



```
a.sequence -> T@U!val!40
```

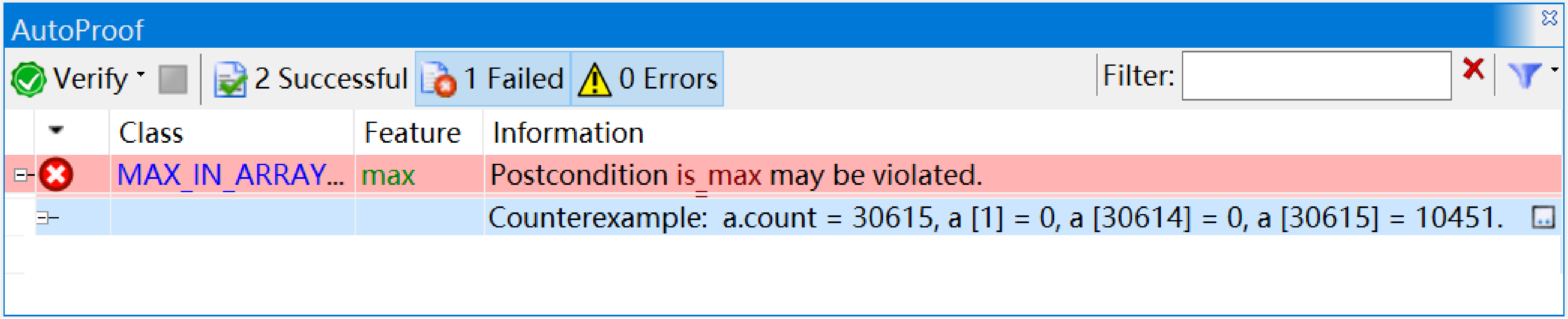


```
Seq#Item -> {
  T@U!val!40 1 -> 0
  T@U!val!40 30614 -> 0
  T@U!val!40 30615 -> 10451
}

Seq#Length -> {
  T@U!val!40 -> 30615
}
```

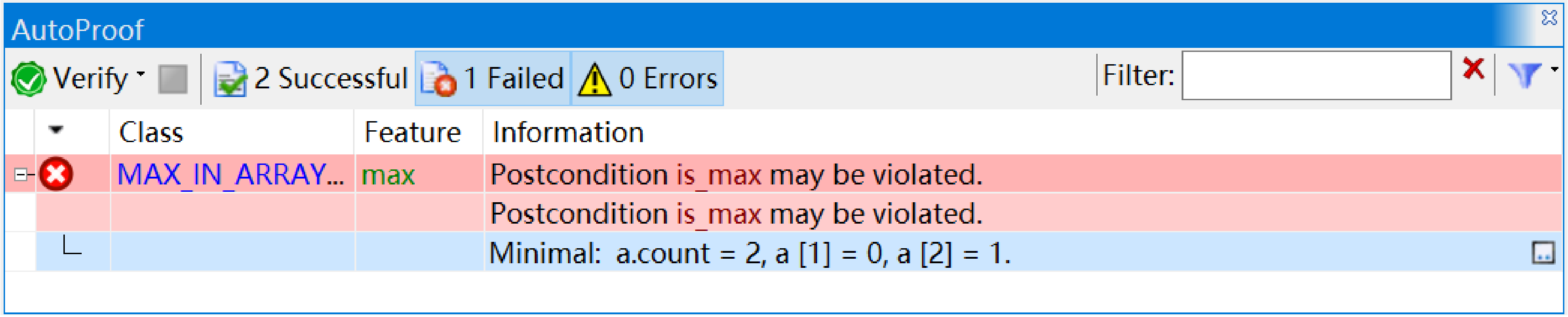
```
a [1] = 0, a [30614] = 0, a [30615] = 10451
```

Counterexample minimization

- Make the counterexample more intuitive through minimization



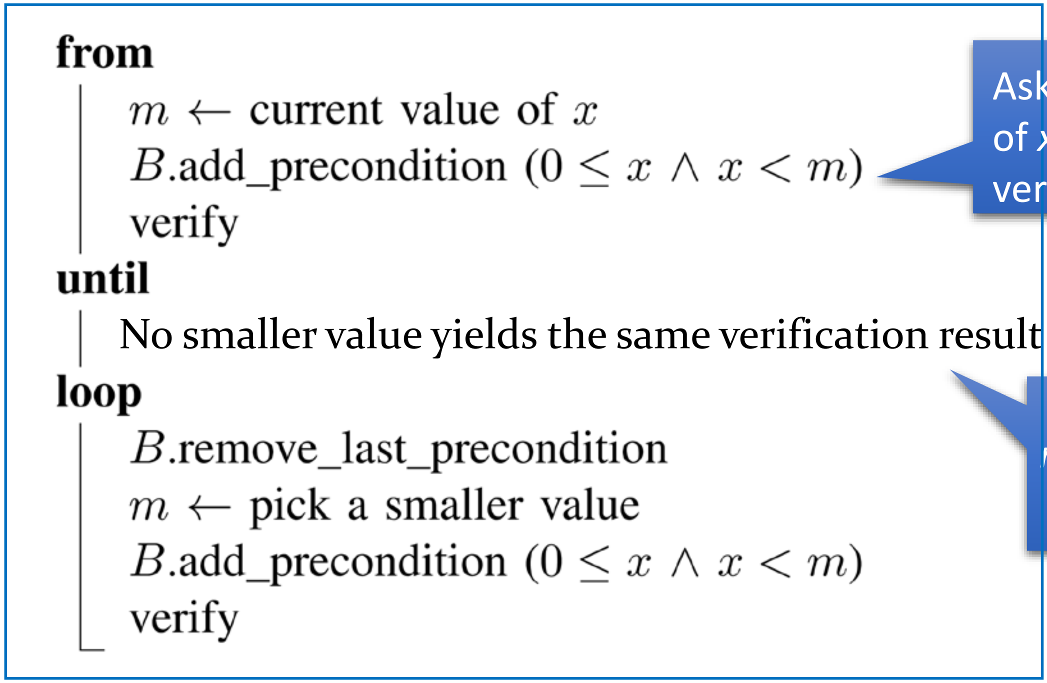
	Class	Feature	Information
⊟ ✖	MAX_IN_ARRAY...	max	Postcondition is_max may be violated.
⊢			Counterexample: a.count = 30615, a [1] = 0, a [30614] = 0, a [30615] = 10451.



	Class	Feature	Information
⊟ ✖	MAX_IN_ARRAY...	max	Postcondition is_max may be violated.
			Postcondition is_max may be violated.
└			Minimal: a.count = 2, a [1] = 0, a [2] = 1.

Counterexample minimization

Minimize each integer in the counterexample

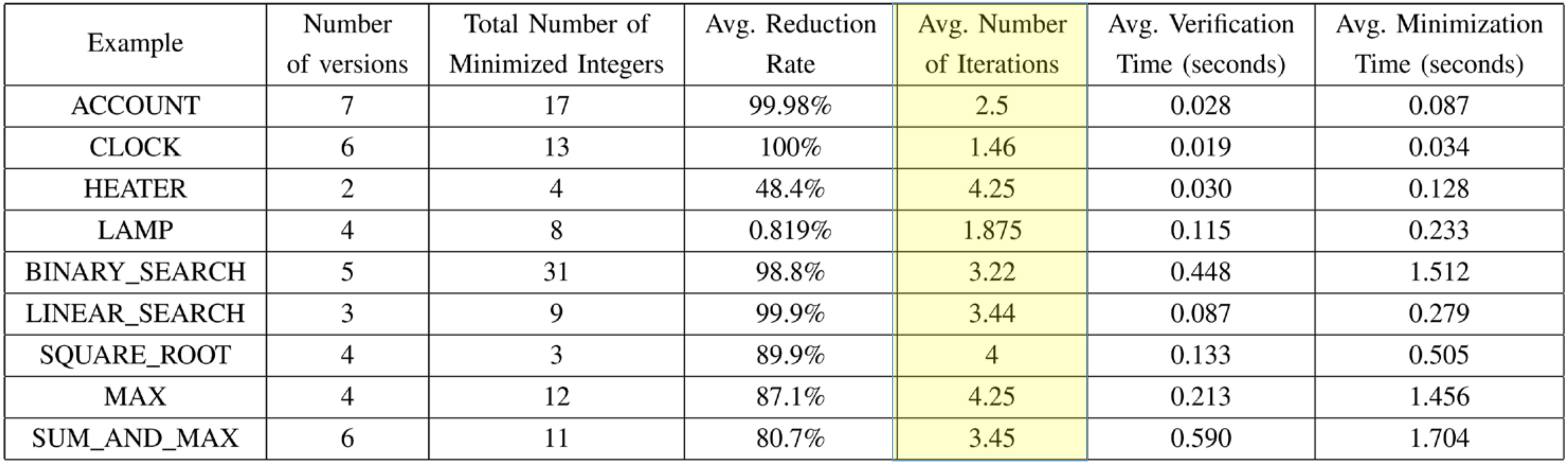


```
from
    $m \leftarrow$ current value of $x$
    $B$.add_precondition $(0 \leq x \wedge x < m)$
    verify
until
    No smaller value yields the same verification result
loop
    $B$.remove_last_precondition
    $m \leftarrow$ pick a smaller value
    $B$.add_precondition $(0 \leq x \wedge x < m)$
    verify
```

Ask prover whether it’s possible to get a value of *x (0 ≤ x < m)* and still yields the same verification results.

When the algorithm ends (no smaller value of *m* can be found), the counterexample from the last verification run is the minimal possible.

Experiment result



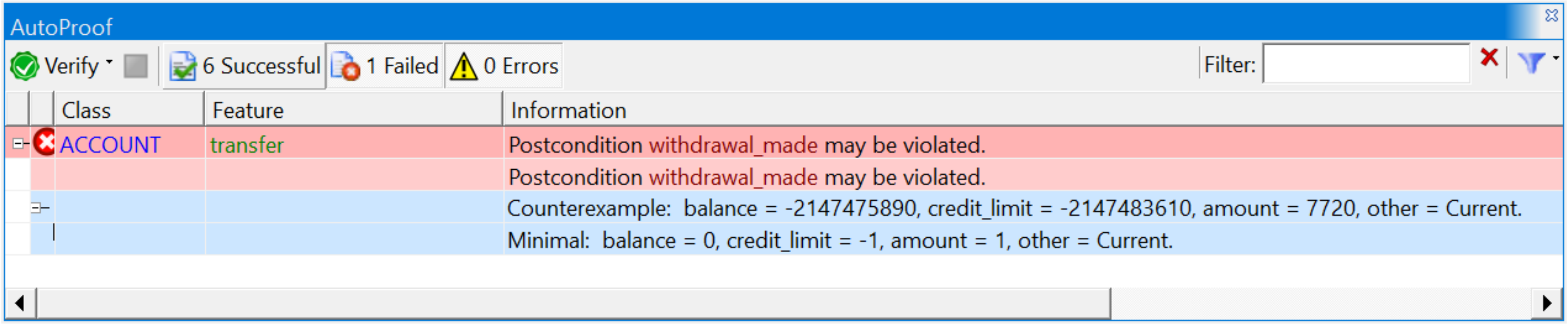
Example	Number of versions	Total Number of Minimized Integers	Avg. Reduction Rate	Avg. Number of Iterations	Avg. Verification Time (seconds)	Avg. Minimization Time (seconds)
ACCOUNT	7	17	99.98%	2.5	0.028	0.087
CLOCK	6	13	100%	1.46	0.019	0.034
HEATER	2	4	48.4%	4.25	0.030	0.128
LAMP	4	8	0.819%	1.875	0.115	0.233
BINARY_SEARCH	5	31	98.8%	3.22	0.448	1.512
LINEAR_SEARCH	3	9	99.9%	3.44	0.087	0.279
SQUARE_ROOT	4	3	89.9%	4	0.133	0.505
MAX	4	12	87.1%	4.25	0.213	1.456
SUM_AND_MAX	6	11	80.7%	3.45	0.590	1.704

125 integers are minimized in total

108 are minimized into values [-2, 2]

58 are minimized to 0

Generate test script from counterexample



Class	Feature	Information
ACCOUNT	transfer	Postcondition withdrawal_made may be violated.
		Postcondition withdrawal_made may be violated.
		Counterexample: balance = -2147475890, credit_limit = -2147483610, amount = 7720, other = Current.
		Minimal: balance = 0, credit_limit = -1, amount = 1, other = Current.

```
test_ACCOUNT_transfer
    local
        current_object: ACCOUNT
        amount: INTEGER
        other: ACCOUNT
    do
        create current_object.make
        {P_INTERNAL}.set_integer_32_field_ ("balance", current_object, 0)
        {P_INTERNAL}.set_integer_32_field_ ("credit_limit", current_object, (-1))
        amount := 1
        other := current_object
        current_object.transfer (amount, other)
    end
```

Seeding contradiction for full-coverage test suite

```
simple (a: INTEGER)
        do
            if a > 0 then
                        check False end
                        x := 1                -- Instruction 1
            else
                        check False end
                        x := 2                -- Instruction 2
            end
            if $a^2 > a$ then
                        check False end
                        x := 3                -- Instruction 3
            else
                        check False end
                        x := 4                -- Instruction 4
            end
        end
```

Test cases:
a = 0, a = 1

Branches
not covered!

The solution: conditional seeding

N: number of basic blocks

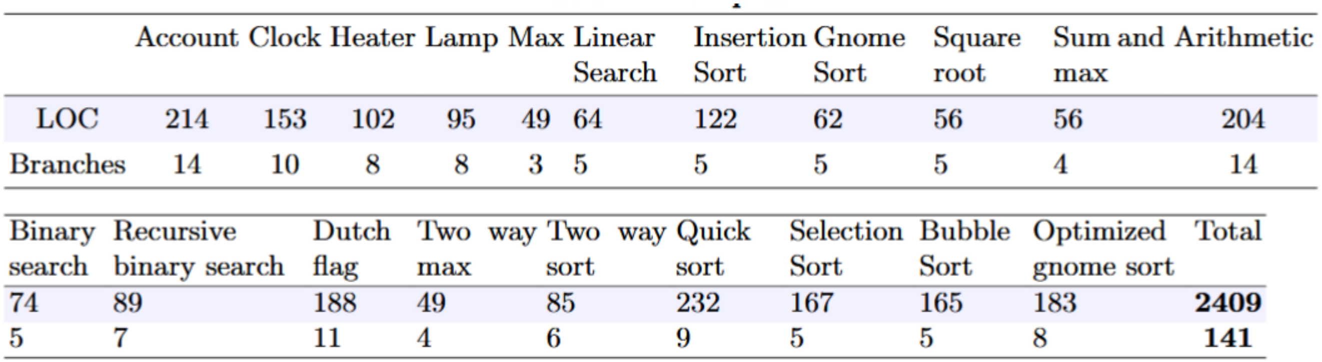
```
bn := non_deterministic (0 .. N)
do
    if a > 0 then
            if bn = 1 then check False end end
            x := 1                  -- Instruction 1      (block 1)
    else
            if bn = 2 then check False end end
            x := 2                  -- Instruction 2      (block 2)
    end
    if $a^2$ > a then
            if bn = 3 then check False end end
            x := 3                  -- Instruction 3      (block 3)
    else
            if bn = 4 then check False end end
            x := 4                  -- Instruction 4      (block 4)
    end
```

Tests cover all branches!

Seeding contradiction: results and comparison (20 examples)

	Seeding Contradiction	**IntelliTest (C#)**	**AutoTest**
Average coverage	99.37%	97.1%	81.2%
Examples reaching exhaustive coverage	20	19	7
Average time to exhaustive coverage	0.5	27	259
Average generated tests to exhaustive coverage	6.3	10.5	623.3

Examples (mostly from verification competitions)



	Account	Clock	Heater	Lamp	Max	Linear Search	Insertion Sort	Gnome Sort	Square root	Sum and max	Arithmetic
LOC	214	153	102	95	49	64	122	62	56	56	204
Branches	14	10	8	8	3	5	5	5	5	4	14

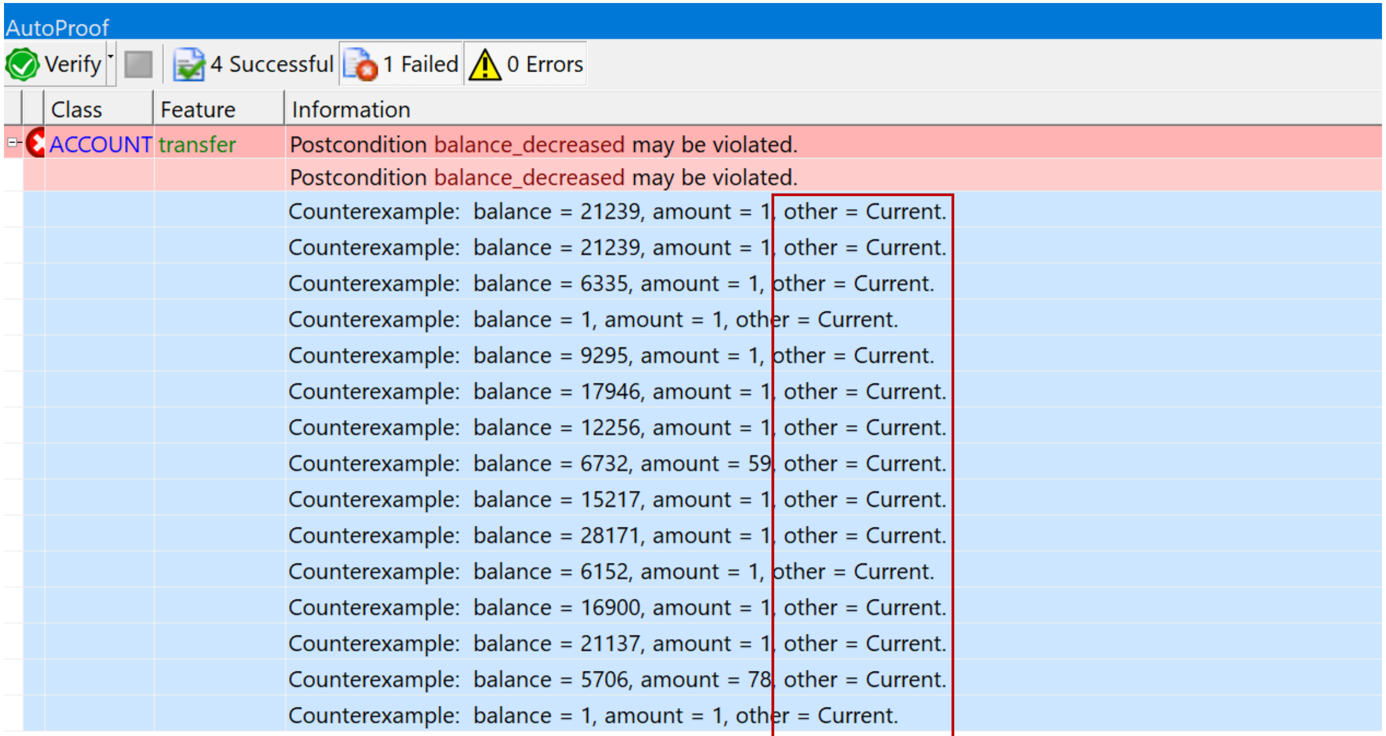
Binary search	Recursive binary search	Dutch flag	Two way max	Two way sort	Quick sort	Selection Sort	Bubble Sort	Optimized gnome sort	Total
74	89	188	49	85	232	167	165	183	**2409**
5	7	11	4	6	9	5	5	8	**141**

Current limitations (and future work)

- Limitations of SMT solver
- Some Eiffel mechanisms (genericity) not yet supported
- Single routines
- Examples still small, although some sophisticated

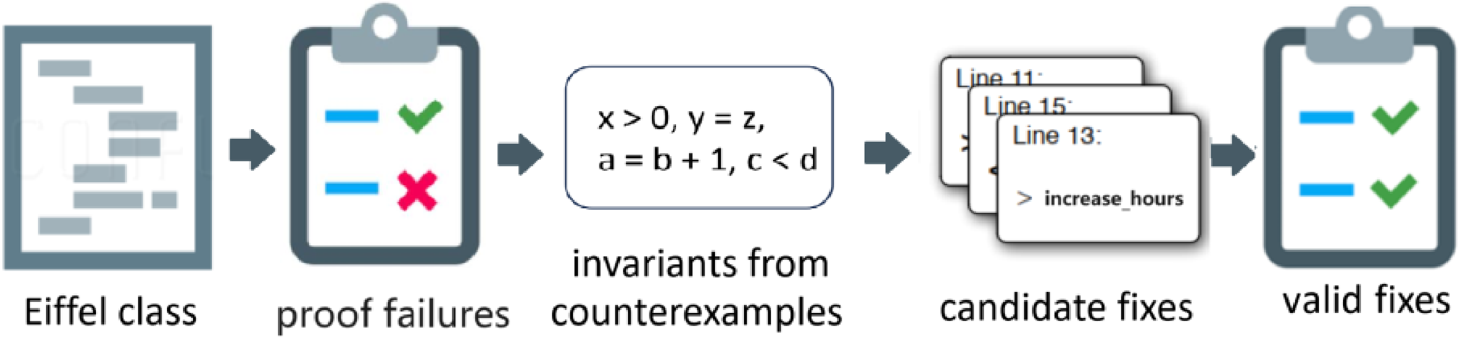
The next step: generating fixes

```
transfer (amount: INTEGER; other: ACCOUNT)
        -- Transfer `amount' from `Current' to `other'.
    note
        explicit: wrapping
    require
        other_not_void: other /= Void
        amount_not_negative: amount >= 0
        amount_not_too_large: amount <= balance
    do
        withdraw (amount)
        other.deposit (amount)
    ensure
            -- Allowed to modify the state of `Current' and `other'
            -- (by default a procedure can only modify `Current'):
        modify (Current, other)
        balance_decreased: balance = old balance - amount
        other_balance_increased: other.balance = old other.balance + amount
    end
```



Class	Feature	Information
ACCOUNT	transfer	Postcondition balance_decreased may be violated.
		Postcondition balance_decreased may be violated.
		Counterexample: balance = 21239, amount = 1, other = Current.
		Counterexample: balance = 21239, amount = 1, other = Current.
		Counterexample: balance = 6335, amount = 1, other = Current.
		Counterexample: balance = 1, amount = 1, other = Current.
		Counterexample: balance = 9295, amount = 1, other = Current.
		Counterexample: balance = 17946, amount = 1, other = Current.
		Counterexample: balance = 12256, amount = 1, other = Current.
		Counterexample: balance = 6732, amount = 59, other = Current.
		Counterexample: balance = 15217, amount = 1, other = Current.
		Counterexample: balance = 28171, amount = 1, other = Current.
		Counterexample: balance = 6152, amount = 1, other = Current.
		Counterexample: balance = 16900, amount = 1, other = Current.
		Counterexample: balance = 21137, amount = 1, other = Current.
		Counterexample: balance = 5706, amount = 78, other = Current.
		Counterexample: balance = 1, amount = 1, other = Current.

Proof2Fix: generating fixes for proof failures



Eiffel class

proof failures

x > 0, y = z,
a = b + 1, c < d

invariants from counterexamples

Line 11:

Line 15:

Line 13:

> increase_hours

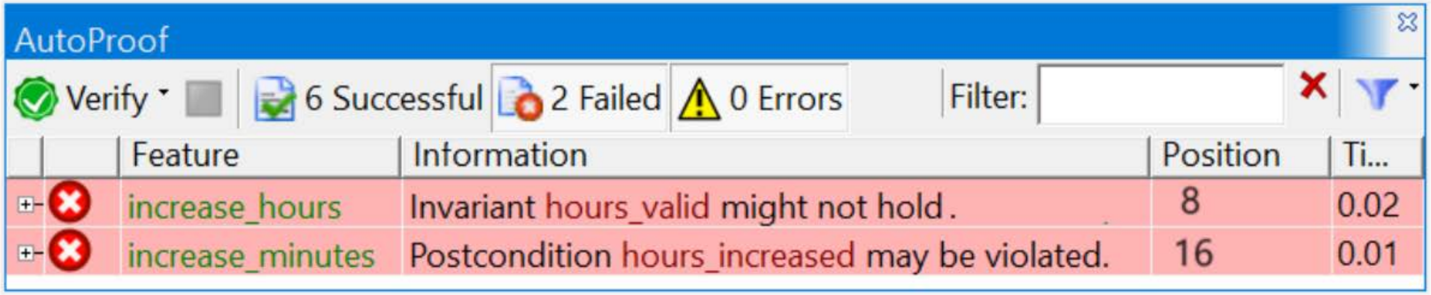
candidate fixes

valid fixes

Example: CLOCK

```
increase_hours
  do




      if hours = 24 then
        hours := 0
      else
        hours := hours + 1
      end


  end
invariant
    hours_valid: 0 ≤ hours ∧ hours ≤  23
```

```
increase_minutes
do




    if minutes <59 then
      minutes := minutes + 1
    else
       minutes := 0
    end
ensure
hours_increased: old minutes= 59 ⟹ hours=(old hours+1)\\24
  end
```

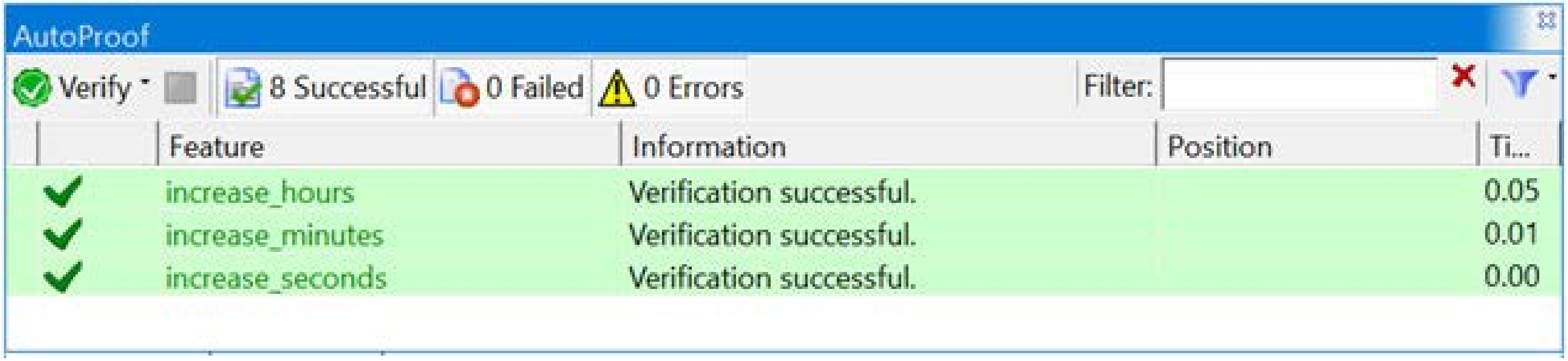


	Feature	Information	Position	Ti...
⊞ ⊗	increase_hours	Invariant hours_valid might not hold.	8	0.02
⊞ ⊗	increase_minutes	Postcondition hours_increased may be violated.	16	0.01

Example: CLOCK

```
increase_hours_fixed
  do
      if hours = 23 then
          hours := 0
      else
        if hours = 24 then
           hours := 0
        else
           hours := hours + 1
        end
      end
  end
```

```
increase_minutes_fixed
do
       if minutes = 59 then
           increase_hours
       end
     if minutes <59 then
        minutes := minutes + 1
     else
         minutes := 0
     end
end
```



	Feature	Information	Position	Ti...
✓	increase_hours	Verification successful.		0.05
✓	increase_minutes	Verification successful.		0.01
✓	increase_seconds	Verification successful.		0.00

Generating candidate fixes

Candidate fixes based on a counterexample invariant ϕ

Fixes on contracts

- **Precondition strengthening**: add **not** ϕ to r's precondition, to rule out the faulty cases characterized by ϕ.

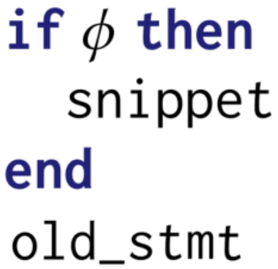
- **Postcondition weakening**: if ψ is the postcondition clause that causes the proof to fail, replace it by **not** ϕ **implies** ψ, so that the previously failing cases will now verify.

Generating candidate fixes

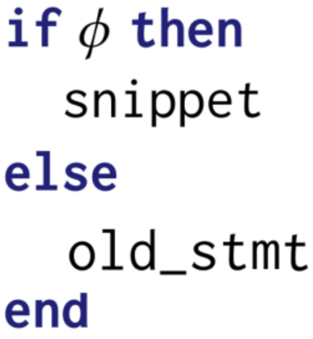
Candidate fixes based on a counterexample invariant ϕ

Fixes on implementation

Replace the implementation with the code snippet generated based on the following schema:

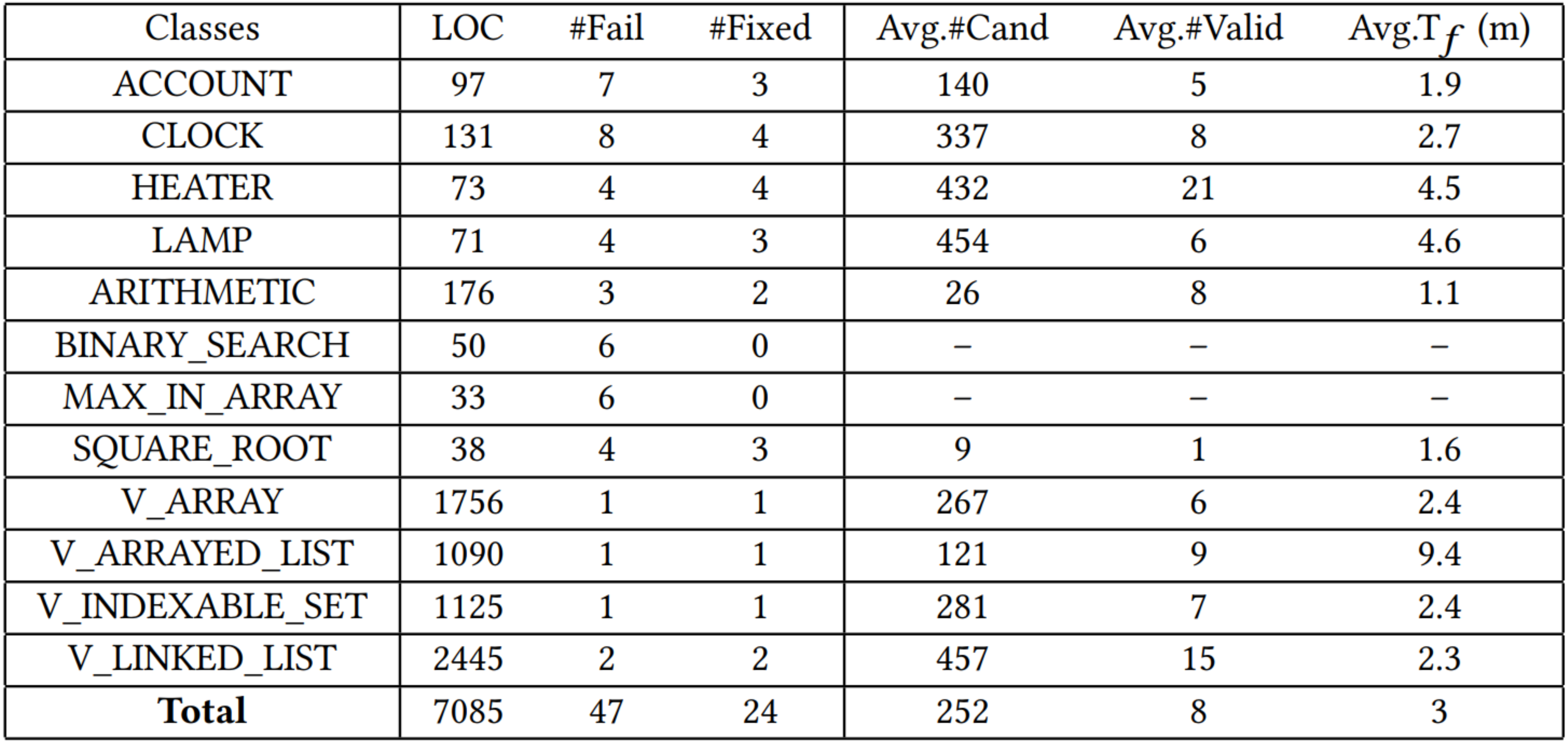


```
if ϕ then
   snippet
end
old_stmt
```



```
if ϕ then
   snippet
else
   old_stmt
end
```

Fixing Results of Proof2Fix



Classes	LOC	#Fail	#Fixed	Avg.#Cand	Avg.#Valid	Avg.T_f (m)
ACCOUNT	97	7	3	140	5	1.9
CLOCK	131	8	4	337	8	2.7
HEATER	73	4	4	432	21	4.5
LAMP	71	4	3	454	6	4.6
ARITHMETIC	176	3	2	26	8	1.1
BINARY_SEARCH	50	6	0	–	–	–
MAX_IN_ARRAY	33	6	0	–	–	–
SQUARE_ROOT	38	4	3	9	1	1.6
V_ARRAY	1756	1	1	267	6	2.4
V_ARRAYED_LIST	1090	1	1	121	9	9.4
V_INDEXABLE_SET	1125	1	1	281	7	2.4
V_LINKED_LIST	2445	2	2	457	15	2.3
Total	7085	47	24	252	8	3

Summary

Take advantage of the test-proofs complementarity

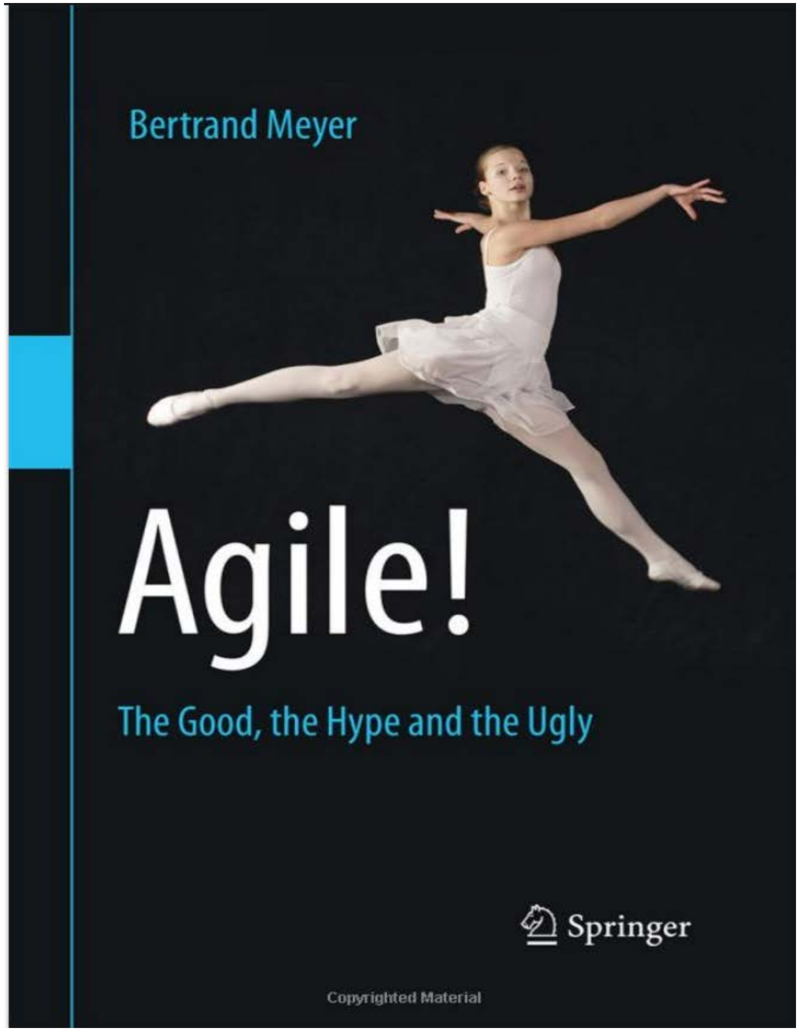
- Generate failing tests from failing proofs
- Make these tests meaningful to programmers
- For a correct program, generate a test suite:
    - Guaranteed exhaustive coverage
    - Does not require any test data
    - Based on the program text only
    - Entirely automatic
    - Extremely fast
- Next: Automatic Program Repair with the same benefits

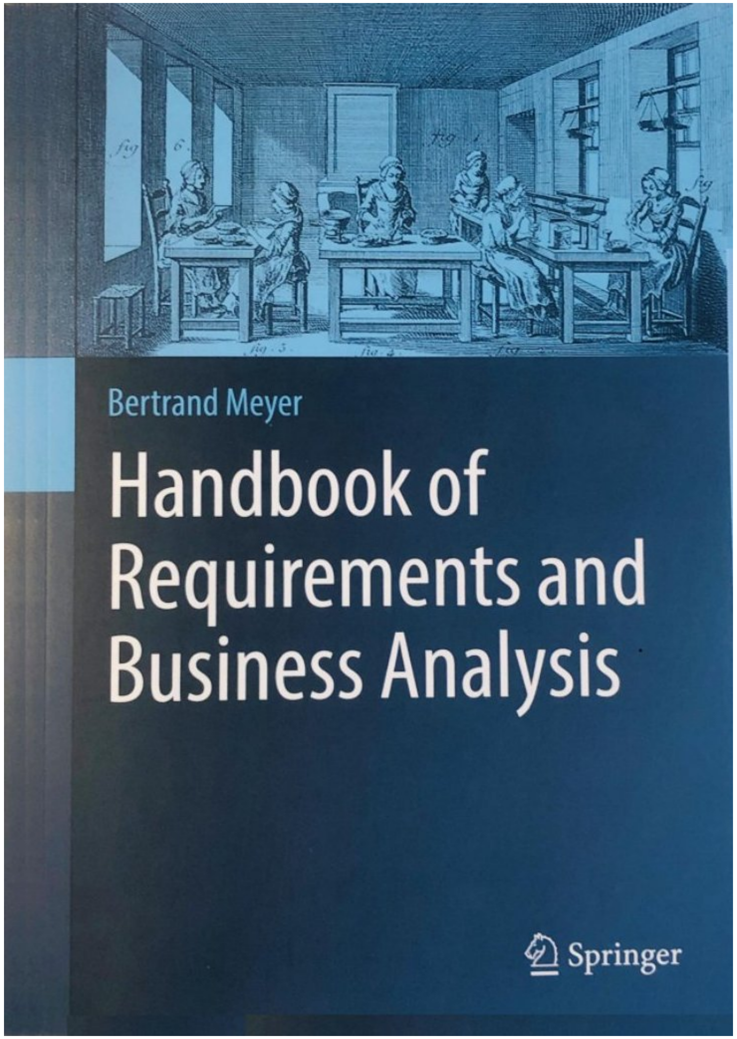
Constructor Institute, Schaffhausen

Master programs (CSSE-Leadership/Quantum)

PhD and postdoc positions in SE, quantum, verification…

(Fairly) recent books





For more!

AutoProof	http://autoproof.sit.org
Eiffel	https://eiffel.com https://eiffel.org
Constructor Institute	https://constructor.org/institute