Resource-Parameterized Program Analysis using Observation Sequences

Peizun Liu

Northeastern University

Static analysis and concurrency

Complicated shared-state interaction

Static analysis and concurrency

Complicated shared-state interaction

⬇

Subtle bugs, easily missed by humans

Static analysis and concurrency

Complicated thread-local behavior

Static analysis and concurrency

Complicated thread-local behavior

⬇

Undecidable in combination with shared state

Static analysis and concurrency

Complicated thread-local behavior

⬇

Undecidable in combination with shared state

Can decide under a fixed resource bound:

"No violations within 8 context switches"

Static analysis and concurrency

Complicated thread-local behavior

⬇

Undecidable in combination with shared state

Can decide under a fixed resource bound:

"No violations within 8 context switches"

But what if violation would appear after 9?

New proof technique

Checking safety in concurrent systems

New proof technique

Checking safety in concurrent systems

Prove, not just refute

New proof technique

Checking safety in concurrent systems

Prove, not just refute

Practical automation despite undecidability

Contents

- Past work: Context-bounded analysis

Contents

- Past work: Context-bounded analysis
- New proof technique: Context-unbounded analysis

Contents

- Past work: Context-bounded analysis
- New proof technique: Context-unbounded analysis
- The CUBA Algorithm

Contents

Contents

Pushdown system (PDS)

Machine: (Q, Σ, Δ, q^I)

Pushdown system (PDS)

Machine: (Q, Σ, Δ, q^I)

Q	States (shared)
Σ	Stack alphabet
Δ	Program: relates $(Q \times \Sigma^n)$ and $(Q \times \Sigma^{n\pm1})$
$q^I \in Q$	Starting state

Pushdown system (PDS)

Machine: (Q, Σ, Δ, q^I)

Q	States (shared)
Σ	Stack alphabet
Δ	Program: relates $(Q \times \Sigma^n)$ and $(Q \times \Sigma^{n\pm1})$
$q^I \in Q$	Starting state

Action may preserve stack OR pop one item OR push one item

Pushdown system (PDS)

Machine: (Q, Σ, Δ, q^I)

Q	States (shared)
Σ	Stack alphabet
Δ	Program: relates $(Q \times \Sigma^n)$ and $(Q \times \Sigma^{n\pm 1})$
$q^I \in Q$	Starting state

Action may preserve stack OR pop one item OR push one item

Σ^n models control stack contents

Pushdown system (PDS)

Machine: (Q, Σ, Δ, q^I)

Q	States (shared)
Σ	Stack alphabet
Δ	Program: relates $(Q \times \Sigma^n)$ and $(Q \times \Sigma^{n\pm1})$
$q^I \in Q$	Starting state

Action may preserve stack OR pop one item OR push one item

Σ^n models control stack contents

Nondeterminism models acting on input

Concurrent pushdown system (CPDS)

Machine: Fixed collection of PDSes

$$\mathcal{P}_i = (Q, \Sigma_i, \Delta_i, q^I)$$

Concurrent pushdown system (CPDS)

Machine: Fixed collection of PDSes

$$\mathcal{P}_i = (Q, \Sigma_i, \Delta_i, q^I)$$

Shared: state set, current state

Per-PDS: stack alphabet, program

Decidability

✓ Control stack only

Decidability

- ✓ Control stack only
- ✓ Shared state only

Decidability

✓ Control stack only

✓ Shared state only

✗ Both at once

Context-bounded anaysis (CBA)

Finite bound on context switches

Context-bounded anaysis (CBA)

Finite bound on context switches

Can we violate safety property with only 5 context switches?

Context-bounded anaysis (CBA)

Finite bound on context switches

Can we violate safety property with only 5 context switches?

$\Downarrow$

Finitely many single-PDS reachability questions

Context-bounded anaysis (CBA)

Finite bound on context switches

Can we violate safety property with only 5 context switches?

⬇

Finitely many single-PDS reachability questions

Can only refute safety properties, not prove them

Contents

- Past work: Context-bounded analysis
- New proof technique: Context-unbounded analysis
- The CUBA Algorithm
- Evaluation

Context-unbounded analysis (CUBA)

Proof technique built on CBA

Context-unbounded analysis (CUBA)

Proof technique built on CBA

CBA: "No safety violation within *k* context switches"

Context-unbounded analysis (CUBA)

Proof technique built on CBA

CBA: "No safety violation within *k* context switches"

CUBA: "No *k* is big enough for CBA to observe safety violation"

Observation Sequence

Definition parameterized over

$\mathcal{P}$	a CPDS (program we're investigating)
$\mathcal{D}$	a poset (space of possible observations)
$\mathcal{C}$	a property of resource-bounded CPDSes (what we're trying to check)

Observation Sequence

An observation sequence for property $\mathcal{C}$ on machine $\mathcal{P}$ over a poset $\mathcal{D}$ is a sequence $(O_k)_{k=0}^{\infty}$ with the following properties

Monotonicity: $\forall k \in \mathbb{N}. O_k \sqsubseteq O_{k+1}$

Computability: There is a computable function $f : \mathbb{N} \to \mathcal{D}$ such that $f(k) = O_k$

Expressibility: There is a computable predicate p on $\{O_k | k \in \mathcal{N}\}$ such that $p(O_k)$ holds iff $\mathcal{P}$ has property $\mathcal{C}$ when subject to resource bound k

Why observation sequences?

Definition captures important properties for safety proofs

Why observation sequences?

Definition captures important properties for safety proofs

Monotonicity: With more resources, more results are possible

Why observation sequences?

Definition captures important properties for safety proofs

Monotonicity:	With more resources, more results are possible
Computability:	We can actually make these observations

Why observation sequences?

Definition captures important properties for safety proofs

Monotonicity:	With more resources, more results are possible
Computability:	We can actually make these observations
Expressibility:	What we observe informs us about what we care about

Guaranteeing convergence

Unbounded stack size ➡ Infinite state space

Guaranteeing convergence

Unbounded stack size ➡ Infinite state space

Observe only top of stack ➡ Finite observation poset

Guaranteeing convergence

Unbounded stack size ➡ Infinite state space

Observe only top of stack ➡ Finite observation poset

Monotonic sequence must eventually converge

Monotone data flow analysis?

Data flow analysis: iterate function to find least fixed point

$$f : D \to D$$

Monotone data flow analysis?

Data flow analysis: iterate function to find least fixed point

$$f : D \to D$$

CUBA: grow input until convergence

$$f : \mathbb{N} \to D$$

Monotone data flow analysis?

Data flow analysis: iterate function to find least fixed point

$$f : D \to D$$

CUBA: grow input until convergence

$$f : \mathbb{N} \to D$$

No fixed points!

$$f(n+1) = f(n) \not\Rightarrow f(n+2) = f(n)$$

Stuttering

How do we know when to stop?

$$f(n+1) = f(n)$$

Stuttering

How do we know when to stop?

$f(n+1) = f(n)$ Are we done yet? We might be...

Stuttering

How do we know when to stop?

$$f(n+1) = f(n)$$ Are we done yet? We might be...

$$f(n+2) = f(n)$$

Stuttering

How do we know when to stop?

$f(n+1) = f(n)$ Are we done yet? We might be...

$f(n+2) = f(n)$ How about now? Maybe...

Stuttering

How do we know when to stop?

$f(n+1) = f(n)$ Are we done yet? We might be...

$f(n+2) = f(n)$ How about now? Maybe...

$f(n+3) \sqsupset f(n)$

Stuttering

How do we know when to stop?

$f(n+1) = f(n)$	Are we done yet? We might be...
$f(n+2) = f(n)$	How about now? Maybe...
$f(n+3) \sqsupset f(n)$	Guess not

Stuttering

How do we know when to stop?

$f(n+1) = f(n)$	Are we done yet? We might be...
$f(n+2) = f(n)$	How about now? Maybe...
$f(n+3) \sqsupset f(n)$	Guess not

Recall: CPDS reachability is undecidable!

Stuttering

How do we know when to stop?

$f(n+1) = f(n)$	Are we done yet? We might be...
$f(n+2) = f(n)$	How about now? Maybe...
$f(n+3) \sqsupset f(n)$	Guess not

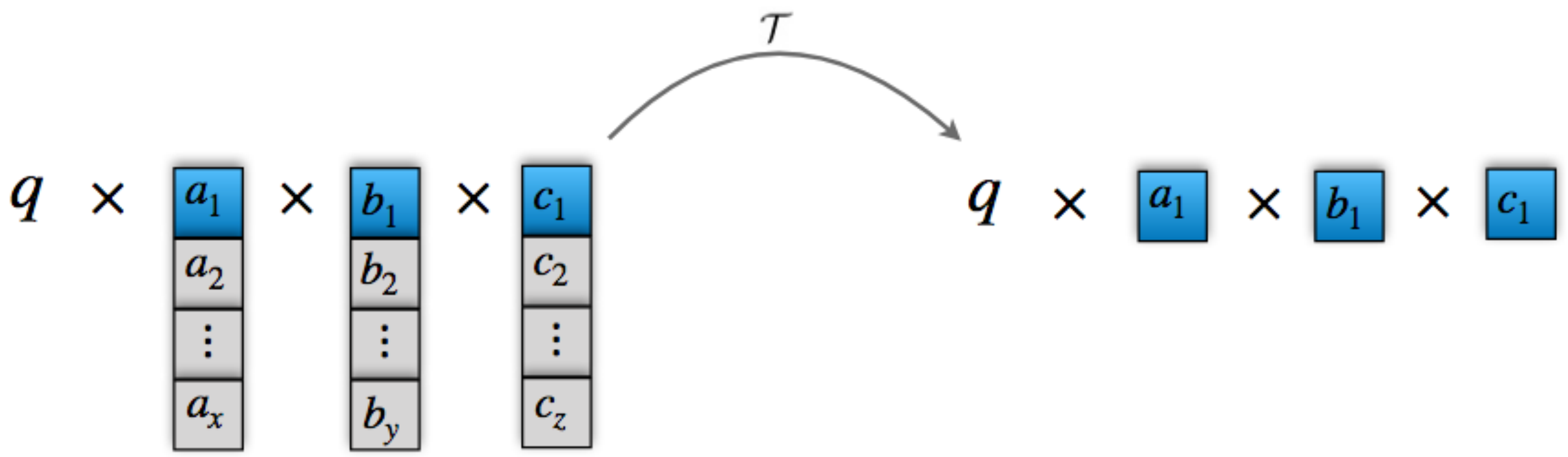
Recall: CPDS reachability is undecidable!

Key automation challenge: distinguish stuttering from convergence

Contents

- Past work: Context-bounded analysis
- New proof technique: Context-unbounded analysis
- The CUBA Algorithm
- Evaluation

Finitize the observation sequences to guarantee convergence



$q \times a_1, a_2, \vdots, a_x \times b_1, b_2, \vdots, b_y \times c_1, c_2, \vdots, c_z$
$\mathcal{T}$
$q \times a_1 \times b_1 \times c_1$

Stutter Detection is undecidable

Since convergence is guaranteed, but the problem is undecidable, stutter detection is undecidable

Stutter Detection is undecidable

Since convergence is guaranteed, but the problem is undecidable, stutter detection is undecidable

Must approximate!

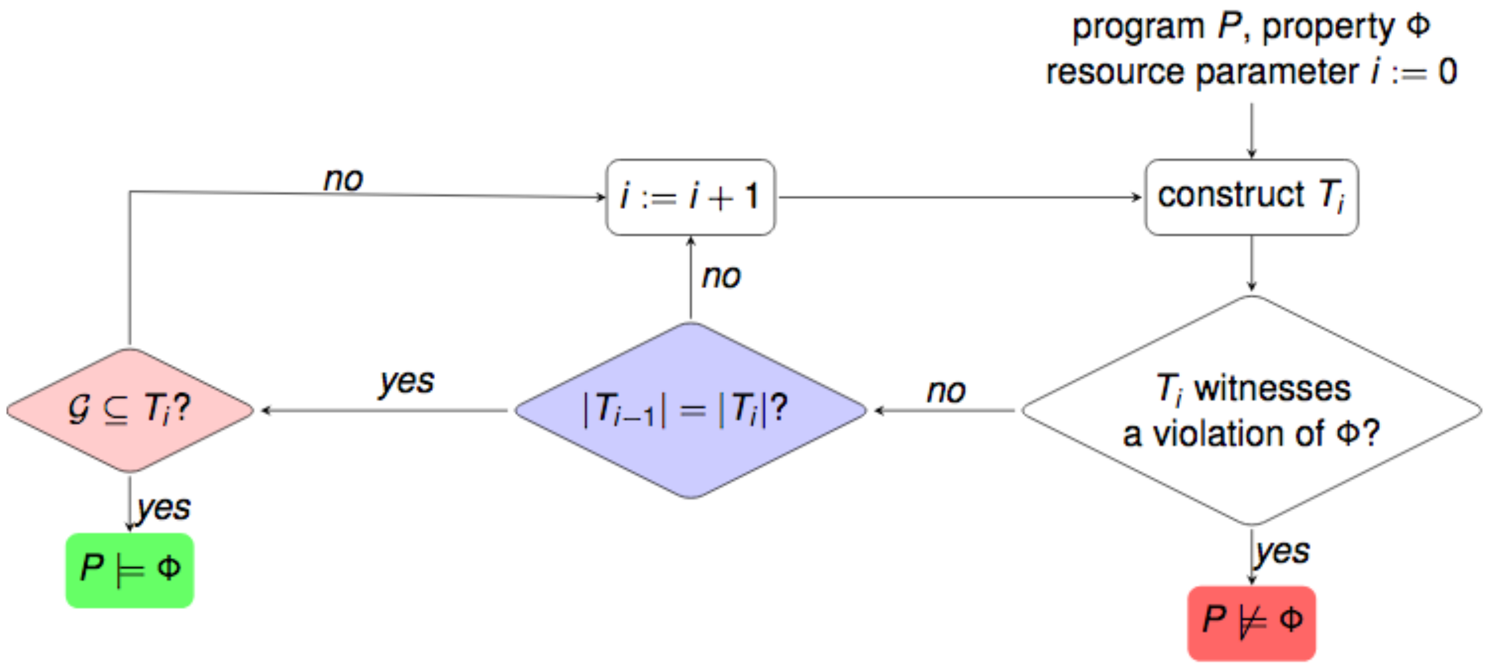
Only certain instructions can change observation set

If a stutter ends it has to be the consequence of a stack pop.

The observation sequence converges if all reachable popping states are in the observation set

We give an overapproximation of this set to stay decidable

Algorithm Summary


program P, property Φ
resource parameter $i := 0$
construct T_i
$i := i + 1$
no
T_i witnesses a violation of Φ?
yes
$P \not\models \Phi$
no
$|T_{i-1}| = |T_i|$?
no
yes
$\mathcal{G} \subseteq T_i$?
no
yes
$P \models \Phi$

$$\mathcal{G} := \{\textit{reachable generators}\}$$

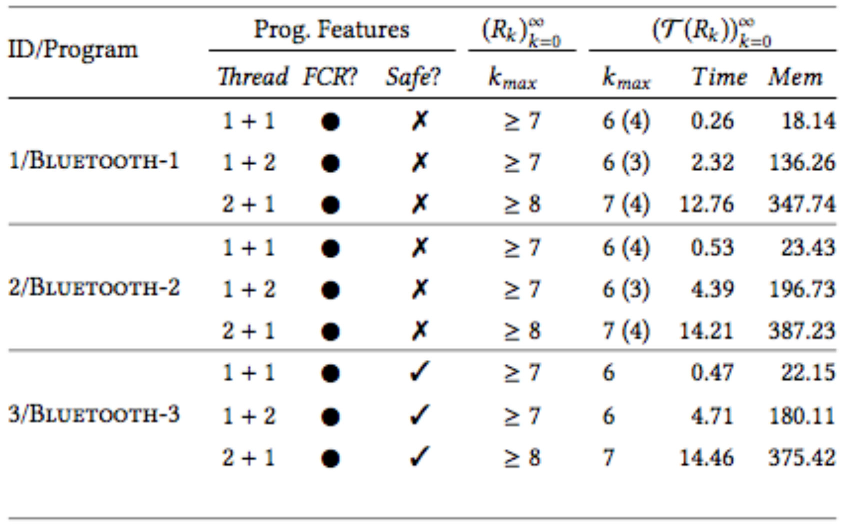
Contents

Empirical Evaluation

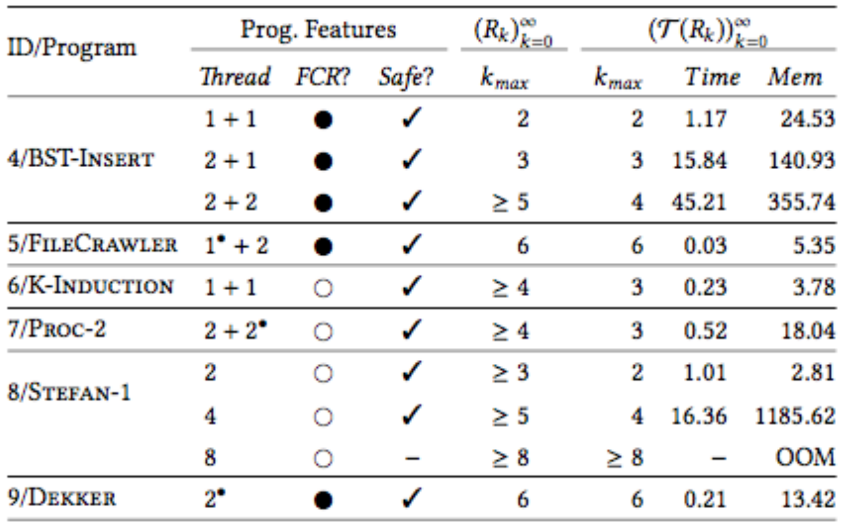
Evaluated our CUBA algorithm on 9 CPDSs converted from concurrent programs in C/Java

For comparison, evaluated most of those using JMoped, a CBA implementation

CUBA is effective at proving and refuting safety



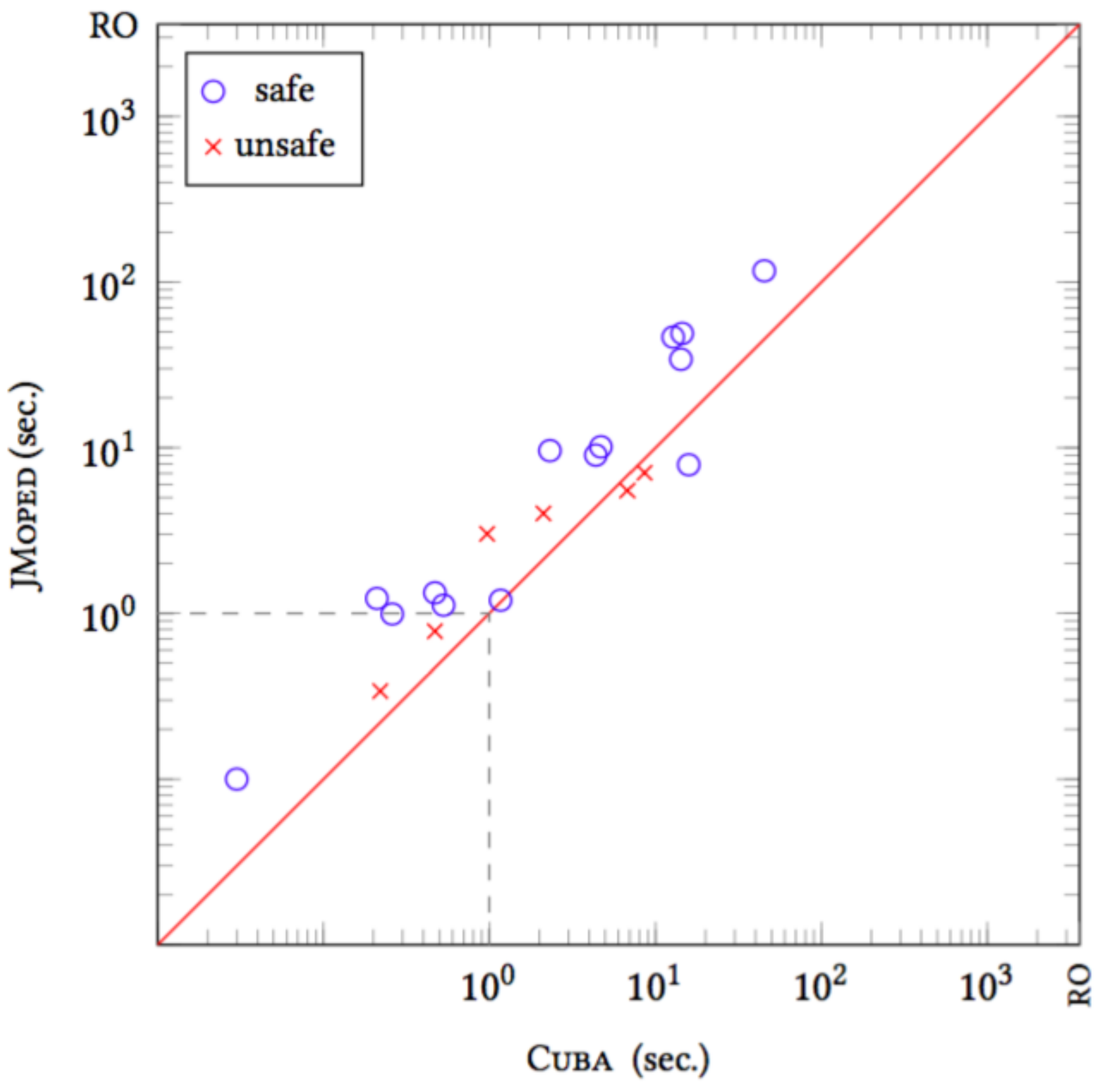
ID/Program	Prog. Features			$(R_k)_{k=0}^{\infty}$	$(\mathcal{T}(R_k))_{k=0}^{\infty}$		
	Thread	*FCR?*	*Safe?*	k_{max}	k_{max}	*Time*	*Mem*
1/Bluetooth-1	$1+1$	●	✗	≥ 7	6 (4)	0.26	18.14
	$1+2$	●	✗	≥ 7	6 (3)	2.32	136.26
	$2+1$	●	✗	≥ 8	7 (4)	12.76	347.74
2/Bluetooth-2	$1+1$	●	✗	≥ 7	6 (4)	0.53	23.43
	$1+2$	●	✗	≥ 7	6 (3)	4.39	196.73
	$2+1$	●	✗	≥ 8	7 (4)	14.21	387.23
3/Bluetooth-3	$1+1$	●	✓	≥ 7	6	0.47	22.15
	$1+2$	●	✓	≥ 7	6	4.71	180.11
	$2+1$	●	✓	≥ 8	7	14.46	375.42



ID/Program	Prog. Features			$(R_k)_{k=0}^{\infty}$	$(\mathcal{T}(R_k))_{k=0}^{\infty}$		
	Thread	*FCR?*	*Safe?*	k_{max}	k_{max}	*Time*	*Mem*
4/BST-Insert	$1+1$	●	✓	2	2	1.17	24.53
	$2+1$	●	✓	3	3	15.84	140.93
	$2+2$	●	✓	≥ 5	4	45.21	355.74
5/FileCrawler	$1^{\bullet}+2$	●	✓	6	6	0.03	5.35
6/K-Induction	$1+1$	○	✓	≥ 4	3	0.23	3.78
7/Proc-2	$2+2^{\bullet}$	○	✓	≥ 4	3	0.52	18.04
8/Stefan-1	2	○	✓	≥ 3	2	1.01	2.81
	4	○	✓	≥ 5	4	16.36	1185.62
	8	○	–	≥ 8	≥ 8	–	OOM
9/Dekker	$2^{\bullet}$	●	✓	6	6	0.21	13.42

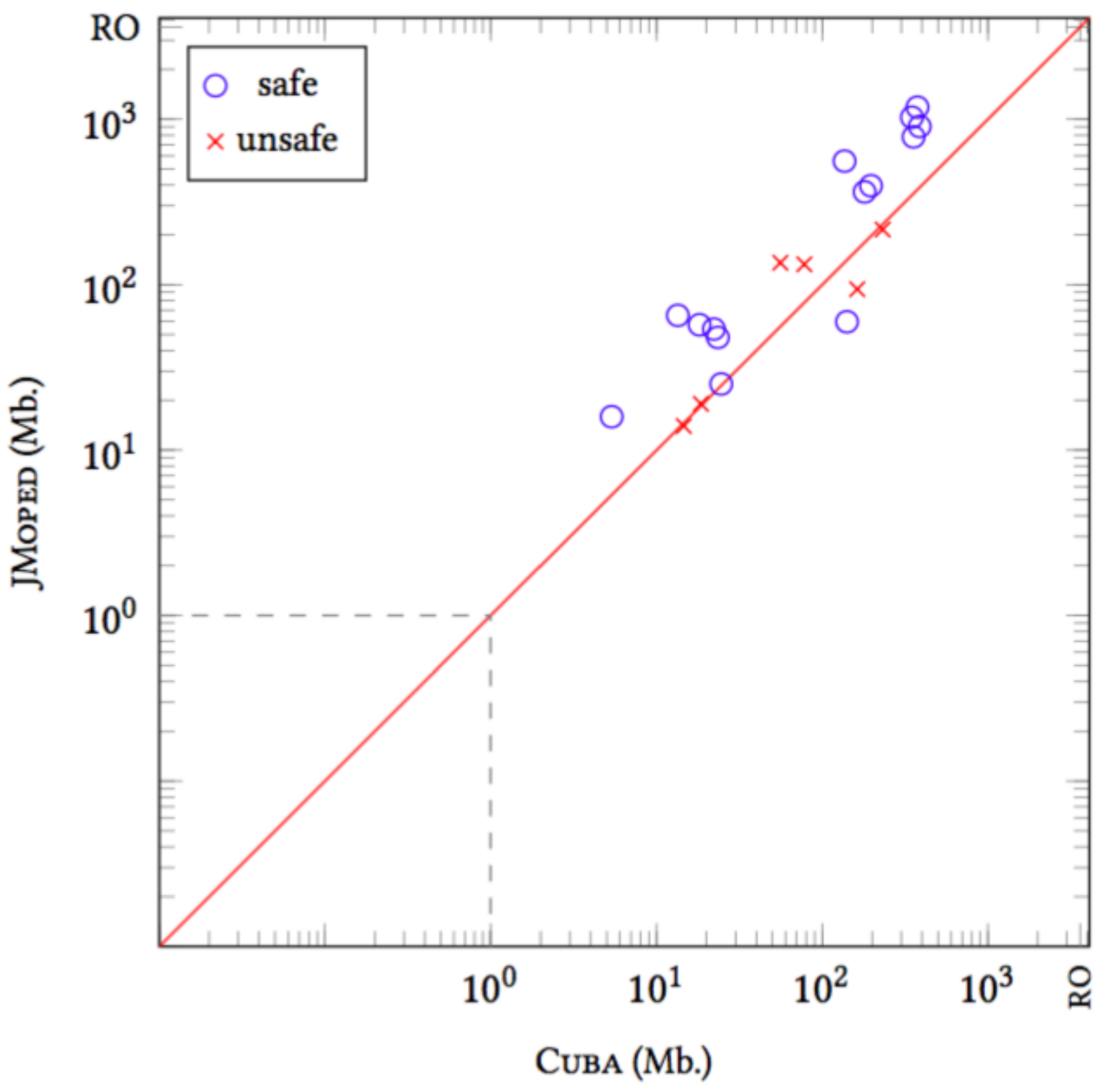
CUBA has competitive performance with CBA

Runtime



CUBA has competitive performance with CBA

Memory Usage



More in the paper!

- Systematic theory of observation sequences
- How we represent obvervation sequences efficiently?
- How to compute generator sets

Conclusions

- Context UnBounded Analysis can automatically find bugs and prove safety of concurrent programs
- Observation sequences provide unifying perspective for understanding resource constrained analyses
- As efficient as previous, weaker analyses