vanHelsing: A Fast Proof Checker for Debuggable Compiler Verification

Roland Lezuo*, Ioan Dragan†, Gergö Barany‡, Andreas Krall*
rlezuo@complang.tuwien.ac.at

*Vienna University of Technology, Institute of Computer Languages

†Institute e-Austria, Timisoara, Romania

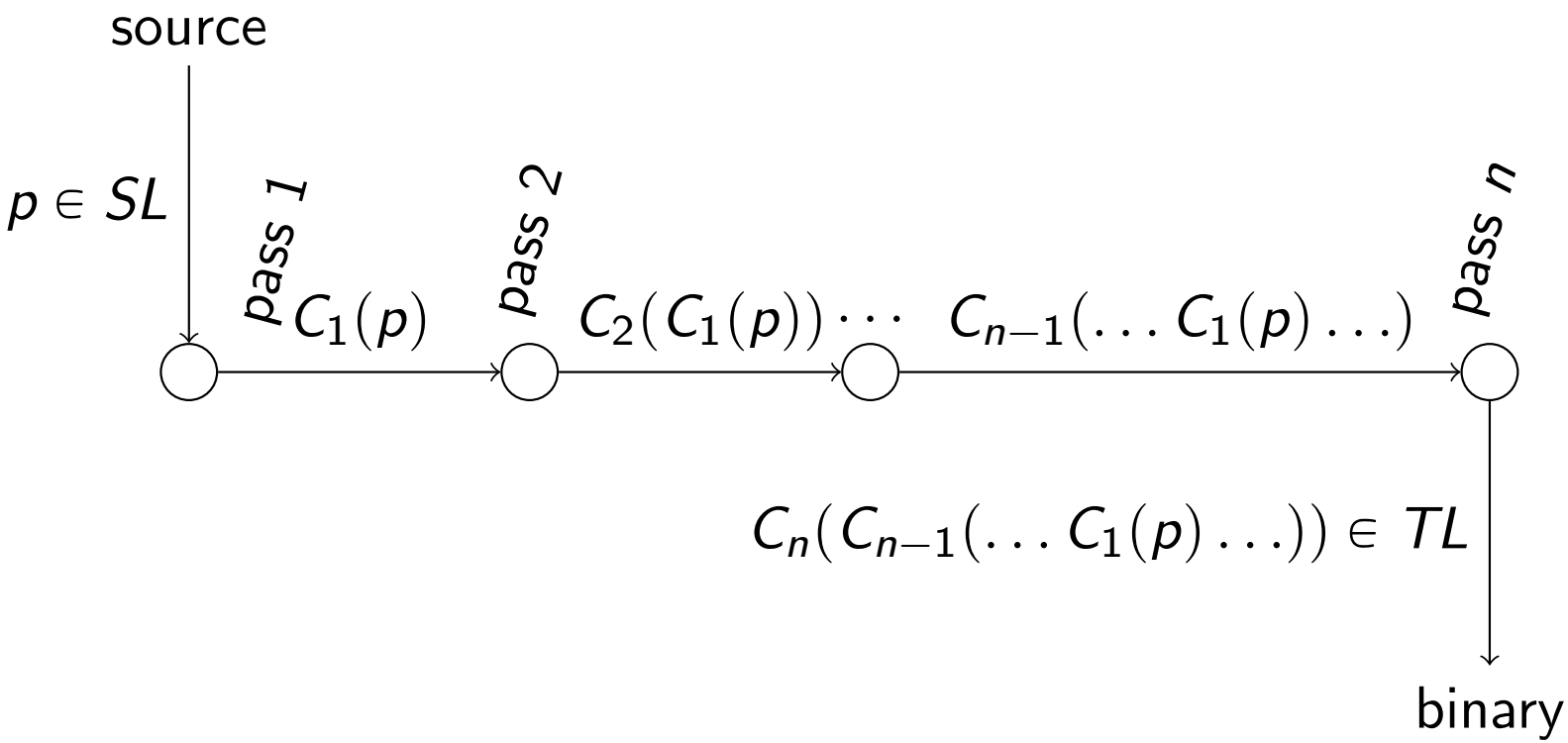
‡CEA LIST Software Reliability Laboratory, Gif-sur-Yvette, France

September 21, 2015

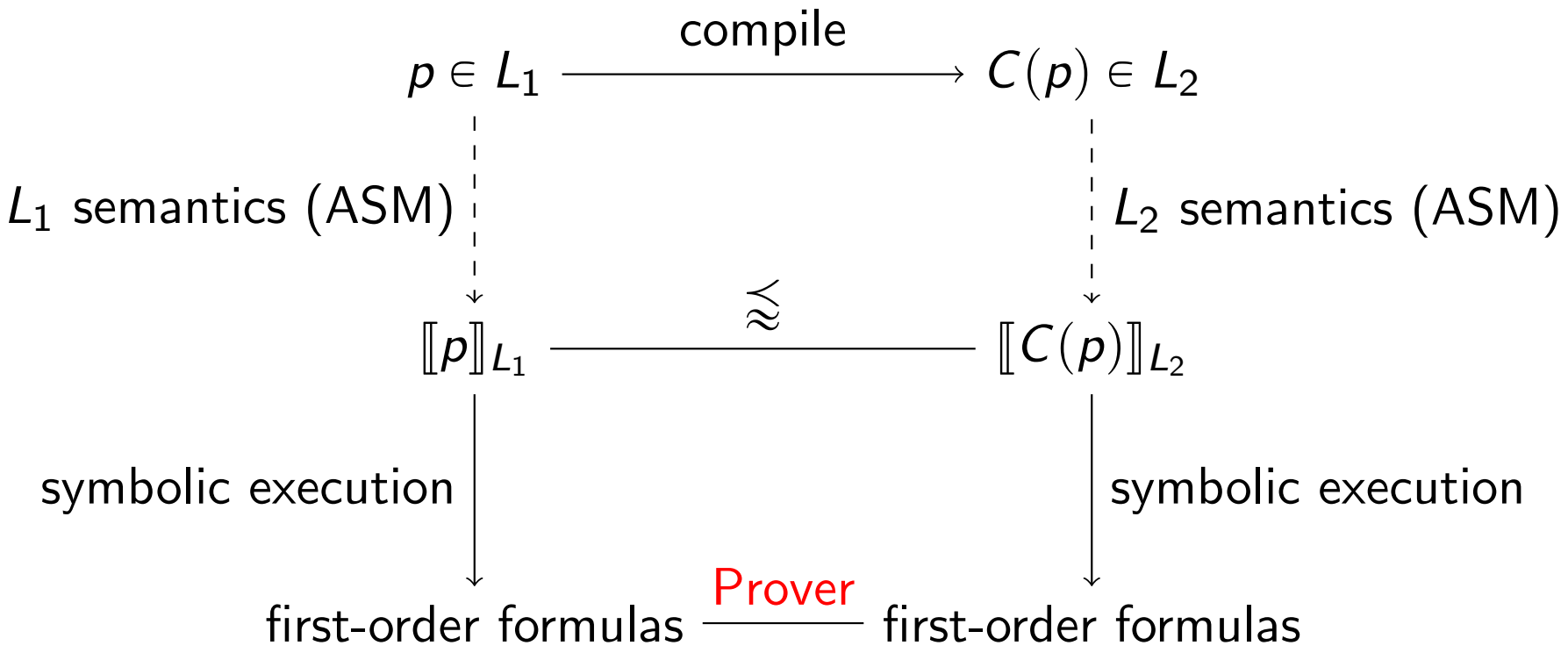
This work is supported in part by the Austrian Research Promotion Agency (FFG) and by Catena DSP GmbH

Overview

Used in Translation Validation Framework



Per Pass Simulation Proofs



Cf. *vanHelsing: A Fast Proof Checker for Debuggable Compiler Verification (SYNASC'15)* and *Scalable Translation Validation, Ph.D. thesis, Roland Lezuo 2014*

Requirements for Prover

- Evidence (traceable, constructive)
- Performance (prover dominates validation time)
- Debugging failed proofs (identify mis-compilation)
- Problem formulation (as-is, deal with it)

Comparison: Off-the-Shelf Provers

	Resolution based	SMT	vanHelsing
Evidence	$\surd^1$		$\surd$
Debugging		$\surd^1$	$\surd$
Performance	$\varnothing$	$-$	$+$

Debugging matters:

- Tool used by compiler experts
- Limited theorem proving knowledge
- Proofs are large, multiple thousands of formulas

[1] problem formulation dependent

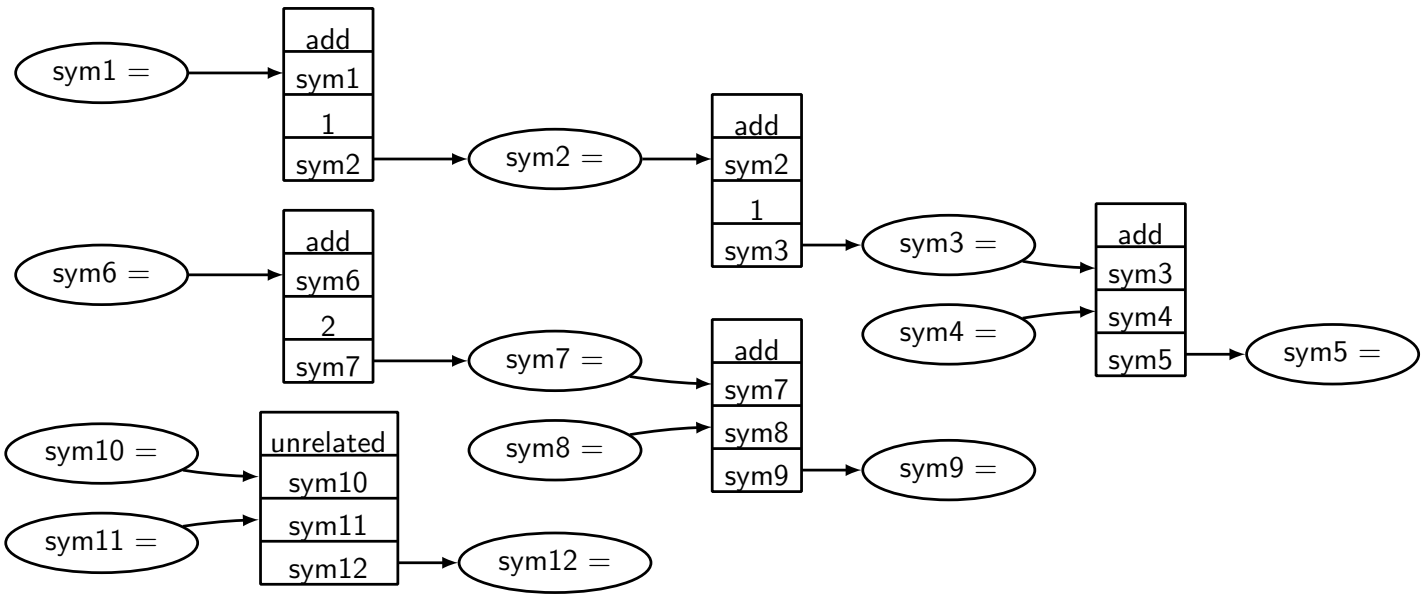
Special Proof Structure

- Data-Flow Equivalence Problem (DFE)
- expression equivalence problem
- Visualization as trees (GraphViz)

```
fof(id0,hypothesis,add(sym1,1,sym2)).
fof(id1,hypothesis,add(sym2,1,sym3)).
fof(id2,hypothesis,add(sym3,sym4,sym5)).

fof(id3,hypothesis,add(sym6,2,sym7)).
fof(id4,hypothesis,add(sym7,sym8,sym9)).

fof(id5,hypothesis,unrelated(sym10,
        sym11, sym12)).
```



Unification by a Forward Chaining Strategy

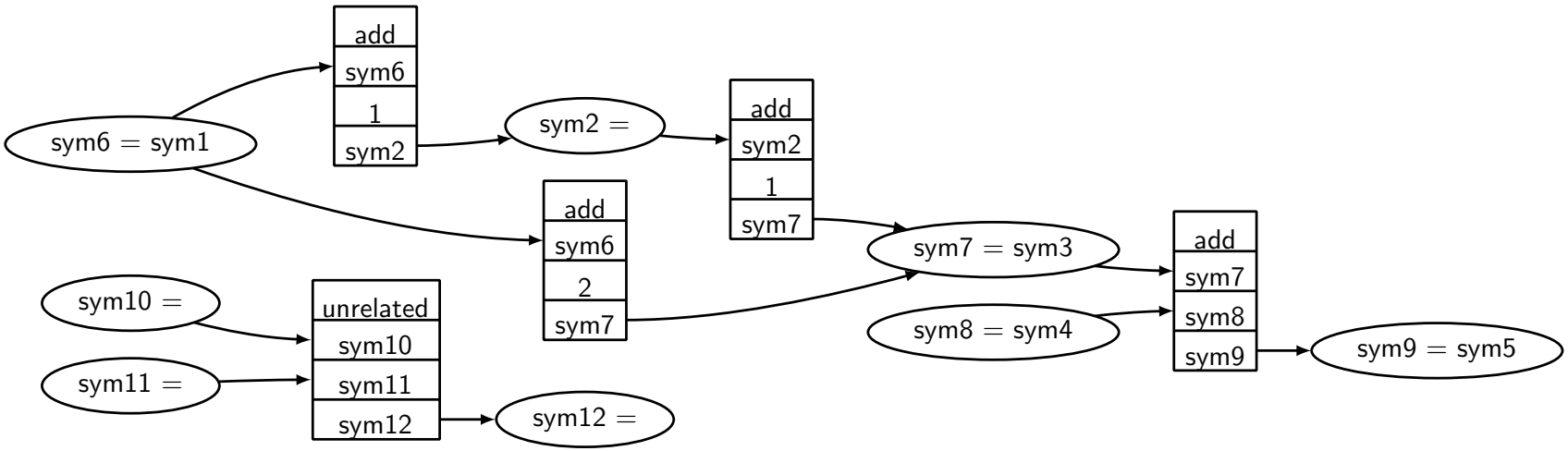
Witness Information

```
fof(op1,hypothesis, sym1=sym6).
fof(op2,hypothesis, sym4=sym8).

fof(cj1,conjecture, sym5=sym9).
```

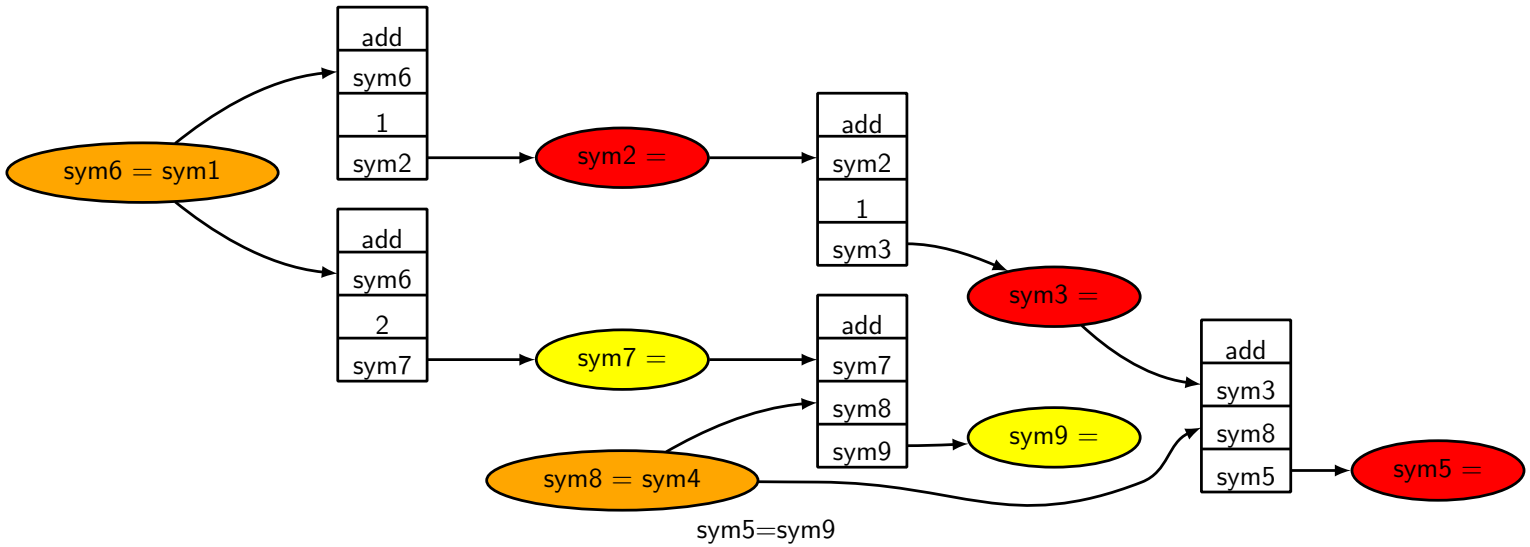
Axioms of Transformations

```
fof(ax1,axiom,(add(A,B,X) & add(A,B,Y))
        => X=Y).
fof(ax2,axiom,(add(A,1,B) & add(B,1,C)
        & add(A,2,D)) => C=D)
```



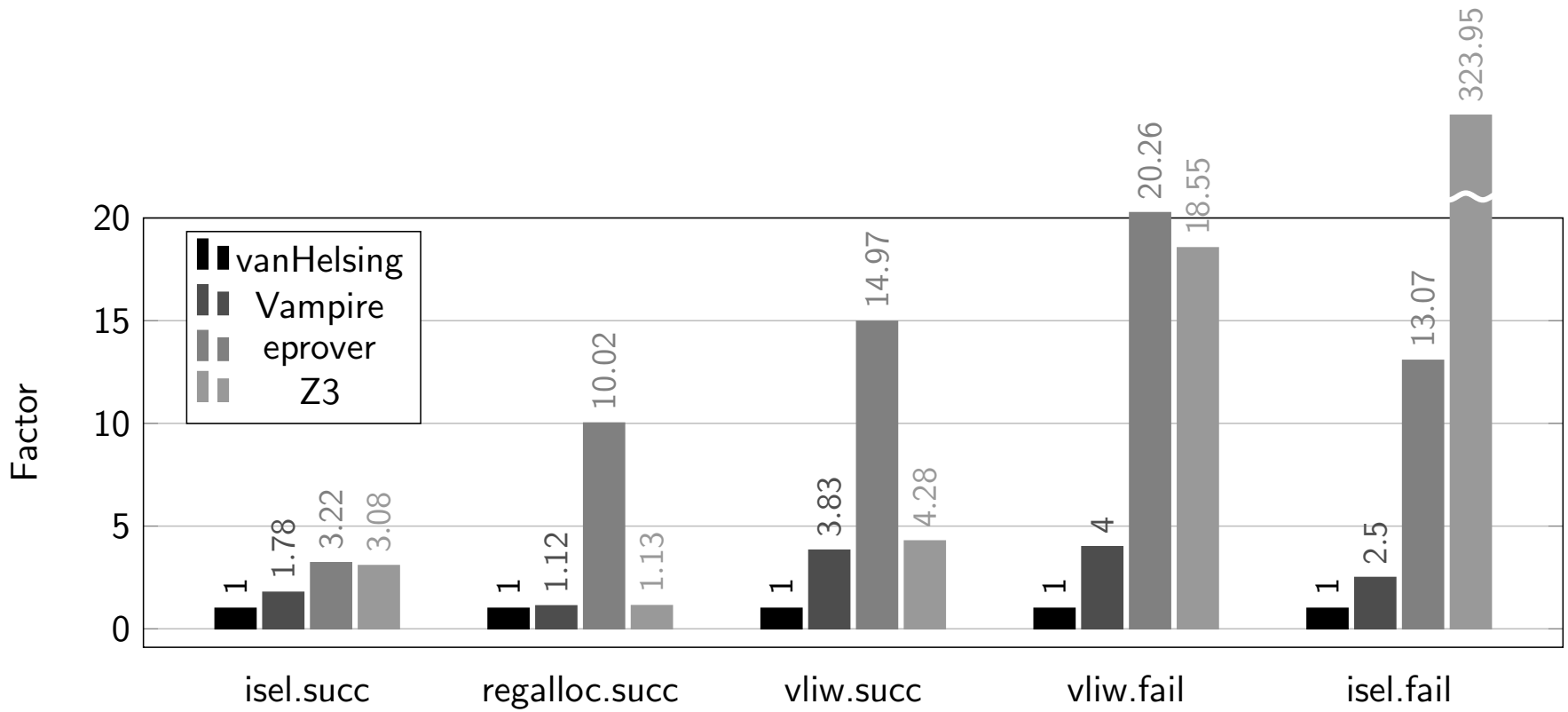
Debugging reduces Tree to relevant Subtrees

- start from pair of non-unified symbols of conjecture
- compute reachability (backwards)
- colorize nodes by membership of data-flows (i.e. left, right, both)



In practice reduces from thousands of formulas to a few dozen!

Performance



Conclusion

- fast (up to factor 3)
- small (4k LOC)
- visual debugging

Questions?