The Transformation Game: Joining Forces for Verification

Dirk Beyer
LMU Munich, Germany

SoSy-Lab @ LMU Munich

June 25, 2025, at Petri Nets '25 in Paris

Background of this presentation: Automatic Software Verification

C Program

```
int main() {
    int a = foo();
    int b = bar(a);

    assert(a == b);
}
```

Verification Tool

TRUE+witness
i.e., specification is satisfied

FALSE+witness
i.e., bug found

Status on Software Verifiers

- From lack of verifiers to plentitude
- 76 verification tools publicly available [41]
- SV-COMP 2025: 62 verification tools and 18 witness validation tools

Competitions in Software Verification and Testing

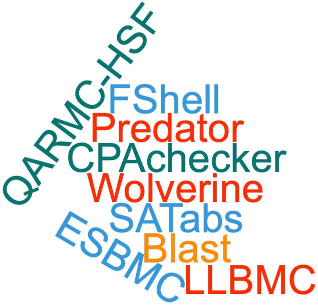
Mature research area, and there are tool competitions (alphabetic order):

- RERS: off-site, tools, free-style [57]
- SV-COMP: off-site, automatic tools, controlled [10]
- Test-Comp: off-site, automatic tools, controlled [11]
- VerifyThis: on-site, interactive, teams [58]

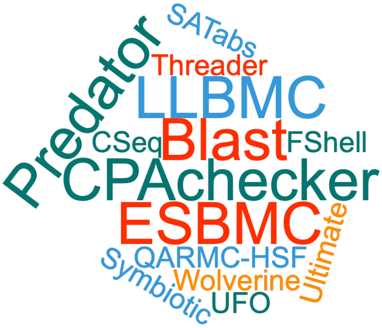
Broader in formal methods:

- MCC [3]
- SAT-COMP [8]
- SMT-COMP [9]
- TPTP [70]
- HWMCC [43]

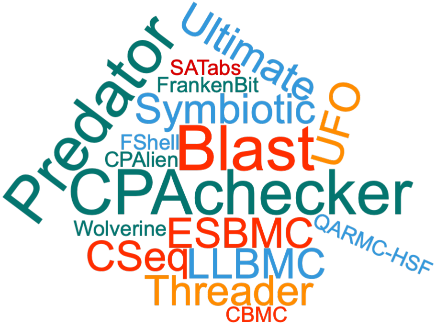
SV-COMP (Automatic Tools 2012)



SV-COMP (Automatic Tools 2013, cumulative)

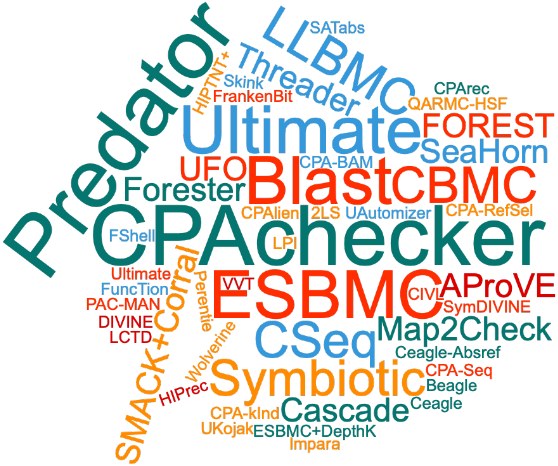


SV-COMP (Automatic Tools 2014, cumulative)

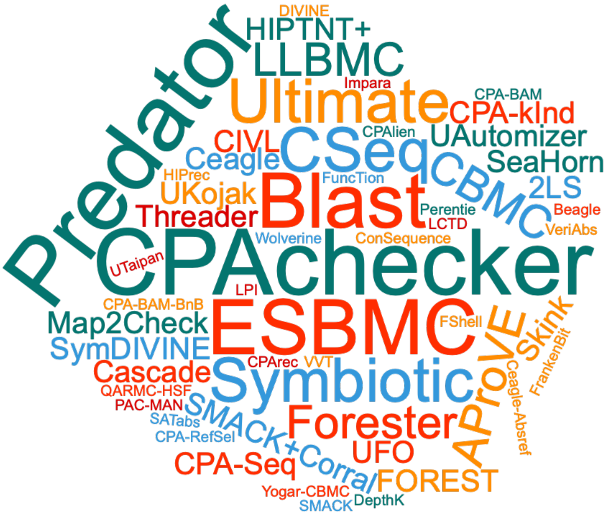


SV-COMP (Automatic Tools 2015, cumulative)

SV-COMP (Automatic Tools 2016, cumulative)

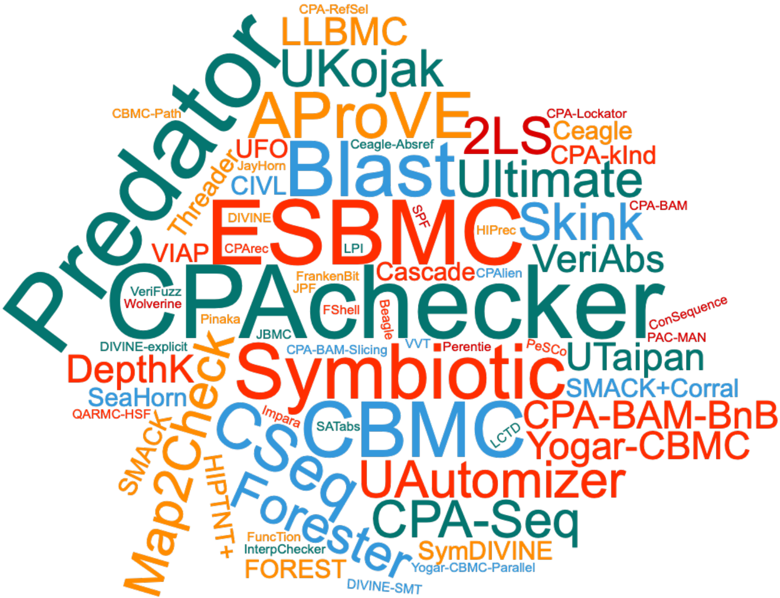


SV-COMP (Automatic Tools 2017, cumulative)



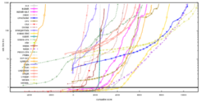
SV-COMP (Automatic Tools 2018, cumulative)

SV-COMP (Automatic Tools 2019, cumulative)

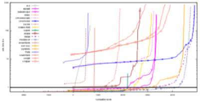


Different Strengths

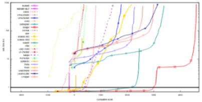
ReachSafety
1. CPAchecker
2. ESBMC-kind
3. CPV



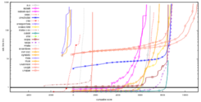
MemSafety
1. CPAchecker
2. Symbiotic
3. UAutomizer



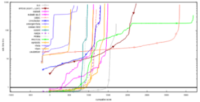
ConcurrencySafety
1. Deagle
2. Dartagnan
3. UGemCutter



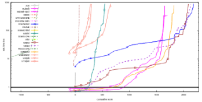
NoOverflows
1. UAutomizer
2. UTaipan
3. UKojak



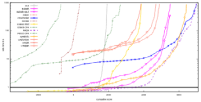
Termination
1. PROTON
2. UAutomizer
3. AProVE (KoAT + LoAT)



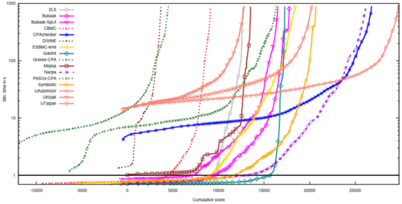
SoftwareSystems
1. CPAchecker
2. Mopsa
3. Symbiotic



FalsificationOverall
1. CPAchecker
2. Symbiotic
3. Bubaak



Overall
1. UAutomizer
2. CPAchecker
3. Symbiotic



https://sv-comp.sosy-lab.org/2025/results

Different Techniques (Extract from Report)

Table 8: Algorithms and techniques used by the participating tools;
∅ for inactive, meta for meta verifiers, and new for first-time participants

Tool	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	Algorithm Selection	Portfolio	Task Translation
2LS				✓	✓			✓	✓		✓						✓				
AISE			✓																		
APROVE																	✓				
BRICK	✓		✓	✓				✓								✓					
BUBAAK			✓								✓					✓	✓			✓	
BUBAAK-SPLIT			✓		✓						✓				✓	✓	✓		✓	✓	
CBMC ∅				✓							✓					✓					
COASTAL ∅			✓																		
CONCURRENTW2T																✓					
COOPERACE meta new																✓			✓	✓	
CPACHECKER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	
CPALOCKATOR ∅	✓	✓					✓				✓	✓	✓	✓		✓					
CPA-BAM-BNB ∅	✓	✓					✓				✓	✓	✓	✓							
CPA-BAM-SMG ∅																					
CPA-W2T ∅						✓						✓			✓						
CPROVER-W2T ∅				✓																	
CPV	✓	✓		✓	✓	✓					✓			✓					✓	✓	✓
CRUX ∅			✓																		
CSEQ ∅				✓							✓					✓					

(continues on next page)

Competition Report [38]

https://doi.org/10.1007/978-3-031-90660-2_9

Example CPACHECKER [29]: Many Concepts

- Included Concepts:
  - CEGAR [49] Interpolation [33, 21]
  - Configurable Program Analysis [24, 25]
  - Adjustable-block encoding [30]
  - Conditional model checking [23]
  - Verification witnesses [19, 17]
  - Various abstract domains: predicates, intervals, BDDs, octagons, explicit values
- Available analyses approaches:
  - Predicate abstraction [15, 30, 25, 34]
  - IMPACT algorithm [65, 40, 21]
  - Bounded model checking [50, 21]
  - k-Induction [20, 21]
  - IC3/Property-directed reachability [16]
  - Explicit-state model checking [33]
  - Interpolation-based model checking [31]

Insights from Software Model Checking

- Verifiers have different strengths
- There are plenty of tools
- $\Rightarrow$ Combination of Verification Approaches

Cooperative Verification — Think big!

- Introduce a new level!
- Current tools should become "low level" components (engines)
- Construct combinations
- Clear Interfaces
  via, e.g., Conditions, Witnesses, Test Suites
- Success: SAT, SMT (common interfaces, usable as libraries)
- See also: Little Engines [69], Evidential Tool Bus [51]

Verification by Transformations

Vision: Modular Transformation Paradigm

- Standalone and reusable transformers to construct verifiers
- Well-defined interfaces and exchange formats
- Construction recipes: easy to build new verifiers for different applications

Inputs and Outputs of Transformers: Artifacts

Type	Notation	Usage
Model	$\mathcal{M}$	Description of the system under verification
Specification	Φ	Expected behavior of the system under verification
Verdict	$\mathcal{R}$	Decision on whether a model satisfies a specification
Witness	Ω	Certificate explaining the verdict of a tool
Verification condition	$\mathcal{VC}$	Set of constraints that encode the behavior of a model

Example Transformers

Type	Signature	Functionality
Translator	$\mathcal{M} \mapsto \mathcal{M}$	Translates a model to a behaviorally equivalent one in a different language
Encoder	$\mathcal{M} \mapsto \mathcal{VC}$	Describes partial or complete behavior of a model as a verification condition
Specification transformer	$\mathcal{M} \times \Phi \mapsto \mathcal{M} \times \Phi$	Converts a verification task to an equisatisfiable one with a different specification
Witness transformer	$\mathcal{M} \times \Omega \mapsto \Omega$	Transforms a witness for a model to another witness, e.g., by making it more precise
Pruner	$\mathcal{M} \times \Omega \mapsto \mathcal{M}$	Removes irrelevant or fully-explored parts of a model based on a witness

Literature

The Transformation Game: Joining Forces for Verification, Festschrift 60th Birthday Jost-Pieter Katoen, 2024, available at doi:10.1007/978-3-031-75778-5_9

Application Examples

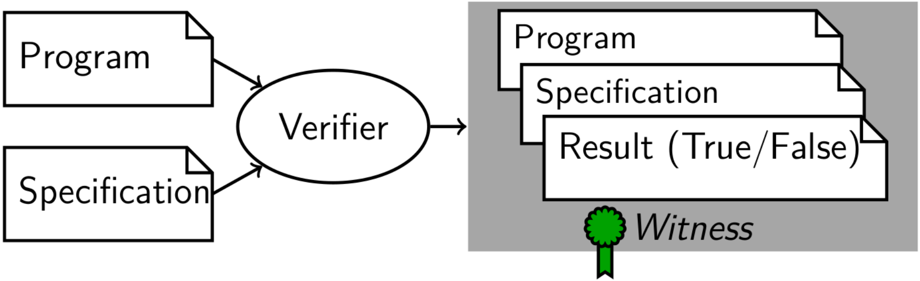
- (A1) Verification Witnesses and Validation
- (A2) LIV: Decomposing Validator
- (A3) CoVeriTeam: Language and Tool for Combination
- (A4) Simple Combinations
- (A5) Btor2C: Transforming from Hardware to Software
- (A6) Certifying Verification for BTOR2 with SV Tools
- (A7) Transformation-Based Verification with MoXI

Application Examples

- (A8) Transformation of Specifications
- (A9) Conditional Model Checking (CMC)
- (A10) Reducer-Based CMC
- (A11) Modularization of CEGAR
- (A12) Combining Interactive and Automatic Methods
- (A13) Loop Abstraction

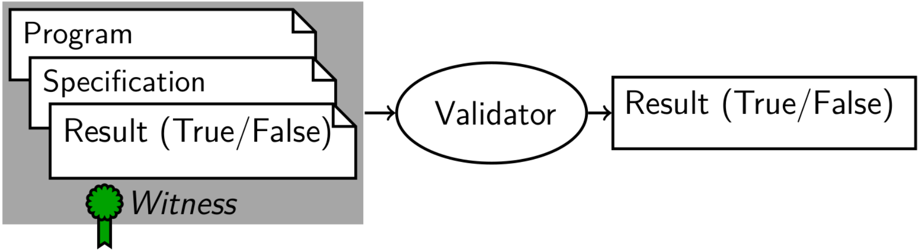
(A1) Software Verification with Witnesses

Witnesses are an important interface between tools.



[19, Proc. FSE 2015] [17, Proc. FSE 2016] [18, TOSEM 2022]

(A1) Witness-Based Result Validation



- Validate untrusted results
- Reestablish proof of correctness or violation
- Easier than full verification

(A1) Verification and Validation

Given program P and specification φ

- Verification: **prove** that $P \models \varphi$
  (mainly invariant construction)
- Validation with witness w: **re-prove** that $P \models \varphi$

AI can be used to

- **write** programs
- **suggest** invariants for programs

(A1) Correctness Witnesses

Program P, specification φ, proof π

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

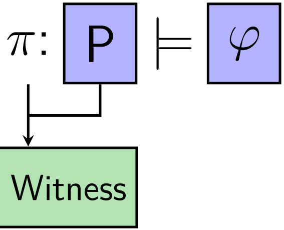
(A1) Correctness Witnesses

Program P, specification φ, proof π

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

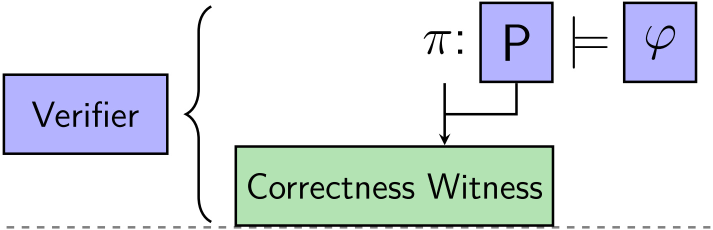
(A1) Correctness Witnesses

Program P, specification φ, proof π



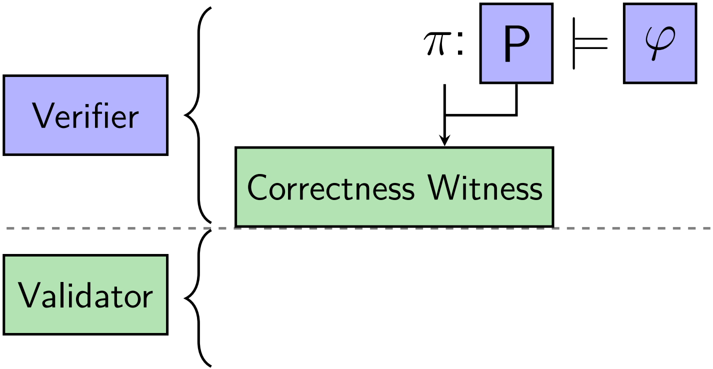
(A1) Correctness Witnesses

Program P, specification φ, proof π



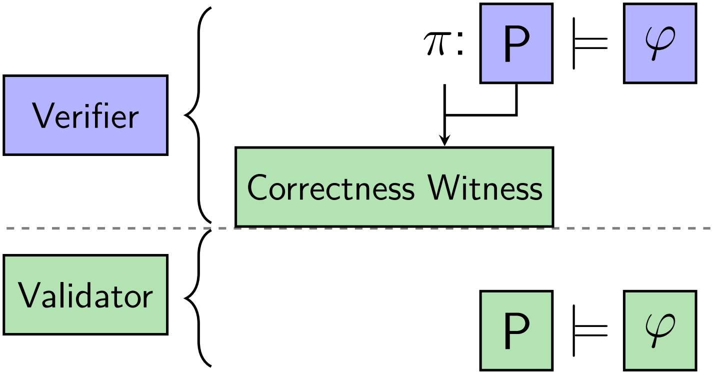
(A1) Correctness Witnesses

Program P, specification φ, proof π



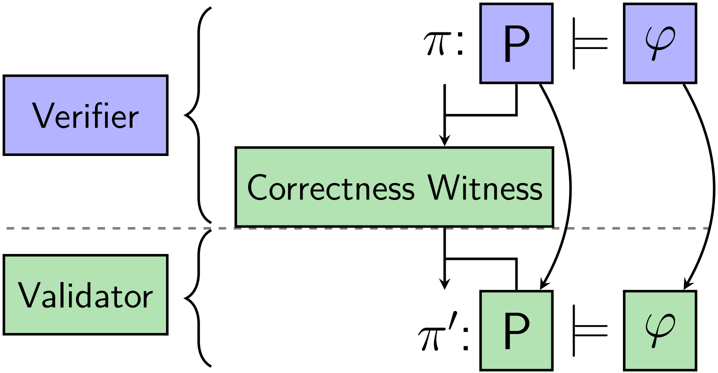
(A1) Correctness Witnesses

Program P, specification φ, proof π



(A1) Correctness Witnesses

Program P, specification φ, proof π



(A1) Example Program and Witness

Program:

```
int main() {
  unsigned char n = __nondet_uchar();
  if (n == 0) {
    return 0;
  }
  unsigned char v = 0;
  unsigned int  s = 0;
  unsigned int  i = 0;
  while (i < n) {
    v = __nondet_uchar();
    s += v;
    ++i;
  }
  if (s < v) {
    reach_error();
    return 1;
  }
  if (s > 65025) {
    reach_error();
    return 1;
  }
  return 0;
}
```

Witness (format v2.0):

```
content:
  - invariant:
      type: loop_invariant
      location:
        file_name: "inv-a.c"
        line: 12
        column: 1
        function: main
      value: "s <= i*255 && 0 <= i && i <= 255 && n <= 255"
      format: c_expression
```

(A1) State of the Art

- 18 validators exist for C and Java
- 4 formats for witnesses exist
  (GraphML and YAML, correctness and validation)
- Competition on Software Verification (SV-COMP) has a validation track

Certifying Algorithms [64] are used also in SAT and SMT.

(A2) LIV — Decomposing Validator

[36, Proc. ASE 2023], Idea from A. Appel

Program:

```
int x = 0;
int sum = 0 ;
//@ loop invariant I;
while (x<10) {
   x++;
   sum+=x;
}
assert(sum<=55);
```

Proof Obligations:

- $\{P\}s_0\{Inv\}$
- $\{Inv \wedge Cond\}Body\{Inv\}$
- $Inv \Rightarrow Q$

(A2) From Proof Obligations to Straight-Line Programs

Proof Obligations:

- $\{P\}s_0\{Inv\}$ (Base Case)
- $\{Inv \wedge Cond\}Body\{Inv\}$ (Inductiveness)
- $Inv \wedge \neg Cond \Rightarrow Q$ (Safety)

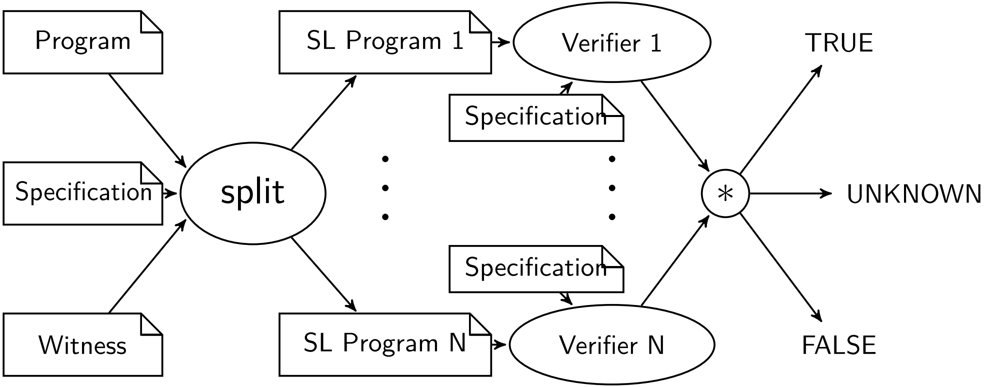
Straight-Line Programs:

```
int x = 0;
int sum = 0;
assert (Inv);
```

```
int x = nondet();
int sum = nondet();
assume(Inv && C);
x++;
sum += x;
assert (Inv);
```

```
int x = nondet();
int sum = nondet();
assume(Inv && ! C);
assert (Q);
```

(A2) Workflow of LIV



- Can use any off-the-shelf verifier from SV-COMP as backend
- Small frontend using pycparser for AST-based splitting

(A3) Example Combination (in DSL CoVeriTeam)

CoVeriTeam: Language and Tool [27, Proc. TACAS 2022]

Algorithm Witness Validation [19, 17]

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)
```

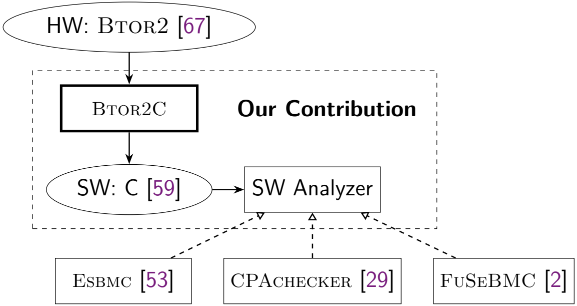
(A4) Simple Combination without Cooperation

Often, even simple combinations help!

Portfolio construction using off-the-shelf
verification tools [28, 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)

(A5) Btor2C: Transforming from Hardware to Software



- **43** HW-verification tasks uniquely solved by SW analyzers in our evaluation
  $\rightarrow$ enhance HW quality assurance using SW analyzers

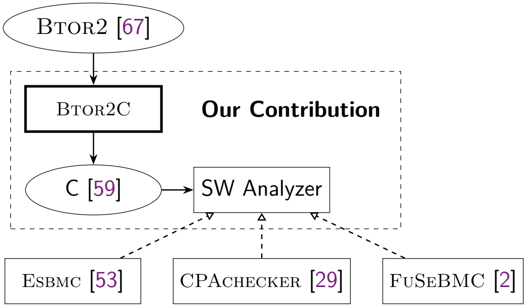
[14, Proc. TACAS ’23]

(A5) Btor2C: Btor2-to-C Translator

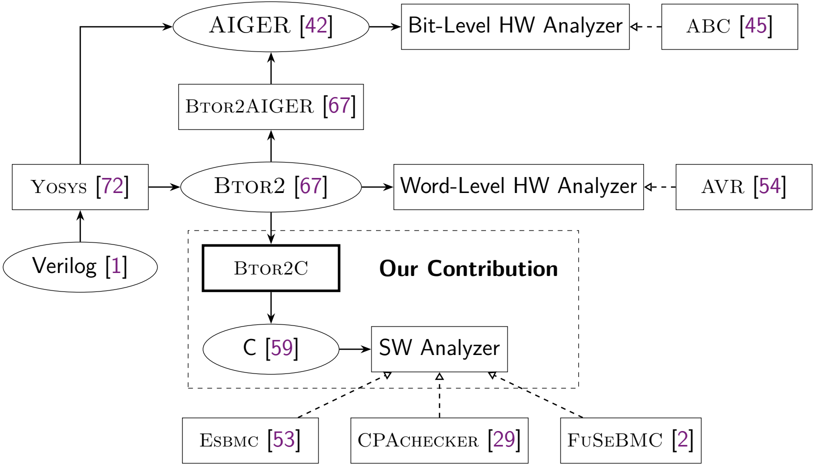
- A lightweight tool
  - Written in C++ with ~2 K LOC
  - Use the frontend parser provided by Btor2Tools [66]
- Open-source under Apache License 2.0

https://gitlab.com/sosy-lab/software/btor2c

(A5) BTOR2C in Action



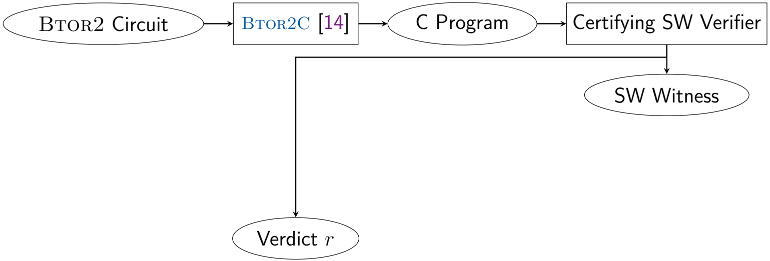
(A5) Btor2C in Action



(A5) Results using Btor2C

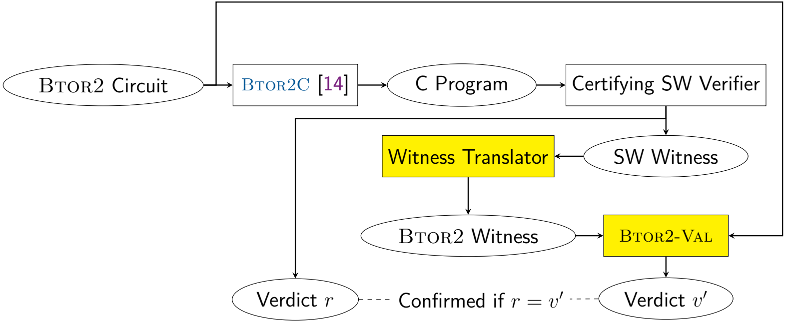
Tool	ABC	AVR	CPAchecker	Esbmc	VeriAbs
Algorithm	PDR	PDR	PA	KI	LA
Input	AIGER	Btor2	C (bit-masking applied lazily)		
Correct results	**862**	736	280	**410**	393
BV proofs	**524**	458	**189**	93	49
BV alarms	**338**	233	91	315	**342**
Array proofs	–	**45**	0	0	0
Array alarms	–	0	0	**2**	**2**

(A6) Certifying Verification for BTOR2 with SV Tools



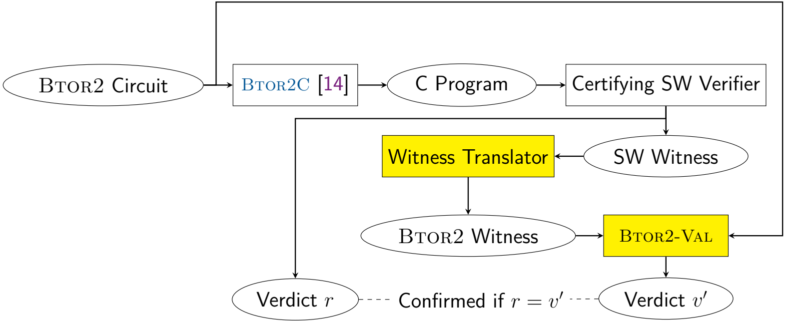
- BTOR2 [67] word-level circuits and translator BTOR2C [14]
- Software verifiers in SV-COMP [13]

(A6) Certifying Verification for Btor2 with SV Tools



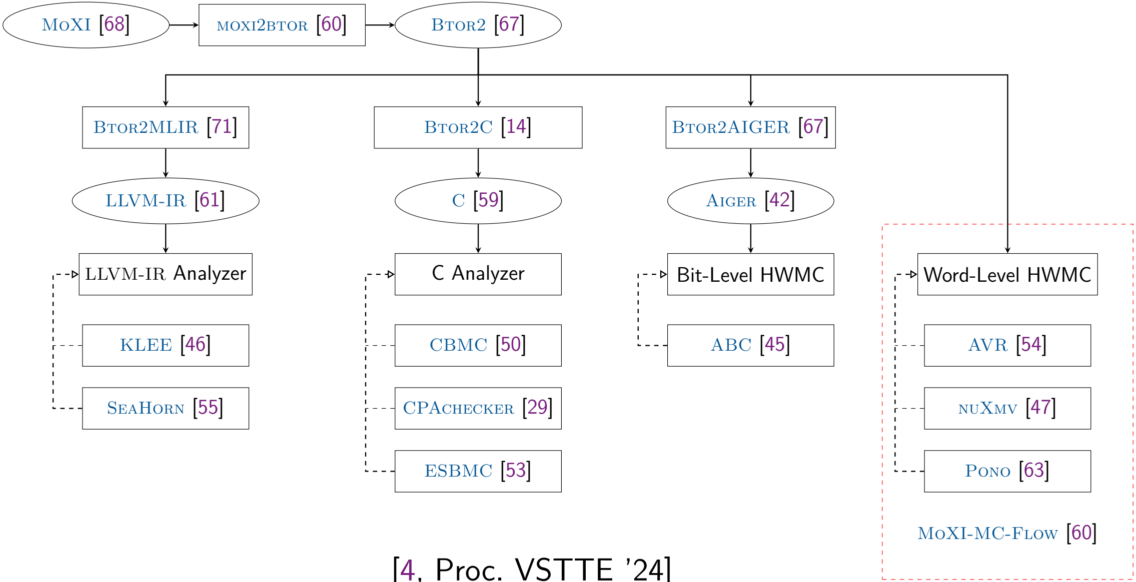
- Btor2 [67] word-level circuits and translator Btor2C [14]
- Software verifiers in SV-COMP [13]
- SW-to-HW witness translation and Btor2-Val

(A6) Certifying Verification for BTOR2 with SV Tools



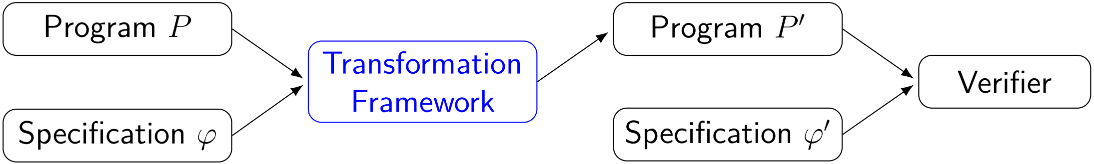
- BTOR2 [67] word-level circuits and translator BTOR2C [14]
- Software verifiers in SV-COMP [13]
- SW-to-HW witness translation and BTOR2-VAL
- On 1214 BTOR2 circuits, BTOR2-CERT achieved that
  - CBMC [50] found 37 bugs that ABC [45] missed
  - derived invariants by CPACHECKER [29] accelerated ABC

(A7) Transformation-Based Verification with MoXI



[4, Proc. VSTTE '24]

(A8) Transformation of Specifications

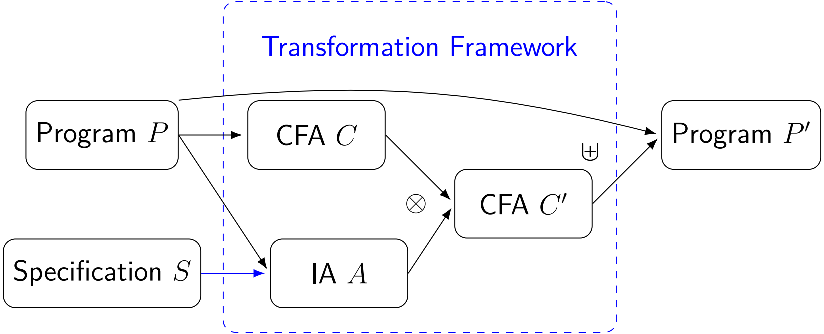


Our framework:

- *Easy to adopt* $\rightarrow$ Used by three tools in SV-COMP 25
- *Moudular* $\rightarrow$ Can be used by any verifier supporting SV-COMP syntax
- *Configurable* $\rightarrow$ The transformations given by *Instrumentation Automata (IA)*

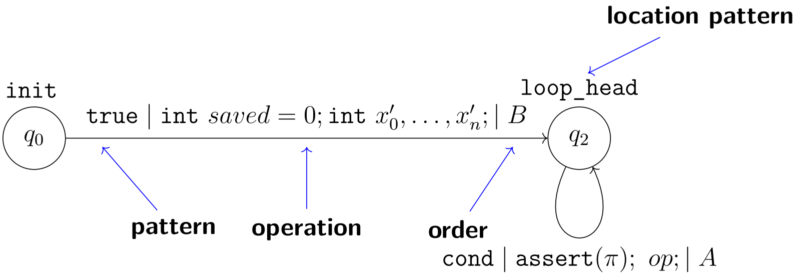
Proc. SPIN 2025

(A8) Transformation of $P \models \varphi$ to $P' \models \varphi'$



- Instrumentation Automaton (IA)
- Sequentialization Operation ($\otimes$)
- Instrumentation Operation ($\uplus$)

(A8) An Instrumentation Automaton for Termination

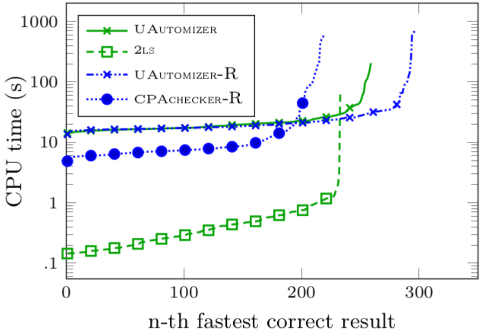


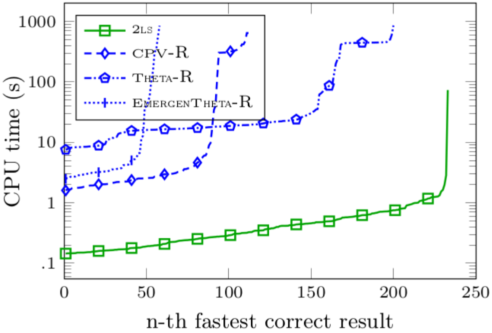
$op \equiv$ [nondet() $\wedge$ $saved = 0$] ? $x'_0 = x_0; \dots ; x'_n = x_n; saved = 1;$
$\pi \equiv (saved = 1) \Rightarrow (x'_0 \neq x_0 \vee x'_1 \neq x_1 \vee \cdots \vee x'_n \neq x_n);$

(A8) Tools and Their Specifications

Tool	reachability	no overflow	memory cleanup	termination
CPACHECKER [5]	✓	✓	✓	✓
UAUTOMIZER [56]	✓	✓	✓	✓
UTAIPAN [52]	✓	✓	✗	✗
2LS [62]	✓	✗	✗	✓
THETA [7]	✓	✗	✗	✗
EMERGENTHETA [6]	✓	✗	✗	✗
CPV [48]	✓	✗	✗	✗

(A8) Results for Termination → Reachability



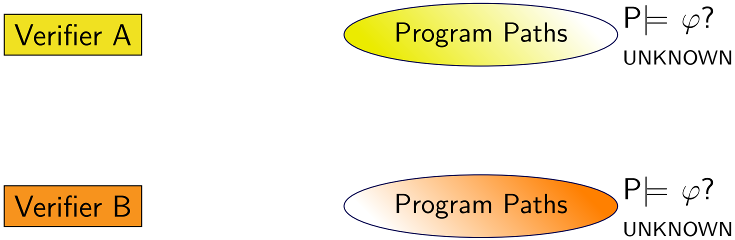


(A8) Results on Termination Reduction

Results (#Tasks)		UAutomizer	2ls	UAutomizer-R	CPAchecker-R
Correct	377	312	259	**333**	121
Proofs	264	250	189	**264**	55
Alarms	69	62	**70**	69	66

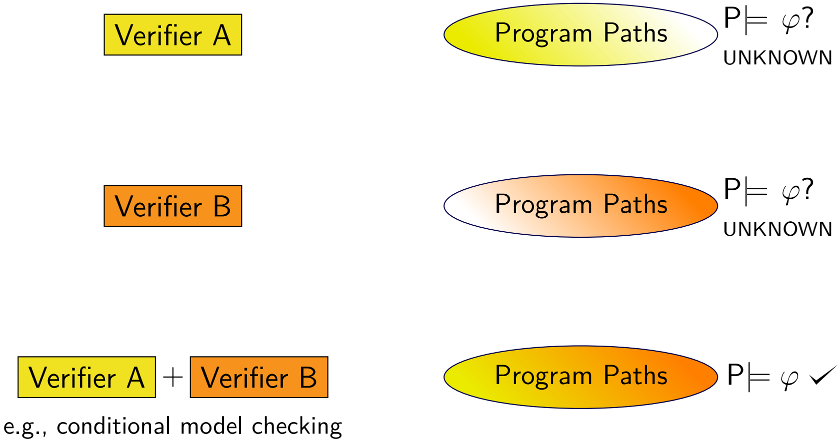
(A9) Facing Hard Verification Tasks

Given: Program $P \models \varphi$?

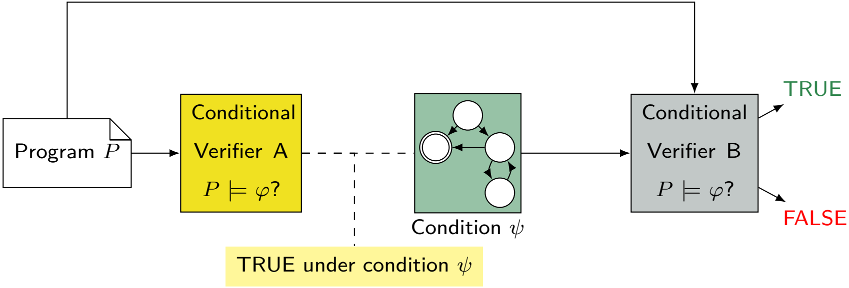


(A9) Facing Hard Verification Tasks

Given: Program P $\models \varphi$?

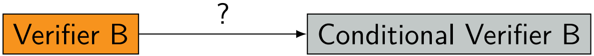


(A9) Conditional Model Checking



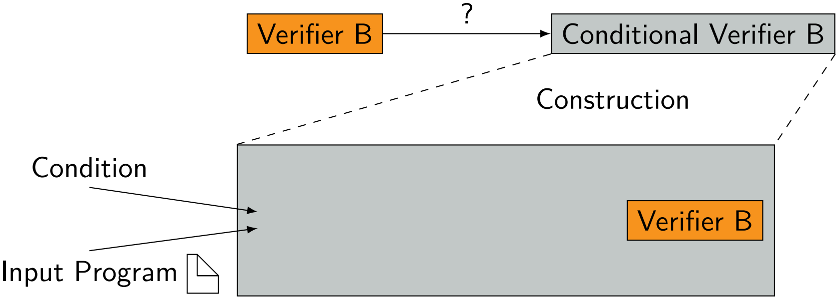
Proc. FSE 2012 [23]

(A10) Reducer-Based Construction



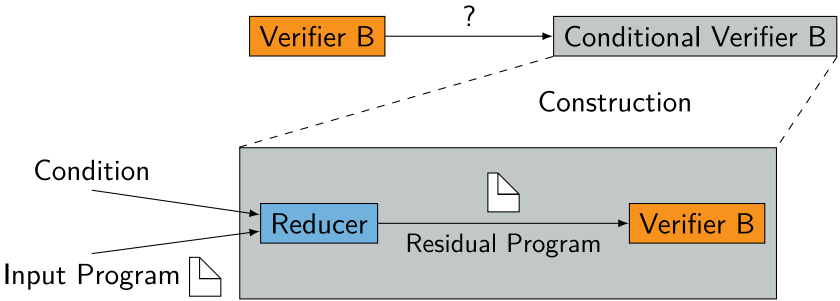
Proc. ICSE 2018 [26]

(A10) Reducer-Based Construction



Proc. ICSE 2018 [26]

(A10) Reducer-Based Construction

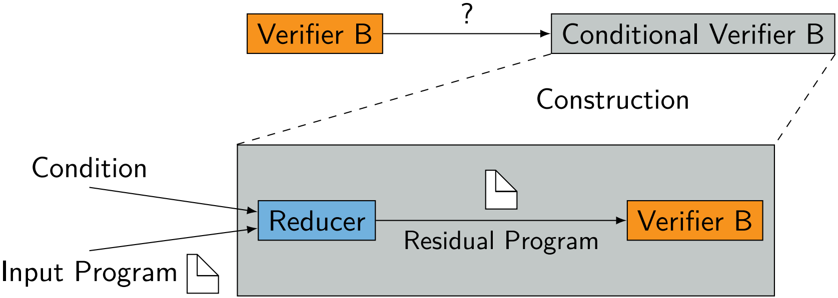


Reducer (preprocessor)

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

Proc. ICSE 2018 [26]

(A10) Reducer-Based Construction



Reducer (preprocessor)

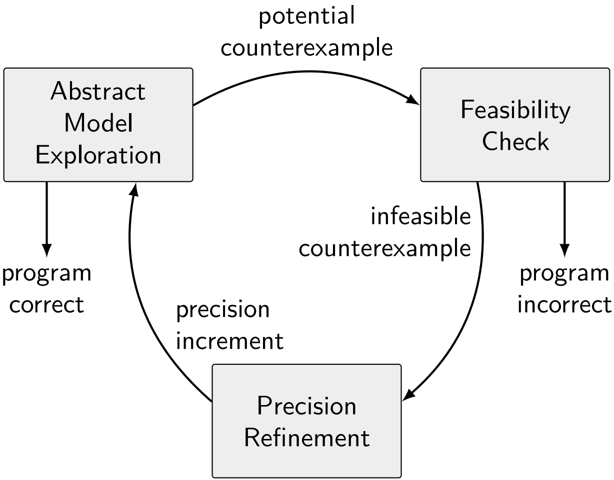
- Builds standard input (C program)
- Representing a subset of paths
- Contains at least all non-verified paths

\+ Verifier-unspecific approach

\+ Many conditional verifiers possible

Proc. ICSE 2018 [26]

(A11) CEGAR



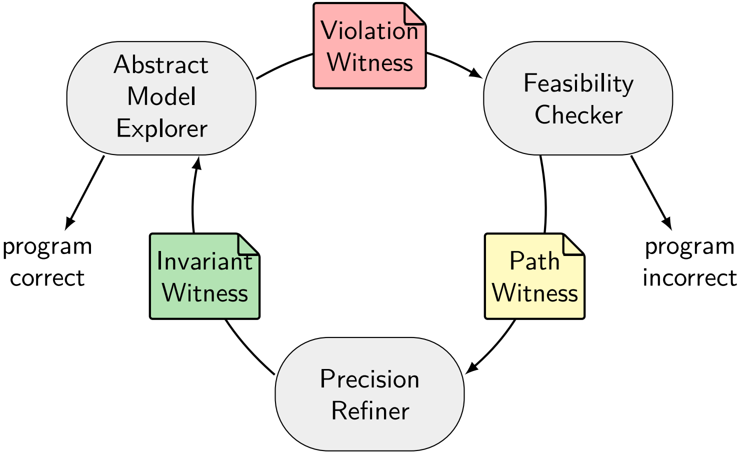
potential counterexample
Abstract Model Exploration
Feasibility Check
infeasible counterexample
program correct
program incorrect
precision increment
Precision Refinement

(A11) Modularization of CEGAR

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

$\Rightarrow$ Modular reformulation

(A11) Workflow of Modular CEGAR



Proc. ICSE 2022 [22]

(A12) Interactive and Automatic Methods

- How to achieve cooperation between automatic and interactive verifiers?
- Idea: Try to use existing interfaces for information exchange
- [37, Proc. SEFM '22]

```
//@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;
}
```

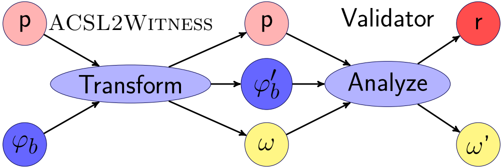
ACSL-annotated program, as used by FRAMA-C


```
...
<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>
...
```

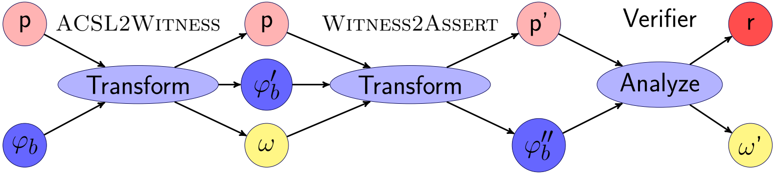

GraphML-based witness automaton generated by automatic verifiers

(A12) From Components: Construct Interactive Verifiers

- Turn a witness validator into an interactive verifier:



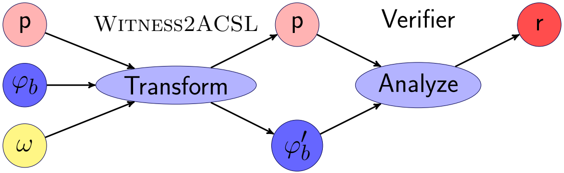
- Turn an automatic verifier into an interactive verifier:



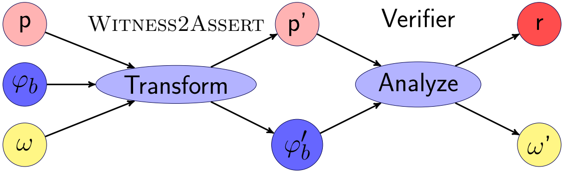
- Annotating in ACSL is more human-readable than witness automata
- Works for a wide range of automatic verifiers/validators

(A12) Component Framework: Constructing Validators

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

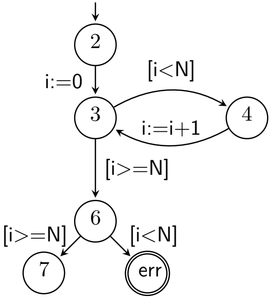


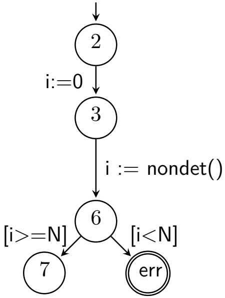
- Turn an automatic verifier into a validator [35, CAV '20]:



(A13) 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

(A13) Example: Havoc Abstraction

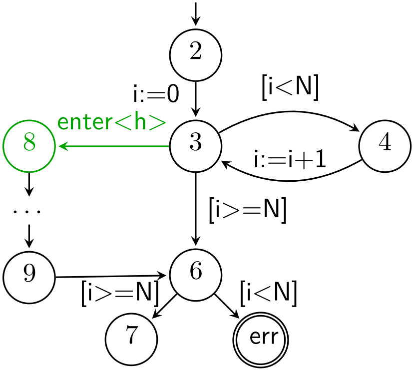
```
void main() {
  int i = 0;
  while (i<N) {
    i=i+1;
  }
  assert (i>=N);
}
```

```
void main() {
  int i = 0;
  if (i<N) {
    i = nondet();
    assume(!(i<N));
  }
  assert (i>=N);
}
```

- Havoc Abstraction: if loop is entered, havoc all input variables of the loop and perform one loop iteration, then assume the loop is left
- Only sound if the loop body does not contain assertions
- Overapproximation, but sometimes enough (as in this example)

(A13) 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: $\sigma : L \to S$
- In the example:
  $S = \{b, h\}$
  $\sigma(8) = h$
  $\sigma(l) = b$ for all l except 8
- Select allowed strategies during state-space exploration using σ
- [32, Proc. SEFM '22]



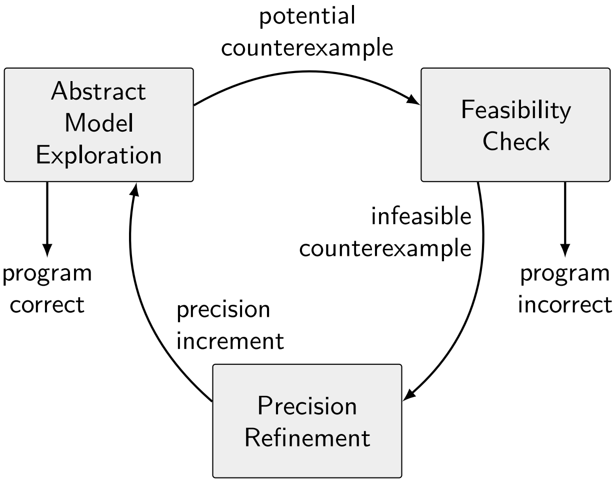
(A12) Accessibility of Loop Abstractions via Patches

- 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?
  $\Rightarrow$ 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));
```

(A11) CEGAR

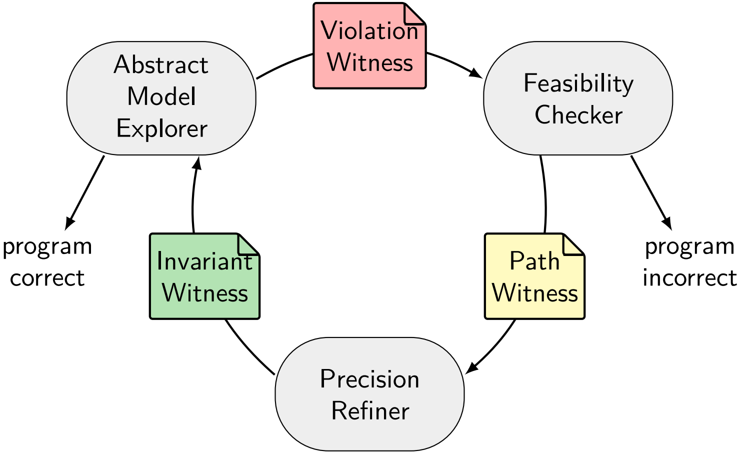

potential
counterexample
Abstract
Model
Exploration
Feasibility
Check
program
correct
infeasible
counterexample
program
incorrect
precision
increment
Precision
Refinement

(A11) Modularization of CEGAR

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

$\Rightarrow$ Modular reformulation

(A11) Workflow of modular CEGAR



Proc. ICSE 2022 [22]

Why Transformation?

- Join forces
- Re-use verifier components off-the-shelf
- Divide and conquer
- Robust components because more widely used and tested
- Community involvement
- Have a tool chain and replace components for better ones
- Transformation tools as separate off-the-shelf components

Part 2: Verification Tools as Exchangable Components

Vision:

- All tools for formal methods work together to solve hard verification problems and make our world safer and more secure.
- Model checkers and theorem provers can be integrated into the software-development process as seamless as unit testing today.
- Model checkers, theorem provers, SMT solvers, and testers use common interfaces for interaction and composition.

Some Steps Towards the Vision

- **Find**: Which tools for software verification exist?
- ... for test-case generation?
- ... for SMT solving?
- ... for hardware verification?
- **Reuse**: How to get executables?
- Where to find documentation?
- Am I allowed to use it?
- How to use them?
- **Conserve**: Which operating system, libraries, environment?

Requirements for Solution

- Support documentation and reuse
- Easy to query and generate knowledge base
- Long-term availability/executability of tools
- Must come with tool support
- Approach must be compatible with competitions

Solution [12]

One central repository:
https://gitlab.com/sosy-lab/benchmarking/fm-tools which gives information about:

- Location of the tool (via DOI, just like other literature)
- License
- Contact (via ORCID)
- Project web site
- Options
- Requirements (certain Docker container / VM)
- Limits

Maintained by formal-methods community

Example: Entry for LTSmin [44]


```yaml
id: ltsmin
name: LTSmin
description: |
  LTSmin is a language-independent model-checking ...
input_languages:
  - B
  - DVE
  - ETF
  - PNML
  - Promela
  - ...
project_url: https://ltsmin.utwente.nl/
repository_url: https://github.com/utwente-fmt/ltsmin
spdx_license_identifier: BSD-3-Clause
benchexec_toolinfo_module:
   "https://www.cip.ifi.lmu.de/~wachowitz/ltsmin.py"
fmtools_format_version: "2.0"
fmtools_entry_maintainers:
  - ricffb
```

Example: LTSmin's Contacts


```yaml
maintainers:
  - orcid: 0000-0002-2433-4174
    name: Alfons Laarman
    institution: Leiden Institute for Advanced Computer Science
    country: Netherlands
    url: https://alfons.laarman.com/
```

Example: LTSmin's Versions

```yaml
versions:
  - version: "pnml2lts-sym-3.0.2"
    url:
       "https://github.com/utwente-fmt/ltsmin/releases/download/v3.0.2/ltsmin-v3.0.2-l
    benchexec_toolinfo_options: ["pnml2lts-sym"]
    required_ubuntu_packages: []
    base_container_image: ["ubuntu:24.04"]
```

Example: LTSmin's Documentation


```yaml
literature:
  - doi: 10.1007/978-3-662-46681-0_61
    title: "LTSmin:␣High-Performance␣Language-Independent␣Model␣
        Checking"
    year: 2015
  - doi: 10.1007/978-3-642-20398-5_40
    title: "Multi-Core␣LTSmin:␣Marrying␣Modularity␣and␣
        Scalability"
    year: 2011
  - doi: 10.1007/978-3-642-14295-6_31
    title: "LTSmin:␣Distributed␣and␣Symbolic␣Reachability"
    year: 2010
```

Example: LTSmin's Web-Page Entry

fm-tools.sosy-lab.org/#tool-ltsmin

Tools for Formal Methods: Tools

Tools · Techniques · Competitions · Frameworks · Input Languages · Documentation of the YAML Schema ↗ · Code on GitLab

Table of Contents

LTSmin

LTSmin is a language-independent model-checking toolset supporting multiple input languages and advanced state-space generation techniques via a unified PINS interface. It features modular architecture, symbolic/distributed/multi-core reachability, and easy extensibility for new languages.

Project URL: https://ltsmin.utwente.nl/

Repository URL: https://github.com/utwente-fmt/ltsmin

Maintainers: • Alfons Laarman

Supported input languages: • B • DVE • ETF • Event-B • MCRL • MCRL2 • PNML • Promela • TLA+ • Z

License: • BSD-3-Clause

Releases: • pnml2lts-sym-3.0.2

Literature:
- *LTSmin: High-Performance Language-Independent Model Checking. 2015.* DOI: 10.1007/978-3-662-46681-0_61
- *Multi-Core LTSmin: Marrying Modularity and Scalability. 2011.* DOI: 10.1007/978-3-642-20398-5_40
- *LTSmin: Distributed and Symbolic Reachability. 2010.* DOI: 10.1007/978-3-642-14295-6_31

FM-Tools is FAIR

- **F**indable:
  overview is available on internet,
  generated knowledge base
- **A**ccessible:
  data retrievable via Git, format is YAML
- **I**nteroperable:
  Format is defined in schema,
  archives identified by DOIs, researchers by ORCIDs
- **R**eusable:
  Data are CC-BY, each tool comes with a license,
  format of tool archive standardized

FM-WECK: Run Tools in Conserved Environment [39, Proc. FM 2024]

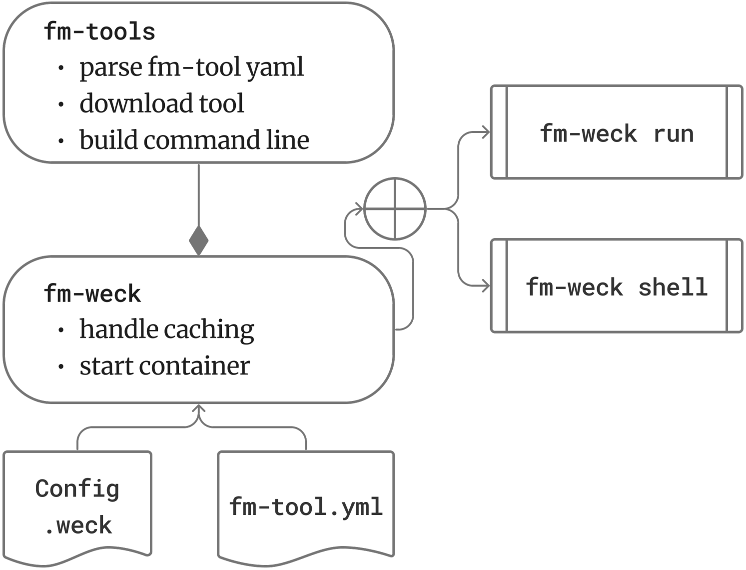
Refer to known fm-tools by name:version

`fm-weck` `run` `ltsmin` `philo.pnml`

Download, Install and run the tool

- No knowledge of the tools CLI needed
- Tool runs in a container (no dependencies on host system)

FM-WECK: Architecture



fm-weck

`fm-weck run` `fm-weck run -m` `fm-weck shell`

- Download and execute tool in container
- No knowledge of tool needed

- Download and execute tool in container
- Expert knowledge about tool required

- Spin up interactive shell in tool environment

Conclusion FM-Tools and FM-Weck

FM-Tools collects and stores essential information to:

- Generate a knowledge base about formal-methods tools [12]
  https://fm-tools.sosy-lab.org
- Conserve tool versions and their required environment
  (with help by Zenodo and Podman/Docker)
- Run a tool in conserved environment via FM-Weck [39]
- Please add your tool

https://fm-tools.sosy-lab.org

Conclusion

- Many verification tools and techniques
- External combinations are important
- Interfaces (artifacts, actors)
- Combinations and Cooperation
- Leverage Cooperation between Tools
- Conserve tools and make findable in FM-Tools

References I

[1] IEEE standard for Verilog hardware description language (2006). https://doi.org/10.1109/IEEESTD.2006.99495

[2] Alshmrany, K.M., Aldughaim, M., Bhayat, A., Cordeiro, L.C.: FuSeBMC: An energy-efficient test generator for finding security vulnerabilities in C programs. In: Proc. TAP. pp. 85–105. Springer (2021). https://doi.org/10.1007/978-3-030-79379-1_6

[3] Amat, N., Amparore, E.G., Berthomieu, B., Bouvier, P., Dal-Zilio, S., Hulin-Hubard, F., Jensen, P.G., Jezequel, L., Kordon, F., Li, S., Paviot-Adet, E., Petrucci, L., Srba, J., Thierry-Mieg, Y., Wolf, K.: Behind the scene of the model checking contest, analysis of results from 2018 to 2023. In: Proc. TOOLympics Challenge 2023. pp. 52–89. LNCS 14550, Springer (2023). https://doi.org/10.1007/978-3-031-67695-6_3

[4] Ates, S., Beyer, D., Chien, P.C., Lee, N.Z.: MoXIchecker: An extensible model checker for MoXI. In: Proc. VSTTE 2024. pp. 1–14. LNCS 15525, Springer (2025). https://doi.org/10.1007/978-3-031-86695-1_1

[5] Baier, D., Beyer, D., Chien, P.C., Jankola, M., Kettl, M., Lee, N.Z., Lemberger, T., Lingsch-Rosenfeld, M., Spiessl, M., Wachowitz, H., Wendler, P.: CPAchecker 2.3 with strategy selection (competition contribution). In: Proc. TACAS (3). pp. 359–364. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_21

[6] Bajczi, L., Szekeres, D., Mondok, M., Ádám, Z., Somorjai, M., Telbisz, C., Dobos-Kovács, M., Molnár, V.: EmergenTheta: Verification beyond abstraction refinement (competition contribution). In: Proc. TACAS (3). pp. 371–375. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_23

References II

[7] Bajczi, L., Telbisz, C., Somorjai, M., Ádám, Z., Dobos-Kovács, M., Szekeres, D., Mondok, M., Molnár, V.: THETA: Abstraction based techniques for verifying concurrency (competition contribution). In: Proc. TACAS (3). pp. 412–417. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_30

[8] Balyo, T., Heule, M.J.H., Järvisalo, M.: SAT Competition 2016: Recent developments. In: Proc. AAAI. pp. 5061–5063. AAAI Press (2017)

[9] Barrett, C., Deters, M., de Moura, L., Oliveras, A., Stump, A.: 6 years of SMT-COMP. J. Autom. Reasoning **50**(3), 243–277 (2013). https://doi.org/10.1007/s10817-012-9246-5

[10] 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

[11] 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

[12] Beyer, D.: Find, use, and conserve tools for formal methods. In: Proc. Festschrift Podelski 65th Birthday. Springer (2024), available online: https://www.sosy-lab.org/research/pub/2024-Podelski65.Find_Use_and_Conserve_Tools_for_Formal_Methods.pdf

[13] Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: Proc. TACAS (3). pp. 299–329. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_15

References III

[14] Beyer, D., Chien, P.C., Lee, N.Z.: Bridging hardware and software analysis with Btor2C: A word-level-circuit-to-C translator. In: Proc. TACAS (2). pp. 152–172. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_12

[15] 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

[16] 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

[17] 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

[18] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Lemberger, T., Tautschnig, M.: Verification witnesses. ACM Trans. Softw. Eng. Methodol. **31**(4), 57:1–57:69 (2022). https://doi.org/10.1145/3477579

[19] 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

[20] 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

[21] 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 IV

[22] 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

[23] 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

[24] 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

[25] 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

[26] 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

[27] 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

[28] 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

[29] 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

References V

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

[31] Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. J. Autom. Reasoning **69**(1), 5 (2025). https://doi.org/10.1007/s10817-024-09702-9

[32] Beyer, D., Lingsch-Rosenfeld, M., Spiessl, M.: A unifying approach for control-flow-based loop abstraction. In: Proc. SEFM. pp. 3–19. LNCS 13550, Springer (2022). https://doi.org/10.1007/978-3-031-17108-6_1

[33] 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

[34] 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

[35] Beyer, D., Spiessl, M.: MetaVal: Witness validation via verification. In: Proc. CAV. pp. 165–177. LNCS 12225, Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_10

[36] Beyer, D., Spiessl, M.: LIV: A loop-invariant validation using straight-line programs. In: Proc. ASE. pp. 2074–2077. IEEE (2023). https://doi.org/10.1109/ASE56229.2023.00214

[37] Beyer, D., Spiessl, M., Umbricht, S.: Cooperation between automatic and interactive software verifiers. In: Proc. SEFM. p. 111–128. LNCS 13550, Springer (2022). https://doi.org/10.1007/978-3-031-17108-6_7

References VI

[38] Beyer, D., Strejček, J.: Improvements in software verification and witness validation: SV-COMP 2025. In: Proc. TACAS (3). pp. 151–186. LNCS 15698, Springer (2025). https://doi.org/10.1007/978-3-031-90660-2_9

[39] Beyer, D., Wachowitz, H.: FM-Weck: Containerized execution of formal-methods tools. In: Proc. FM. pp. 39–47. LNCS 14934, Springer (2024). https://doi.org/10.1007/978-3-031-71177-0_3

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

[41] Beyer, D., Podelski, A.: Software model checking: 20 years and beyond. In: Principles of Systems Design. pp. 554–582. LNCS 13660, Springer (2022). https://doi.org/10.1007/978-3-031-22337-2_27

[42] Biere, A.: The AIGER And-Inverter Graph (AIG) format version 20071012. Tech. Rep. 07/1, Institute for Formal Models and Verification, Johannes Kepler University (2007). https://doi.org/10.35011/fmvtr.2007-1

[43] Biere, A., van Dijk, T., Heljanko, K.: Hardware model-checking competition 2017. In: Proc. FMCAD. p. 9. IEEE (2017). https://doi.org/10.23919/FMCAD.2017.8102233

[44] Blom, S., van de Pol, J., Weber, M.: LTSmin: Distributed and symbolic reachability. In: Proc. CAV. pp. 354–359. LNCS 6174, Springer (2010). https://doi.org/10.1007/978-3-642-14295-6_31

[45] Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: Proc. CAV. pp. 24–40. LNCS 6174, Springer (2010). https://doi.org/10.1007/978-3-642-14295-6_5

References VII

[46] Cadar, C., Dunbar, D., Engler, D.R.: KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. OSDI. pp. 209–224. USENIX Association (2008)

[47] Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The NUXMV symbolic model checker. In: Proc. CAV. pp. 334–342. LNCS 8559, Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_22

[48] Chien, P.C., Lee, N.Z.: CPV: A circuit-based program verifier (competition contribution). In: Proc. TACAS (3). pp. 365–370. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_22

[49] 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

[50] 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

[51] 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

[52] Dietsch, D., Heizmann, M., Klumpp, D., Schüssele, F., Podelski, A.: ULTIMATE TAIPAN 2023 (competition contribution). In: Proc. TACAS (2). pp. 582–587. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_40

References VIII

[53] Gadelha, M.R., Monteiro, F.R., Morse, J., Cordeiro, L.C., Fischer, B., Nicole, D.A.: ESBMC 5.0: An industrial-strength C model checker. In: Proc. ASE. pp. 888–891. ACM (2018). https://doi.org/10.1145/3238147.3240481

[54] Goel, A., Sakallah, K.: AVR: Abstractly verifying reachability. In: Proc. TACAS. pp. 413–422. LNCS 12078, Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_23

[55] Gurfinkel, A., Kahsai, T., Komuravelli, A., Navas, J.A.: The SeaHorn verification framework. In: Proc. CAV. pp. 343–361. LNCS 9206, Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_20

[56] Heizmann, M., Bentele, M., Dietsch, D., Jiang, X., Klumpp, D., Schüssele, F., Podelski, A.: Ultimate Automizer and the abstraction of bitwise operations (competition contribution). In: Proc. TACAS (3). pp. 418–423. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_31

[57] 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

[58] 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

[59] ISO/IEC JTC 1/SC 22: ISO/IEC 9899-2018: Information technology — Programming Languages — C. International Organization for Standardization (2018), https://www.iso.org/standard/74528.html

References IX

[60] Johannsen, C., Nukala, K., Dureja, R., Irfan, A., Shankar, N., Tinelli, C., Vardi, M.Y., Rozier, K.Y.: The MoXI model exchange tool suite. In: Proc. CAV. pp. 203–218. LNCS 14681, Springer (2024). https://doi.org/10.1007/978-3-031-65627-9_10

[61] Lattner, C., Adve, V.S.: LLVM: A compilation framework for lifelong program analysis and transformation. In: Proc. CGO. pp. 75–88. IEEE (2004). https://doi.org/10.1109/CGO.2004.1281665

[62] Malík, V., Schrammel, P., Vojnar, T., Nečas, F.: 2LS: Arrays and loop unwinding (competition contribution). In: Proc. TACAS (2). pp. 529–534. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_31

[63] Mann, M., Irfan, A., Lonsing, F., Yang, Y., Zhang, H., Brown, K., Gupta, A., Barrett, C.W.: Pono: A flexible and extensible SMT-based model checker. In: Proc. CAV. pp. 461–474. LNCS 12760, Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_22

[64] McConnell, R.M., Mehlhorn, K., Näher, S., Schweitzer, P.: Certifying algorithms. Computer Science Review **5**(2), 119–161 (2011). https://doi.org/10.1016/j.cosrev.2010.09.009

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

[66] Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Source-code repository of Btor2, BtorMC, and Boolector 3.0. https://github.com/Boolector/btor2tools, accessed: 2023-01-29

[67] Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2, BtorMC, and Boolector 3.0. In: Proc. CAV. pp. 587–595. LNCS 10981, Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_32

References X

[68] Rozier, K.Y., Dureja, R., Irfan, A., Johannsen, C., Nukala, K., Shankar, N., Tinelli, C., Vardi, M.Y.: MoXI: An intermediate language for symbolic model checking. In: Proc. SPIN. pp. 26–46. LNCS 14624, Springer (2024). https://doi.org/10.1007/978-3-031-66149-5_2

[69] 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

[70] Sutcliffe, G.: The CADE ATP system competition: CASC. AI Magazine **37**(2), 99–101 (2016). https://doi.org/10.1609/aimag.v37i2.2620

[71] Tafese, J., Garcia-Contreras, I., Gurfinkel, A.: Btor2MLIR: A format and toolchain for hardware verification. In: Proc. FMCAD. pp. 55–63. TU Wien Academic Press (2023). https://doi.org/10.34727/2023/ISBN.978-3-85448-060-0_13

[72] Wolf, C.: Yosys open synthesis suite. https://yosyshq.net/yosys/, accessed: 2023-01-29