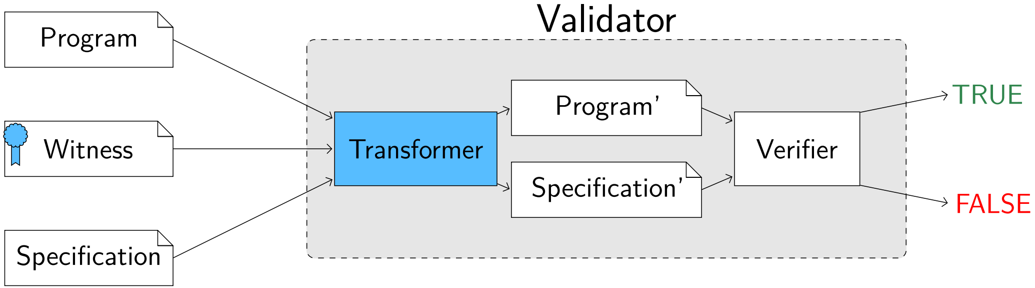
LIV: Invariant Validation Using Straight-Line Programs

Dirk Beyer, and **Martin Spiessl**

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

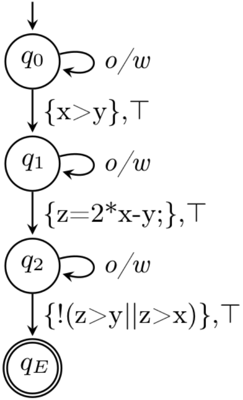
SoSy-Lab Software Systems

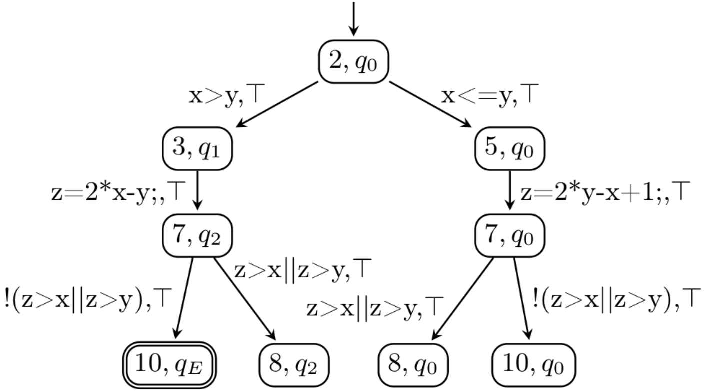
Recap: The MetaVal Approach (CAV 2020)



- MetaVal uses verifiers as validators
- The transformer translates validation into a verification problem

Automaton Product of MetaVal





- MetaVal dumps the product of CFA and witness automaton into a C program
- lots of gotos, not very readable

Strength and Weaknesses of METAVAL

Strengths:

- Reuse of off-the-shelf verifiers
- Implementation is an one-time-effort

Weaknesses:

- Transformation is brittle and for unsound in edge cases
- Transformed programs / validation result is hard to understand
- Transformer based on CPAchecker ⇒ technology bias
- "True" witnesses might be validated (general problem with graphml witnesses)

Goal of proposed new approach:

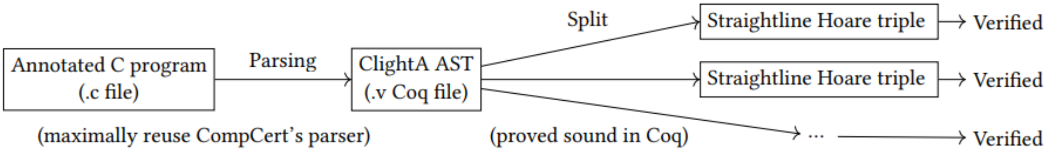
- Keep the Pros and improve on the Cons

Inspiration: Hoare-Style Proofs

```c
int x = nondet();
if (x>=0) {
  while (x>0) { // loop invariant: x>=0
    x--;
  }
} else x++;
y = x;
```

$$\dfrac{\{P\}s_0\{R_1\} \qquad \dfrac{\dfrac{R_1 \wedge C_1 \Rightarrow \gamma \qquad \{\gamma \wedge C_2\}\texttt{B}\{\gamma\} \qquad \gamma \Rightarrow R_2}{\{R_1 \wedge C_1\}\texttt{while}\ C_2\ \texttt{do}\ B\ \{R_2\}}\,\text{while} \qquad \{R_1 \wedge \neg C_1\}s_1\{R_2\}}{\{R_1\}\texttt{if}\ C_1\,(\texttt{while}\ C_2\ \texttt{do}\ B)\,s_1\{R_2\}}\,\text{if} \qquad \{R_2\}s_2\{Q\}}{\{P\}s_0\,(\texttt{if}\ C_1\,(\texttt{while}\ C_2\ \texttt{do}\ B\ )\texttt{else}\ s_1)\,s_2\{Q\}}$$

Inspiration 2:
VST-A: Foundationally Sound Annotation Verifier[2]

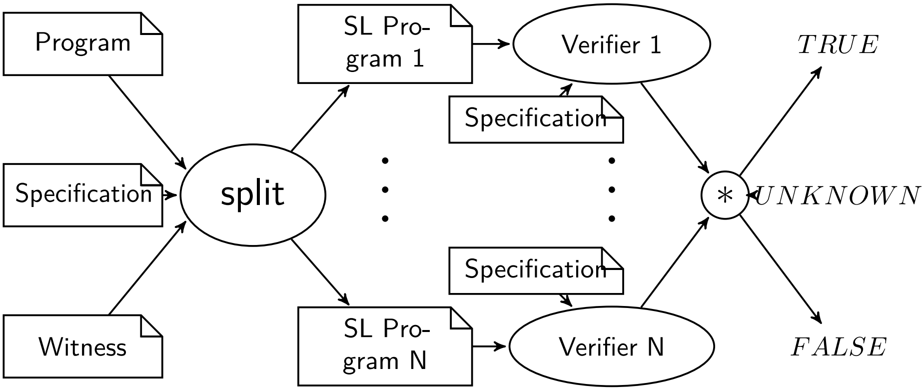


- Splits C program into straightline hoare triples
- Split procedure proven to be sound in Coq

Downsides:

- actual transformation implementation works on the CFG, not AST
- depends on Clight/Coq in the backend

Worflow of LIV



- can use any off-the-shelf verifier as backend
- small frontend using pycparser for splitting
- intended use case: inductive invariants provided in the new YAML format

Strength of LIV

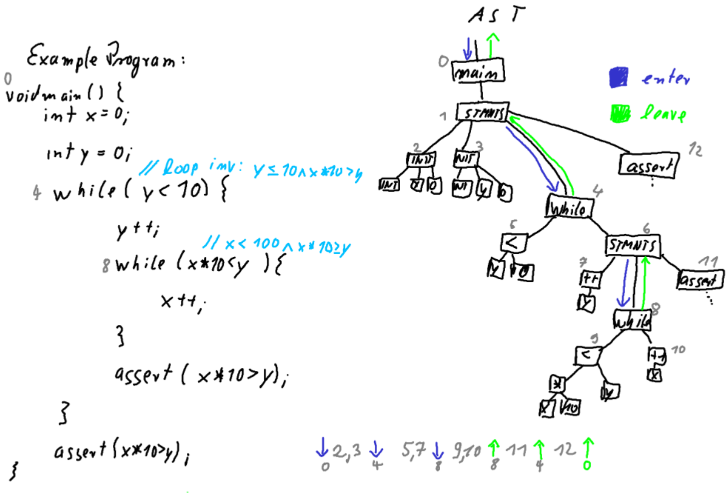
Strengths:

- reuse of off-the-shelf verifiers (now proudly powered by CoVeriTeam[1])
- implementation is an one-time-effort

addressed weaknesses:

- MetaVal transformation was brittle
  ⇒ now AST-based, "minimally invasive", easier to check for errors
- programs + validation result now easier to understand
- technology bias: we use pycparser for parsing C files, no bias towards CPAchecker anymore
- "True" witnesses no longer validated unless "True" is sufficient invariant

For Hoare triples we do not need a CFA/CFG



- Transformation into Hoare triples based on AST
- No notion of CFA/CFG needed
- Translation of triples into straightline programs is very natural

For Hoare triples we do not need a CFA/CFG (cont.)

0) ↓ true   0) ↑ $x > 0 \wedge y > 0$

4) $y \le 10 \wedge x * 10 \ge y$

8) $x < 100 \wedge x * 10 \ge y$

↓ 2,3 ↓ (1, 4)

{true} int x=0; int y=0; {y ≤ 10 ∧ 10x ≥ y}

```
assume(true);
  int x=0;
  int y=0;
assert( y<=10 && 10*y>=y);
```

↓ 5,7 ↓ (4, 8)

{y ≤ 10 ∧ x * 10 ≥ y} [y<10] y++; {x < 100 ∧ x * 10 ≥ y}

```
int x= nondet(); int y =nondet()
assume (y<=10 && x*10 >=y);
assume(y <10); //assume we enter
                 the loop
y++;
assert (x<100 && x*10 >=y);
```

Splitting Procedure in LIV

$$
\begin{aligned}
\text{split}(\{P\}, \epsilon) &= (\emptyset, \{(\{P\}, \epsilon)\} && \text{(empty)} \\
\text{split}(\{P\}, S_0 s_1) &= \text{split}(\{P\}, S_0) \cdot s_1 && \text{(atomic)} \\
\text{split}(\{P\}, S_0 if C do S_1 else S_2) &= \text{split}(\{P\}, S_0[c]S_1) + && \text{(if)} \\
&\quad \text{split}(\{P\}, S_0![c]S_2) \\
\text{split}(\{P\}, S_0 while^{\gamma} C do B) &= \text{split}(\{P\}, S_0) \odot \{\gamma\} + && \text{(while-inv)} \\
&\quad \text{split}(\{\gamma\}, [c]B\{\gamma\}) + \\
&\quad \text{split}(\{\gamma\}, \emptyset) \\
\text{split}(\{P\}, S_0 S_1^{\gamma}) &= \text{split}(\{P\}, S_0) \odot \{\gamma\} + && \text{(loc-inv)} \\
&\quad \text{split}(\{\gamma\}, S_1)
\end{aligned}
$$

- split collects sets of closed and open (missing postcondition) Hoare triples
- finalize by appending the post condition to all open Hoare triples
- simplified version of splitting from VST-A[2]
  (but probably every deductive verifier does this one way or the other)

Preliminary Benchmarks

Verifier	total	nontrivial	confirmed	rejected	unknown	error
2ls	13	12	6	7	0	0
cbmc	7	0	0	0	0	7
cvt-algosel	16	9	2	11	1	2
cvt-parport	19	5	4	15	0	0
cpachecker	21	6	5	14	0	2
graves	22	9	5	14	2	1
pesco	21	16	11	5	1	4
uautomizer	22	22	9	12	1	0
ukojak	21	21	10	10	1	0
utaipan	22	16	6	10	0	6

- analyzed on witnesses of zilu benchmarks from SV-COMP 2023 (containing only one loop)
- some of the invariants already inductive, but far from perfect

Further Research Directions

- use LIV in SV-COMP 2024 as validator, work towards having inductive invariants
- expressing new YAML witness syntax as source code transformation:
  - for correctness: easy, just insert invariant assertion (essentially METAVAL)
  - for violation: would need to encode current segment number e.g. via ghost variables
- add support for ACSL
- add other simple transformations like NoOverflow to reachability
- extend with more annotation types like function contracts etc.

More Information Online

Gitlab: `https://gitlab.com/sosy-lab/software/liv/`
Demonstration video: `https://youtu.be/mZhoGAa08Rk`

References I

Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. pp. 561–579. LNCS 13243, Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_31

Wang, Q., Cao, Q.: VST-A: A foundationally sound annotation verifier. CoRR **abs/1909.00097** (2019). https://doi.org/10.48550/arXiv.1909.00097