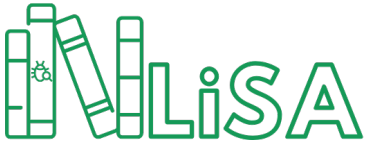


JLiSA: the Java Frontend of LiSA

Vincenzo Arceri[2], **Luca Negrini**[1], Giacomo Zanatta[1], Filippo Bianchi[2], Teodors Lisovenko[1], Luca Olivieri[1], Pietro Ferrara[1]

[1]Ca' Foscari University of Venice, Italy

[2]University of Parma, Italy

luca.negrini@unive.it

TACAS 2026, Turin, Italy

April 13th, 2026

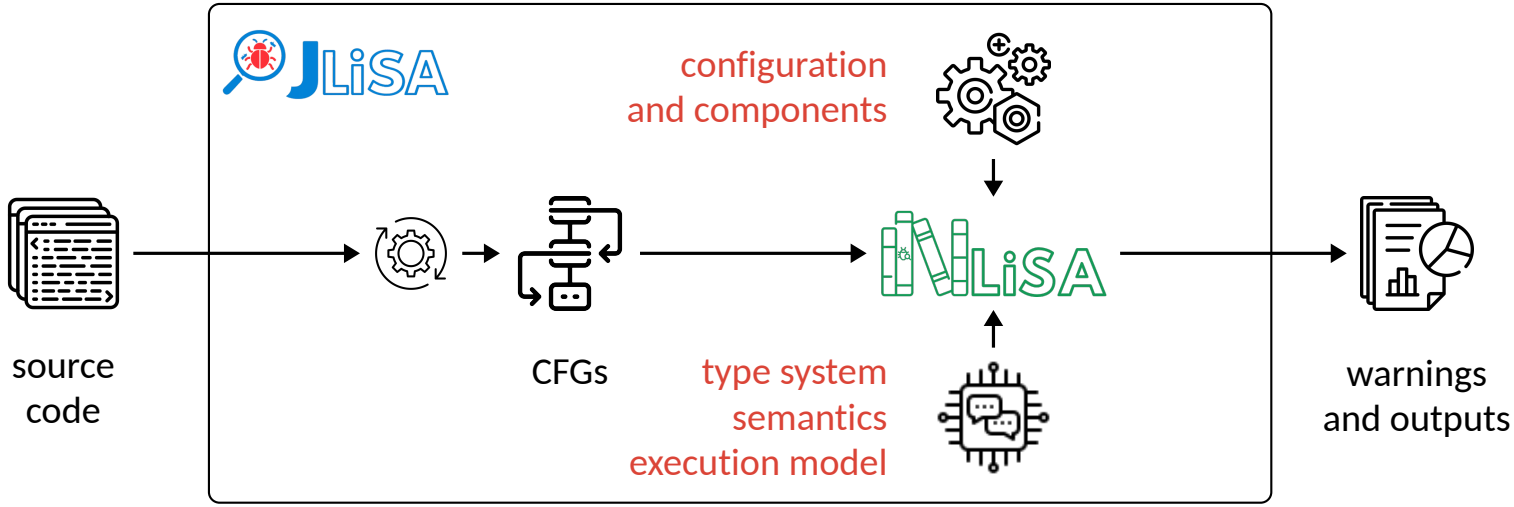
JLiSA: the Java Frontend for LiSA

Based on **LiSA** [NFAC23, Neg23], a modular library for building static analyzers based on the Abstract Interpretation theory [CC77, Cou21]

JLiSA: the Java Frontend for LiSA

Based on **LiSA** [NFAC23, Neg23], a modular library for building static analyzers based on the Abstract Interpretation theory [CC77, Cou21]

LiSA provides the infrastructure, **JLiSA** parses the code and selects the components to run



SV-COMP Setup and Results

For SV-COMP, **JLiSA** sets up **LiSA** to:

SV-COMP Setup and Results

For SV-COMP, **JLiSA** sets up **LiSA** to:

- perform a context-based [PS81] analysis based on inlining (max call stack: 150)

SV-COMP Setup and Results

For SV-COMP, **JLiSA** sets up **LiSA** to:

- perform a context-based [PS81] analysis based on inlining (max call stack: 150)
- abstract the memory with a field-sensitive Andersen-style [And94] analysis

SV-COMP Setup and Results

For SV-COMP, **JLiSA** sets up **LiSA** to:

- perform a context-based [PS81] analysis based on inlining (max call stack: 150)
- abstract the memory with a field-sensitive Andersen-style [And94] analysis
- abstract the values with a reduced product of intervals [CC77] and constant propagation

SV-COMP Setup and Results

For SV-COMP, **JLiSA** sets up **LiSA** to:

- perform a context-based [PS81] analysis based on inlining (max call stack: 150)
- abstract the memory with a field-sensitive Andersen-style [And94] analysis
- abstract the values with a reduced product of intervals [CC77] and constant propagation

▷ **Sound by design but has many false positives**

SV-COMP Setup and Results

For SV-COMP, **JLiSA** sets up **LiSA** to:

- perform a context-based [PS81] analysis based on inlining (max call stack: 150)
- abstract the memory with a field-sensitive Andersen-style [And94] analysis
- abstract the values with a reduced product of intervals [CC77] and constant propagation

▷ **Sound by design but has many false positives**

We use a reachability analysis: if an instruction is not always executed, don't issue warnings

SV-COMP Setup and Results

For SV-COMP, **JLiSA** sets up **LiSA** to:

- perform a context-based [PS81] analysis based on inlining (max call stack: 150)
- abstract the memory with a field-sensitive Andersen-style [And94] analysis
- abstract the values with a reduced product of intervals [CC77] and constant propagation

▷ **Sound by design but has many false positives**

We use a reachability analysis: if an instruction is not always executed, don't issue warnings

▷ **No false positives, 40 true positives**

SV-COMP Setup and Results

For SV-COMP, **JLiSA** sets up **LiSA** to:

- perform a context-based [PS81] analysis based on inlining (max call stack: 150)
- abstract the memory with a field-sensitive Andersen-style [And94] analysis
- abstract the values with a reduced product of intervals [CC77] and constant propagation

▷ **Sound by design but has many false positives**

We use a reachability analysis: if an instruction is not always executed, don't issue warnings

▷ **No false positives, 40 true positives**

Category	# Tasks	JLiSA	Position	Best Score	Best Tool
Valid Assert	986	273	5	753	SWAT [LMSE24]
				753	JAVA-RANGER [SHW+20]
No Runtime Exception	745	922	2 🏆	1130	JBMC [CKK+18]
Overall	1731	1311	3 🏆	1561	JBMC [CKK+18]

Conclusion

SV-COMP was challenging but fun!

Conclusion

SV-COMP was challenging but fun!

This year’s challenge was just participating

Conclusion

SV-COMP was challenging but fun!

This year's challenge was just participating

▷ In Java: only abstract interpreter, sound by design, managed to identify (few) correct violations

Conclusion

SV-COMP was challenging but fun!

This year's challenge was just participating

▷ In Java: only abstract interpreter, sound by design, managed to identify (few) correct violations

▷ **Goals for next year:**

Analysis improvements: symbolic reasoning, unbounded recursions support, smarter and more precise domains

Witness generation: beyond trivial violation witnesses, explore correctness witnesses

Thanks!

luca.negrini@unive.it

Bibliography I

[And94] Lars Ole Andersen. **Program analysis and specialization for the c programming language.** *PhD Thesis, University of Copenhagen*, 1994.

[CC77] Patrick Cousot and Radhia Cousot. **Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.** In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, page 238–252, New York, NY, USA, 1977. Association for Computing Machinery.

[CKK+18] Lucas Cordeiro, Pascal Kesseli, Daniel Kroening, Peter Schrammel, and Marek Trtik. **Jbmc: A bounded model checking tool for verifying java bytecode.** In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, pages 183–190, Cham, 2018. Springer International Publishing.

[Cou21] Patrick Cousot. ***Principles of Abstract Interpretation.*** MIT Press, 2021.

Bibliography II

[LMSE24] Nils Loose, Felix Mächtle, Florian Sieck, and Thomas Eisenbarth. **Swat: Modular dynamic symbolic execution for java applications using dynamic instrumentation (competition contribution).** In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 399–405, Cham, 2024. Springer Nature Switzerland.

[Neg23] Luca Negrini. ***A generic framework for multilanguage analysis.*** Doctoral Thesis, Università Ca' Foscari Venezia, January 2023.

[NFAC23] Luca Negrini, Pietro Ferrara, Vincenzo Arceri, and Agostino Cortesi. **LiSA: A Generic Framework for Multilanguage Static Analysis.** In Vincenzo Arceri, Agostino Cortesi, Pietro Ferrara, and Martina Olliaro, editors, *Challenges of Software Verification*, pages 19–42. Springer Nature, Singapore, 2023.

[PS81] M Pnueli and Micha Sharir. **Two approaches to interprocedural data flow analysis.** *Program flow analysis: theory and applications*, pages 189–234, 1981.

Bibliography III

[SHW+20] Vaibhav Sharma, Soha Hussein, Michael W. Whalen, Stephen McCamant, and Willem Visser. **Java ranger: statically summarizing regions for efficient symbolic execution of java.** In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 123–134, New York, NY, USA, 2020. Association for Computing Machinery.