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

Vision

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.

Outline

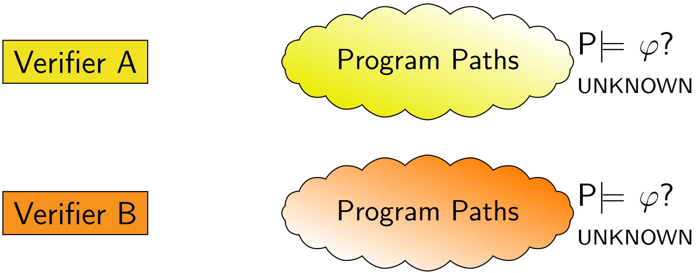
Dream is not utopian, will illustrate a few approaches ...

- 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

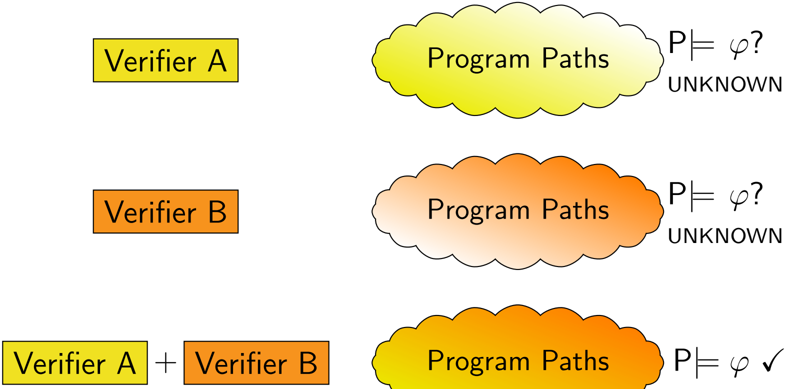
Facing Hard Verification Tasks

Given: Program P$\models \varphi$?



Facing Hard Verification Tasks

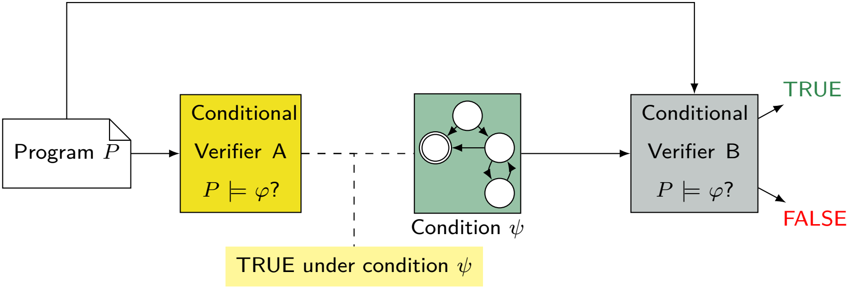
Given: Program $P \models \varphi$?



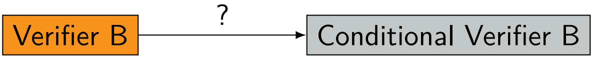
e.g., conditional model checking

Conditional Model Checking

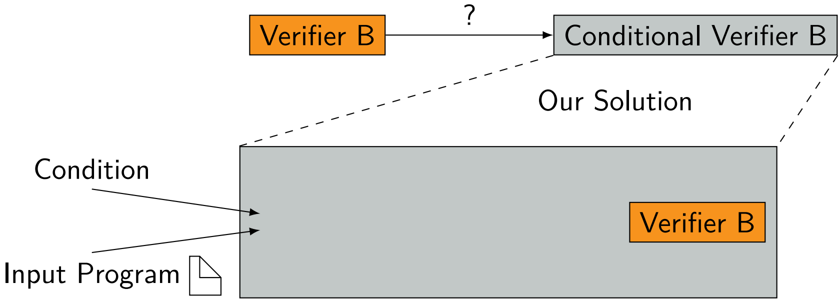
[Beyer/Henzinger/Keremoglu/Wendler FSE'12, DOI Link, Preprint Link]]



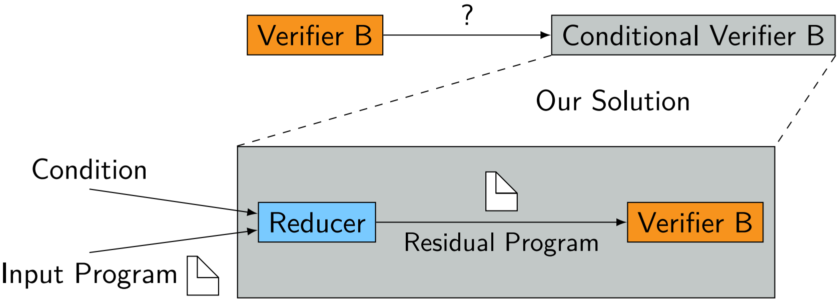
Reducer-Based Conditional Verifier Construction



Reducer-Based Conditional Verifier Construction



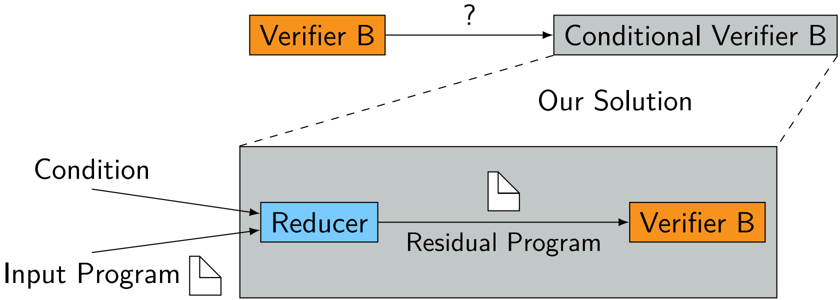
Reducer-Based Conditional Verifier Construction



Reducer (preprocessor)

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

Reducer-Based Conditional Verifier Construction



Reducer (preprocessor)

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

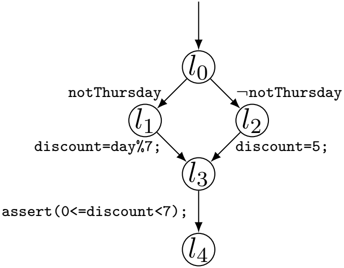
\+ Verifier-unspecific approach

\+ Many conditional verifiers possible

Example Program and Condition

Program

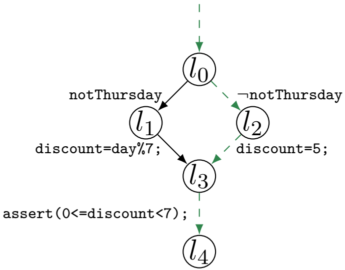
```
0: if(notThursday)
1:  discount=day%7;
   else
2:  discount=5;
3: assert(0<=discount<7);
4:
```



Example Program and Condition

Program

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

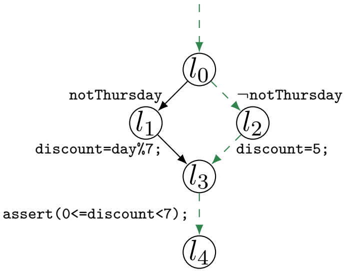


Verifier A only proofs else branch

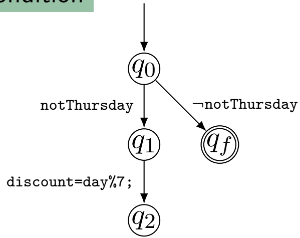
Example Program and Condition

Program

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

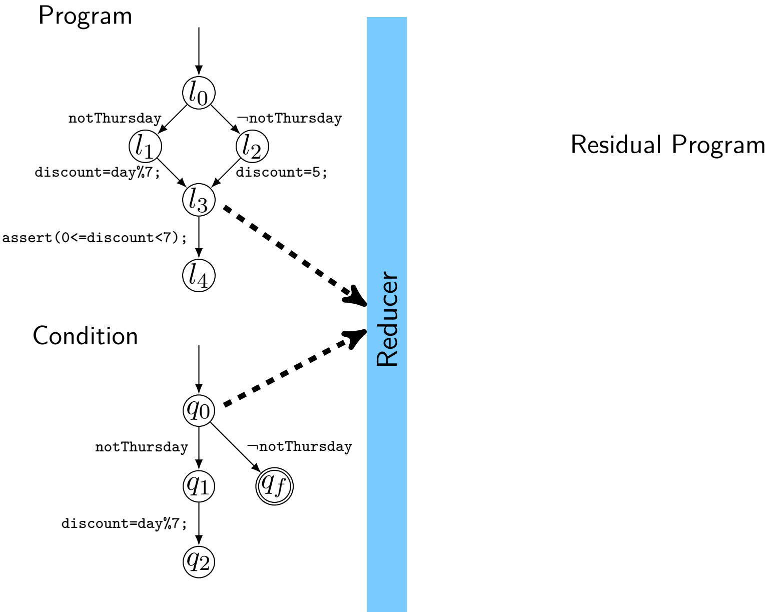


Condition

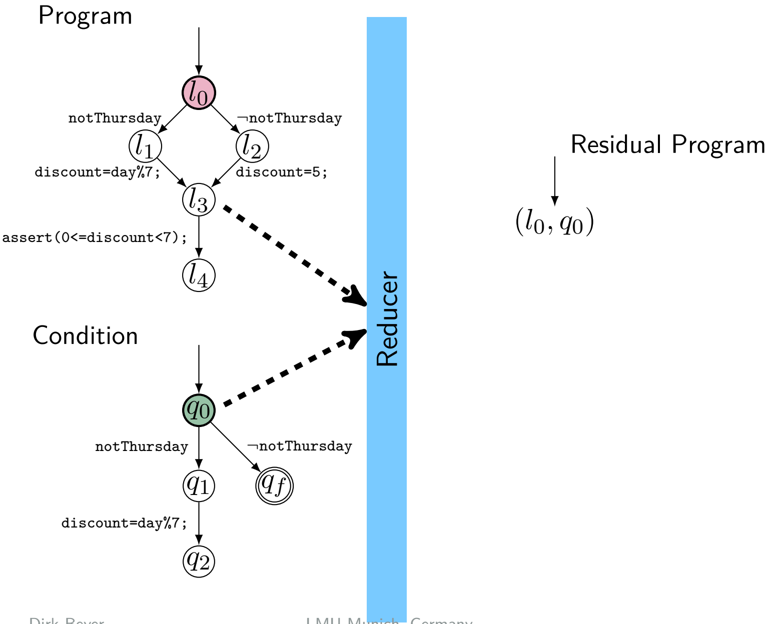


Verifier A only proofs else branch

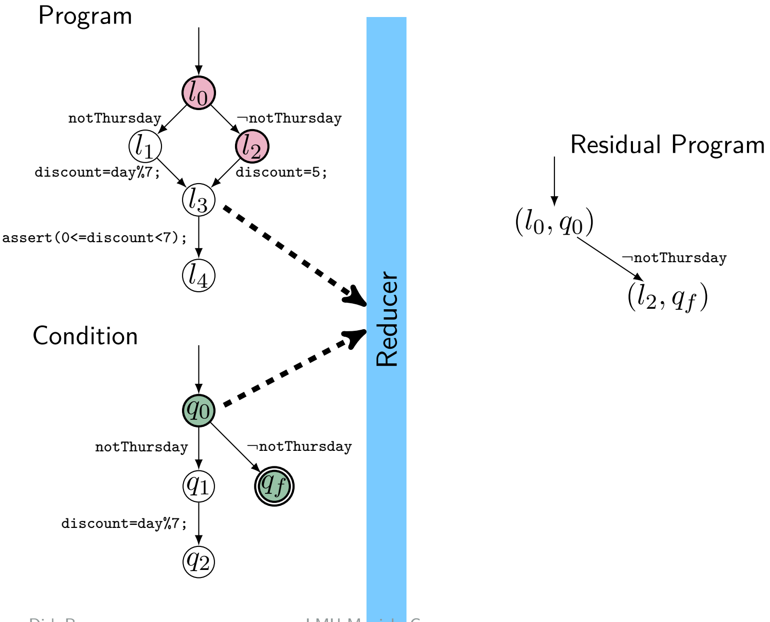
Reducer: Residual Program Construction



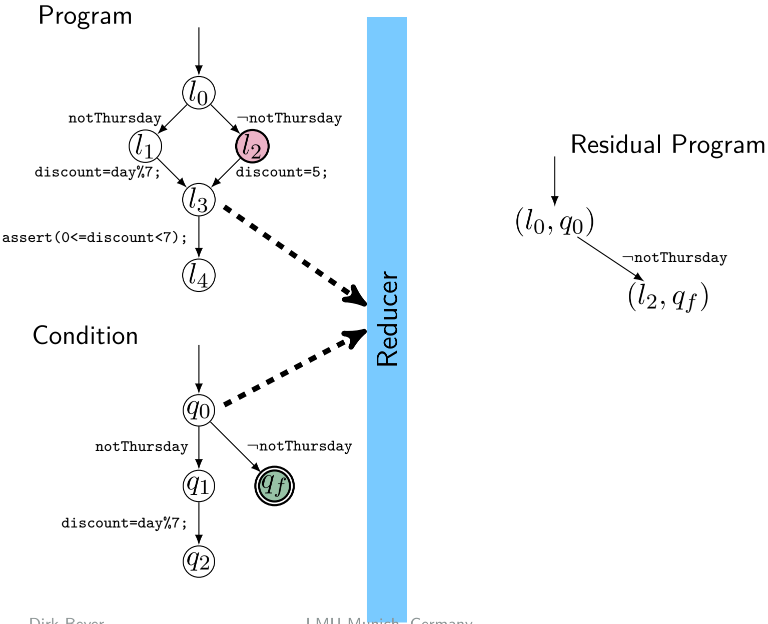
Reducer: Residual Program Construction



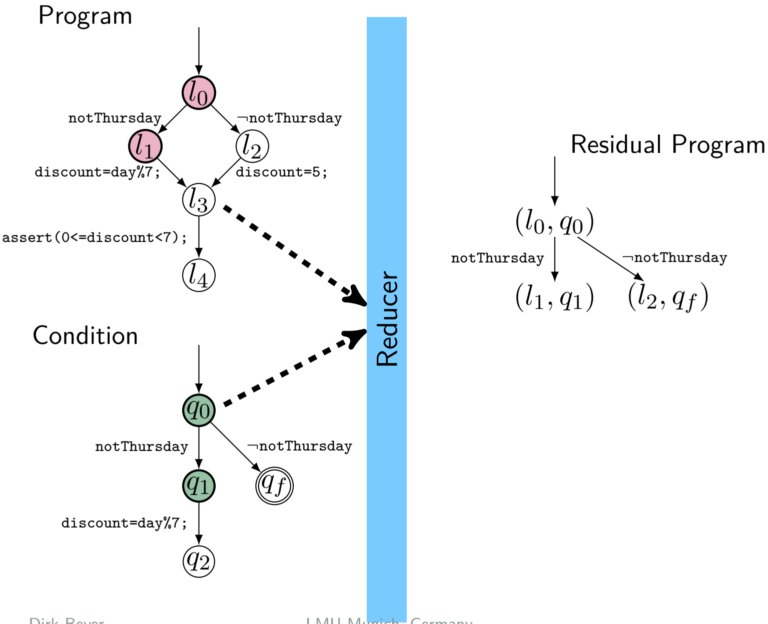
Reducer: Residual Program Construction



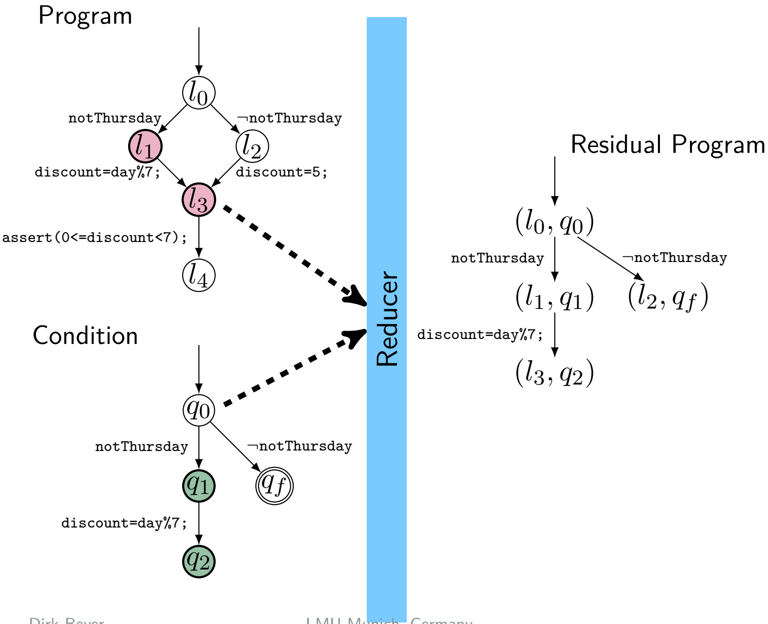
Reducer: Residual Program Construction



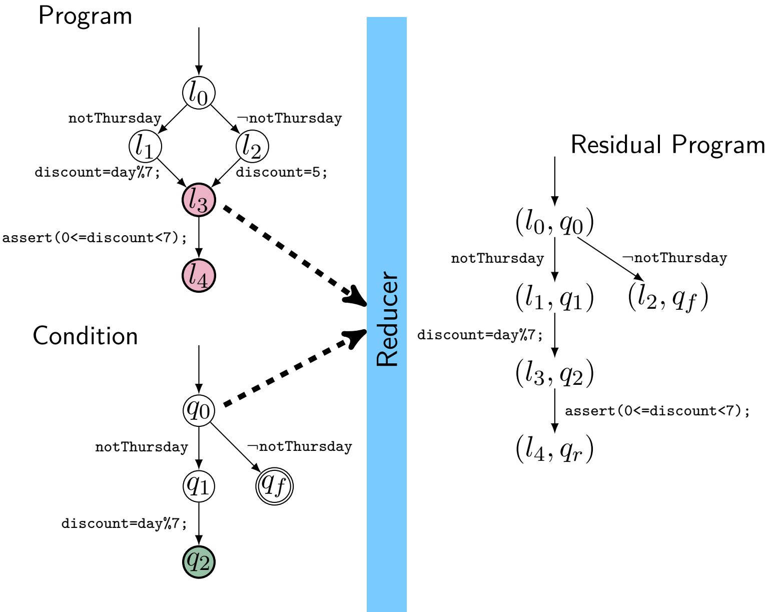
Reducer: Residual Program Construction



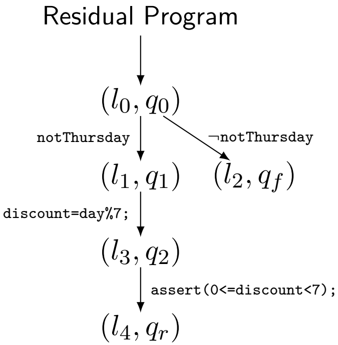
Reducer: Residual Program Construction



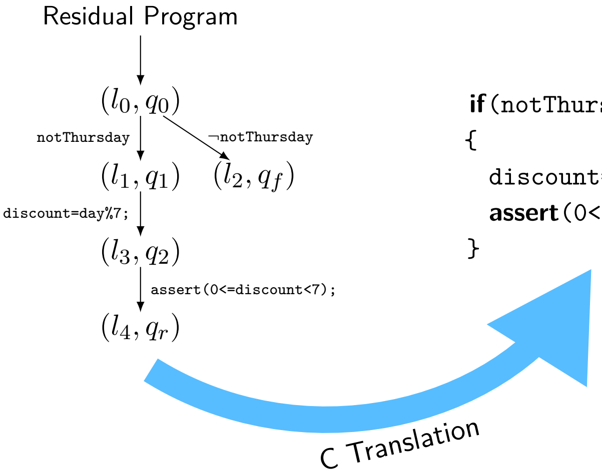
Reducer: Residual Program Construction



Reducer: C Transformation



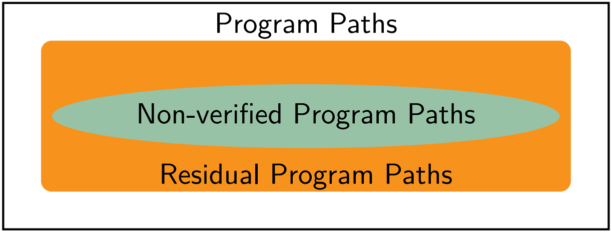
Reducer: C Transformation




```
if(notThursday)
{
  discount=day%7;
  assert(0<=discount<7);
}
```

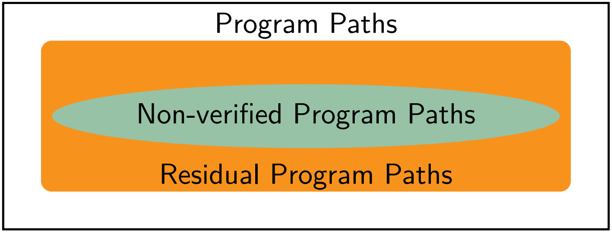
Reducer: Soundness

Residual Condition



Reducer: Soundness

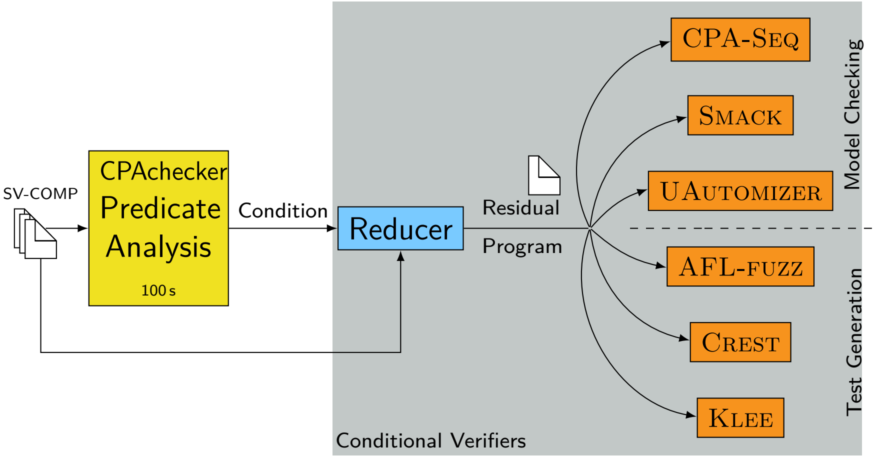
Residual Condition



Theorem

Presented reducer fulfills residual condition.

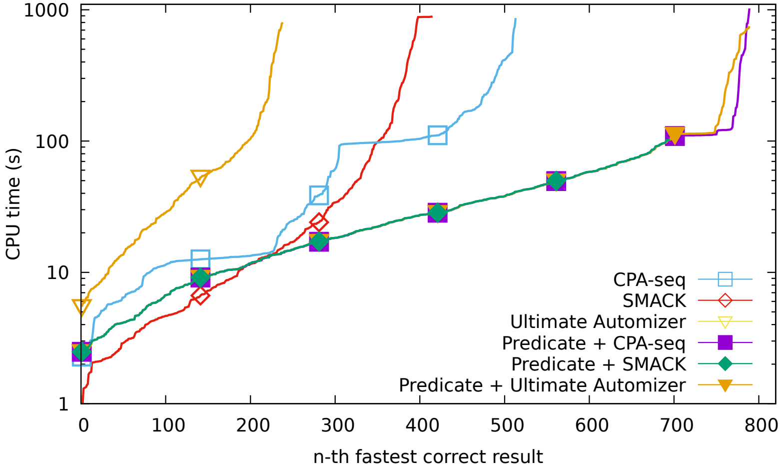
Evaluation Setup



Small Extract of Results

Task	R	CPA-Seq S	CPA-Seq t(s)	UAutomizer S	UAutomizer t(s)	Predicate +Reducer +CPA-Seq S	Predicate +Reducer +CPA-Seq t(s)	Predicate +Reducer +UAutomizer S	Predicate +Reducer +UAutomizer t(s)
P15l01	T	✗	910	✗	900	✓	120	✓	130
flood4	T	✗	910	✗	910	✓	450	✗	1100
newt3_6	F	✗	950	✗	490	✗	910	✓	260
P07l38	T	✗	950	✗	910	✗	1100	✓	470

Effectiveness on Hard Tasks

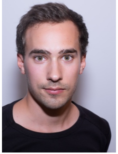


More Information:
Reducer-Based Construction of Conditional Verifiers

[Proc. ICSE 2018, pages 1182–1193, ACM. DOI Link, Preprint Link]

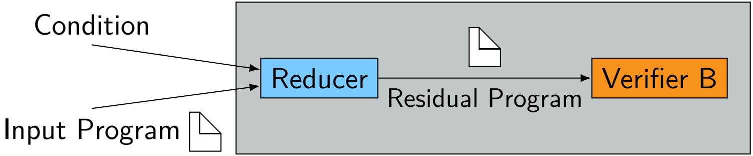
Dirk Beyer, Marie-Christine Jakobs, Thomas Lemberger, and Heike Wehrheim

LMU Munich, Germany and Paderborn University, Germany

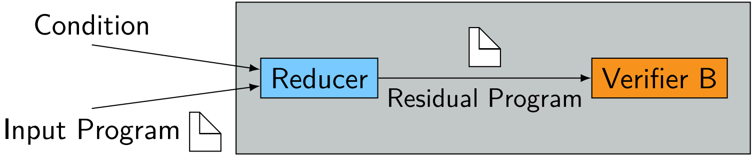
Conclusion — Approach 1

- Template-based conditional verifier construction



Conclusion — Approach 1

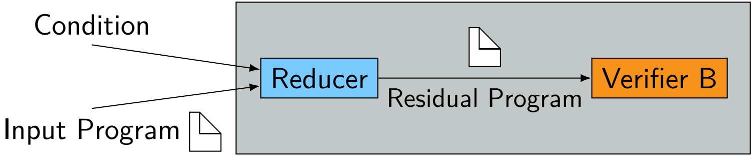
- Template-based conditional verifier construction



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

Conclusion — Approach 1

- Template-based conditional verifier construction

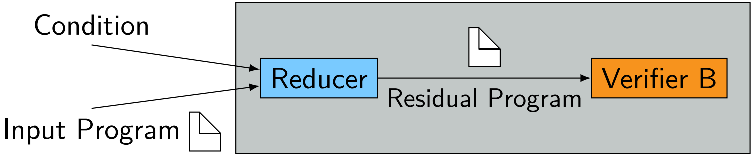


- One Reducer
  - Proven sound
  - Used in many conditional verifiers

- Effective on hard tasks for verifiers and test tools

Conclusion — Approach 1

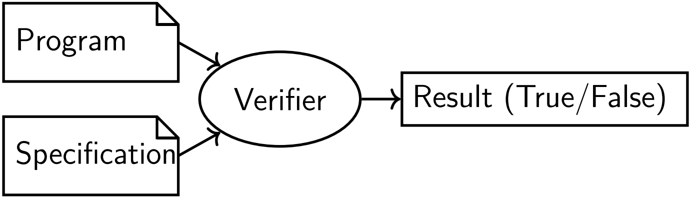
- Template-based conditional verifier construction



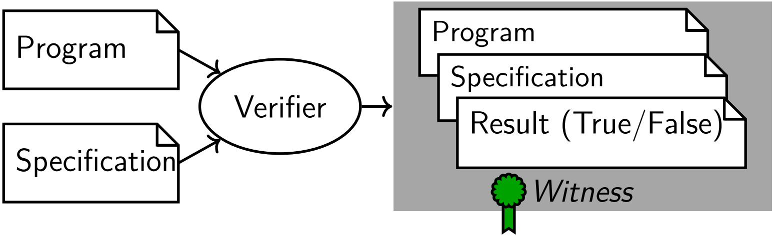
- One Reducer
  - Proven sound
  - Used in many conditional verifiers
- Effective on hard tasks for verifiers and test tools
- Future Work
  - More reducers
  - Using conditions from other tools

Approach 2: Cooperative Verification by Verification Witnesses

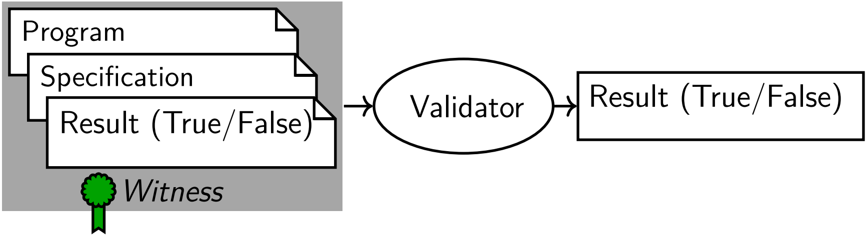
Software Verification



Software Verification with Witnesses

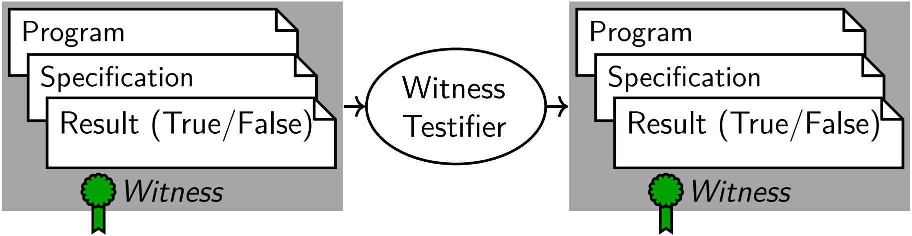


Witness Validation



- Validate untrusted results
- Easier than full verification

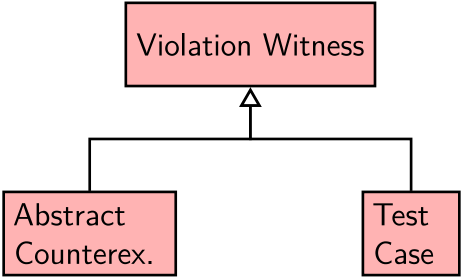
Stepwise Refinement



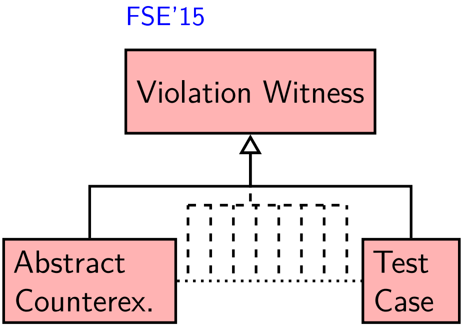
Violation Witnesses

Violation Witness

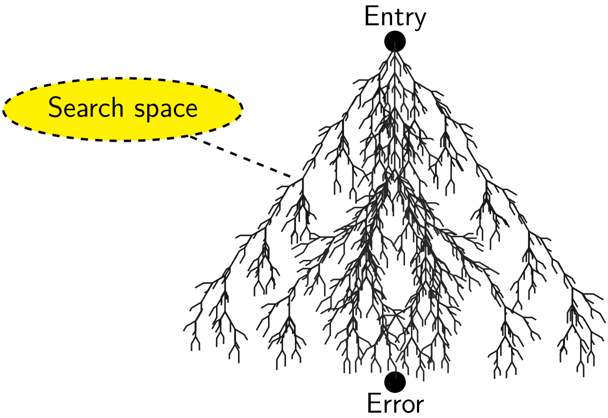
Violation Witnesses



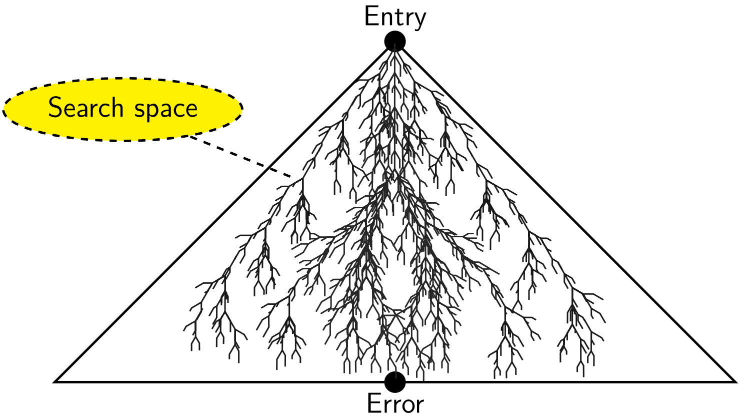
Violation Witnesses



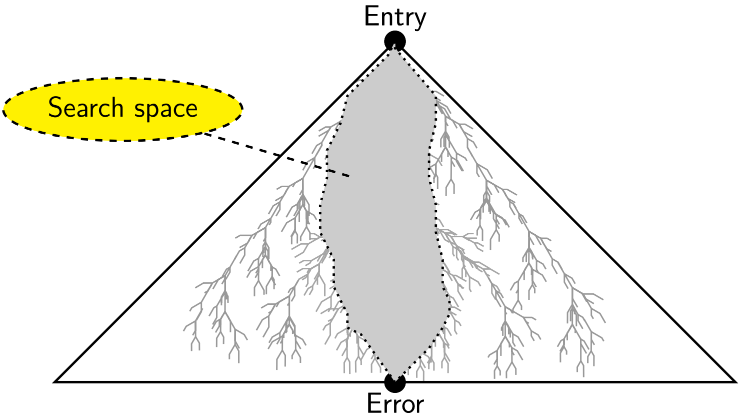
Search-Space Reduction for Stepwise Witness Refinement



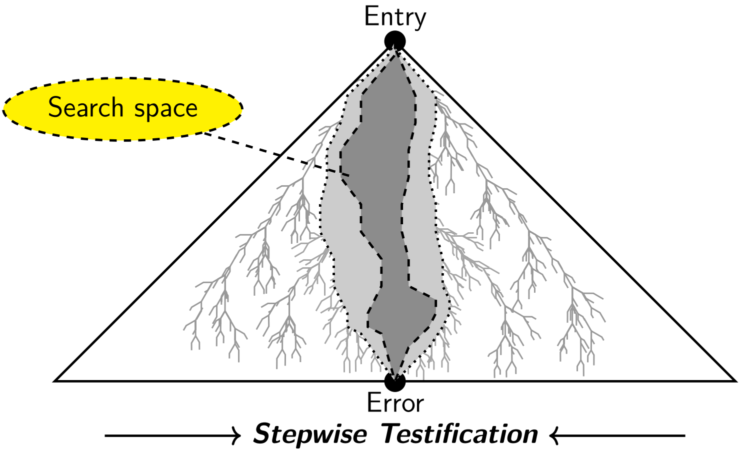
Search-Space Reduction for Stepwise Witness Refinement



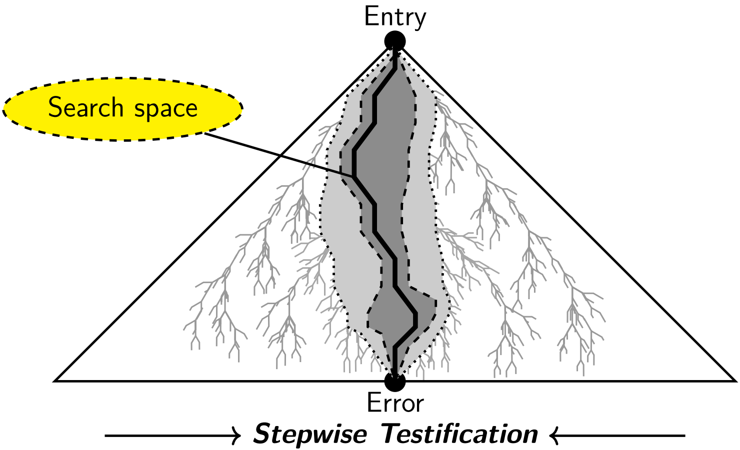
Search-Space Reduction for Stepwise Witness Refinement



Search-Space Reduction for Stepwise Witness Refinement



Search-Space Reduction for Stepwise Witness Refinement



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**

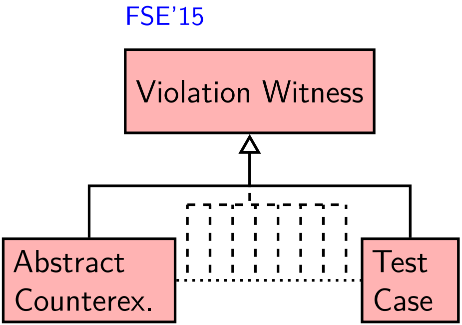
Open Problems

1. **Standardized way** to document verification results to enhance engineering processes **required**
2. **Difficult to establish trust** in results from an untrusted verifier

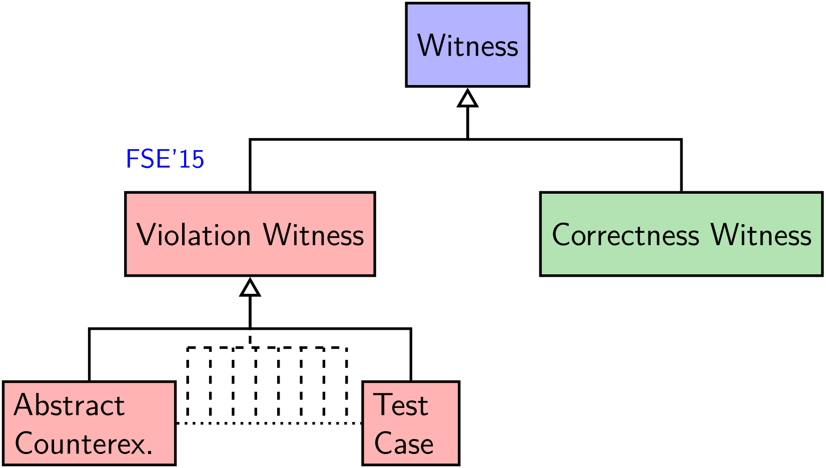
Open Problems

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**

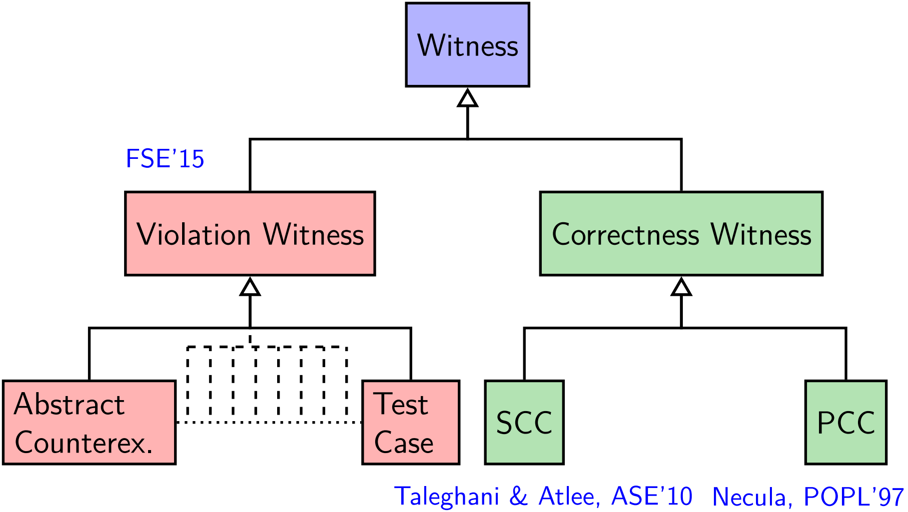
Verification Witnesses: Classification



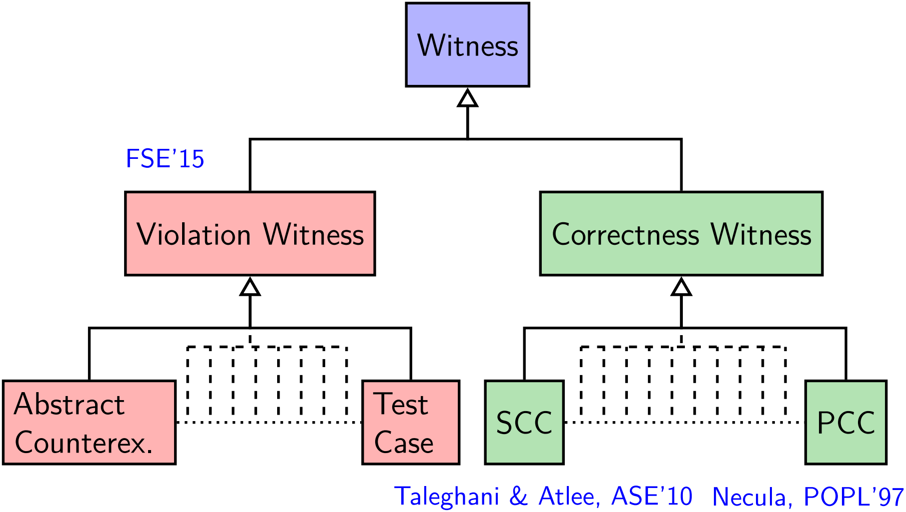
Verification Witnesses: Classification



Verification Witnesses: Classification



Verification Witnesses: Classification



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**

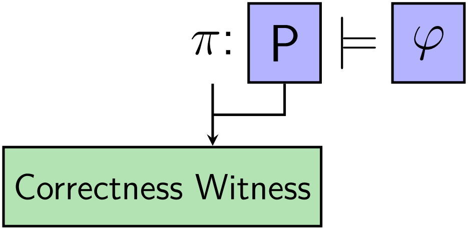
Correctness Witnesses

$$\boxed{P} \models \boxed{\varphi}$$

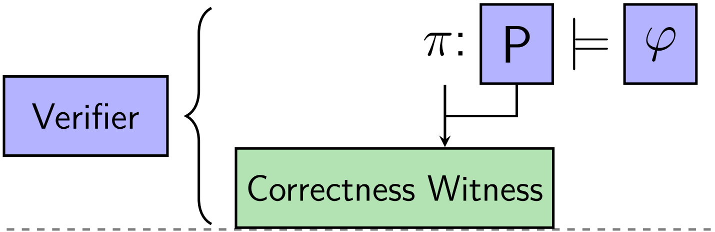
Correctness Witnesses

$$\pi : \boxed{\mathsf{P}} \models \boxed{\varphi}$$

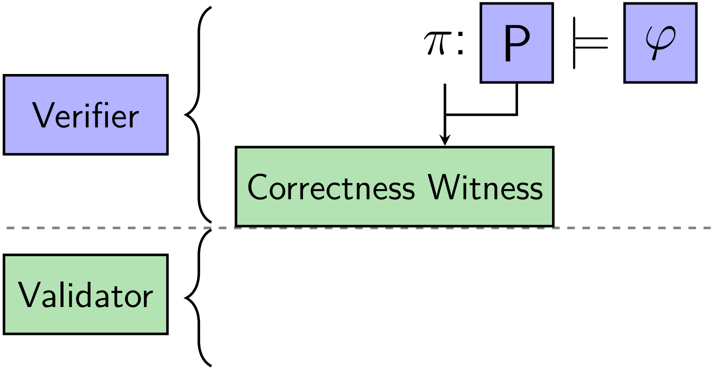
Correctness Witnesses



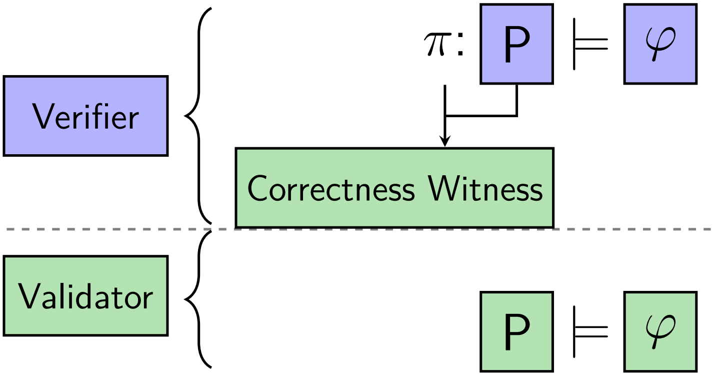
Correctness Witnesses



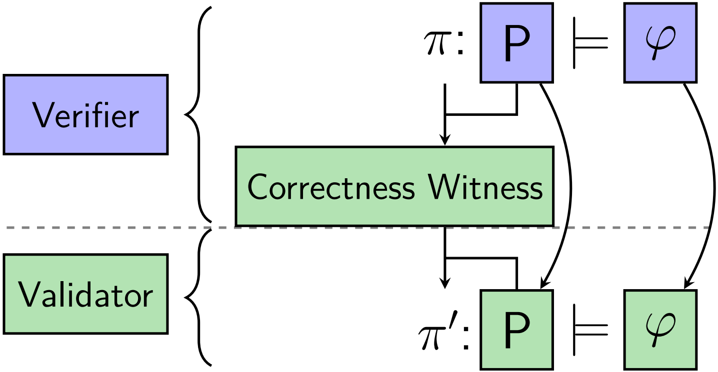
Correctness Witnesses



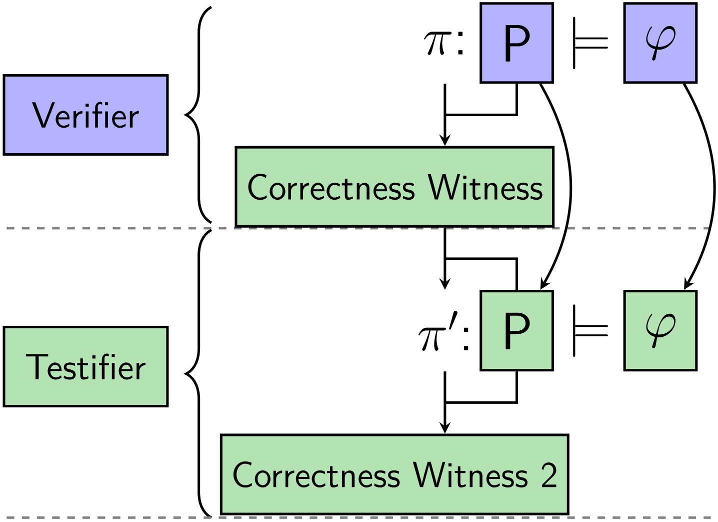
Correctness Witnesses



Correctness Witnesses



Correctness Witnesses



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**

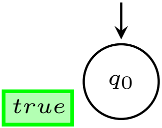
Witness Automata

- 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;
}
```

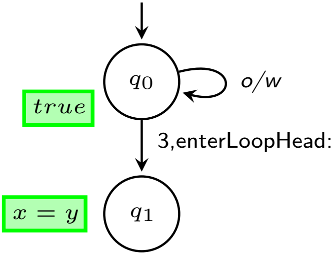
Example: Inject Invariants




```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;
}
```

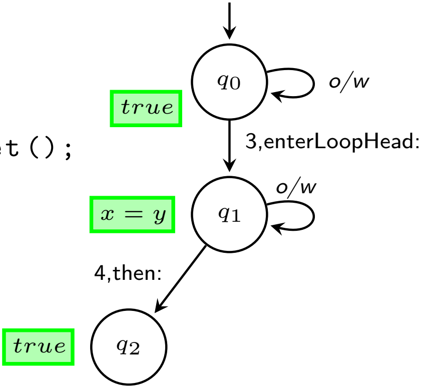
Example: Inject Invariants

```
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;
}
```



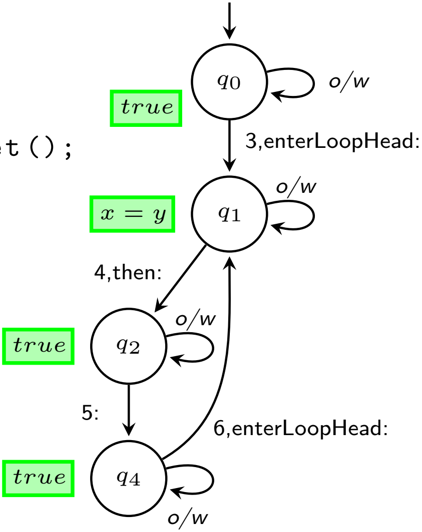
Example: Inject Invariants

```
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;
}
```



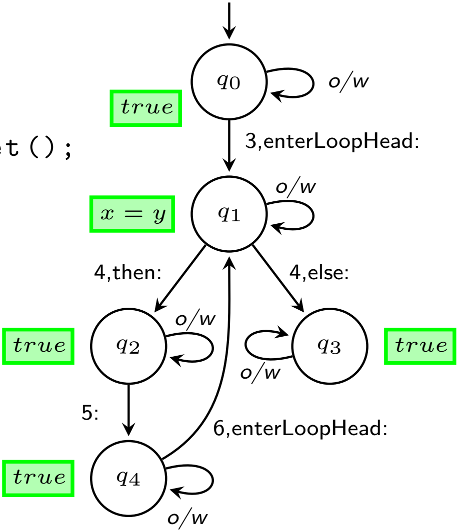
Example: Inject Invariants

```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

```
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;
}
```



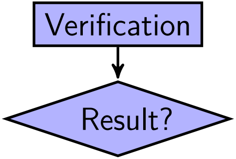
Producing and Consuming Witnesses: SV-COMP

Table 8: Confirmation rate of witnesses

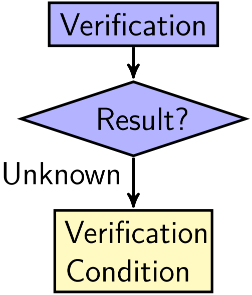
Result	TRUE			FALSE		
	Total	Confirmed	Unconfirmed	Total	Confirmed	Unconfirmed
UAUTOMIZER	3 558	3 481	77	1 173	1 121	52
SMACK	2 947	2 695	252	1 929	1 768	161
CPA-SEQ	3 357	3 078	279	2 342	2 315	27

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].

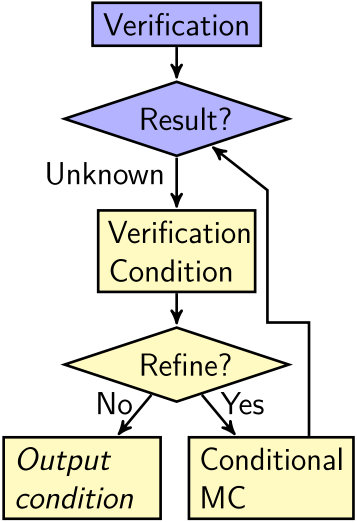
Stepwise Refinement: Classification



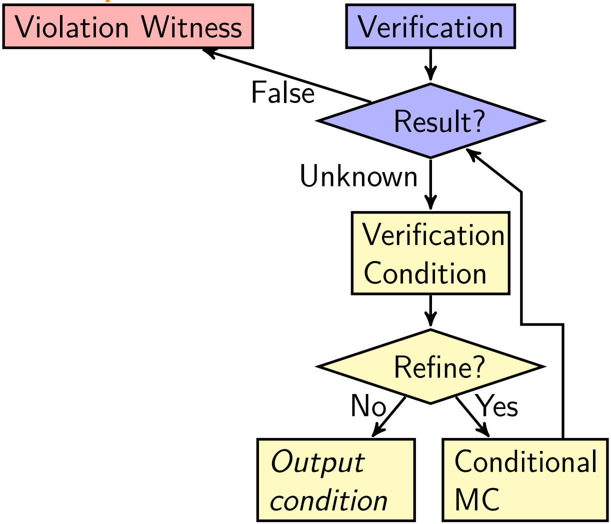
Stepwise Refinement: Classification



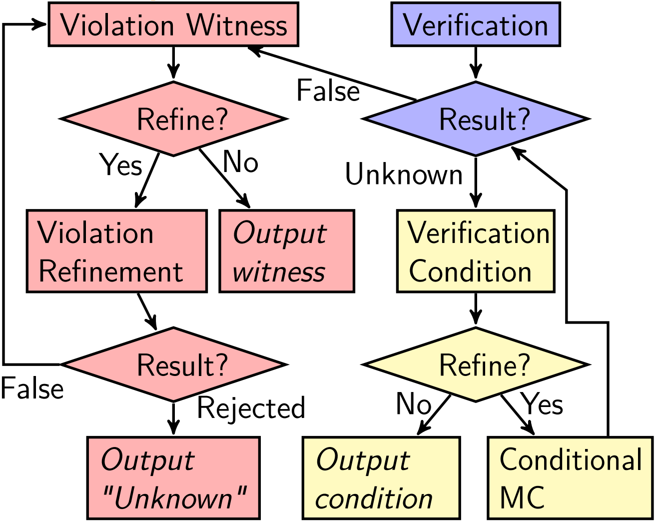
Stepwise Refinement: Classification



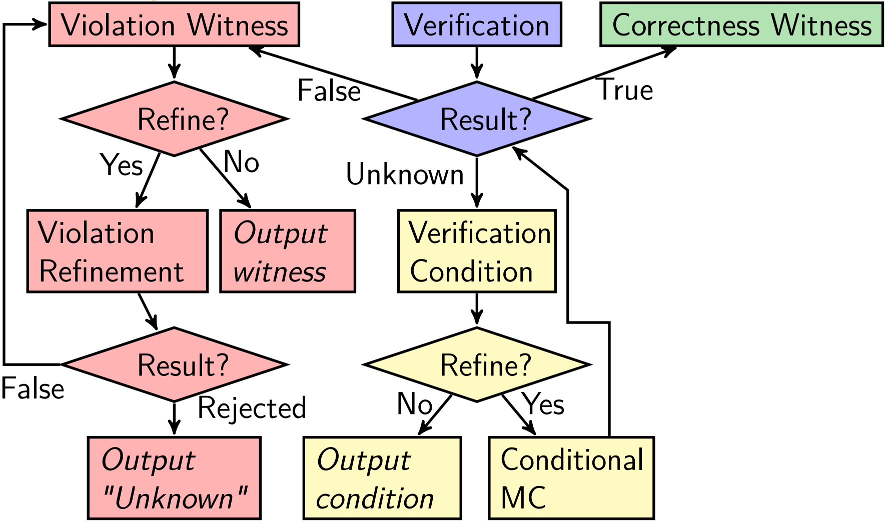
Stepwise Refinement: Classification



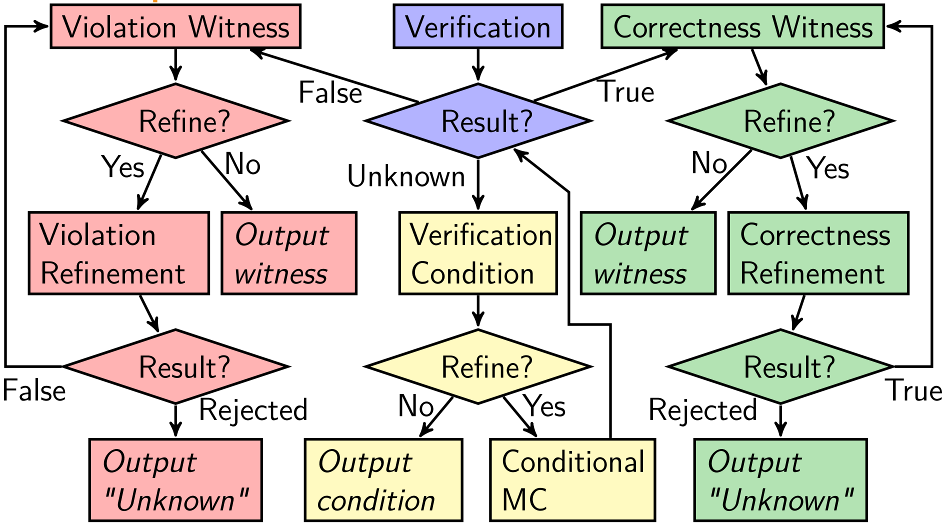
Stepwise Refinement: Classification



Stepwise Refinement: Classification



Stepwise Refinement: Classification



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.
$\Rightarrow$ Automatic verification.

Software contains bugs.
$\Rightarrow$ Automatic verification.

But software contains bugs.

Software contains bugs.
$\Rightarrow$ Automatic verification.

But software contains bugs.
$\Rightarrow$ Automatic validation of results.

Software contains bugs.
$\Rightarrow$ Automatic verification.

But software contains bugs.
$\Rightarrow$ Automatic validation of results.

But software contains bugs.

Software contains bugs.
$\Rightarrow$ Automatic verification.

But software contains bugs.
$\Rightarrow$ Automatic validation of results.

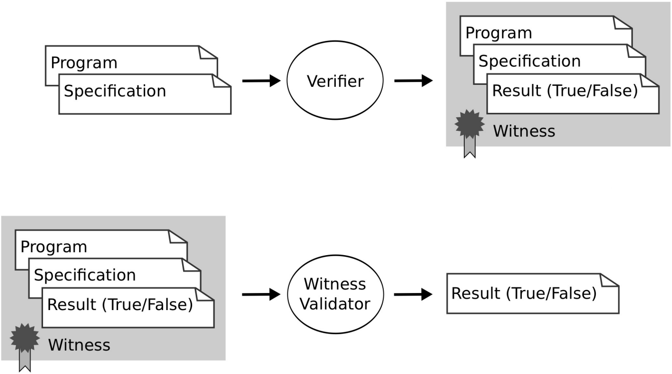
But software contains bugs.
$\Rightarrow$ Execution as proof.

Old Idea:
Tests from Counterexamples

- “Generating Tests from Counterexamples”
  D. Beyer, A. J. Chlipala, T. A. Henzinger, R. Jhala, R. Majumdar
  ICSE 2004 DOI Link, Preprint Link
- “Test-Input Generation with Java PathFinder”
  W. Visser, C. S. Păsăreanu, S. Khurshid
  ISSTA 2004 DOI Link
- Influencial papers, but:
- Problem: No exchange format; proprietary technology, proprietary format for test vector

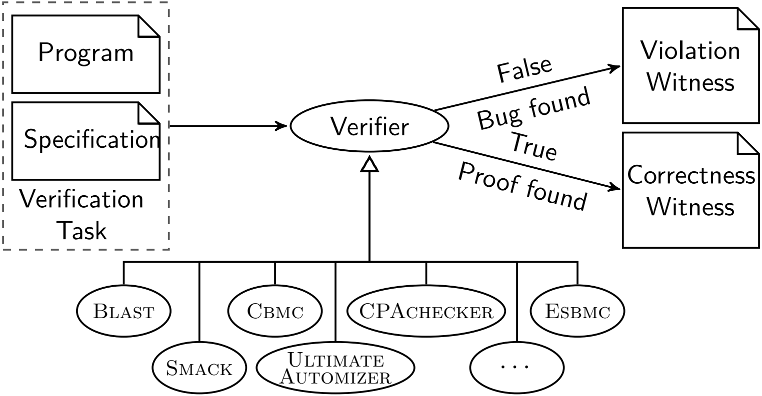
Witness Validation

- Problem: Confidence in verifiers
- Approach: Witness validation



[Beyer/Dangl/Dietsch/Heizmann/Stahlbauer FSE'15]

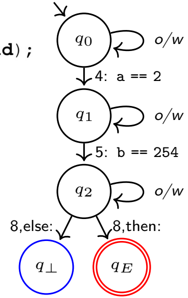
Witness Creation



Violation Witness Format: Witness Automaton

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

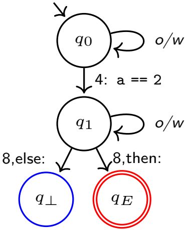
```c
extern void __VERIFIER_error(void);
extern unsigned char __VERIFIER_nondet_uchar(void);
int main(void) {
  unsigned char a = __VERIFIER_nondet_uchar();
  unsigned char b = __VERIFIER_nondet_uchar();
  unsigned char sum = a + b;
  unsigned char mean = sum / 2;
  if (mean < a / 2) {
    __VERIFIER_error();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



Witness Refinement

- Problem 1: Abstract witnesses

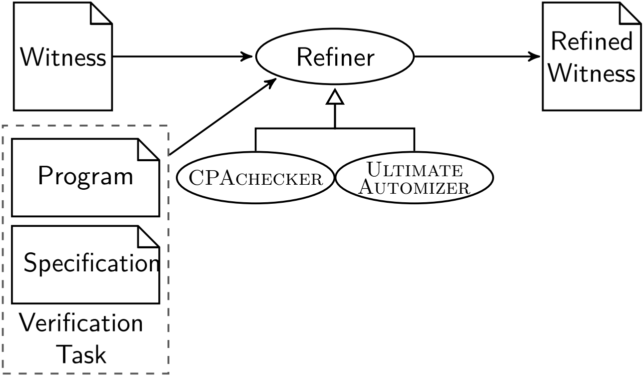
```
1 extern void __VERIFIER_error(void);
2 extern unsigned char __VERIFIER_nondet_uchar(void);
3 int main(void) {
4   unsigned char a = __VERIFIER_nondet_uchar();
5   unsigned char b = __VERIFIER_nondet_uchar();
6   unsigned char sum = a + b;
7   unsigned char mean = sum / 2;
8   if (mean < a / 2) {
9     __VERIFIER_error();
10  }
11  return 0;
12 }
```



`b == ?`

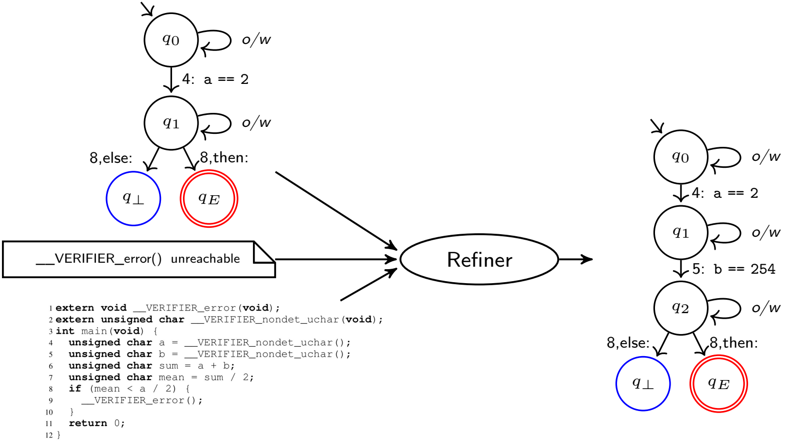
Witness Refinement

- Problem 1: Abstract witnesses
- Solution: Witness refinement



Witness Refinement

- Problem 1: Abstract witnesses
- Solution: Witness refinement



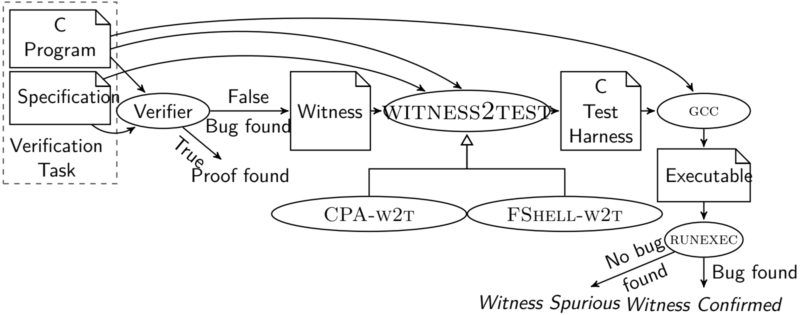

```c
extern void __VERIFIER_error(void);
extern unsigned char __VERIFIER_nondet_uchar(void);
int main(void) {
  unsigned char a = __VERIFIER_nondet_uchar();
  unsigned char b = __VERIFIER_nondet_uchar();
  unsigned char sum = a + b;
  unsigned char mean = sum / 2;
  if (mean < a / 2) {
    __VERIFIER_error();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

Witness Validation

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

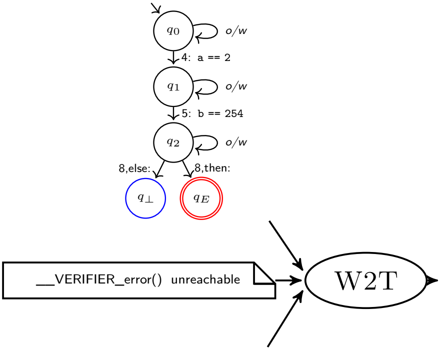
Execution-based Witness Validation

Execution-based Witness Validation



Execution-based Witness Validation

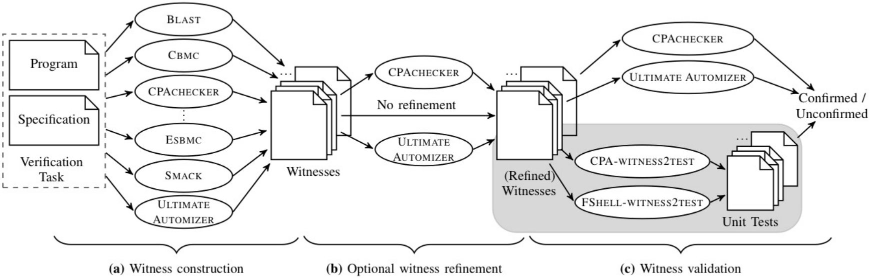
- Build executable counterexample from witness
- State-space guards $\mapsto$ input variables/functions




```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;
}
```

```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;
}
```

Full Workflow



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

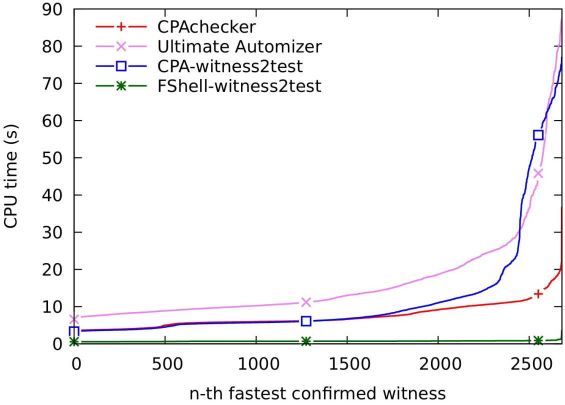
- 18 965 witnesses in total
- Not only increase of confidence,
  but also increase of overall effectivity

TABLE III: Validation results of static/dynamic validators

	Static	Dynamic	Union
Confirmed results	12 671	8 702	14 434
Incorrectly confirmed results	21	6	27

Time Performance

- Time comparison over 2 680 witnesses that all validators confirmed



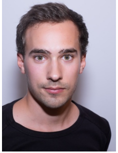
More Information:

Tests from Witnesses: Execution-Based Validation of Verification Results

[Proc. TAP 2018, pages 3–23, Springer. DOI Link, Preprint Link]

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

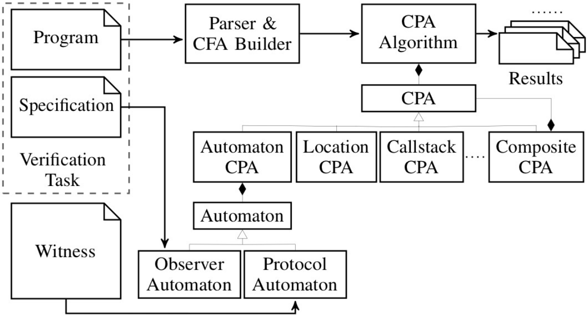
Overall Conclusion

- 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

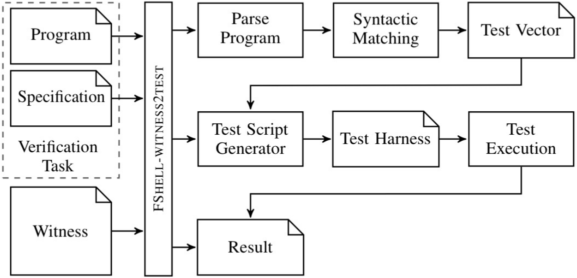
Thank You!

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 I: Subject verification tasks from the SV-BENCHMARKS repository

Sub-category	**Number of verification tasks**
ReachSafety-Arrays	135
ReachSafety-BitVectors	50
ReachSafety-ControlFlow	94
ReachSafety-ECA	1 149
ReachSafety-Floats	172
ReachSafety-Heap	173
ReachSafety-Loops	156
ReachSafety-ProductLines	597
ReachSafety-Recursive	98
ReachSafety-Sequentialized	273
Systems_DeviceDriversLinux64_ReachSafety	2 795
Total	5 692

Witness Set

TABLE II: Number of violation witnesses produced by verifiers from the subject verification tasks

Verifier	Ref.	Produced witnesses	Refined witnesses	Total witnesses
2LS	[39]	992	384	1 376
BLAST	[41]	778	202	980
CBMC	[31]	831	467	1 298
CEAGLE		619	426	1 045
CPA-BAM-BNB	[2]	851	175	1 026
CPA-KIND	[9]	263	193	456
CPA-SEQ	[19]	883	767	1 650
DEPTHK	[37]	1 159	305	1 464
ESBMC	[34]	653	148	801
ESBMC-FALSI	[34]	981	395	1 376
ESBMC-INCR	[34]	970	392	1 362
ESBMC-KIND	[20]	847	352	1 199
FORESTER	[27]	51	0	51
PREDATORHP	[30]	86	61	147
SKINK	[13]	30	25	55
SMACK	[36]	871	632	1 503
SYMBIOTIC	[15]	927	411	1 338
SYMDIVINE	[29]	247	223	470
ULTIMATE AUTOMIZER	[25]	514	70	584
UKOJAK	[35]	309	67	376
UTAIPAN	[22]	338	70	408
Total		13 200	5 765	18 965