PrideMM: Second Order Model Checking for Memory Consistency Models

Simon Cooksey[1] Sarah Harris[1] Mark Batty[1]
Radu Grigore[1] **Mikoláš Janota**[2]

[1] University of Kent, Canterbury
[2] IST/INESC-ID, University of Lisbon, Portugal

What?

initially `x = 0, y = 0`	
`y = 1`	`x = 1`
`r1 = x`	`r2 = y`
`r1 == 0, r2 == 0` allowed?	

- Study and reason about memory models
- Interleaving semantics does not correspond to the real-world (e.g. optimizer may swap operations)

Memory Model Types

- axiomatic $\longrightarrow$ SAT
- Jeffrey-Riely $\longrightarrow$ QBF
  - toy: does not account for many real-world concerns
  - nice: explains some tricky aspects of the Java behaviour, in a concise&declarative way
- . . . and many others, not(?) expressible in QBF

Axiomatic Models

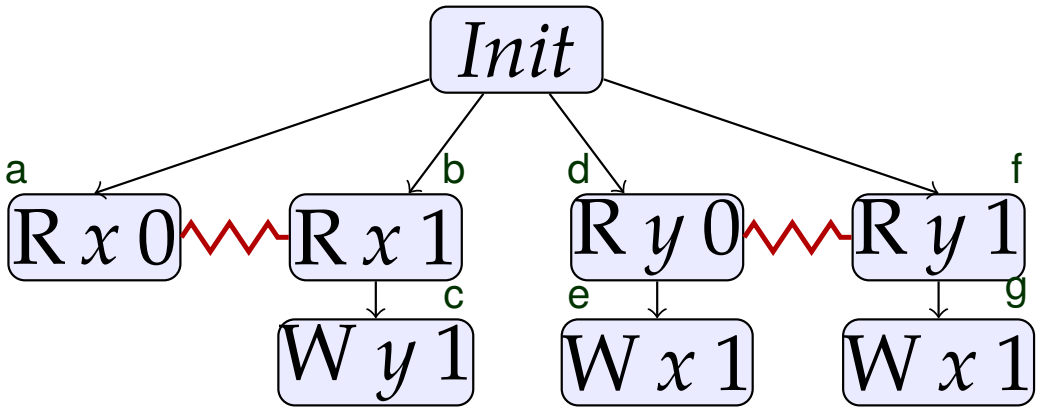
initially `x = 0, y = 0`

`y = 1`	`x = 1`
`r1 = x (value 0)`	`r2 = y (value 0)`

`r1 == 0, r2 == 0` allowed?

- Pick events (instantiate values for memory read/writes).
- Find relations between events such that certain constraints are satisfied (e.g. acyclicity).

Event Structures



Jeffrey–Riely

Axiomatic

Pick potential execution. Check validity by looking inside a single execution.

JR

Pick potential execution. Check validity by looking inside but also at other potential executions.

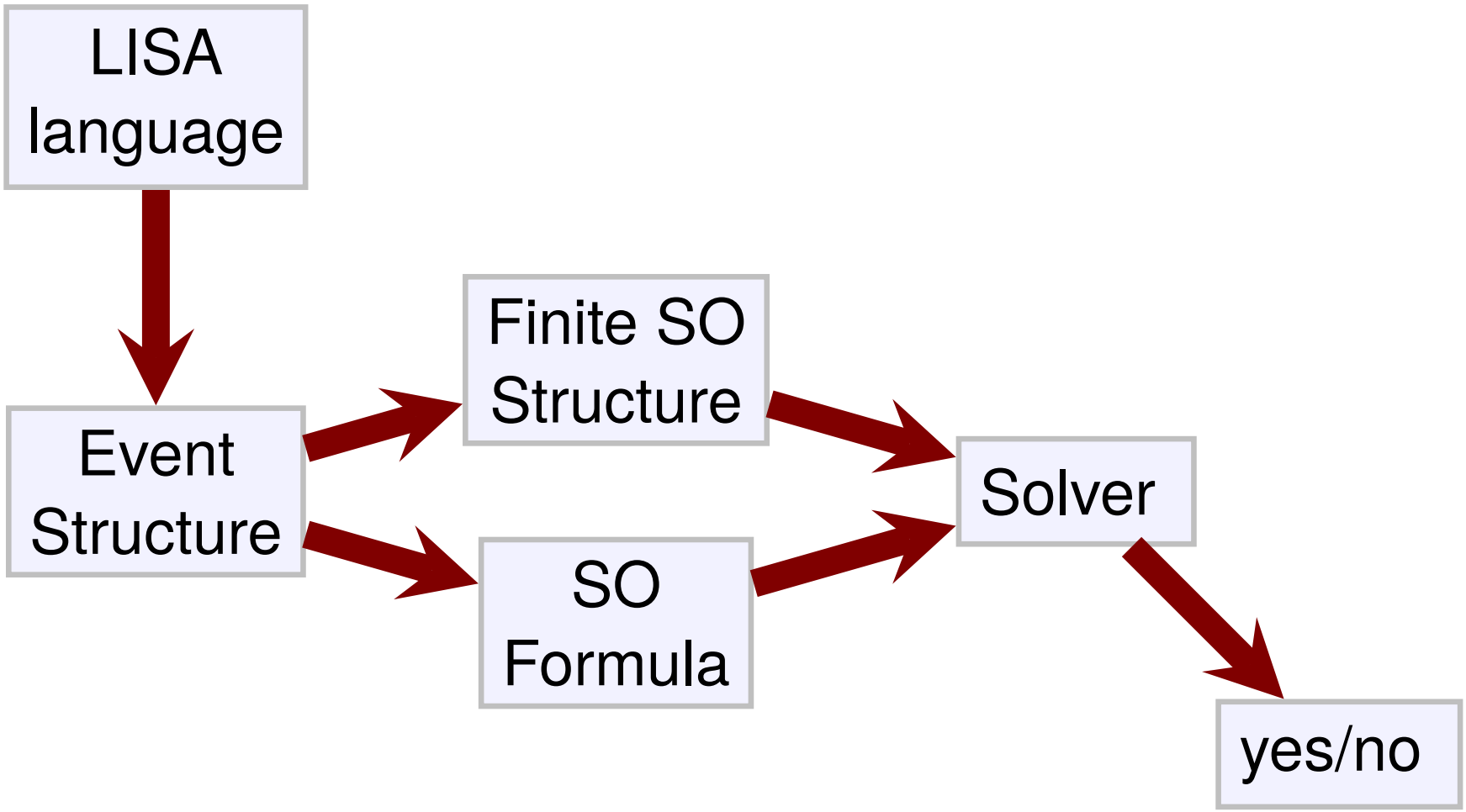
Idea: confluence – an execution is allowed if picking some alternative would essentially do the same anyway.

PrideMM: SO model-checking

PrideMM: Expressing memory models in Second-Order Logic represents a sweet spot:

1. memory models express naturally in SO
2. more expressive than SAT
3. *model checking* in SO is decidable

PrideMM Architecture



PrideMM: Input & Output

Structure

Universe size and relation/predicate interpretations

```
{ .size: 1..3
    .relations:
       a:1 := { (3) (4) }
       r:2 := { (1 3) (2 4) } }
```

PrideMM: Input & Output

Quantification over relations/predicates and FO variables, references to interpreted symbols

Formula

```
(?S:1 .
  (!x .
     ~S(x) |
     (?y . a(y) & r(y, x))))
```

Memory Models in PrideMM

Jeffrey-Riely: game like semantics requires $\exists\forall\exists$ formulas
Axiomatic: $\exists$ formulas, Sequential Consistency, Release-Acquire, C++ (follows Herd7)

Remark:

Some formulations require care, e.g. transitive closure on relations, acyclicity of relations

Solving

- Method 1: convert to quantfied Boolean formulas (QBF)
  1. grounding of FO variables
  2. for grounded predicate atom introduce Boolean variable
- Method 2: dedicated solver QFM enabling lazy grounding (CEGAR)
  1. small benefit for small arities
  2. overall works better due to native handling of predicates

Results

Java Causality Test Cases

Prob.	SAT	caqe (s)	qfun (s)	qfm (s)
1	N	⊥	610	**2**
2	N	⊥	23	**2**
3	Y	⊥	⊥	**222**
4	Y	⊥	**2**	5
5	Y	⊥	78	**51**
6	N	5	4	**1**
7	Y	⊥	280	**56**
8	N	⊥	**2**	**2**
9	N	⊥	2	**1**
10	Y	⊥	36	**10**
11	Y	⊥	598	**335**
13	Y	**1**	**1**	**1**
14	Y	⊥	**29**	33
15	Y	⊥	512	**157**
16	N	⊥	⊥	**12**
17	N	⊥	**39**	311
18	N	⊥	359	**190**
#17		#2	#15	#17

Conclusion

- PrideMM memory models via SO modelchecking
- automatic, expressive
- Various options in the backend solver, see also [Janota and Suda, LPAR 18]