Mopsa at SV-Comp

Raphaël Monat

rmonat.fr

APR à Lille
2 June 2025

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

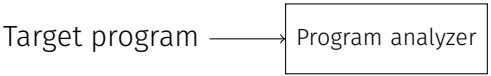
Target program

Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

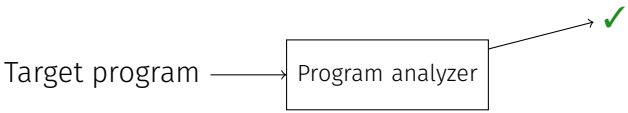


Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

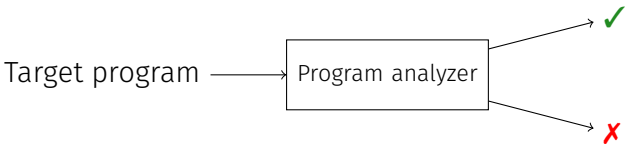


Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up

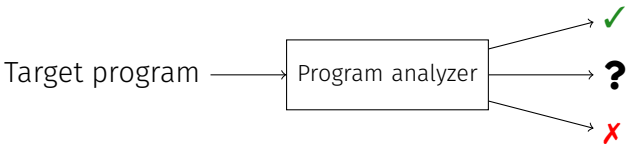


Automated Program Analysis

Motivation

Sheer quantity of programs and changes during their life:

Automated analyses will help scaling up



Turing & Rice to the Rescue

Sound All errors in program reported by analyzer

Turing & Rice to the Rescue

All errors reported by analyzer are replicable in program

Complete

Sound

All errors in program reported by analyzer

Turing & Rice to the Rescue

Guaranteed Termination

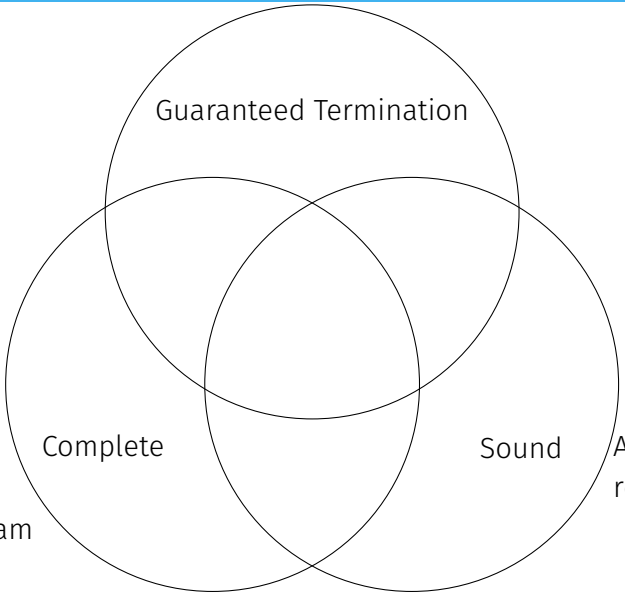
All errors reported by analyzer are replicable in program

Complete

Sound

All errors in program reported by analyzer

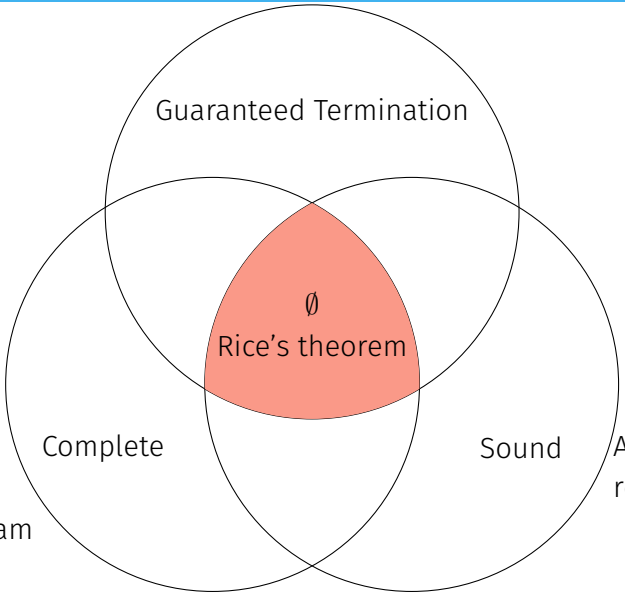
Turing & Rice to the Rescue



All errors reported by analyzer are replicable in program

All errors in program reported by analyzer

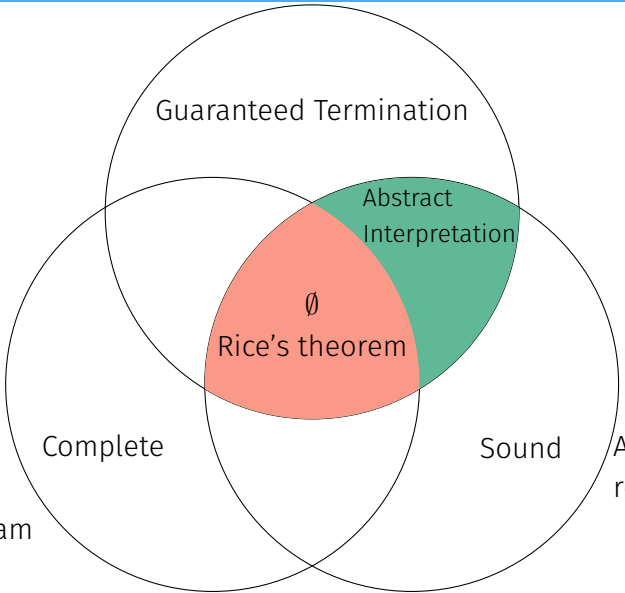
Turing & Rice to the Rescue (or not?)



All errors reported by analyzer are replicable in program

All errors in program reported by analyzer

Turing & Rice to the Rescue (or not?)



All errors reported by analyzer are replicable in program

All errors in program reported by analyzer

Advancing the state-of-the-art

How to evaluate, compare and improve automated program verification?

What about an academic competition?

SV-Comp since 2012, introduced by Dirk Beyer (LMU).

Outline

Mopsa Overview

Modular **O**pen **P**latform for **S**tatic **A**nalysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

Modular **O**pen **P**latform for **S**tatic **A**nalysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

Goals

- Explore new designs
  - Including multi-language support

Modular **O**pen **P**latform for **S**tatic **A**nalysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

Goals

- Explore new designs
  - Including multi-language support
- Ease development of relational static analyses
  - High expressivity

Modular **O**pen **P**latform for **S**tatic **A**nalysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

Goals

- Explore new designs
  - Including multi-language support
- Ease development of relational static analyses
  - High expressivity
- Open-source (LGPL)

Modular **O**pen **P**latform for **S**tatic **A**nalysis [Jou+19]

`gitlab.com/mopsa/mopsa-analyzer` or `opam install mopsa`

Started by ERC Consolidator Grant (2016-2021) of Antoine Miné (LIP6, SU)

Goals

- Explore new designs
  - Including multi-language support
- Ease development of relational static analyses
  - High expressivity
- Open-source (LGPL)
- Can be used as an experimentation platform

Contributors (2018–2025, chronological arrival order)

- A. Miné
- A. Ouadjaout
- M. Journault
- A. Fromherz
- D. Delmas
- R. Monat
- G. Bau
- F. Parolini
- M. Milanese
- M. Valnet
- J. Boillot

Contributors (2018–2025, chronological arrival order)

- **A. Miné**
- **A. Ouadjaout**
- M. Journault
- A. Fromherz
- D. Delmas
- **R. Monat**
- G. Bau
- F. Parolini
- M. Milanese
- M. Valnet
- J. Boillot

Maintainers in bold.

Analysis = composition of abstract domains

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms
- separation of concerns

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms
- separation of concerns
- allows reuse of domains across languages

Analysis = composition of abstract domains

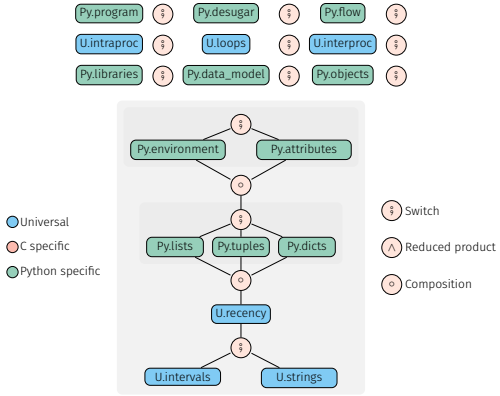
unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms
- separation of concerns
- allows reuse of domains across languages
- defined as json files in `share/mopsa/configs`

Analysis = composition of abstract domains

unified domain signature $\Longrightarrow$ iterators are abstract domains

- flexible architecture suitable for various programming paradigms
- separation of concerns
- allows reuse of domains across languages
- defined as json files in `share/mopsa/configs`



Summary of analyses

Languages
C [JMO18; OM20], Python [MOM20a; MOM20b]

Summary of analyses

Languages
C [JMO18; OM20], Python [MOM20a; MOM20b] Multilanguage Python+C [MOM21]

Summary of analyses

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b] Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23; VMM25], Catala (date arithmetic [MFM24])...

Summary of analyses

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b] Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23; VMM25], Catala (date arithmetic [MFM24])...

Properties

Summary of analyses

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b] Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23; VMM25], Catala (date arithmetic [MFM24])...

Properties

- Absence of RTEs

Summary of analyses

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b] Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23; VMM25], Catala (date arithmetic [MFM24])...

Properties

- **Absence of RTEs**
- Patch analysis [DM19]

Summary of analyses

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b] Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23; VMM25], Catala (date arithmetic [MFM24])...

Properties

- **Absence of RTEs**
- Patch analysis [DM19]
- Endianness portability [DOM21]

Summary of analyses

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b] Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23; VMM25], Catala (date arithmetic [MFM24])...

Properties

- **Absence of RTEs**
- Patch analysis [DM19]
- Endianness portability [DOM21]
- Non-exploitability [PM24]

Summary of analyses

Languages

C [JMO18; OM20], Python [MOM20a; MOM20b] Multilanguage Python+C [MOM21]

WIP: Michelson [Bau+22], OCaml [VMM23; VMM25], Catala (date arithmetic [MFM24])...

Properties

- **Absence of RTEs**
- Patch analysis [DM19]
- Endianness portability [DOM21]
- Non-exploitability [PM24]
- Sufficient precondition inference [MM24a; MM24b]

SV-Comp

Presentation of SV-Comp

Software-Verification Competition

- Yearly, since 2012

Presentation of SV-Comp

Software-Verification Competition

- Yearly, since 2012
- Part of ETAPS

Presentation of SV-Comp

Software-Verification Competition

- Yearly, since 2012
- Part of ETAPS
- Organized by Dirk Beyer (Munich) + Jan Strejček (since 2025)

Presentation of SV-Comp

Software-Verification Competition

- Yearly, since 2012
- Part of ETAPS
- Organized by Dirk Beyer (Munich) + Jan Strejček (since 2025)
- 62 participating tools in 2025

Presentation of SV-Comp

Software-Verification Competition

- Yearly, since 2012
- Part of ETAPS
- Organized by Dirk Beyer (Munich) + Jan Strejček (since 2025)
- 62 participating tools in 2025
- Initially for model checkers
  Abstract interpreters dabbling since 2021 (Goblint)

Presentation of SV-Comp

Software-Verification Competition

- Yearly, since 2012
- Part of ETAPS
- Organized by Dirk Beyer (Munich) + Jan Strejček (since 2025)
- 62 participating tools in 2025
- Initially for model checkers
  Abstract interpreters dabbling since 2021 (Goblint)
- Rewards “incremental improvements” (software dev./maintenance)

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

- Preprocessed C programs

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

- Preprocessed C programs
- Lots of handcrafted or small examples

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

- Preprocessed C programs
- Lots of handcrafted or small examples
- “SoftwareSystems” category, more realistic

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

- Preprocessed C programs
- Lots of handcrafted or small examples
- “SoftwareSystems” category, more realistic
- Community-curated, including oracle verdicts

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

- Preprocessed C programs
- Lots of handcrafted or small examples
- “SoftwareSystems” category, more realistic
- Community-curated, including oracle verdicts

Properties

Presentation of SV-Comp (II)

"Tasks"

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

- Preprocessed C programs
- Lots of handcrafted or small examples
- "SoftwareSystems" category, more realistic
- Community-curated, including oracle verdicts

Properties

- Reachability

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

- Preprocessed C programs
- Lots of handcrafted or small examples
- “SoftwareSystems” category, more realistic
- Community-curated, including oracle verdicts

Properties

- Reachability
- Memory safety

Presentation of SV-Comp (II)

"Tasks"

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

- Preprocessed C programs
- Lots of handcrafted or small examples
- "SoftwareSystems" category, more realistic
- Community-curated, including oracle verdicts

Properties

- Reachability
- Memory safety
- Integer overflows

Presentation of SV-Comp (II)

"Tasks"

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

- Preprocessed C programs
- Lots of handcrafted or small examples
- "SoftwareSystems" category, more realistic
- Community-curated, including oracle verdicts

Properties

- Reachability
- Memory safety
- Integer overflows
- Termination

Presentation of SV-Comp (II)

“Tasks”

- **Input** check if a given program satisfies a property
- **Constraints** 15 minutes CPU time, 8GB RAM
- **Output** result (true, false or unknown) & witness
- **Scoring** discussed later

Programs

- Preprocessed C programs
- Lots of handcrafted or small examples
- “SoftwareSystems” category, more realistic
- Community-curated, including oracle verdicts

Properties

- Reachability
- Memory safety
- Integer overflows
- Termination
- Data race

Presentation of SV-Comp (III)

Category	# tasks	Median loc.
ReachSafety	6282	1267
MemSafety	6280	86
ConcurrencySafety	2370	127
NoOverflows	6539	49
Termination	3324	901
SoftwareSystems	5825	6655

Subcategories in SoftwareSystems

Presentation of SV-Comp (III)

Category	# tasks	Median loc.
ReachSafety	6282	1267
MemSafety	6280	86
ConcurrencySafety	2370	127
NoOverflows	6539	49
Termination	3324	901
SoftwareSystems	5825	6655

Subcategories in SoftwareSystems

- AWS C commons

Presentation of SV-Comp (III)

Category	# tasks	Median loc.
ReachSafety	6282	1267
MemSafety	6280	86
ConcurrencySafety	2370	127
NoOverflows	6539	49
Termination	3324	901
SoftwareSystems	5825	6655

Subcategories in SoftwareSystems

- AWS C commons
- BusyBox (coreutils alternative)

Presentation of SV-Comp (III)

Category	# tasks	Median loc.
ReachSafety	6282	1267
MemSafety	6280	86
ConcurrencySafety	2370	127
NoOverflows	6539	49
Termination	3324	901
SoftwareSystems	5825	6655

Subcategories in SoftwareSystems

- AWS C commons
- BusyBox (coreutils alternative)
- Coreutils

Presentation of SV-Comp (III)

Category	# tasks	Median loc.
ReachSafety	6282	1267
MemSafety	6280	86
ConcurrencySafety	2370	127
NoOverflows	6539	49
Termination	3324	901
SoftwareSystems	5825	6655

Subcategories in SoftwareSystems

- AWS C commons
- BusyBox (coreutils alternative)
- Coreutils
- Linux Device Drivers

Presentation of SV-Comp (III)

Category	# tasks	Median loc.
ReachSafety	6282	1267
MemSafety	6280	86
ConcurrencySafety	2370	127
NoOverflows	6539	49
Termination	3324	901
SoftwareSystems	5825	6655

Subcategories in SoftwareSystems

- AWS C commons
- BusyBox (coreutils alternative)
- Coreutils
- Linux Device Drivers
- OpenBSD

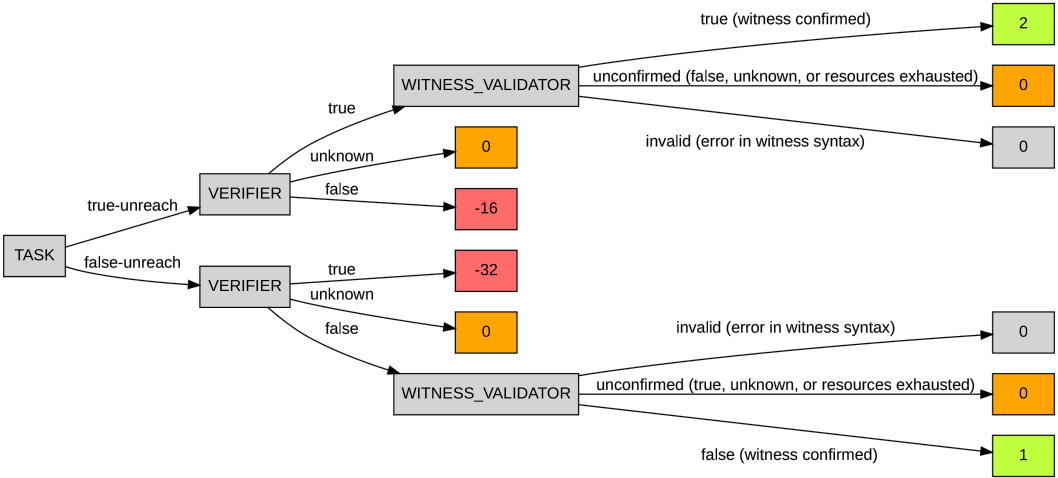
Presentation of SV-Comp (III)

Category	# tasks	Median loc.
ReachSafety	6282	1267
MemSafety	6280	86
ConcurrencySafety	2370	127
NoOverflows	6539	49
Termination	3324	901
SoftwareSystems	5825	6655

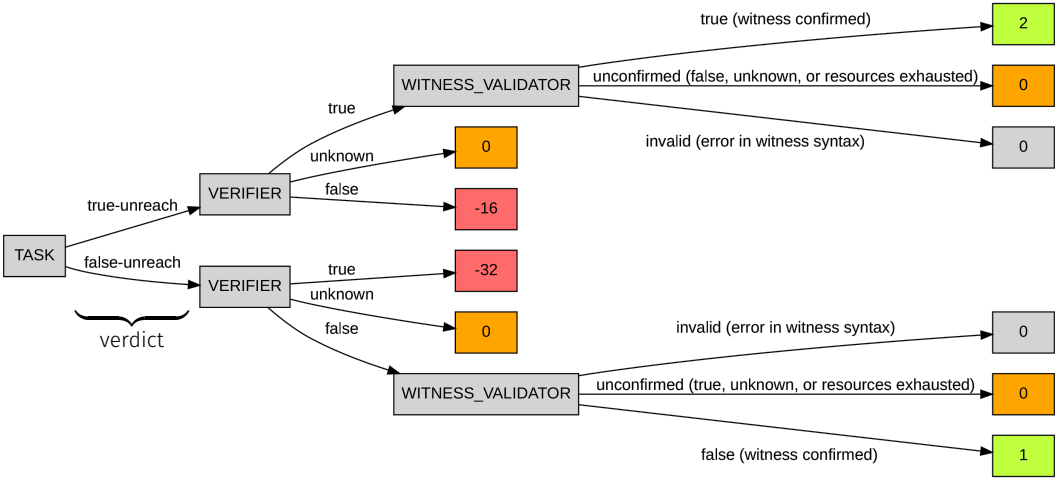
Subcategories in SoftwareSystems

- AWS C commons
- BusyBox (coreutils alternative)
- Coreutils
- Linux Device Drivers
- OpenBSD
- uthash

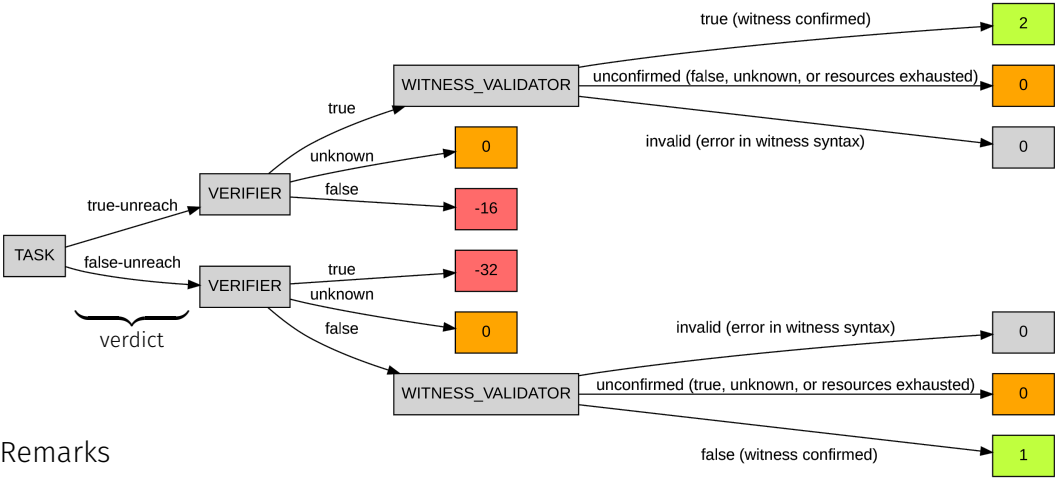
SV-Comp's Scoring System



SV-Comp's Scoring System



SV-Comp's Scoring System



Remarks

- community-based curation of verdicts
- 187 manual fixes in 2023, 20 fixes in 2024...

SV-Comp's Scoring System (II)

Categories are divided into subcategories (a family of benchmarks).

SV-Comp's Scoring System (II)

Categories are divided into subcategories (a family of benchmarks).

Scoring incentive for balanced results among subcategories.

$$\text{overall score} \propto \sum_{s \in \text{subCategory}} \frac{\text{raw score in } s}{\#\text{ tasks in } s}$$

SV-Comp's Scoring System (II)

Categories are divided into subcategories (a family of benchmarks).

Scoring incentive for balanced results among subcategories.

$$\text{overall score} \propto \sum_{s \in \text{subCategory}} \frac{\text{raw score in } s}{\#\text{ tasks in } s}$$

You may have a high raw score but not so good overall score.

SV-Comp's "Witnesses"

Motivation

- Ensure that results can be validated, at a reduced computational cost

SV-Comp's "Witnesses"

Motivation

- Ensure that results can be validated, at a reduced computational cost
- Improve interoperability between verifiers?

SV-Comp's "Witnesses"

Motivation

- Ensure that results can be validated, at a reduced computational cost
- Improve interoperability between verifiers?

Witnesses

Programs annotated with loop invariants

SV-Comp's "Witnesses"

Motivation

- Ensure that results can be validated, at a reduced computational cost
- Improve interoperability between verifiers?

Witnesses

Programs annotated with loop invariants

Issues (in my opinion)

SV-Comp's "Witnesses"

Motivation

- Ensure that results can be validated, at a reduced computational cost
- Improve interoperability between verifiers?

Witnesses

Programs annotated with loop invariants

Issues (in my opinion)

- Inlining-based analysis vs context-free program annotations [Saa20]

SV-Comp's "Witnesses"

Motivation

- Ensure that results can be validated, at a reduced computational cost
- Improve interoperability between verifiers?

Witnesses

Programs annotated with loop invariants

Issues (in my opinion)

- Inlining-based analysis vs context-free program annotations [Saa20]
- Cross-validator scores can be low [Bey+22] – 45%

SV-Comp's "Witnesses"

Motivation

- Ensure that results can be validated, at a reduced computational cost
- Improve interoperability between verifiers?

Witnesses

Programs annotated with loop invariants

Issues (in my opinion)

- Inlining-based analysis vs context-free program annotations [Saa20]
- Cross-validator scores can be low [Bey+22] – 45%
- Until 2025, *time*(program verification) = *time*(witness validation)

Competitive Mopsa?

Adapting Mopsa to SV-Comp's Framework

Differing workflows

Given a configuration and a program, Mopsa computes potential run-time errors.

SV-Comp: check whether a program statisfies one property.

Adapting Mopsa to SV-Comp's Framework

Differing workflows

Given a configuration and a program, Mopsa computes potential run-time errors.

SV-Comp: check whether a program statisfies one property.

A "sequential portfolio" approach

1. Analyze the target program with Mopsa

Adapting Mopsa to SV-Comp's Framework

Differing workflows

Given a configuration and a program, Mopsa computes potential run-time errors.

SV-Comp: check whether a program statisfies one property.

A "sequential portfolio" approach

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds

Adapting Mopsa to SV-Comp's Framework

Differing workflows

Given a configuration and a program, Mopsa computes potential run-time errors.

SV-Comp: check whether a program statisfies one property.

A "sequential portfolio" approach

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds
   - **Yes?** finished, program is safe

Adapting Mopsa to SV-Comp's Framework

Differing workflows

Given a configuration and a program, Mopsa computes potential run-time errors.

SV-Comp: check whether a program statisfies one property.

A "sequential portfolio" approach

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds
   - **Yes?** finished, program is safe
   - **No?** restart with a more precise analysis configuration

Adapting Mopsa to SV-Comp's Framework

Differing workflows

Given a configuration and a program, Mopsa computes potential run-time errors.

SV-Comp: check whether a program statisfies one property.

A "sequential portfolio" approach

1. Analyze the target program with Mopsa
2. Postprocess Mopsa's result to decide whether the property of interest holds
   - **Yes?** finished, program is safe
   - **No?** restart with a more precise analysis configuration

⇝ Mopsa returns unknown or times out when a property is not verified.

Our 2025 sequential portfolio

Roughly:

1. Interval analysis without loop unrolling

Our 2025 sequential portfolio

Roughly:

1. Interval analysis without loop unrolling
2. Add string-length domain, loop unrolling of 2, various unrolling heuristics

Our 2025 sequential portfolio

Roughly:

1. Interval analysis without loop unrolling
2. Add string-length domain, loop unrolling of 2, various unrolling heuristics
3. Add relational analysis (with packing), loop unrolling of 15

Our 2025 sequential portfolio

Roughly:

1. Interval analysis without loop unrolling
2. Add string-length domain, loop unrolling of 2, various unrolling heuristics
3. Add relational analysis (with packing), loop unrolling of 15
4. Add trace partitioning, loop unrolling of 60

Our 2025 sequential portfolio

Roughly:

1. Interval analysis without loop unrolling
2. Add string-length domain, loop unrolling of 2, various unrolling heuristics
3. Add relational analysis (with packing), loop unrolling of 15
4. Add trace partitioning, loop unrolling of 60
5. Fully relational analysis (no packing),
   PPLite Polyhedra instead of Apron's default.

Our 2025 sequential portfolio

Roughly:

1. Interval analysis without loop unrolling
2. Add string-length domain, loop unrolling of 2, various unrolling heuristics
3. Add relational analysis (with packing), loop unrolling of 15
4. Add trace partitioning, loop unrolling of 60
5. Fully relational analysis (no packing),
   PPLite Polyhedra instead of Apron's default.

+ Some task-specific heuristic autosuggestions.

Global results per configuration

Max. Conf.	Tasks proved correct	Tasks reaching 900s timeout
1	7002	389

Global results per configuration

Max. Conf.	Tasks proved correct		Tasks reaching 900s timeout	
1	7002		389	
2	7743	(+741)	970	(+581)

Global results per configuration

Max. Conf.	Tasks proved correct		Tasks reaching 900s timeout	
1	7002		389	
2	7743	(+741)	970	(+581)
3	8489	(+746)	3377	(+2407)

Global results per configuration

Max. Conf.	Tasks proved correct		Tasks reaching 900s timeout	
1	7002		389	
2	7743	(+741)	970	(+581)
3	8489	(+746)	3377	(+2407)
4	8660	(+171)	5378	(+2001)

Global results per configuration

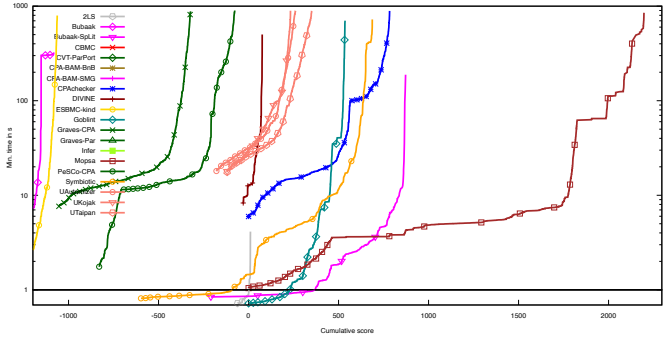
Max. Conf.	Tasks proved correct		Tasks reaching 900s timeout	
1	7002		389	
2	7743	(+741)	970	(+581)
3	8489	(+746)	3377	(+2407)
4	8660	(+171)	5378	(+2001)
5	8933	(+273)	8440	(+3062)

Results

- 2023: 3rd in *SoftwareSystems*.
  Category winner has been participating for 10 years!

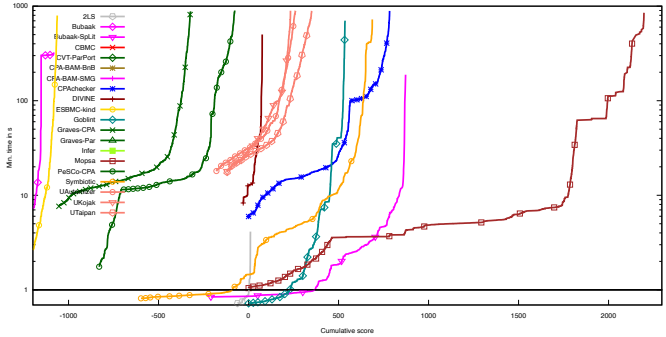
Results

- 2023: 3rd in *SoftwareSystems*.
  Category winner has been participating for 10 years!
- 2024: 1st in *SoftwareSystems*.



Results

- 2023: 3rd in *SoftwareSystems*.
  Category winner has been participating for 10 years!
- 2024: 1st in *SoftwareSystems*.



- 2025: 2nd in *SoftwareSystems*.
  Trying to improve ranking in *NoOverflows* category (no luck!).

But, why?

- Community

But, why?

- Community
- Gamified side

But, why?

- Community
- Gamified side
- Allocate time to improve Mopsa's precision

But, why?

- Community
- Gamified side
- Allocate time to improve Mopsa's precision
- "What do you win?"

But, why?

- Community
- Gamified side
- Allocate time to improve Mopsa's precision
- "What do you win?"
  - A piece of wood!

But, why?

- Community
- Gamified side
- Allocate time to improve Mopsa's precision
- "What do you win?"
  - A piece of wood!
  - Some visibility?

But, why?

- Community
- Gamified side
- Allocate time to improve Mopsa's precision
- "What do you win?"
  - A piece of wood!
  - Some visibility?
  - Mopsa supports de facto benchmarks

Improvements made in 2025

- Heuristic Autosuggestions

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations
  - Single loop in program unrolling

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations
  - Single loop in program unrolling
- Trace partitioning

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations
  - Single loop in program unrolling
- Trace partitioning
- Widening with thresholds

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations
  - Single loop in program unrolling
- Trace partitioning
- Widening with thresholds
- Memory deallocation

Improvements made in 2025

- Heuristic Autosuggestions
  - Bounded recursion unrolling
  - Loop unrolling for precise allocations
  - Single loop in program unrolling
- Trace partitioning
- Widening with thresholds
- Memory deallocation
- Sound bitfield support

Our 2025 improvements

Category	Prop.	|tasks|	Mopsa'24	Mopsa'25	Best score, verifier (2025)	
Hardness	R	4012	432	518	7426	SVF-SVC [CMY25]
Heap	R	240	190	226	314	PredatorHP [PSV20]
Loops	R	774	298	376	1031	AISE [WC24; YZJ25]
Recursive	R	160	12	60	150	UTaipan [Die+23]
Heap	M	247	40	154	331	PredatorHP [PSV20]
Juliet	M	3271	2224	2530	4709	CPAchecker [Bai+24]
LinkedLists	M	134	58	96	220	PredatorHP [PSV20]
Main	N	1989	1920	2138	2756	UAutomizer [Hei+23]
AWS	R	341	36	76	326	Bubaak [CH23; MC25]
DDL	R	2420	3476	3602	3602	Mopsa
uthash	M	192	96	108	246	Bubaak* [CH23; MC25; CR24]
uthash	N	162	204	300	300	Mopsa

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)
- Sound analysis $\rightsquigarrow$ contesting some verdicts

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)
- Sound analysis $\rightsquigarrow$ contesting some verdicts

Weaknesses

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)
- Sound analysis $\rightsquigarrow$ contesting some verdicts

Weaknesses

- Sequential portfolio could leverage incremental approaches

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)
- Sound analysis $\rightsquigarrow$ contesting some verdicts

Weaknesses

- Sequential portfolio could leverage incremental approaches
- Do we really need sequential portfolio?

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)
- Sound analysis $\rightsquigarrow$ contesting some verdicts

Weaknesses

- Sequential portfolio could leverage incremental approaches
- Do we really need sequential portfolio?
- Verifying a single property vs all RTEs

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)
- Sound analysis $\rightsquigarrow$ contesting some verdicts

Weaknesses

- Sequential portfolio could leverage incremental approaches
- Do we really need sequential portfolio?
- Verifying a single property vs all RTEs
- Unable to provide counterexamples yet

ongoing work by Marco [MM24a; MM24b]

Strengths & Weaknesses

Strengths

- Scalability, esp. *SoftwareSystems* category
- Progress on *NoOverflows* (Ultimate family difficult to beat!)
- Sound analysis $\rightsquigarrow$ contesting some verdicts

Weaknesses

- Sequential portfolio could leverage incremental approaches
- Do we really need sequential portfolio?
- Verifying a single property vs all RTEs
- Unable to provide counterexamples yet

  ongoing work by Marco [MM24a; MM24b]
- Fixed sequence of configurations

Conclusion

Conclusion

- Participation during Fall of 2023, 2024 and 2025.
- Mopsa is competitive!
- Raises interesting longer-term research questions (ANR RAISIN)
  Beyond Termination: Resource-Aware Static Analyses?
- ⚠ benchmark bias, scalability AnalyzeThat workshop?

References – I

[Bai+24] Daniel Baier et al. **“CPAchecker 2.3 with Strategy Selection - (Competition Contribution)”.** In: Lecture Notes in Computer Science. Springer, 2024, pp. 359–364.

[Bau+22] Guillaume Bau et al. **“Abstract interpretation of Michelson smart-contracts”.** In: ed. by Laure Gonnord and Laura Titolo. ACM, 2022, pp. 36–43. DOI: 10.1145/3520313.3534660.

[Bey+22] Dirk Beyer et al. **“Verification Witnesses”.** In: ACM Trans. Softw. Eng. Methodol. 4 (2022), 57:1–57:69.

References – II

[CH23] Marek Chalupa and Thomas A. Henzinger. “Bubaak: Runtime Monitoring of Program Verifiers - (Competition Contribution)”. In: Lecture Notes in Computer Science. Springer, 2023, pp. 535–540.

[CMY25] C. McGowan, M. Richards, and Y. Sui. “SVF-SVC: Software Verification Using SVF (Competition Contribution)”. In: LNCS. Springer, 2025.

[CR24] Marek Chalupa and Cedric Richter. “Bubaak-SpLit: Split what you cannot verify (Competition contribution)”. In: Lecture Notes in Computer Science. Springer, 2024, pp. 353–358.

[Die+23] Daniel Dietsch et al. “Ultimate Taipan and Race Detection in Ultimate - (Competition Contribution)”. In: Lecture Notes in Computer Science. Springer, 2023, pp. 582–587.

References – III

[DM19] David Delmas and Antoine Miné. **“Analysis of Software Patches Using Numerical Abstract Interpretation”.** In: ed. by Bor-Yuh Evan Chang. Lecture Notes in Computer Science. Springer, 2019, pp. 225–246. DOI: 10.1007/978-3-030-32304-2_12.

[DOM21] David Delmas, Abdelraouf Ouadjaout, and Antoine Miné. **“Static Analysis of Endian Portability by Abstract Interpretation”.** In: Lecture Notes in Computer Science. Springer, 2021, pp. 102–123.

[Hei+23] Matthias Heizmann et al. **“Ultimate Automizer and the CommuHash Normal Form - (Competition Contribution)”.** In: Lecture Notes in Computer Science. Springer, 2023, pp. 577–581.

References – IV

[JMO18] Matthieu Journault, Antoine Miné, and Abdelraouf Ouadjaout. **“Modular Static Analysis of String Manipulations in C Programs”.** In: ed. by Andreas Podelski. Lecture Notes in Computer Science. Springer, 2018, pp. 243–262. DOI: 10.1007/978-3-319-99725-4_16.

[Jou+19] M. Journault et al. **“Combinations of reusable abstract domains for a multilingual static analyzer”.** In: New York, USA, July 2019, pp. 1–17.

[MC25] M. Chalupa and C. Richter. **“Bubaak: Dynamic Cooperative Verification (Competition Contribution)”.** In: LNCS. Springer, 2025.

References – V

[MFM24] Raphaël Monat, Aymeric Fromherz, and Denis Merigoux. **“Formalizing Date Arithmetic and Statically Detecting Ambiguities for the Law”.** In: ed. by Stephanie Weirich. Lecture Notes in Computer Science. Springer, 2024, pp. 421–450. DOI: 10.1007/978-3-031-57267-8_16.

[MM24a] Marco Milanese and Antoine Miné. **“Generation of Violation Witnesses by Under-Approximating Abstract Interpretation”.** In: ed. by Rayna Dimitrova, Ori Lahav, and Sebastian Wolff. Lecture Notes in Computer Science. Springer, 2024, pp. 50–73. DOI: 10.1007/978-3-031-50524-9_3.

References – VI

[MM24b] Marco Milanese and Antoine Miné. **“Under-Approximating Memory Abstractions”.** In: ed. by Roberto Giacobazzi and Alessandra Gorla. Lecture Notes in Computer Science. Springer, 2024, pp. 300–326. DOI: 10.1007/978-3-031-74776-2_12.

[MOM20a] R. Monat, A. Ouadjaout, and A. Miné. **“Static Type Analysis by Abstract Interpretation of Python Programs”.** In: LIPIcs. 2020.

[MOM20b] R. Monat, A. Ouadjaout, and A. Miné. **“Value and allocation sensitivity in static Python analyses”.** In: ACM, 2020, pp. 8–13. DOI: 10.1145/3394451.3397205.

[MOM21] R. Monat, A. Ouadjaout, and A. Miné. **“A Multilanguage Static Analysis of Python Programs with Native C Extensions”.** In: 2021.

References – VII

[OM20] A. Ouadjaout and A. Miné. **“A Library Modeling Language for the Static Analysis of C Programs”.** In: ed. by David Pichardie and Mihaela Sighireanu. Lecture Notes in Computer Science. Springer, 2020, pp. 223–247. DOI: 10.1007/978-3-030-65474-0_11.

[PM24] Francesco Parolini and Antoine Miné. **“Sound Abstract Nonexploitability Analysis”.** In: Lecture Notes in Computer Science. Springer, 2024, pp. 314–337.

[PSV20] Petr Peringer, Veronika Soková, and Tomás Vojnar. **“PredatorHP Revamped (Not Only) for Interval-Sized Memory Regions and Memory Reallocation (Competition Contribution)”.** In: Lecture Notes in Computer Science. Springer, 2020, pp. 408–412.

References – VIII

[Saa20] Simmo Saan. Witness generation for data-flow analysis. 2020.

[VMM23] Milla Valnet, Raphaël Monat, and Antoine Miné. “Analyse statique de valeurs par interprétation abstraite de programmes fonctionnels manipulant des types algébriques récursifs”. In: ed. by Timothy Bourke and Delphine Demange. Praz-sur-Arly, France, Jan. 2023, pp. 211–242.

[VMM25] Milla Valnet, Raphaël Monat, and Antoine Miné. “Compositional Static Value Analysis for Higher-Order Numerical Programs”. In: ed. by Jonathan Aldrich and Alexandra Silva; Bergen, Norway: Dagstuhl Publishing, June 2025, p. 15. DOI: 10.4230/LIPIcs.ECOOP.2025.15.

References – IX

[WC24] Zhen Wang and Zhenbang Chen. **“AISE: A Symbolic Verifier by Synergizing Abstract Interpretation and Symbolic Execution (Competition Contribution)”.** In: Lecture Notes in Computer Science. Springer, 2024, pp. 347–352.

[YZJ25] Y. Lin, Z. Chen, and J. Wang. **“AISE v2.0: Combining Loop Transformations (Competition Contribution)”.** In: LNCS. Springer, 2025.