Correctness Witness Validation using Predicate Analysis

Maximilian Wiesholler

June 5, 2019

Outline

- Motivation
- Background
- Concepts
- Evaluation
- Outlook

Motivation

Validation of correctness witnesses can increase **trust** in the verification result.

Only two validators for correctness witnesses exist:

- *k*-Induction-based validator in CPAchecker [1]
- Automata-based validator in Ultimate Automizer [1]

Goal: Use predicate analysis in CPAchecker as new validator.

Motivation

Approaches for predicate analysis as new validator:

- Reuse witness invariants for the initial precision
- Define witness invariants as additional verification goal

Outline

- Motivation
- **Background**
- Concepts
- Evaluation
- Outlook

Background: Correctness Witness

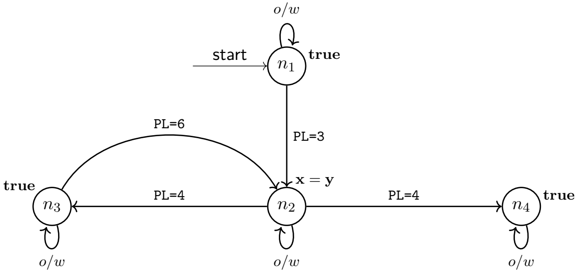
Correctness witness

- is a partial correctness proof when a program fulfills a specification
- can contain invariants which have been found during the verification of the program
- Syntactic level: Witness is stored in an exchangeable format
- Semantic level: Witness is represented by an observer automaton

Background: Example of a Correctness Witness Automaton

Example program and its corresponding correctness witness automaton

```
int main(void) {
  unsigned int x =
              __VERIFIER_nondet_uint();
  unsigned int y = x;
  while (x < 1024) {
    x++;
    y++;
  }
  if (x != y){
    ERROR: return 1;
  }
  return 0;
}
```



Background: Witnesses on the Syntactic Level

Witness file

- includes general information in the header:
  - witness type (violation | **correctness**)
  - producer
  - proved specification
  - program hash
  - ...
- contains the witness automaton

Background: Predicate Analysis

Predicate analysis

Using **predicate abstraction** in a **reachability analysis** with **CEGAR**.

Predicate abstraction

Computation of predicates (boolean expressions) over program variables.

Motivation: Abstract concrete states and their assigned variables.

Predicates can be solved by a SMT solver.

Background: Reachability Analysis

Reachability analysis

A set of reachable abstract states is computed to create an abstract model of the program.

The abstract model has the form of an abstract reachability graph (ARG) [3].

The computation of abstract states is guided by the current precision [3].

Precision

The precision describes the current stored information.

Background: CEGAR

Counterexample-Guided Abstraction Refinement (CEGAR)

Motivation: Find a suitable precision during the verification that is neither too coarse nor too accurate.

CEGAR iteratively calls the reachability analysis with a refined precision

CEGAR terminates when a counterexample is satisfiable or no abstract state violates the specification.

Background: Predicate Precision

Predicate precision

A predicate precision π is a mapping from program locations to sets of predicates over the program variables [2].

$\pi(l)$ describes the predicate precision at program location l.

Outline

- Motivation
- Background
- **Concepts**
- Evaluation
- Outlook

Concepts: Location Invariants

Location invariant

A location invariant is a tuple (l, θ) where l denotes the CFA location and θ denotes the invariant.*

I denotes the set of location invariants.

I can be derived from the correctness witness.

*An implementation of a location invariant already exists for *k*-induction-based validator in CPAchecker

Concepts: Location Invariants

I can be received in CPAchecker by using the following steps:

- Parsing the correctness witness file into a witness automaton*
- Performing a reachability analysis on the witness automaton*
- Extracting the invariants and their corresponding CFA locations from set reached

To guarantee soundness: If (l, θ) is received so that $l = l'$ holds for $(l', \theta') \in I$, $(l, \theta \wedge \theta')$ is created and (l', θ') removed.

*Implementations already exist for *k*-induction-based validator in CPAchecker

Concepts: Precision Reuse with Witness Invariants

Motivation: Witness invariants as precision facts might decrease number of refinements and CPU time.

Approach:

- get I from the correctness witness
- for each $i \in I$: convert θ from i into a predicate ρ
- add ρ to the location, function or global predicate precision
  - CFA location is known because of l

Atomic predicate: Splitting the predicate would create components that are no boolean formulas anymore.

Getting atomic predicates: Split predicates into predicate components until all components have an atomic form.

Concepts: Invariants Specification Automaton (ISA)

Motivation: Validate the invariants in the correctness witness.

Invariants specification automaton (ISA)

- Is a **control automaton** which is constructed by using the invariants from the correctness witness.
- Is used as an additional verification goal.

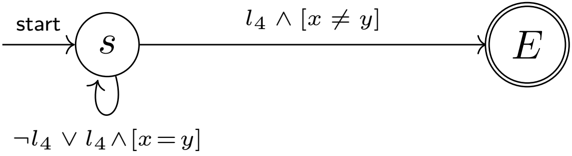
Three ISA concepts are presented.

Concepts: Two States Invariants Specification Automaton (ISA^{2S})

ISA^{2S} has **two states**: an initial state and an error state.

Example:

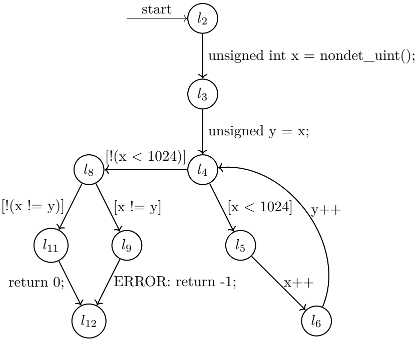
$I = \{(l_4, x = y)\}$



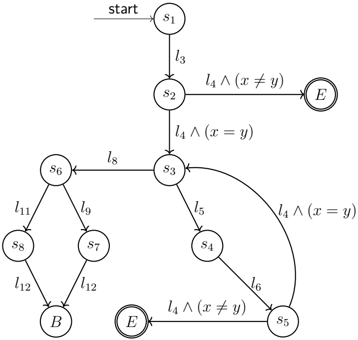
Concepts: CFA based Invariants Specification Automaton (ISA^{CFA})

ISA^{CFA} structure refers to structure of the CFA of the program.
Motivation: better performance expected compared to ISA^{2S}.

Example:
$I = \{(l_4, x = y)\}$



CFA



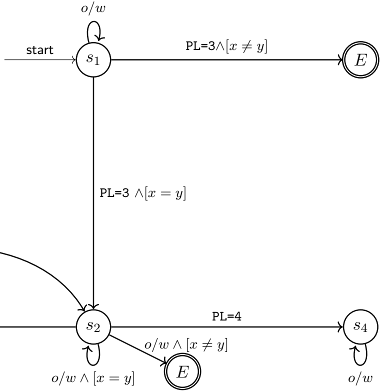
ISA^{CFA}

Concepts: Witness Invariants Specification Automaton (ISA^{WI})

ISA^{WI} extends the original witness with invariant-based assumptions.

Computation of I not required.

Example:



Concepts: Correctness Witness Types

Three types of correctness witnesses are distinguished:

non-trivial-witness	Witness has states with *non-trivial*-Invariants and at least one of these states is reachable in the analysis
true-witness	Witness has no states labeled with *non-trivial*-Invariants.
hidden-true-witness	Witness has states with *non-trivial*-Invariants but each of these states is unreachable in the analysis

Concepts: Correctness Witness Types

Consequences of correctness witness types:

A *true*-witness or *hidden-true*-witness leads to an empty set of location invariants.

	non-trivial-witness	*true*-witness	*hidden-true*-witness
π_0	$\exists l \in L.\pi_0(l) \neq \emptyset$	$\forall l \in L.\pi_0(l) = \emptyset$	$\forall l \in L.\pi_0(l) = \emptyset$
ISA^{2S}	1	2	2
ISA^{CFA}	1	2	2
ISA^{WI}	1	2	3

1: transitions into error states exist

2: transitions into error states do not exist

3: transitions into error states exist but are not reachable

Outline

- Motivation
- Background
- Concepts
- **Evaluation**
- Outlook

Evaluation: Specification and Benchmarks

Specification

Unreachability of error function __VERIFIER_error().

Bechmarks

- Tasks taken from SV-COMP 2019 from categories ReachSafety* and SoftwareSystems**
- Excluding all tasks that violate the specification (focus is on correctness witnesses)
- In total **4668** tasks

*without subcategory ReachSafety-Recursive
**only with subcategory Systems_DeviceDriversLinux64_ReachSafety

Evaluation: System Settings

Benchmark environment

- Machines: 8 core CPUs with 3.40 GHz (Intel Xeon E3-1230 v5) and 33 GB of RAM memory
- Operating system: Ubuntu 18.04 (64 Bit)
- Time limit: 900 s
- Memory limit: 15 GB
- Requirement of 8 CPU cores for a task

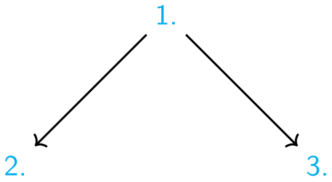
Evaluation: Interesting Questions

- Does reuse of invariants-based predicates lead to less CPU time and fewer CEGAR refinements?
- Is predicate analysis able to validate the original specification and invariants of
  - its own correctness witnesses?
  - correctness witnesses from *k*-induction in CPAchecker and from Ultimate Automizer?

Evaluation: Outline

The evaluation can be divided into three parts:

1. Producing correctness witnesses
2. Initialize predicate precision with invariant-based predicates
3. Validate correctness witnesses by using an ISA



Evaluation: Producing Correctness Witnesses

status	all	**correct-true**	correct-false	incorrect-true	incorrect-false	timeout	error	other
pA-Verification	4668	**2396**	0	0	5	1533	696	38
kI-Verification	4668	**2599**	0	0	1	1784	134	150
uA-Verification	4668	**2688**	0	0	2	1684	90	204

Evaluation: Producing Correctness Witnesses

$\text{Witnesses}^{\text{no-true}}$	Correctness witnesses are no *true*-witnesses.
$\text{Witnesses}^{\text{no-true}^{+h}}$	Correctness witnesses are no *true*-witnesses and no *hidden-true*-witnesses.

Evaluation: Outline

- Producing correctness witnesses
- **Initialize predicate precision with invariant-based predicates**
- Validate correctness witnesses by using an ISA

Evaluation: Witness Invariants for Precision Reuse

Six different configurations:

		atomic predicates	
		no	yes
scope	local	*pA*-Validation-PR$^{\text{lo}}$	*pA*-Validation-PR$^{\text{lo+a}}$
	function	*pA*-Validation-PR$^{\text{fu}}$	*pA*-Validation-PR$^{\text{fu+a}}$
	global	*pA*-Validation-PR$^{\text{gl}}$	*pA*-Validation-PR$^{\text{gl+a}}$

Evaluation: *pA*-Witnesses$^{\text{no-true}^{+h}}$ for Precision Reuse

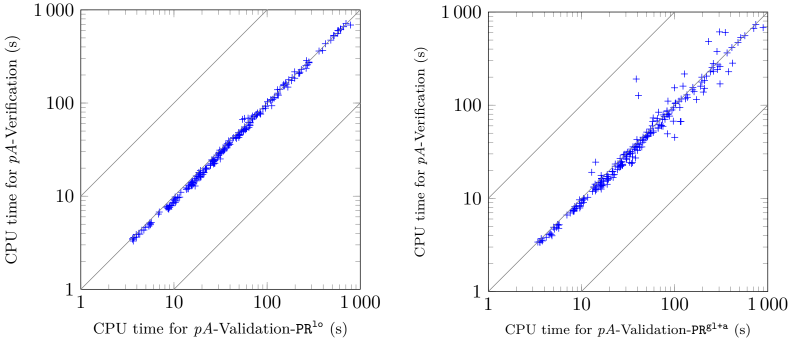
pA-Validation-PR initialized with invariants from **223** *pA*-Witnesses$^{\text{no-true}^{+h}}$.(∗)

pA-Validation-PR	*pA*-Validation-PR^{lo}	*pA*-Validation-PR^{lo+a}	*pA*-Validation-PR^{fu}	*pA*-Validation-PR^{fu+a}	*pA*-Validation-PR^{gl}	*pA*-Validation-PR^{gl+a}
accepted	223	219	223	219	222	217
rejected	0	0	0	0	0	0
error	0	0	0	0	0	0
timeout	0	4	0	4	1	6
other	0	0	0	0	0	0

*filtered from 2396 *pA*-Witnesses

Evaluation: Using Invariants from *pA*-Witnesses$^{\text{no-true}^{+h}}$

Predicate precision in *pA*-Validation-$\texttt{PR}^{\text{lo}}$ and *pA*-Validation-$\texttt{PR}^{\text{gl+a}}$ initialized with invariants from *pA*-Witnesses$^{\text{no-true}^{+h}}$.



→ For *pA*-Validation-$\texttt{PR}^{\text{lo}}$ no changes.

→ For *pA*-Validation-$\texttt{PR}^{\text{gl+a}}$ no speedup.

Evaluation: *pA*-Witnesses$^{\text{no-true}^{+h}}$ for Precision Reuse

CEGAR refinements	0	1	2	3	4	5	[6-10]	[11-20]	[21-30]	30<	Number of tasks
pA-Verification	0	180	3	6	4	1	9	7	1	12	223
pA-Validation-PR^{lo}	0	182	5	3	3	1	10	6	1	12	223
pA-Verification	0	179	3	6	4	1	9	7	1	9	219
pA-Validation-PR^{lo+a}	0	181	5	3	3	1	9	7	2	8	219
pA-Verification	0	180	3	6	4	1	9	7	1	12	223
pA-Validation-PR^{fu}	2	181	4	3	4	0	10	8	0	11	223
pA-Verification	0	179	3	6	4	1	9	7	1	9	219
pA-Validation-PR^{fu+a}	2	180	4	3	4	0	9	8	2	7	219
pA-Verification	0	180	3	6	4	1	9	7	1	11	222
pA-Validation-PR^{gl}	115	69	4	5	4	0	9	7	0	9	222
pA-Verification	0	179	3	6	4	1	9	7	1	7	217
pA-Validation-PR^{gl+a}	114	69	4	5	4	0	8	8	1	4	217

CEGAR refinements only reduced for *pA*-Validation-PR^{gl} and *pA*-Validation-PR^{gl+a}

Evaluation: *pA*-Witnesses$^{\text{no-true}^{+h}}$ for Precision Reuse - Conclusion

CPU time almost never decreased for any *pA*-Validation-PR when compared with *pA*-Verification.

Why no changes in CPU time and CEGAR refinements in particular for *pA*-Validation-PR$^{\text{lo}}$?

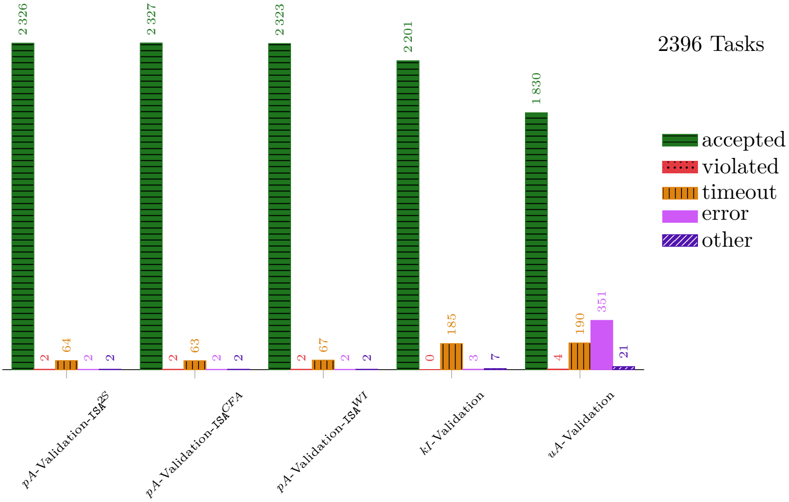
Possible reasons:

- Information from witness invariants not sufficient for an efficient precision reuse
- Bug in the implementation in CPAchecker

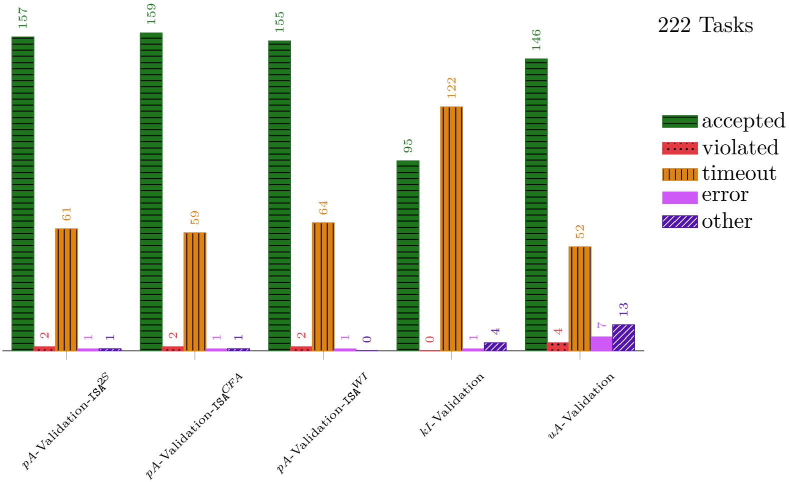
Evaluation: Outline

- Producing correctness witnesses
- Initialize predicate precision with invariant-based predicates
- **Validation of correctness witnesses by using an ISA**

Evaluation: Validating *pA*-Witnesses



Evaluation: Validating pA-Witnesses$^{\text{no-true}}$



Evaluation: Inspecting *pA*-Validation-ISA with *pA*-Witnesses as Input

configuration	*pA*-Validation-ISA^{2S}	*pA*-Validation-ISA^{CFA}	*pA*-Validation-ISA^{WI}
WIV	2	2	2

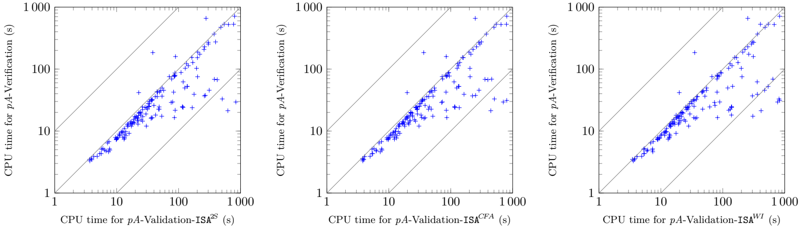
Why does each *pA*-Validation-ISA approach produces for the same two *pA*-Witnesses a witness invariant violation (WIV)?

- Witness is imprecise: Invariant is validated at a location where the invariant variables are not yet assigned
- Witness invariant contains pointer values from the SMT solver which can not be validated

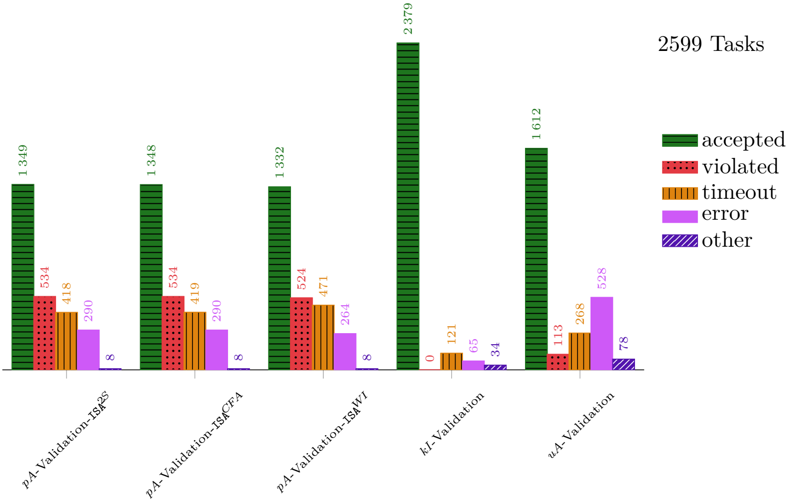
Evaluation: Validating *pA*-Witnesses$^{\text{no-true}}$

Why do the *pA*-Validation-`ISA` approaches sometimes exceed the time limit?

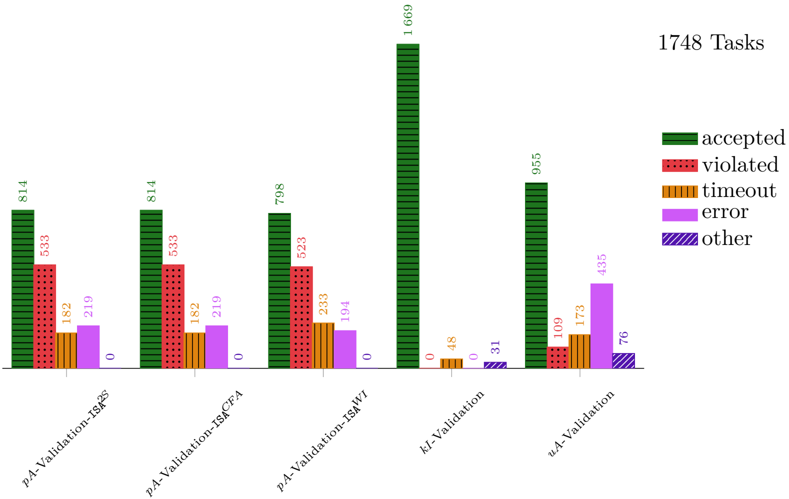
Likely reason: Additional computation effort to validate the invariants.



Evaluation: Validating *kI*-Witnesses



Evaluation: Validating *kI*-Witnesses$^{\text{no-true}}$



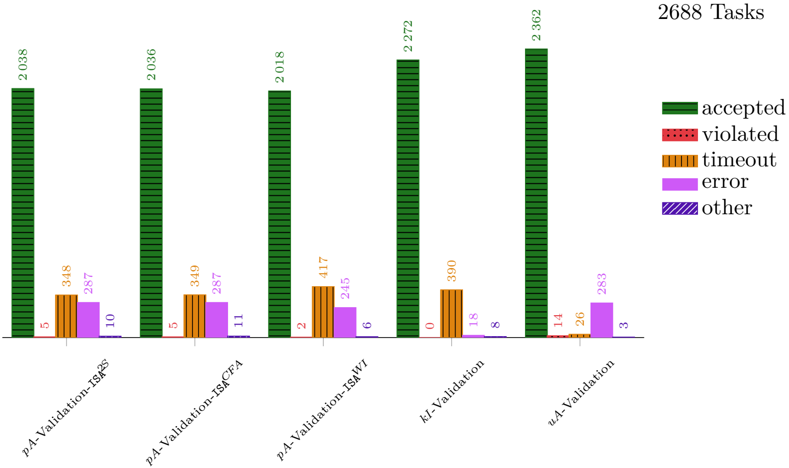
Evaluation: Inspecting *pA*-Validation-ISA with *kI*-Witnesses as Input

configuration	*pA*-Validation-ISA^{2S}	*pA*-Validation-ISA^{CFA}	*pA*-Validation-ISA^{WI}
WIV	533	533	523

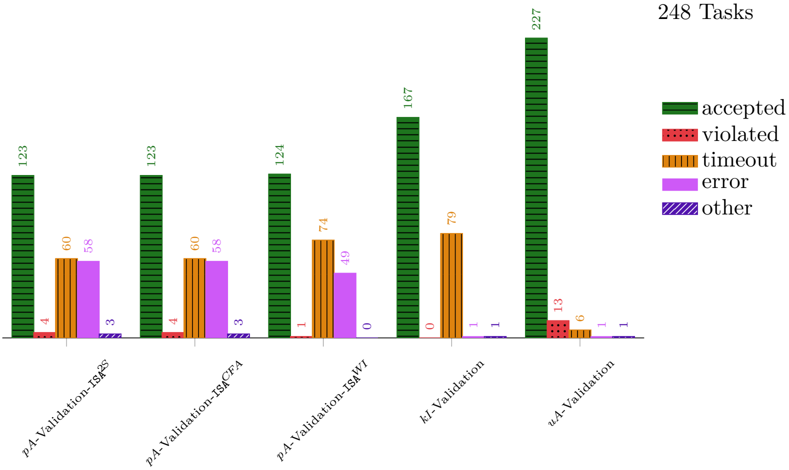
Why are so many WIVs detected?
Found reasons:

- *kI*-Witnesses$^{\text{no-true}}$ are sometimes "imprecise" for the ISA approaches
- Loop invariants in *kI*-Witnesses$^{\text{no-true}}$ do not hold for all loop iterations

Evaluation: Validating *uA*-Witnesses



Evaluation: Validating *uA*-Witnesses$^{\text{no-true}}$



Evaluation: Inspecting *pA*-Validation-ISA for *uA*-Witnesses as Input

configuration	*pA*-Validation-$\texttt{ISA}^{2S}$	*pA*-Validation-$\texttt{ISA}^{CFA}$	*pA*-Validation-$\texttt{ISA}^{WI}$
WIV	4	4	1

For ISA^{2S} or ISA^{CFA} three WIVs are produced more compared to ISA^{WI}.

Reason: Error states in ISA^{2S} and ISA^{CFA} are overapproximated when analyzing the three *uA*-Witnesses.

$\rightarrow$ Only the ISA^{WI} is precise. It can guarantee to reflect the semantics of the original witness.

Evaluation: Detecting *hidden-true*-witnesses

No *hidden-true*-witnesses in *pA*-Witnesses and *kI*-Witnesses.
147 *uA*-Witnesses are *hidden-true*-witnesses in the context of a validation analysis in CPAchecker.

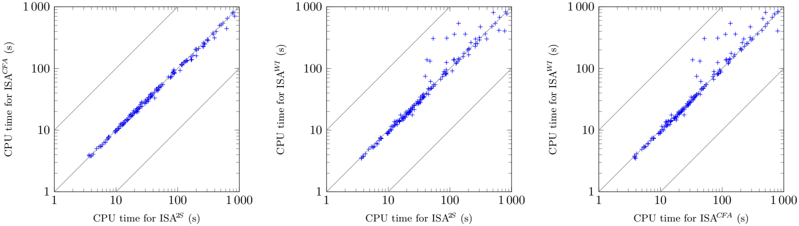
Found reasons:

- Calling `__VERIFIER_nondet_uint();` leads to two states and transitions in *uA*-Witnesses. CPAchecker does not expect this. (e.g. witness for task #2)

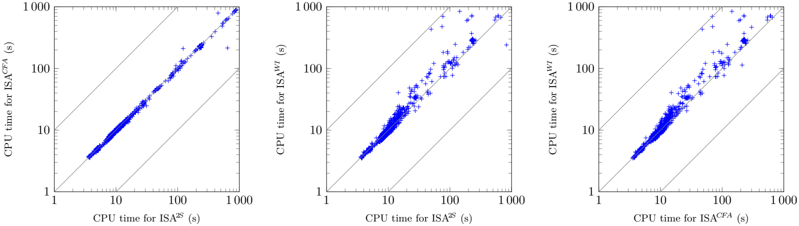
- *uA*-Witnesses can have an invariant 0 to label explicitly unreachable witness states

#2 https://github.com/sosy-lab/sv-benchmarks/blob/svcomp19/c/loop-invariants/eq1_true-unreach-call_true-valid-memsafety_true-no-overflow_false-termination.c

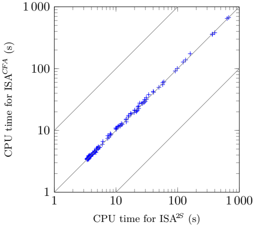
Evaluation: Comparing the *pA*-Validation-ISA Approaches for *pA*-Witnesses$^{\text{no-true}}$

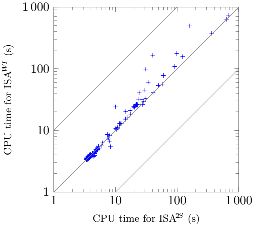


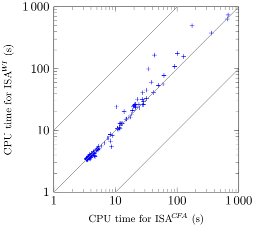
Evaluation: Comparing the *pA*-Validation-ISA Approaches for *kI*-Witnesses$^{\text{no-true}}$



Evaluation: Comparing the *pA*-Validation-`ISA` Approaches for *uA*-Witnesses$^{\text{no-true}}$







Evaluation: Validating Witnesses - Conclusion

Predicate analysis is able to validate the majority of its own witnesses. It understands other correctness witnesses but cannot always validate them.

Verifying invariants in an ISA that is based on an imprecise correctness witness leads in general to a WIV.

Proposing that correctness witnesses should be precise because:

- Transitions in witness file correspond to a certain program operation. If a transition enters a state labeled with an invariant the invariant should indeed hold at the program location that follows the program operation.
- Future validators might have problems with imprecise witnesses as well

Outlook

Validation of negated witness invariants $\rightarrow$ Checking if predicate analysis can reject the intentionally wrong correctness witnesses.

An ISA is independent from the applied abstraction-technique. Hence, it can theoretically be used with other analyzes.

The ISA^{WI} can be used for the program generation concept shown in [4][5][6].

- Concept: Transform program into a behaviorally equal program that is more efficiently verifiable. Use the ARG to create this program.
- Invariant-based error states affect the ARG $\rightarrow$ Verifying the transformed program will also verify invariants.

References

D. Beyer, M. Dangl, D. Dietsch, and M. Heizmann. "Correctness Witnesses: Exchanging Verification Results Between Verifiers". In: *Proc. FSE*. ACM, 2016, pp. 326–337. DOI: 10.1145/2950290.2950351. URL: https://www.sosy-lab.org/research/pub/2016-FSE.Correctness_Witnesses_Exchanging_Verification_Results_between_Verifiers.pdf.

D. Beyer, M. Dangl, and P. Wendler. "A Unifying View on SMT-Based Software Verification". In: *J. Autom. Reasoning* 60.3 (2018), pp. 299–335. ISSN: 1573-0670. DOI: 10.1007/s10817-017-9432-6.

D. Beyer, S.Löwe, E.Novikov, A.Stahlbauer, and P.Wendler. "Precision reuse for efficient regression verification". In: *Proc. FSE*. ACM, 2013, pp. 389–399. DOI: 10.1145/2491411.2491429. URL: https://www.sosy-lab.org/research/pub/2013-FSE.Precision_Reuse_for_Efficient_Regression_Verification.pdf.

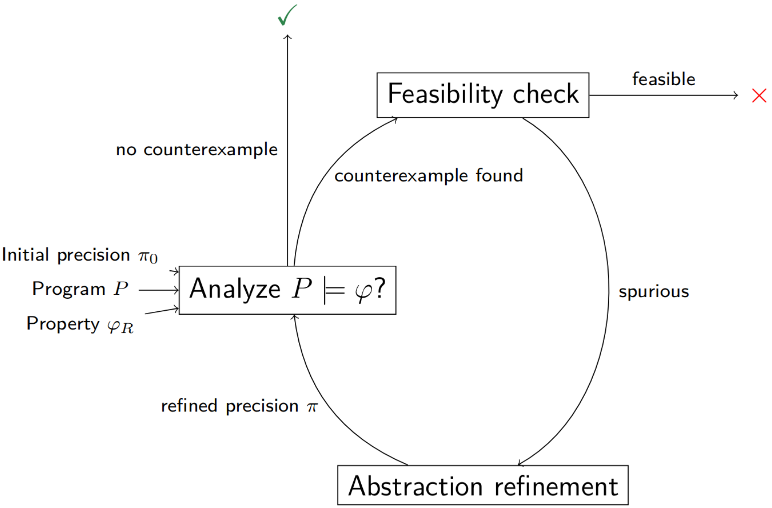
M.-C. Jakobs and H. Wehrheim. "Programs from Proofs of Predicated Dataflow Analyses". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. SAC '15. Salamanca, Spain: ACM, 2015, pp. 1729–1736. ISBN: 978-1-4503-3196-8. DOI: 10.1145/2695664.2695690. URL: http://doi.acm.org/10.1145/2695664.2695690.

M.-C. Jakobs and H. Wehrheim. "Programs from Proofs: A Framework for the Safe Execution of Untrusted Software". In: *ACM Trans. Program. Lang. Syst.* 39.2 (Mar. 2017), 7:1–7:56. ISSN: 0164-0925. DOI: 10.1145/3014427. URL: http://doi.acm.org/10.1145/3014427.

D. Wonisch, A. Schremmer, and H. Wehrheim. "Programs from Proofs – A PCC Alternative". In: *Computer Aided Verification*. Ed. by N. Sharygina and H. Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 912–927. ISBN: 978-3-642-39799-8.

Appendix

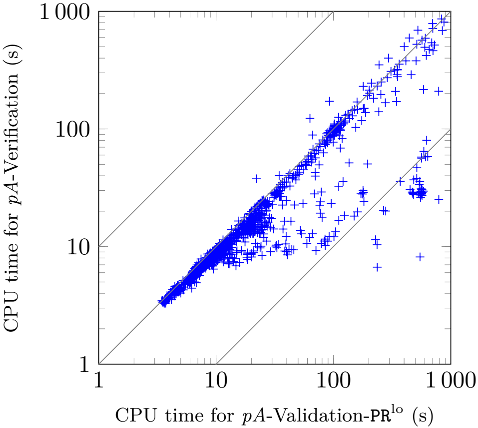
Appendix: CEGAR



Scetch from Lecture "Software Analysis and Verification" (Lecturer: M.-C. Jakobs)

Appendix: Using Invariants from *kI*-Witnesses$^{\text{no-true}^{+h}}$

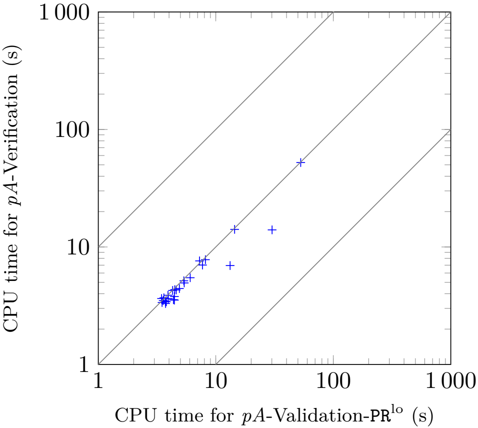
Predicate precision in *pA*-Validation-PR^{lo} is initialized with invariants from *kI*-Witnesses$^{\text{no-true}^{+h}}$.



→ No speedup.

Appendix: Using Invariants from *uA*-Witnesses$^{\text{no-true}^{+h}}$

Predicate precision in *pA*-Validation-PR$^{\text{lo}}$ initialized with invariants from *uA*-Witnesses$^{\text{no-true}^{+h}}$.



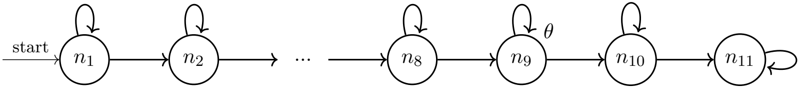
→ No statement possible because too many witnesses are detected as *hidden-true*-witnesses or *true*-witnesses.

Appendix: *pA*-Witnesses for Precision Reuse

pA-Validation-PR	*pA*-Validation-PR^{lo}	*pA*-Validation-PR^{lo+a}	*pA*-Validation-PR^{fu}	*pA*-Validation-PR^{fu+a}	*pA*-Validation-PR^{gl}	*pA*-Validation-PR^{gl+a}
			2396 *pA*-Witnesses			
accepted	2393	2351	2393	2352	2392	2349
rejected	0	0	0	0	0	0
error	1	1	1	1	1	1
timeout	1	43	1	42	2	45
other	1	1	1	1	1	1
			223 *pA*-Witnesses$^{\text{no-true}^{+h}}$			
accepted	223	219	223	219	222	217
rejected	0	0	0	0	0	0
error	0	0	0	0	0	0
timeout	0	4	0	4	1	6
other	0	0	0	0	0	0

Appendix: Inspecting *pA*-Validation-ISA for *uA*-Witnesses as Input

Correctness witness produced by ULTIMATE AUTOMIZER for task #1:



- n_9 corresponds to a loop head in the program and is labeld with invariant θ
- Original witness semantics: θ needs only to be **validated once**
- ISA^{2S} and ISA^{CFA} semantics: θ must be **validated every time** when entering the loop head
- But: θ only valid when entering the loop head the first time $\rightarrow$ ISA^{2S} and ISA^{CFA} trigger a false alarm

$\rightarrow$ Only the ISA^{WI} is precise. It can guarantee to reflect the original witness.

#1 https://github.com/sosy-lab/sv-benchmarks/blob/svcomp19/c/loops/invert_string_true-unreach-call_true-termination.c

Appendix: Comparing Accepted *pA*-Witnesses$^{\text{no-true}}$ for *pA*-Validation-`ISA` and *kI*-Validation

Witness accepted	*pA*-Validation-$\texttt{ISA}^{2S}$	$\neg$ *pA*-Validation-$\texttt{ISA}^{2S}$	Σ
kI-Validation	83	12	95
$\neg$ *kI*-Validation	74	53	127
Σ	157	65	222

Witness accepted	*pA*-Validation-$\texttt{ISA}^{CFA}$	$\neg$ *pA*-Validation-$\texttt{ISA}^{CFA}$	Σ
kI-Validation	84	11	95
$\neg$ *kI*-Validation	75	52	127
Σ	159	63	222

Witness accepted	*pA*-Validation-$\texttt{ISA}^{WI}$	$\neg$ *pA*-Validation-$\texttt{ISA}^{WI}$	Σ
kI-Validation	84	11	95
$\neg$ *kI*-Validation	71	56	127
Σ	155	67	222

Appendix: Comparing Accepted *kI*-Witnesses$^{\text{no-true}}$ for *pA*-Validation-ISA and *kI*-Validation

Witness accepted	*pA*-Validation-ISA^{2S}	¬ *pA*-Validation-ISA^{2S}	Σ
kI-Validation	807	862	1669
¬ *kI*-Validation	7	72	79
Σ	814	934	1748

Witness accepted	*pA*-Validation-ISA^{CFA}	¬ *pA*-Validation-ISA^{CFA}	Σ
kI-Validation	807	862	1669
¬ *kI*-Validation	7	72	79
Σ	814	934	1748

Witness accepted	*pA*-Validation-ISA^{WI}	¬ *pA*-Validation-ISA^{WI}	Σ
kI-Validation	791	878	1669
¬ *kI*-Validation	7	72	79
Σ	798	950	1748

Appendix: Comparing Accepted *uA*-Witnesses$^{\text{no-true}}$ for *pA*-Validation-ISA and *kI*-Validation

Witness accepted	*pA*-Validation-ISA^{2S}	$\neg$ *pA*-Validation-ISA^{2S}	Σ
kI-Validation	104	63	167
$\neg$ *kI*-Validation	19	62	81
Σ	123	125	248

Witness accepted	*pA*-Validation-ISA^{CFA}	$\neg$ *pA*-Validation-ISA^{CFA}	Σ
kI-Validation	104	63	167
$\neg$ *kI*-Validation	19	62	81
Σ	123	125	248

Witness accepted	*pA*-Validation-ISA^{WI}	$\neg$ *pA*-Validation-ISA^{WI}	Σ
kI-Validation	107	60	167
$\neg$ *kI*-Validation	17	64	81
Σ	124	124	248

Appendix: Violation Reason of Correctness Witnesses for *pA*-Validation-ISA

pA-Witnesses:

configuration	*pA*-Validation-ISA^{2S}	*pA*-Validation-ISA^{CFA}	*pA*-Validation-ISA^{WI}
all violations	2	2	2
original specification	0	0	0
WIV	2	2	2

kI-Witnesses:

configuration	*pA*-Validation-ISA^{2S}	*pA*-Validation-ISA^{CFA}	*pA*-Validation-ISA^{WI}
all violations	534	534	524
original specification	1	1	1
WIV	533	533	523

uA-Witnesses:

configuration	*pA*-Validation-ISA^{2S}	*pA*-Validation-ISA^{CFA}	*pA*-Validation-ISA^{WI}
all violations	5	5	2
original specification	1	1	1
WIV	4	4	1

Evaluation: Detecting *true*-witnesses and *hidden-true*-witnesses

Detection:

If the witness

is a *true*-witness or *hidden-true*-witness for ISA^{CFA} or ISA^{2S}

and the witness

is not a *true*-witness for ISA^{WI}

then the witness

is a *hidden-true*-witness.

Appendix: Detecting *true*-witnesses and *hidden-true*-witnesses

	approach	*non-trivial*-witnesses	*hidden-true*-witnesses or *true*-witnesses	*true*-witnesses
2396 *pA*-Witnesses	*pA*-Validation-ISA^{2S}	220	2169	-
	pA-Validation-ISA^{CFA}	220	2169	-
	pA-Validation-ISA^{WI}	220	-	2169
2599 *kI*-Witnesses	*pA*-Validation-ISA^{2S}	1696	559	-
	pA-Validation-ISA^{CFA}	1696	559	-
	pA-Validation-ISA^{WI}	1696	-	559
2688 *uA*-Witnesses	*pA*-Validation-ISA^{2S}	40	2337	-
	pA-Validation-ISA^{CFA}	40	2337	-
	pA-Validation-ISA^{WI}	187	-	2190

→ 147 *uA*-Witnesses are *hidden-true*-witnesses for in the context of an analysis in CPAchecker.