Leveraging Invariant Information Towards Incremental Software Model Checking

Martin Spiessl
spiessl@sosy.ifi.lmu.de

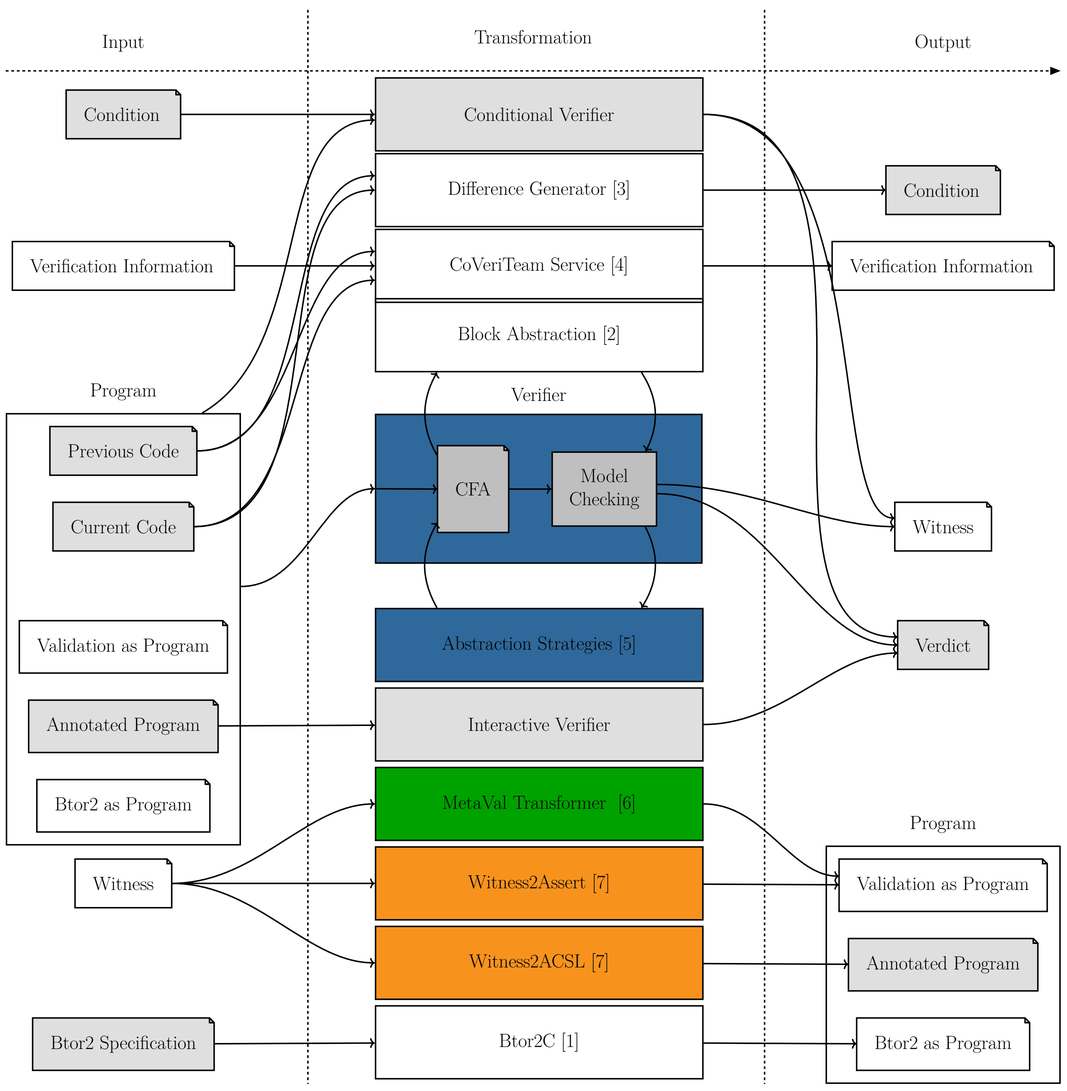
Supervisors: Dirk Beyer & Helmut Seidl

Collaborators: Karlheinz Friedberger (ConVeY) & Sven Umbricht & Marian Lingsch

ConVeY

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Overview of Transformations in ConVeY



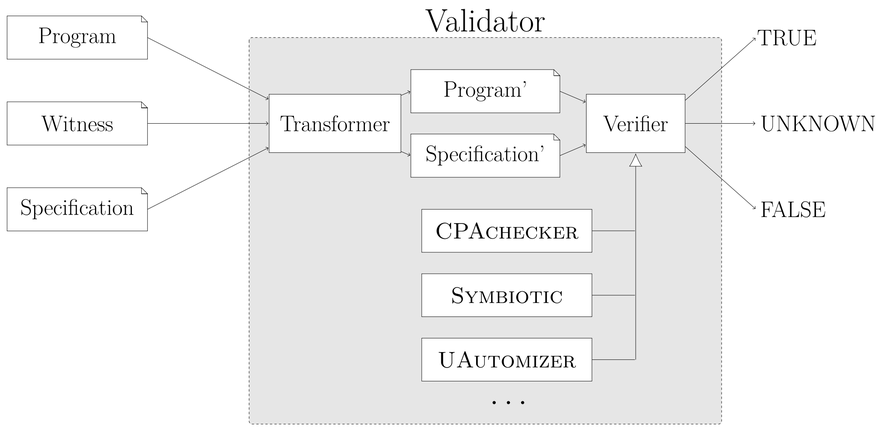
Own Contributions

Leveraging ACSL [7]: ACSL is a specification language for C programs, used by interactive verifiers, while automatic verifiers store invariant information in so-called witnesses. By translating between these formats, we can create novel interactions between interactive and automatic verifiers.

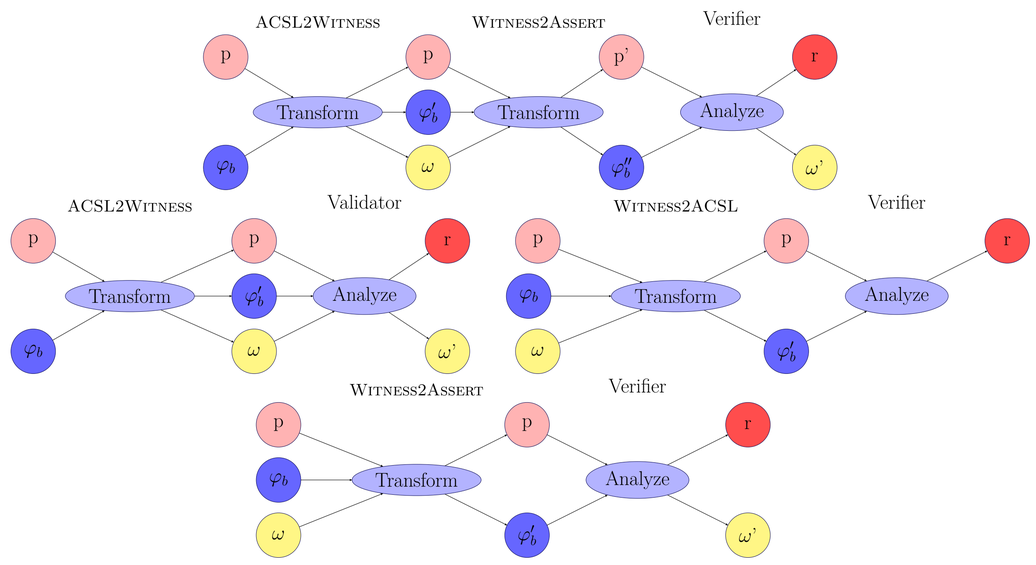
MetaVal [6]: Verification Witnesses increase the trust in the verification result, since they allow to validate the proof. The number of available validators however is limited. MetaVal Encodes the validation as a verification problem into the original program, which allows to use off-the-shelf verifiers for validation.

Unifying Loop Abstractions [5]: Loop abstractions are easy yet often very powerful program transformations which correspond to transition invariants of the loop. Determining which of the available loop abstractions need to be applied for a successful proof is not straight-forward. Our approach allows to apply loop abstractions in a CEGAR-style approach, determining the right level of abstraction automatically.

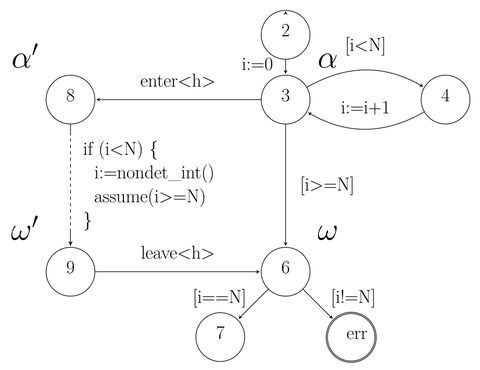
MetaVal



Interfacing Tools with ACSL



Loop Abstractions



Loop abstractions are woven into the CFA of the program. State space exploration is then guided into the abstractions. On spurious counterexamples, CEGAR refinement will lead to other paths being taken. Interfacing with the CFA allows to reuse existing analysis in CPAchecker straight away.

```
1 unsigned int x = 0x0ffffff0;
2 while (x > 0) {
3   x -= 2;
4 }
5 assert(!(x % 2)));
```

(a) Original program

```
1 unsigned int x = 0x0ffffff0;
2 if (x > 0) {
3   x = nondet_uint();
4   if (x > 0) {
5     return 0;
6   }
7 }
8 assert(!(x % 2)));
```

(b) Havoc abstraction

```
1 unsigned int x = 0x0ffffff0;
2 if (x > 0) {
3   long long iterations = x/2;
4   x -= 2*iterations;
5   if (x > 0) {
6     x -= 2;
7   }
8 }
9 assert(!(x % 2)));
```

(c) Constant extrapolation

```
1  unsigned int x = 0x0ffffff0;
2  if (x > 0) {
3    x = nondet_uint();
4    if (x <= 0) {
5      return 0;
6    }
7    x -=2;
8    if (x > 0) {
9      return 0;
10   }
11 }
12 assert(!(x % 2)));
```

(d) Naive abstraction

References

[1] D. Beyer, P.-C. Chien, and N.-Z. Lee. "Bridging Hardware and Software Analysis with Btor2C: A Word-Level-Circuit-to-C Converter". In: *TACAS 2023, Held as Part of ETAPS 2023, Paris, France, 2023, Proceedings*. Springer, 2023.

[2] D. Beyer and K. Friedberger. "Domain-Independent Interprocedural Program Analysis using Block-Abstraction Memoization". In: *ESEC/FSE '20, USA, November 8-13, 2020*. Ed. by P. Devanbu, M. Cohen, and T. Zimmermann. ACM, 2020, pp. 50–62.

[3] D. Beyer, M.-C. Jakobs, and T. Lemberger. "Difference Verification with Conditions". In: *SEFM 2020, Amsterdam, The Netherlands, September 14-18, 2020, Proceedings*. Ed. by F. d. Boer and A. Cerone. LNCS 12310. Springer, 2020, pp. 133–154.

[4] D. Beyer, S. Kanav, and H. Wachowitz. "CoVeriTeam Service: Verification as a Service". In: *ICSE 2023, Melbourne, Australia, May 14–20, 2023, Proceedings, Part II*. Ed. by J. Grundy, L. Pollock, and M. D. Penta. IEEE, 2023.

[5] D. Beyer, M. L. Rosenfeld, and M. Spiessl. "A Unifying Approach for Control-Flow-Based Loop Abstraction". In: *SEFM 2022, Berlin, Germany, September 26-30, 2022, Proceedings*. Ed. by B.-H. Schlingloff and M. Chai. LNCS 13550. Springer, 2022, pp. 3–19.

[6] D. Beyer and M. Spiessl. "MetaVal: Witness Validation via Verification". In: *CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*. Ed. by S. K. Lahiri and C. Wang. LNCS 12225. Springer, 2020, pp. 165–177.

[7] D. Beyer, M. Spiessl, and S. Umbricht. "Cooperation between Automatic and Interactive Software Verifiers". In: *SEFM 2022, Berlin, Germany, September 26-30, 2022, Proceedings*. Ed. by B.-H. Schlingloff and M. Chai. LNCS 13550. Springer, 2022, 111–128.