# Current Trends in CPAchecker

Dirk Beyer
LMU Munich, Germany

October 4, 2022, at CPAchecker Workshop 2022

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

SoSy-Lab Software Systems

# What is going on?

- Implement more and more approaches in CPAchecker
- Make them available as a configuration of CPAchecker
- Reusable components from CPAchecker independently
- Combination approaches

# CPAchecker [21] Offers a Lot!

- Included Concepts:
  - CEGAR [27] Interpolation [24, 12]
  - Configurable Program Analysis [16, 17]
  - Adjustable-block encoding [22]
  - Conditional model checking [15]
  - Verification witnesses [10, 9]
  - Various abstract domains: predicates, intervals, BDDs, octagons, explicit values
- Available analyses approaches:
  - Predicate abstraction [7, 22, 17, 25]
  - Impact algorithm [40, 26, 12]
  - Bounded model checking [29, 12]
  - k-Induction [11, 12]
  - IC3/Property-directed reachability [8]
  - Explicit-state model checking [24]
  - Interpolation-based model checking [23]

# Competitions in Software Verification

CPAchecker should win SV-COMP. ;-)

# Different Techniques

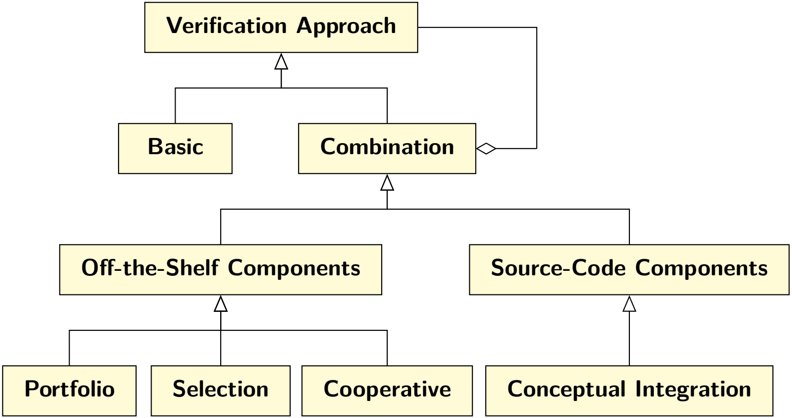
| Participant | CEGAR | Predicate Abstraction | Symbolic Execution | Bounded Model Checking | k-Induction | Property-Directed Reach. | Explicit-Value Analysis | Numeric. Interval Analysis | Shape Analysis | Separation Logic | Bit-Precise Analysis | ARG-Based Analysis | Lazy Abstraction | Interpolation | Automata-Based Analysis | Concurrency Support | Ranking Functions | Evolutionary Algorithms |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2LS | | | | ✓ | ✓ | | | ✓ | | | ✓ | | | | | | ✓ | |
| AProVE | | | ✓ | | | | ✓ | ✓ | | ✓ | ✓ | | | | | | ✓ | |
| CBMC | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | |
| CBMC-Path | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | |
| CPA-BAM-BnB | ✓ | ✓ | | | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | | | |
| CPA-Lockator | ✓ | ✓ | | | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | ✓ | | |
| CPA-Seq | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | |
| DepthK | | | | ✓ | ✓ | | | | | | ✓ | | | | | ✓ | | |
| DIVINE-explicit | | | | | | | ✓ | | | | ✓ | | | | | ✓ | | |
| DIVINE-SMT | | | | | | | ✓ | | | | ✓ | | | | | ✓ | | |
| ESBMC-kind | | | | ✓ | ✓ | | | | | | ✓ | | | | | ✓ | | |
| JayHorn | ✓ | ✓ | | | | ✓ | | ✓ | | | | | ✓ | ✓ | | | | |
| JBMC | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | |
| JPF | | | | ✓ | | | ✓ | ✓ | | | ✓ | | | | | ✓ | | |
| Lazy-CSeq | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | |
| Map2Check | | | | ✓ | | | | | | | ✓ | | | | | | | |
| PeSCo | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | |
| Pinaka | | | ✓ | ✓ | | | | | | | ✓ | | | | | | | |
| PredatorHP | | | | | | | | | ✓ | | | | | | | | | |
| Skink | ✓ | | | | | | ✓ | | | | | | | ✓ | ✓ | | | |
| Smack | ✓ | | | ✓ | | ✓ | | | | | ✓ | | ✓ | | | ✓ | | |
| SPF | | | ✓ | | | | | | ✓ | | | | | | | ✓ | | |
| Symbiotic | | | ✓ | | | | | ✓ | | | ✓ | | | | | | | |
| UAutomizer | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | ✓ | | ✓ | |
| UKojak | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | | | | |
| UTaipan | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | ✓ | | | |
| VeriAbs | ✓ | | | ✓ | ✓ | | ✓ | ✓ | | | | | | | | | | |
| VeriFuzz | | | | ✓ | | | | ✓ | | | | | | | | | | ✓ |
| VIAP | | | | | | | | | | | | | | | | | | |
| Yogar-CBMC | ✓ | | | ✓ | | | | | | | ✓ | | ✓ | | | ✓ | | |
| Yogar-CBMC-Par. | ✓ | | | ✓ | | | | | | | ✓ | | ✓ | | | ✓ | | |

Competition Report [5]
https://doi.org/10.1007/978-3-030-17502-3_9

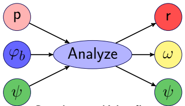
# CPAchecker as Component

- Make CPAchecker usable like a library
- Small tools perspective (Unix philosophy)
- Construct combinations
- More interfaces
  via, e.g., Conditions, Witnesses, Test Suites
- See also: Little Engines [41], Evidential Tool Bus [33]

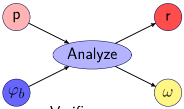
# Approaches for Combinations



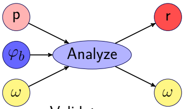
# Graphical Visualization of the Coop Framework



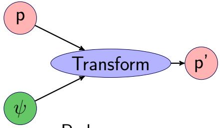
Conditional Verifier



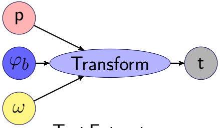
Verifier



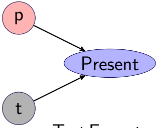
Validator



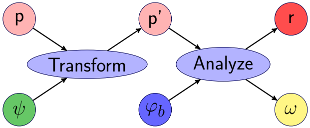
Reducer



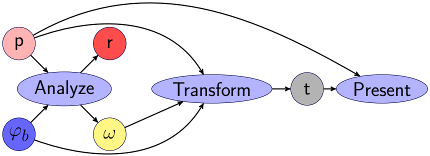
Test Extractor



Test Executor



Conditional Verifier constructed from Reducer and Verifier



Execution-Based Validation constructed from Verifier, Test Extractor, and Test Executor

# Definition of Cooperative Verification

An approach is called **cooperative verification**, if

- identifiable *actors* pass information in form of
- identifiable *artifacts* towards the common objective of
- solving a *verification* problem,

where at least two of these actors are *analyzers*.

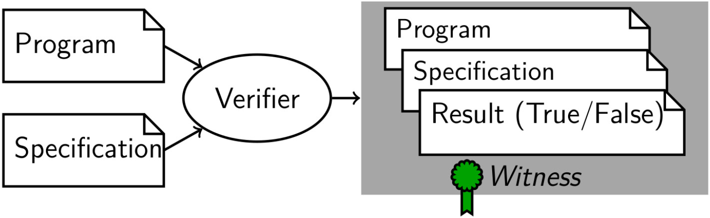
# Definition of Cooperative Verification

Examples for notions:

- Identifiable actor:
  off-the-shelf components, binaries, agents, web services
- Identifiable artifacts:
  programs, witnesses, ARGs, test suites
- Verification problem:
  verification task, test task, feasibility check, refinement

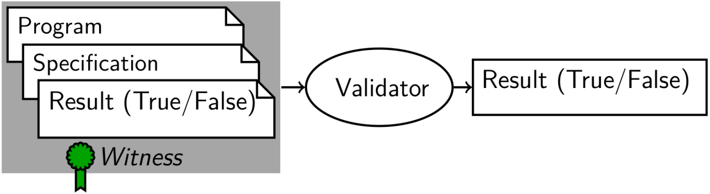
# Software Verification with Witnesses

Witnesses are an important interface between tools.




[10, Proc. FSE 2015] [9, Proc. FSE 2016]

# Witness Validation




- Validate untrusted results
- Easier than full verification

# Example Combination (in DSL CoVeriTeam)

CoVeriTeam: Language and Tool [19, Proc. TACAS 2022]

**Algorithm 1** Witness Validation [10, 9]

**Input:** Program p, Specification s

**Output:** Verdict

```
verifier := Verifier("Ultimate Automizer")
validator := Validator("CPAchecker")
result := verifier.verify(p, s)
if result.verdict ∈ {TRUE, FALSE} then
    result = validator.validate (p, s, result.witness)
return (result.verdict, result.witness)
```

# Simple Combination without Cooperation

Often, even simple combinations help!

Portfolio construction using off-the-shelf verification tools [20, Proc. FASE 2022]

Consider AWS category (177 tasks) in SV-COMP 2022:
CBMC: 69 (8 wrong)
CoVeriTeam-Parallel-Portfolio: 147 (3 wrong)
(improvement did not require any change in a verification tool)

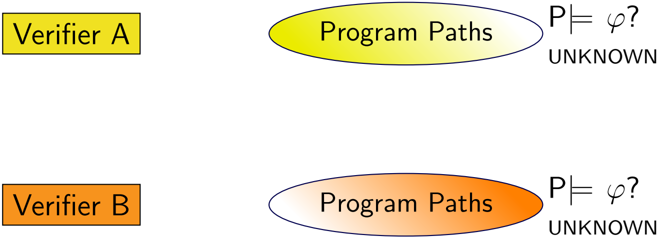
With Nian-Ze Lee and Po-Chun Chien:

- inject invariants (in k-induction, IMC, ISMC)
- parallel portfolio

A lot of improvements are (trivially) possible.

# Facing Hard Verification Tasks

Given: Program P$\models \varphi$?



# Facing Hard Verification Tasks

Given: Program P$\models \varphi$?

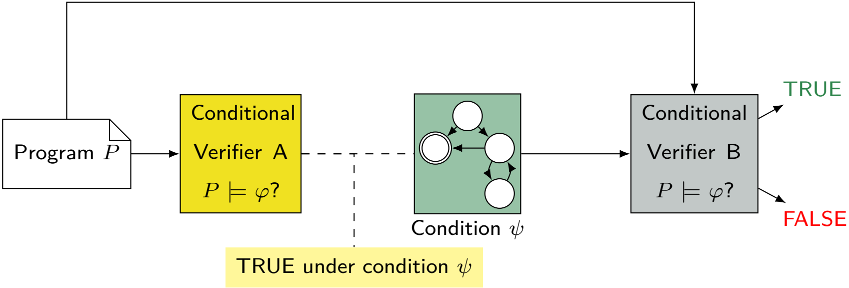
Verifier A — Program Paths — P$\models \varphi$? UNKNOWN

Verifier B — Program Paths — P$\models \varphi$? UNKNOWN

Verifier A + Verifier B — Program Paths — P$\models \varphi$ ✓

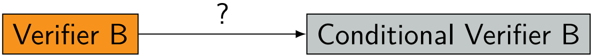
e.g., conditional model checking

# Conditional Model Checking



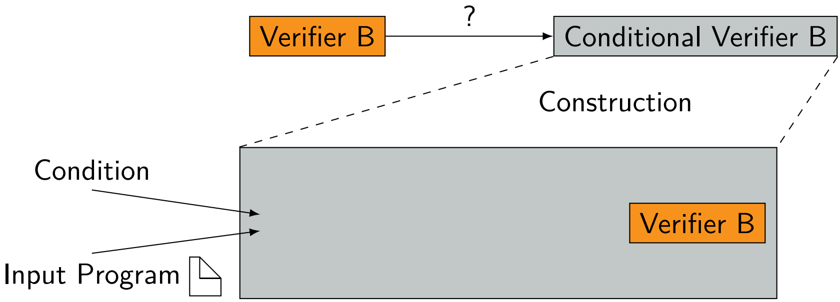

Proc. FSE 2012 [15]

# Reducer-Based Construction



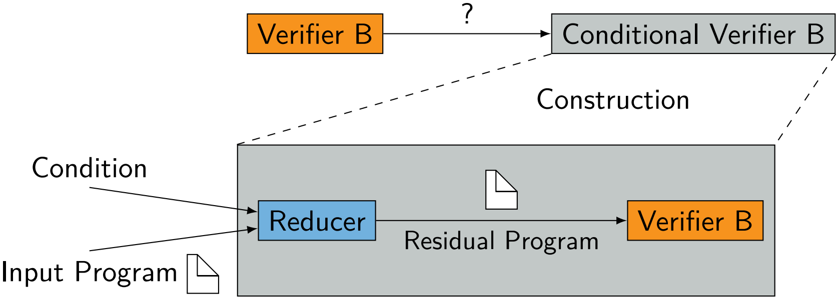

Proc. ICSE 2018 [18]

# Reducer-Based Construction




Proc. ICSE 2018 [18]

# Reducer-Based Construction

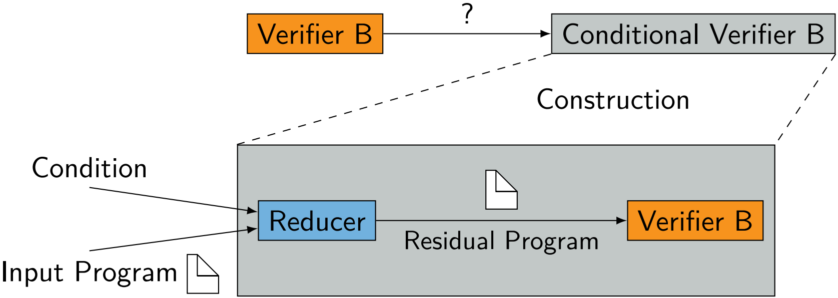



Reducer (preprocessor)

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

Proc. ICSE 2018 [18]

# Reducer-Based Construction



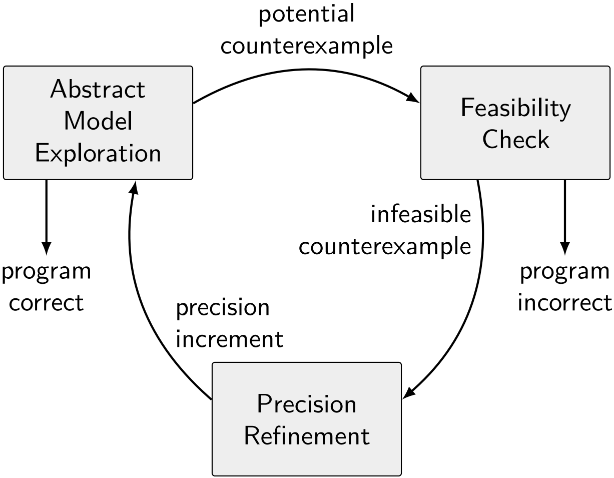

Reducer (preprocessor)

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

+ Verifier-unspecific approach
+ Many conditional verifiers possible

Proc. ICSE 2018 [18]

# CEGAR

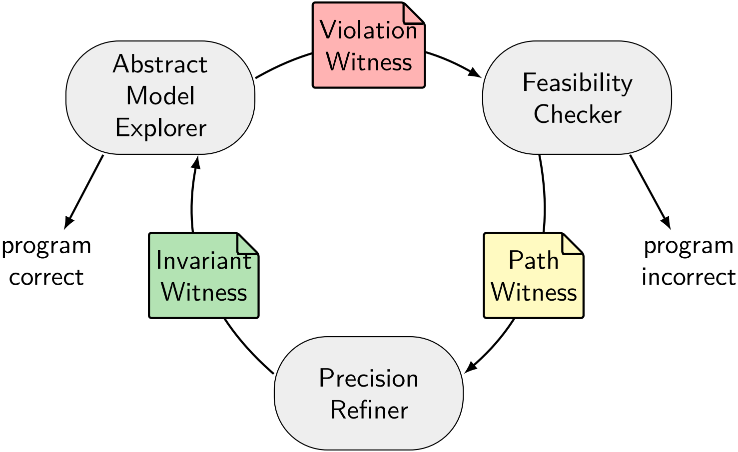

potential
counterexample
Abstract
Model
Exploration
Feasibility
Check
infeasible
counterexample
program
correct
program
incorrect
precision
increment
Precision
Refinement

# Modularization of CEGAR

- CEGARdefines I/O interfaces
- But instances not exchangeable
- Aim: generalize CEGAR, allow exchange of components

$\Rightarrow$ Modular reformulation

# Workflow of modular CEGAR




Proc. ICSE 2022 [13]

# Interactive and Automatic Methods

- How to achieve cooperation between automatic and interactive verifiers?
- Idea: Try to use existing interfaces for information exchange

```
//@ensures \return==0;
int main() {
  unsigned int x = 0;
  unsigned int y = 0;
  //@loop invariant x==y;
  while (nondet_int()) {
    x++;
    //@assert x==y+1;
    y++;
  }
  assert(x==y);
  return 0;
}
```

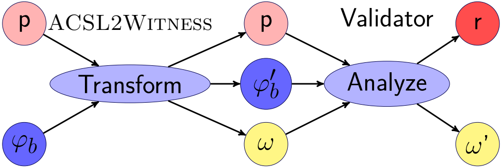
ACSL-annotated program, as used by FRAMA-C

```
...
<node id="q1">
<data key="invariant">( y == x )</data>
<data key="invariant.scope">main</data>
</node>
<edge source="q0" target="q1">
<data key="enterLoopHead">true</data>
<data key="startline">6</data>
<data key="endline">6</data>
<data key="startoffset">157</data>
<data key="endoffset">165</data>
</edge>
...
```

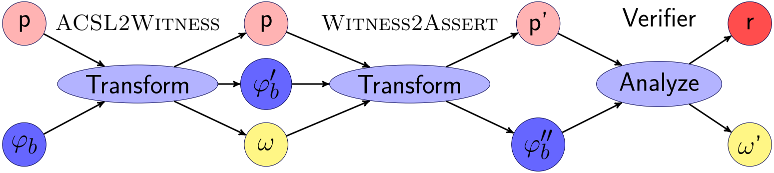
GraphML-based witness automaton generated by automatic verifiers

# From Components: Construct Interactive Verifiers

- Turn a witness validator into an interactive verifier:



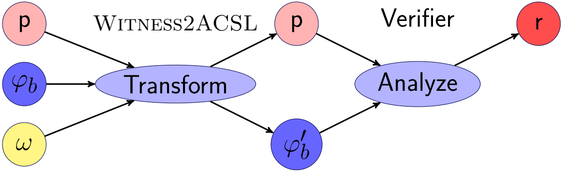

- Turn an automatic verifier into an interactive verifier::



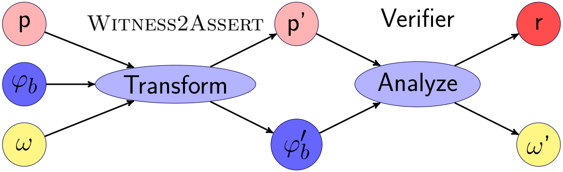

- Annotating in ACSL is more human-readable than witness automata
- Works for a wide range of automatic verifiers/validators

# Component Framework: Constructing Validators

- Turn an interactive verifier (Frama-C) into a validator:

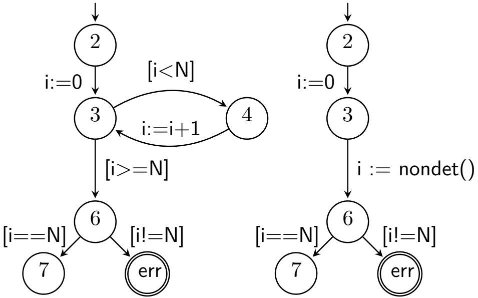



- Turn an automatic verifier into a validator:



# Loop Abstraction

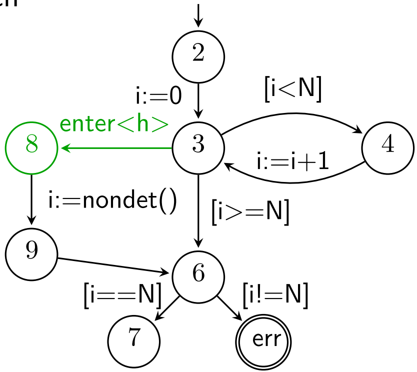
```
void main() {
  int i = 0;
  while (i<N) {
    i=i+1;
  }
  assert (i==N);
}
```




- Instead of a precise acceleration, we can also apply an overapproximating *abstraction*
- Here we just havoc all variables that are modified in the loop, but more elaborate abstraction strategies exist

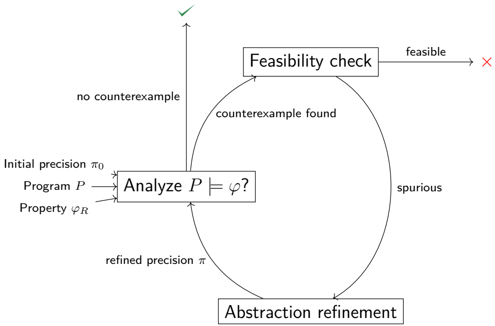
# Configurable Solution a la CPAchecker

- Use the CFA as interface
- Add our loop abstractions next to the original loop
- Mark the entry nodes of each added alternative with an identifier for the applied strategy: $\sigma : L \rightarrow S$
- In the example:
  $S = \{b, h\}$
  $\sigma(8) = h$
  $\sigma(l) = b$ for
  $l \in \{2, 3, 4, 6, 7, err, 9\}$
- Select allowed strategies during state-space exploration using $\sigma$



# CEGAR: Feasibility of Counterexamples

- In general, CEGAR works as shown on the right
- For our approach, we need to rethink what it means if a counterexample is feasible:
  Even if the path formula is satisfiable, the counterexample is only feasible if there are no over-approximating strategies used along the path!



# Accessibility of Loop Abstractions via Patches

- We provide loop abstractions as patches
- We also output a the abstracted version of the program in case we found a proof
- Can be used independently by other tools
- Does this work in practice?
  ⇒ Experiments

```
--- havoc.c
+++ havoc.c
-14,13 +14,16
    return ;
 }

 int main(void) {
   unsigned int x = 1000000;
- while (x > 0) {
- x -= 4;
+ // START HAVOCSTRATEGY
+ if (x > 0) {
+ x = __VERIFIER_nondet_uint();
   }
+ if (x > 0) abort();
+ // END HAVOCSTRATEGY
   __VERIFIER_assert(!(x % 4));
```

# Conclusion

- External combinations
- Interfaces
- Combinations and Cooperation

# References I

[1] Ball, T., Levin, V., Rajamani, S.K.: A decade of software model checking with SLAM. Commun. ACM **54**(7), 68–76 (2011). `https://doi.org/10.1145/1965724.1965743`

[2] Ball, T., Majumdar, R., Millstein, T.D., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: Proc. PLDI. pp. 203–213. ACM (2001). `https://doi.org/10.1145/378795.378846`

[3] Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.5. Tech. rep., University of Iowa (2015), available at `https://smtlib.cs.uiowa.edu/`

[4] Beyer, D.: Competition on software verification (SV-COMP). In: Proc. TACAS. pp. 504–524. LNCS 7214, Springer (2012). `https://doi.org/10.1007/978-3-642-28756-5_38`

[5] Beyer, D.: Automatic verification of C and Java programs: SV-COMP 2019. In: Proc. TACAS (3). pp. 133–155. LNCS 11429, Springer (2019). `https://doi.org/10.1007/978-3-030-17502-3_9`

[6] Beyer, D.: Competition on software testing (Test-Comp). In: Proc. TACAS (3). pp. 167–175. LNCS 11429, Springer (2019). `https://doi.org/10.1007/978-3-030-17502-3_11`

# References II


[7] Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: Proc. FMCAD. pp. 25–32. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351147

[8] Beyer, D., Dangl, M.: Software verification with PDR: An implementation of the state of the art. In: Proc. TACAS (1). pp. 3–21. LNCS 12078, Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_1

[9] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M.: Correctness witnesses: Exchanging verification results between verifiers. In: Proc. FSE. pp. 326–337. ACM (2016). https://doi.org/10.1145/2950290.2950351

[10] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Stahlbauer, A.: Witness validation and stepwise testification across software verifiers. In: Proc. FSE. pp. 721–733. ACM (2015). https://doi.org/10.1145/2786805.2786867

[11] Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Proc. CAV. pp. 622–640. LNCS 9206, Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_42

[12] Beyer, D., Dangl, M., Wendler, P.: A unifying view on SMT-based software verification. J. Autom. Reasoning **60**(3), 299–335 (2018). https://doi.org/10.1007/s10817-017-9432-6

# References III


[13] Beyer, D., Haltermann, J., Lemberger, T., Wehrheim, H.: Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR. In: Proc. ICSE. pp. 536–548. ACM (2022). https://doi.org/10.1145/3510003.3510064

[14] Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker Blast. Int. J. Softw. Tools Technol. Transfer **9**(5-6), 505–525 (2007). https://doi.org/10.1007/s10009-007-0044-z

[15] Beyer, D., Henzinger, T.A., Keremoglu, M.E., Wendler, P.: Conditional model checking: A technique to pass information between verifiers. In: Proc. FSE. ACM (2012). https://doi.org/10.1145/2393596.2393664

[16] Beyer, D., Henzinger, T.A., Théoduloz, G.: Configurable software verification: Concretizing the convergence of model checking and program analysis. In: Proc. CAV. pp. 504–518. LNCS 4590, Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_51

[17] Beyer, D., Henzinger, T.A., Théoduloz, G.: Program analysis with dynamic precision adjustment. In: Proc. ASE. pp. 29–38. IEEE (2008). https://doi.org/10.1109/ASE.2008.13

[18] Beyer, D., Jakobs, M.C., Lemberger, T., Wehrheim, H.: Reducer-based construction of conditional verifiers. In: Proc. ICSE. pp. 1182–1193. ACM (2018). https://doi.org/10.1145/3180155.3180259

# References IV


[19] 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

[20] Beyer, D., Kanav, S., Richter, C.: Construction of Verifier Combinations Based on Off-the-Shelf Verifiers. In: Proc. FASE. pp. 49–70. Springer (2022). https://doi.org/10.1007/978-3-030-99429-7_3

[21] Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_16

[22] Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD. pp. 189–197. FMCAD (2010)

[23] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. arXiv/CoRR **2208**(05046) (July 2022). https://doi.org/10.48550/arXiv.2208.05046

[24] Beyer, D., Löwe, S.: Explicit-state software model checking based on CEGAR and interpolation. In: Proc. FASE. pp. 146–162. LNCS 7793, Springer (2013). https://doi.org/10.1007/978-3-642-37057-1_11

# References V


[25] Beyer, D., Löwe, S., Novikov, E., Stahlbauer, A., Wendler, P.: Precision reuse for efficient regression verification. In: Proc. FSE. pp. 389–399. ACM (2013). https://doi.org/10.1145/2491411.2491429

[26] Beyer, D., Wendler, P.: Algorithms for software model checking: Predicate abstraction vs. IMPACT. In: Proc. FMCAD. pp. 106–113. FMCAD (2012)

[27] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643

[28] Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R.: Handbook of Model Checking. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8

[29] Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988, Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_15

[30] Clarke, E.M., Kröning, D., Sharygina, N., Yorav, K.: SATABS: SAT-based predicate abstraction for ANSI-C. In: Proc. TACAS. pp. 570–574. LNCS 3440, Springer (2005). https://doi.org/10.1007/978-3-540-31980-1_40

[31] Cok, D.R., Déharbe, D., Weber, T.: The 2014 SMT competition. JSAT **9**, 207–242 (2016)

# References VI


[32] Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. J. Symb. Log. **22**(3), 250–268 (1957). `https://doi.org/10.2307/2963593`

[33] Cruanes, S., Hamon, G., Owre, S., Shankar, N.: Tool integration with the Evidential Tool Bus. In: Proc. VMCAI. pp. 275–294. LNCS 7737, Springer (2013). `https://doi.org/10.1007/978-3-642-35873-9_18`

[34] Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: Proc. CAV. pp. 72–83. LNCS 1254, Springer (1997). `https://doi.org/10.1007/3-540-63166-6_10`

[35] Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: Proc. POPL. pp. 58–70. ACM (2002). `https://doi.org/10.1145/503272.503279`

[36] Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D.: The RERS grey-box challenge 2012: Analysis of event-condition-action systems. In: Proc. ISoLA. pp. 608–614. LNCS 7609, Springer (2012). `https://doi.org/10.1007/978-3-642-34026-0_45`

[37] Huisman, M., Klebanov, V., Monahan, R.: VerifyThis 2012: A program verification competition. STTT **17**(6), 647–657 (2015). `https://doi.org/10.1007/s10009-015-0396-8`

[38] Kildall, G.A.: A unified approach to global program optimization. In: Proc. POPL. pp. 194–206. ACM (1973). `https://doi.org/10.1145/512927.512945`

# References VII

[39] McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV. pp. 1–13. LNCS 2725, Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_1

[40] McMillan, K.L.: Lazy abstraction with interpolants. In: Proc. CAV. pp. 123–136. LNCS 4144, Springer (2006). https://doi.org/10.1007/11817963_14

[41] Shankar, N.: Little engines of proof. In: Proc. FME. pp. 1–20. LNCS 2391, Springer (2002). https://doi.org/10.1007/3-540-45614-7_1

[42] Turing, A.: Checking a large routine. In: Report on a Conference on High Speed Automatic Calculating Machines. pp. 67–69. Cambridge Univ. Math. Lab. (1949)