Cooperative Verification: The Art of Combining Verification Tools

Dirk Beyer

LMU Munich, Germany

Keynote at TAP 2018, Toulouse, 2018-06-27
Many Verification Tools Available
I have a dream ...

- ... that one day, all tools for formal methods work together to solve hard verification problems and make our world safer and more secure.
- ... that one day, model checkers and theorem provers can be integrated into the software-development process as seamless as unit testing today.
- ... that one day, model checkers, theorem provers, SMT solvers, and testers use common interfaces for interaction and composition.
Dream is not utopian, will illustrate a few approaches ...

- Approach 1: Conditional Model Checking [FSE’12]
- Approach 2: Verification Witnesses [FSE’15, FSE’16]
- Approach 3: Tests from Witnesses [TAP’18]
Approach 1:
Cooperative Verification by Conditional Model Checking and Reducers
Facing Hard Verification Tasks

Given: Program $P \models \varphi$?
Facing Hard Verification Tasks

Given: Program $P \models \varphi$?

Verifier A

Verifier B

Verifier A + Verifier B

e.g., conditional model checking
Conditional Model Checking

Program $P$ → Conditional Verifier A $P \models \varphi$? → Conditional Verifier B $P \models \varphi$?

Condition $\psi$ → TRUE under condition $\psi$

TRUE → FALSE
Reducer-Based Conditional Verifier Construction

Verifier B ⟷ Conditional Verifier B

Reducer (preprocessor)
▶ Builds standard input (C program)
▶ Representing a subset of paths
▶ Contains at least all non-verified paths

+ Verifier-unspecific approach
+ Many conditional verifiers possible
Reducer-Based Conditional Verifier Construction

Verifier B → Conditional Verifier B

Our Solution

Condition

Input Program

Verifier B

Dirk Beyer
LMU Munich, Germany
Reducer-Based Conditional Verifier Construction

Verifier B

Conditional Verifier B

Our Solution

Reducer

Residual Program

Verifier B

Condition

Input Program

Reducer (preprocessor)

- Builds standard input (C program)
- Representing a subset of paths
- Contains at least all non-verified paths
Reducer-Based Conditional Verifier Construction

Verifier B

Our Solution

Reducer

Verifier B

Reducer (preprocessor)
- Builds standard input (C program)
- Representing a subset of paths
- Contains at least all non-verified paths
+ Verifier-unspecific approach
+ Many conditional verifiers possible
Example Program and Condition

0: if (notThursday)
1:    discount = day % 7;
else
2:    discount = 5;
3:    assert (0 <= discount < 7);
4:
Example Program and Condition

0: if (notThursday)
1: discount = day%7;
else
2: discount = 5;
3: assert (0 <= discount < 7);
4: 

Verifier A only proofs else branch
Example Program and Condition

0: if (notThursday)
   1: discount = day % 7;
   else
   2: discount = 5;
  3: assert (0 <= discount < 7);
  4: 

Verifier A only proofs else branch
Reducer: Residual Program Construction

Program

\[
\begin{align*}
&l_0 \quad \text{notThursday} \quad \lnot \text{notThursday} \\
&l_1 \quad \text{discount=day}\%7; \\
&l_2 \quad \text{discount=5;} \\
&l_3 \quad \text{assert(0<=discount<7);} \\
&l_4
\end{align*}
\]

Residual Program

Condition

\[
\begin{align*}
&q_0 \quad \text{notThursday} \quad \lnot \text{notThursday} \\
&q_1 \quad \text{discount=day}\%7; \\
&q_2 \\
&\quad \lnot \\
&q_f
\end{align*}
\]
Reducer: Residual Program Construction

Program

\[
\begin{align*}
\text{notThursday} & \quad \neg \text{notThursday} \\
\text{discount} = \text{day} \mod 7; & \quad \text{discount} = 5; \\
\text{assert}(0 \leq \text{discount} < 7); & \\
\end{align*}
\]

Condition

\[
\begin{align*}
\text{notThursday} & \quad \neg \text{notThursday} \\
\text{discount} = \text{day} \mod 7; & \\
\end{align*}
\]

Reducer

\[
(l_0, q_0)
\]
Reducer: Residual Program Construction

Program

Condition

Residual Program
Reducer: Residual Program Construction

Program

\[ l_0 \quad \text{notThursday} \quad \neg \text{notThursday} \]
\[ l_1 \quad \text{discount=day} \mod 7; \]
\[ l_2 \quad \text{discount=5;} \]
\[ l_3 \quad \text{assert(0} \leq \text{discount}<7); \]
\[ l_4 \]

Condition

\[ q_0 \quad \text{notThursday} \quad \neg \text{notThursday} \]
\[ q_1 \quad \text{discount=day} \mod 7; \]
\[ q_2 \]
\[ q_f \]

Residual Program

\[ (l_0, q_0) \quad \neg \text{notThursday} \]
\[ (l_2, q_f) \]
Reducer: Residual Program Construction

Program

\( l_0 \)  
\( l_1 \)  
\( l_2 \)  
\( l_3 \)  
\( l_4 \)

\( \text{notThursday} \)  
\( \text{discount} = \text{day}\%7; \)  
\( \text{discount} = 5; \)  
\( \text{assert}(0 \leq \text{discount} < 7); \)  

Condition

\( q_0 \)  
\( q_1 \)  
\( q_2 \)

\( \text{notThursday} \)  
\( \text{discount} = \text{day}\%7; \)  

Residual Program

\( (l_0, q_0) \)  
\( (l_1, q_1) \)  
\( (l_2, q_f) \)
Reducer: Residual Program Construction

Program

\[ l_0 \rightarrow \text{notThursday} \]
\[ l_1 \rightarrow \neg \text{notThursday} \]
\[ l_2 \rightarrow \text{discount=day}\%7; \]
\[ l_3 \rightarrow \text{discount}=5; \]
\[ l_4 \rightarrow \text{assert}(0<=\text{discount}<7); \]

Residual Program

\[ (l_0, q_0) \rightarrow \text{notThursday} \]
\[ (l_1, q_1) \rightarrow \neg \text{notThursday} \]
\[ (l_2, q_f) \rightarrow \text{discount=day}\%7; \]
\[ (l_3, q_2) \rightarrow \]

Dirk Beyer
LMU Munich, Germany
Reduction: Residual Program Construction

Program

\[ l_0 \rightarrow \neg \text{notThursday} \]
\[ l_1 \rightarrow \text{discount}=\text{day}\%7; \]
\[ l_2 \rightarrow \text{discount}=5; \]
\[ l_3 \rightarrow \text{assert}(0\leq\text{discount}<7); \]
\[ l_4 \rightarrow \neg \text{notThursday} \]

Residual Program

\[ (l_0, q_0) \rightarrow \neg \text{notThursday} \]
\[ (l_1, q_1) \rightarrow \text{discount}=\text{day}\%7; \]
\[ (l_2, q_f) \rightarrow \text{assert}(0\leq\text{discount}<7); \]
\[ (l_3, q_2) \rightarrow \neg \text{notThursday} \]
\[ (l_4, q_r) \rightarrow \text{notThursday} \]
Reducer: C Transformation

Residual Program

\[(l_0, q_0)\]

\[\text{notThursday} \rightarrow (l_1, q_1) \quad \neg\text{notThursday} \rightarrow (l_2, q_f)\]

\[\text{discount=day}\%7;\]

\[\text{assert}(0\leq\text{discount}<7);\]

\[(l_3, q_2)\]

\[\text{assert}(0\leq\text{discount}<7);\]

\[(l_4, q_r)\]
Reducer: C Transformation

Residual Program

\[(l_0, q_0) \quad \overset{\text{notThursday}}{\longrightarrow} \quad \neg \text{notThursday} \quad (l_1, q_1) \quad (l_2, q_f) \]

\[(l_3, q_2) \quad \overset{\text{discount=day\%7;}}{\longrightarrow} \quad \text{assert}(0\leq\text{discount}<7); \quad (l_4, q_r)\]

C Translation

\[
\text{if}(\text{notThursday}) \\
{\}
\quad \text{discount=day\%7;} \\
\quad \text{assert}(0\leq\text{discount}<7); \]

\}{
Reducer: Soundness

Residual Condition

Program Paths

Non-verified Program Paths

Residual Program Paths
Reduction: Soundness

Residual Condition

Theorem

*Presented reducer fulfills residual condition.*
Evaluation Setup
## Small Extract of Results

<table>
<thead>
<tr>
<th>Task</th>
<th>R</th>
<th>CPA-Seq S t(s)</th>
<th>UAutomizer S t(s)</th>
<th>Predicate +Reducer S t(s)</th>
<th>Predicate +Reducer S t(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P15l01</td>
<td>T</td>
<td>X 910</td>
<td>X 900</td>
<td>✓ 120</td>
<td>✓ 130</td>
</tr>
<tr>
<td>flood4</td>
<td>T</td>
<td>X 910</td>
<td>X 910</td>
<td>✓ 450</td>
<td>X 1100</td>
</tr>
<tr>
<td>newt3_6</td>
<td>F</td>
<td>X 950</td>
<td>X 490</td>
<td>X 910</td>
<td>✓ 260</td>
</tr>
<tr>
<td>P07l38</td>
<td>T</td>
<td>X 950</td>
<td>X 910</td>
<td>X 1100</td>
<td>✓ 470</td>
</tr>
</tbody>
</table>
Effectiveness on Hard Tasks

![Graph showing the effectiveness of different methods on hard tasks. The x-axis represents the n-th fastest correct result, and the y-axis represents CPU time in seconds. The graph compares CPA-seq, SMACK, Ultimate Automizer, Predicate + CPA-seq, Predicate + SMACK, and Predicate + Ultimate Automizer. The methods are represented by different line styles and markers.](image-url)
More Information:
Reducer-Based Construction of Conditional Verifiers


Dirk Beyer, Marie-Christine Jakobs, Thomas Lemberger, and Heike Wehrheim

LMU Munich, Germany and Paderborn University, Germany
Conclusion — Approach 1

- Template-based conditional verifier construction

```
Condition
\[\text{Reducer}\]
Residual Program
Verifier B
\[\text{Input Program}\]
```
Conclusion — Approach 1

- Template-based conditional verifier construction

- One Reducer
  - Proven sound
  - Used in many conditional verifiers
Conclusion — Approach 1

- Template-based conditional verifier construction

![Diagram showing the process of conditional verification using a reducer and verifier B.]

- One Reducer
  - Proven sound
  - Used in many conditional verifiers

- Effective on hard tasks for verifiers and test tools
Conclusion — Approach 1

- Template-based conditional verifier construction

  ![Diagram](image)

  - Reducer
  - Input Program
  - Condition
  - Residual Program
  - Verifier B

- One Reducer
  - Proven sound
  - Used in many conditional verifiers

- Effective on hard tasks for verifiers and test tools

- Future Work
  - More reducers
  - Using conditions from other tools
Approach 2: Cooperative Verification by Verification Witnesses
Software Verification

Diagram:

- Program
- Specification
- Verifier
- Result (True/False)
Software Verification with Witnesses

Program

Verifier

Specification

Result (True/False)

Witness

Program

Specification

Result (True/False)

Witness
Witness Validation

- Validate untrusted results
- Easier than full verification
Stepwise Refinement

Witness Program Specification Result (True/False)

Witness Testifier

Witness Program Specification Result (True/False)
Violation Witnesses

Violation Witness
Violation Witnesses

Violation Witness

Abstract Counterex.

Test Case
Violation Witnesses

FSE'15

Violation Witness

Abstract Counterex.

Test Case
Search-Space Reduction for Stepwise Witness Refinement

Search space
Search-Space Reduction for Stepwise Witness Refinement
Search-Space Reduction for Stepwise Witness Refinement

Search space
Search-Space Reduction for Stepwise Witness Refinement
Search-Space Reduction for Stepwise Witness Refinement

Entry

Error

Search space

Stepwise Testification
Correctness: State of the Art

1. Rarely any additional information
Correctness: State of the Art

1. Rarely any additional information

2. Not human readable
Correctness: State of the Art

1. Rarely any additional information
2. Not human readable
3. Not easily exchangeable across tools
Open Problems

1. **Standardized way** to document verification results to enhance engineering processes **required**
1. **Standardized way** to document verification results to enhance engineering processes **required**

2. **Difficult to establish trust** in results from an untrusted verifier
Open Problems

1. **Standardized way** to document verification results to enhance engineering processes **required**

2. **Difficult to establish trust** in results from an untrusted verifier

3. Potential for synergies between tools and techniques is **left unused**
Verification Witnesses: Classification

FSE’15

Violation Witness

Abstract Counterex. Test Case
Verification Witnesses: Classification

- **Witness**
  - **Violation Witness**
    - **Abstract Counterex.**
    - **Test Case**
  - **Correctness Witness**

- **FSE’15**
Verification Witnesses: Classification

- **Witness**
  - **Violation Witness**
    - **Abstract Counterex.**
    - **Test Case**
  - **Correctness Witness**
    - **SCC**
    - **PCC**

Taleghani & Atlee, ASE’10  Necula, POPL’97
Verification Witnesses: Classification

- **Witness**
  - **Violation Witness**
    - **Abstract Counterex.**
    - **Test Case**
  - **Correctness Witness**
    - **SCC**
    - **PCC**

- **FSE’15**
- Taleghani & Atlee, ASE’10  Necula, POPL’97
Correctness Witnesses and Proof Certificates

- **Full proofs** seem nice, but in practice become **too large**

- Witnesses **support**, but do **not enforce** full proofs

- Instead, correctness witnesses may also represent **proof sketches**
Correctness Witnesses

\[
\begin{array}{c}
P \vdash \varphi
\end{array}
\]
Correctness Witnesses

\[ \pi : [P] \models \varphi \]
Correctness Witnesses

\[ \pi : P \models \varphi \]

Correctness Witness

\[ \pi : P \models \varphi \]

Diagram: Correctness Witness
Correctness Witnesses

Verifier

\[ \pi : \mathcal{P} \rightarrow \varnothing \]

Correctness Witness
Correctness Witnesses

Verifier

\[ \pi : P \models \varphi \]

Validator

Correctness Witness

Dirk Beyer
Correctness Witnesses

Verifier

\( \pi \): \( P \) \models \( \varphi \)

Correctness Witness

Validator

\( P \) \models \( \varphi \)
Correctness Witnesses

\[ \pi : P \models \varphi \]

\[ \pi' : P \models \varphi \]
Correctness Witnesses

Verifier

\[ \pi : P \models \varphi \]

Correctness Witness

Testifier

\[ \pi' : P \models \varphi \]

Correctness Witness 2
Witness Automata

- Express witness as automaton
Witness Automata

- Express witness as **automaton**
- Witness Validation **matches** the **witness** to the **program**
Witness Automata

- Express witness as **automaton**
- Witness Validation **matches** the **witness** to the **program**
- **Decoupled from** specific verification **techniques** and implementations
Witness Automata

- Express witness as **automaton**
- Witness Validation **matches** the **witness** to the **program**
- **Decoupled from** specific verification **techniques** and implementations
- **One common exchange format** for violation witnesses and correctness witnesses
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```
```
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```

Diagram: [Diagram of the program flow and assertions]
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```
Example: Inject Invariants

```c
int main() {
    unsigned int x = nondet();
    unsigned int y = x;
    while (x < 1024) {
        x = x + 1;
        y = y + 1;
    }
    // Safety property
    assert(x == y);
    return 0;
}
```
Producing and Consuming Witnesses: SV-COMP

Table 8: Confirmation rate of witnesses

<table>
<thead>
<tr>
<th>Result</th>
<th>True</th>
<th></th>
<th>False</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Confirmed</td>
<td>Unconfirmed</td>
<td>Total</td>
</tr>
<tr>
<td>UAutomizer</td>
<td>3 558</td>
<td>3 481</td>
<td>77</td>
<td>1 173</td>
</tr>
<tr>
<td>SMACK</td>
<td>2 947</td>
<td>2 695</td>
<td>252</td>
<td>1 929</td>
</tr>
<tr>
<td>CPA-SEQ</td>
<td>3 357</td>
<td>3 078</td>
<td>279</td>
<td>2 342</td>
</tr>
</tbody>
</table>

Verifiable Witnesses. For SV-COMP, it is not sufficient to answer with just True or False: each answer must be accompanied by a verification witness. For correctness witnesses, an unconfirmed answer True was still accepted, but was assigned only 1 point instead of 2 (cf. Table 2). All verifiers in categories that required witness validation support the common exchange format for violation and correctness witnesses. We used the two independently developed witness validators that are integrated in CPAchecker and UAutomizer [7, 8].
Stepwise Refinement: Classification

Verification

Result?

Verification

Condition

Unknown

Refine?

Output

condition

No

Conditional

MC

Yes

Violation Witness

False

Refine?

Violation

Refinement

Yes

Output

witness

No

Result?

False

Output

"Unknown"

Rejected

Correctness Witness

True

Refine?

Output

witness

No

Correctness

Refinement

Yes

Result?

True

Output

"Unknown"

Rejected
Stepwise Refinement: Classification

Verification

Result?

Unknown

Verification Condition
Stepwise Refinement: Classification

- Verification
  - Result?
    - Unknown
      - Verification Condition
        - Refine?
          - No: Output condition
          - Yes: Conditional MC

- Violation Witness
  - False
    - Refine?
      - Output witness
        - Yes: Result? True
        - No: Result? False
        - Output "Unknown"

- Correctness Witness
  - True
    - Refine?
      - Output "Unknown"
Stepwise Refinement: Classification

- Violation Witness
- Verification
  - Result?
    - False
    - Unknown
      - Verification Condition
        - Refine?
          - No: Output condition
          - Yes: Conditional MC
Stepwise Refinement: Classification

1. **Violation Witness**
   - Refine? (Yes/No)
     - Yes: Violation Refinement
     - No: Output "Unknown" (Rejected)
   - Result? (False)
2. **Verification**
   - Result? (False/Unknown)
     - False: Output "Unknown" (Rejected)
     - Unknown: Refine? (Yes/No)
       - Yes: Conditional MC
       - No: Output condition
Stepwise Refinement: Classification

- **Violation Witness**
  - Refine? (Yes) → Violation Refinement
  - Refine? (No) → Output witness
- **Verification**
  - Result? (False) → False
  - Result? (Unknown) → False
  - Result? (True) → True
    - Refine? (Yes) → Output "Unknown"
    - Refine? (No) → Output condition
    - Correctness Witness
      - Correctness Refinement
        - Refine? (Yes) → Yes
        - Refine? (No) → No
      - Result?
        - False → Rejected
        - True → "Unknown"
Stepwise Refinement: Classification

- **Violation Witness**
  - Refine? (Yes) → Violation Refinement
  - Refine? (No) → Verification
    - Result? (False) → Rejected
      - Output "Unknown"
    - Result? (Unknown) → Refine?
      - Refine? (Yes) → Correctness Refinement
      - Refine? (No) → Correctness Witness
        - Result? (True) → Rejected
          - Output "Unknown"
        - Result? (False) → Output "Unknown"
More Information:
Correctness Witnesses:
Exchanging Verification Results between Verifiers

[Proc. FSE 2016, pages 326–337, ACM. DOI Link, Preprint Link]
Dirk Beyer, Matthias Dangl, Daniel Dietsch, and Matthias Heizmann
Conclusion — Approach 2

Correctness-Witnesses...

1. are easy to implement for verifiers that already support violation witnesses
Conclusion — Approach 2

Correctness-Witnesses...

1. are easy to implement for verifiers that already support violation witnesses

2. enable information exchange across different software verifiers
Conclusion — Approach 2

Correctness-Witnesses...

1. are easy to implement for verifiers that already support violation witnesses

2. enable information exchange across different software verifiers

3. efficiently increase confidence in results by validation
Approach 3: Cooperative Verification by Tests from Witnesses
Software contains bugs.
Software contains bugs.
⇒ Automatic verification.
Software contains bugs.
⇒ Automatic verification.

But software contains bugs.
Software contains bugs.  
⇒ Automatic verification. 

But software contains bugs.  
⇒ Automatic validation of results.
Software contains bugs.
⇒ Automatic verification.

But software contains bugs.
⇒ Automatic validation of results.

But software contains bugs.
Software contains bugs.
⇒ Automatic verification.
But software contains bugs.
⇒ Automatic validation of results.
But software contains bugs.
⇒ Execution as proof.
Old Idea: Tests from Counterexamples

- “Generating Tests from Counterexamples”
  ICSE 2004 DOI Link, Preprint Link

- “Test-Input Generation with Java PathFinder”
  W. Visser, C. S. Păsăreanu, S. Khurshid
  ISSTA 2004 DOI Link

Influential papers, but:

- Problem: No exchange format; proprietary technology, proprietary format for test vector
Witness Validation

- **Problem**: Confidence in verifiers
- **Approach**: Witness validation

[Beyer/Dangl/Dietsch/Heizmann/Stahlbauer FSE’15]
Witness Creation

- Program
- Specification
- Verification Task

Verifier

- Violation Witness
- Correctness Witness

- Blast
- CBMC
- CPAchecker
- ESBMC
- Smack
- Ultimate Automizer
- ...
Violation Witness Format: Witness Automaton

- Automaton
- Describes set of error paths
- State-space + source-code guards

```c
extern void __VERIFIER_error(void);
extern unsigned char __VERIFIER_nondet_uchar(void);
int main(void) {
    unsigned char a = __VERIFIER_nondet_uchar();
    unsigned char b = __VERIFIER_nondet_uchar();
    unsigned char sum = a + b;
    unsigned char mean = sum / 2;
    if (mean < a / 2) {
        __VERIFIER_error();
    }
    return 0;
}
```

Diagram of the automaton:
- $q_0$ (start state)
- $q_1$
- $q_2$
- $q_E$
- $q_\perp$

Transitions:
- $q_0 \rightarrow q_1$: $a == 2$
- $q_0 \rightarrow q_2$: $b == 254$
- $q_2 \rightarrow q_\perp$: $\text{else:}$
- $q_2 \rightarrow q_E$: $\text{then:}$
Problem 1: Abstract witnesses

```c
extern void __VERIFIER_error(void);
extern unsigned char __VERIFIER_nondet_uchar(void);
int main(void) {
  unsigned char a = __VERIFIER_nondet_uchar();
  unsigned char b = __VERIFIER_nondet_uchar();
  unsigned char sum = a + b;
  if (mean < a / 2) {
    __VERIFIER_error();
  }
  return 0;
}
```
Witness Refinement

- Problem 1: Abstract witnesses
- Solution: Witness refinement

Diagram:
- Witness
- Refiner
- Refined Witness
- Program
- Specification
- Verification Task
- CPAChecker
- Ultimate Automizer
Witness Refinement

- Problem 1: Abstract witnesses
- Solution: Witness refinement

```
extern void __VERIFIER_error(void);
extern unsigned char __VERIFIER_nondet_uchar(void);
int main(void) {
  unsigned char a = __VERIFIER_nondet_uchar();
  unsigned char b = __VERIFIER_nondet_uchar();
  unsigned char sum = a + b;
  unsigned char mean = sum / 2;
  if (mean < a / 2) {
    __VERIFIER_error();
  }
  return 0;
}
```
Witness Validation

- Existing validators are model checkers
- Problem 2: Confidence in validators
- Problem 3: Found errors difficult to debug
- Solution: Executable counterexamples
Execution-based Witness Validation
Execution-based Witness Validation

C Program

Specification

Verifier

Witness

WITNESS2TEST

Witness Spurious Witness Confirmed

CPA-w2t

FSHELL-w2t

GCC

Executable

Witness

False

Bug found

True

proof found

Dirk Beyer
Execution-based Witness Validation

- Build executable counterexample from witness
- State-space guards $\mapsto$ input variables/functions

```c
#include <stdlib.h>

void __VERIFIER_error() { exit(107); } 

unsigned char __VERIFIER_nondet_uchar() {
  static unsigned int test_vector_index = 0;
  unsigned char retval;

  switch (test_vector_index) {
    case 0: retval = 2U; break;
    case 1: retval = 254U; break;
  }
  ++test_vector_index;
  return retval;
}

extern void __VERIFIER_error(void);
extern unsigned char __VERIFIER_nondet_uchar(void);

int main(void) {
  unsigned char a = __VERIFIER_nondet_uchar();
  unsigned char b = __VERIFIER_nondet_uchar();
  unsigned char sum = a + b;
  unsigned char mean = sum / 2;
  if (mean < a / 2) {
    __VERIFIER_error();
  }
  return 0;
}
```
Full Workflow

(a) Witness construction
(b) Optional witness refinement
(c) Witness validation
Experimental Results
Experiments

- Implementations: CPA-w2t and FShell-w2t
- Witness Refiner: CPAchecker
- Benchmark set:
  - 18,965 witnesses
  - From 21 verifiers
  - From 5,692 verification tasks (1,490 false tasks)
Validation Performance

- 18,965 witnesses in total
- Not only increase of confidence, but also increase of overall effectivity

**TABLE III: Validation results of static/dynamic validators**

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
<th>Union</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confirmed results</td>
<td>12,671</td>
<td>8,702</td>
<td>14,434</td>
</tr>
<tr>
<td>Incorrectly confirmed results</td>
<td>21</td>
<td>6</td>
<td>27</td>
</tr>
</tbody>
</table>
Time Performance

- Time comparison over 2,680 witnesses that all validators confirmed
More Information:

Tests from Witnesses: Execution-Based Validation of Verification Results


Dirk Beyer, Matthias Dangl, Thomas Lemberger, and Michael Tautschnig

LMU Munich, Germany and Queen Mary University of London, UK
Conclusion — Approach 3

- Validate more witnesses
- Validate witnesses faster
- Provide debuggable counterexamples
- Provide executable tests
- Increase confidence in results
Overall Conclusion

- Dream **can** become reality!
- Conditional Model Checking makes sure to inform other verifier about progress
- Verification Witnesses increase trust in results, first-class object to save
- Verification results validated by Testing makes sure developers can use debuggers to explore bug

Thank You!
Additional Material
Architecture CPA-w2t
Architecture FShell-w2t
Experiment Environment

> Machines:
  > Intel Xeon E3-1230 v5 CPU, 8 units, 3.4 GHz
  > 33 GB RAM
  > Ubuntu 16.04

> Limits verifiers:
  > 4 processing units
  > 7 GB RAM
  > 15 min CPU time

> Limits validators:
  > 2 processing units
  > 4 GB RAM
  > 1.5 min CPU time
## Verification Tasks

<table>
<thead>
<tr>
<th>Sub-category</th>
<th>Number of verification tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReachSafety-Arrays</td>
<td>135</td>
</tr>
<tr>
<td>ReachSafety-BitVectors</td>
<td>50</td>
</tr>
<tr>
<td>ReachSafety-ControlFlow</td>
<td>94</td>
</tr>
<tr>
<td>ReachSafety-ECA</td>
<td>1149</td>
</tr>
<tr>
<td>ReachSafety-Floats</td>
<td>172</td>
</tr>
<tr>
<td>ReachSafety-Heap</td>
<td>173</td>
</tr>
<tr>
<td>ReachSafety-Loops</td>
<td>156</td>
</tr>
<tr>
<td>ReachSafety-ProductLines</td>
<td>597</td>
</tr>
<tr>
<td>ReachSafety-Recursive</td>
<td>98</td>
</tr>
<tr>
<td>ReachSafety-Sequentialized</td>
<td>273</td>
</tr>
<tr>
<td>Systems_DeviceDriversLinux64_ReachSafety</td>
<td>2795</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>5692</strong></td>
</tr>
</tbody>
</table>
TABLE II: Number of violation witnesses produced by verifiers from the subject verification tasks

<table>
<thead>
<tr>
<th>Verifier</th>
<th>Ref.</th>
<th>Produced witnesses</th>
<th>Refined witnesses</th>
<th>Total witnesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>2LS</td>
<td>[39]</td>
<td>992</td>
<td>384</td>
<td>1 376</td>
</tr>
<tr>
<td>BLAST</td>
<td>[41]</td>
<td>778</td>
<td>202</td>
<td>980</td>
</tr>
<tr>
<td>CBMC</td>
<td>[31]</td>
<td>831</td>
<td>467</td>
<td>1 298</td>
</tr>
<tr>
<td>CEAGLE</td>
<td></td>
<td>619</td>
<td>426</td>
<td>1 045</td>
</tr>
<tr>
<td>CPA-BAM-BNB</td>
<td>[2]</td>
<td>851</td>
<td>175</td>
<td>1 026</td>
</tr>
<tr>
<td>CPA-KIND</td>
<td>[9]</td>
<td>263</td>
<td>193</td>
<td>456</td>
</tr>
<tr>
<td>CPA-SEQ</td>
<td>[19]</td>
<td>883</td>
<td>767</td>
<td>1 650</td>
</tr>
<tr>
<td>DEPTHK</td>
<td>[37]</td>
<td>1 159</td>
<td>305</td>
<td>1 464</td>
</tr>
<tr>
<td>ESBMC</td>
<td>[34]</td>
<td>653</td>
<td>148</td>
<td>801</td>
</tr>
<tr>
<td>ESBMC-FALSI</td>
<td>[34]</td>
<td>981</td>
<td>395</td>
<td>1 376</td>
</tr>
<tr>
<td>ESBMC-INCER</td>
<td>[34]</td>
<td>970</td>
<td>392</td>
<td>1 362</td>
</tr>
<tr>
<td>ESBMC-KIND</td>
<td>[20]</td>
<td>847</td>
<td>352</td>
<td>1 199</td>
</tr>
<tr>
<td>FORESTER</td>
<td>[27]</td>
<td>51</td>
<td>0</td>
<td>51</td>
</tr>
<tr>
<td>PREDATOR-HP</td>
<td>[30]</td>
<td>86</td>
<td>61</td>
<td>147</td>
</tr>
<tr>
<td>SMACK</td>
<td>[36]</td>
<td>871</td>
<td>632</td>
<td>1 503</td>
</tr>
<tr>
<td>SYMBIOTIC</td>
<td>[15]</td>
<td>927</td>
<td>411</td>
<td>1 338</td>
</tr>
<tr>
<td>SYMDIVINE</td>
<td>[29]</td>
<td>247</td>
<td>223</td>
<td>470</td>
</tr>
<tr>
<td>ULTIMATE AUTOMIZER</td>
<td>[25]</td>
<td>514</td>
<td>70</td>
<td>584</td>
</tr>
<tr>
<td>UKOJAK</td>
<td>[35]</td>
<td>309</td>
<td>67</td>
<td>376</td>
</tr>
<tr>
<td>UTAPIAN</td>
<td>[22]</td>
<td>338</td>
<td>70</td>
<td>408</td>
</tr>
</tbody>
</table>

Total            | 13 200 | 5 765       | 18 965            |