

Resource-Parameterized Program Analysis using Observation Sequences

Peizun Liu

Ph.D. Proposal

CCIS, Northeastern University

October 30, 2018

Outline

- Overview of the Research
- A Paradigm: Observation Sequences
- Application: Context-UnBounded Analysis
- Proposed Work: Applications and Beyond
- Conclusion and Schedule

Outline

- **Overview of the Research**
- **A Paradigm: Observation Sequences**
- **Application: Context-UnBounded Analysis**
- **Proposed Work: Applications and Beyond**
- **Conclusion and Schedule**

Problem Statement

Target is ...

- ☞ *resource-parameterized programs*, which are designed over a *variable* number of *discrete resources*.

“Resources” could mean:

threads

context switches

memory writes

executions

message channels

...

...

Problem Statement

Target is ...

- ☞ *resource-parameterized programs*, which are designed over a *variable* number of *discrete resources*.

“Resources” could mean:



threads



context switches



memory writes



executions



message channels

...

Problem Statement

Analysis is to ...

- ☞ ensure *safety* of such programs for an *unspecified* number of resource instances

Safety could mean ...

- ☞ free of *data race / race condition* in shared-memory multi-threaded programs
- ☞ *responsiveness* in message-passing programs
- ☞ *deadlock-free* or *mutual exclusion* in distributed systems
- ☞ *assertions* ...

Motivation

Reason 1

☞ Resource-parameterized programs are *ubiquitous*.



Motivation

Reason 2

- ☞ Ensuring their safety is **desirable** and **significant**.



Motivation

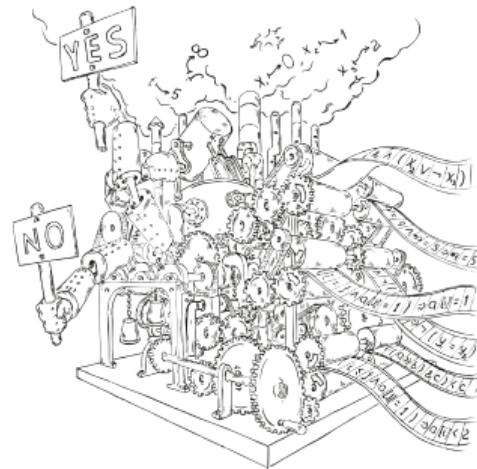
However, ...

- resource-parameterized program analysis is *challenging*.



I can't find an efficient algorithm, but neither can all these famous people.

intractable



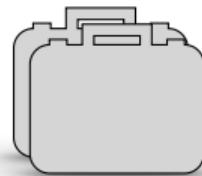
*or even **undecidable***

A Sidestep

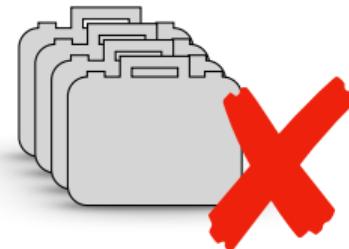
Resource-bounded analysis



start with one resource



iteratively increase resource



until a violation is found



until a pre-specified upper bound

A Sidestep

Tested empirically [ASPLOS'08]

- ☞ Most bugs can be exposed with a *small* number of resources.

We thus have



*a **bug-finding** technique*

A Sidestep

Still, uncertainty remains ...

- ☛ beyond the pre-specified bound.



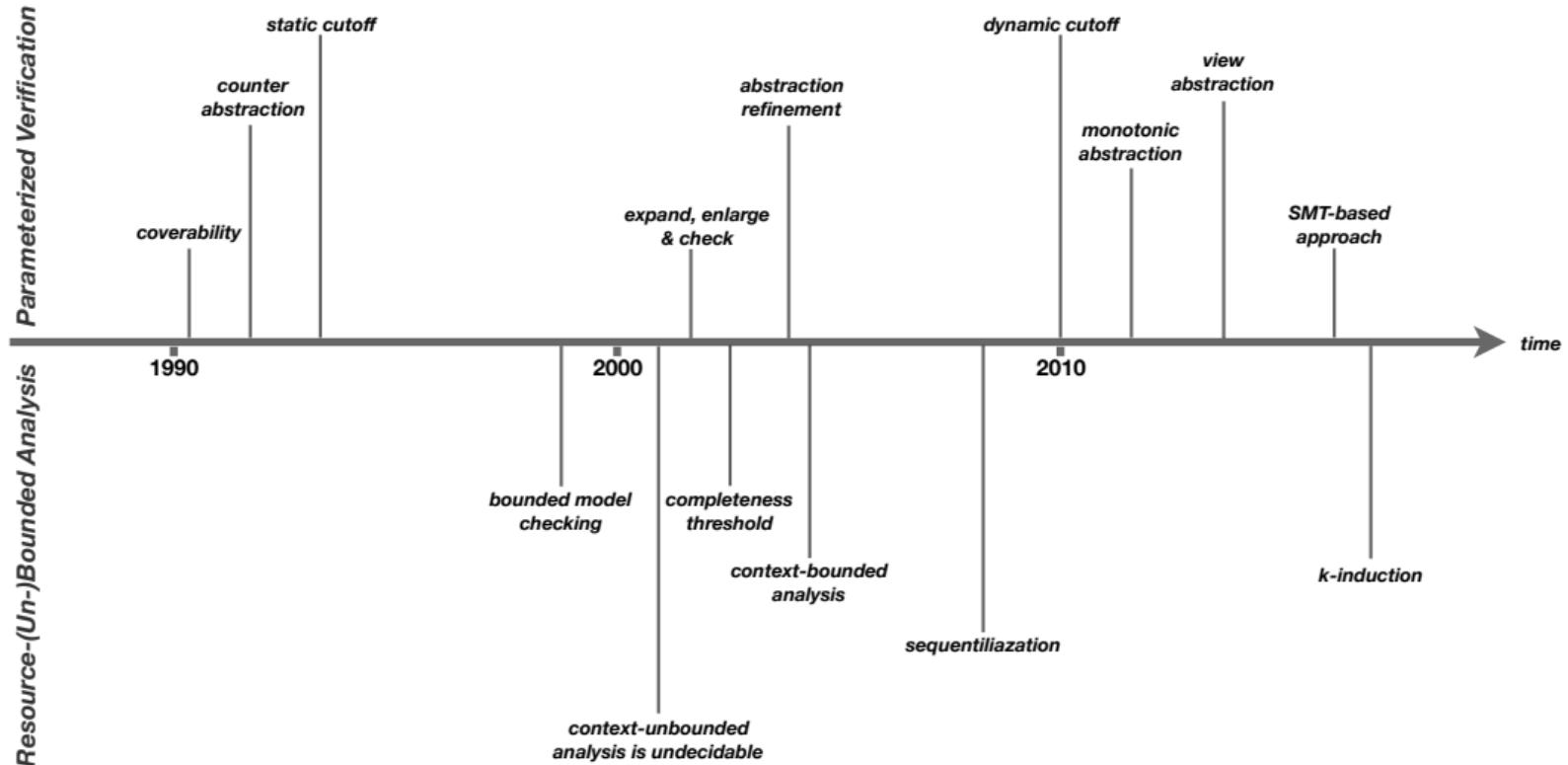
Beyond the Sidestep

Can we lift the bug-finding technique to resource-unbounded analysis?



Gotcha! No place
is safe to hide in
formal testbench!

Status of Research



Research Goal

To provide ...

- ☞ a *uniform* paradigm, which can
 - lift resource-bounded bug-finding technique to *resource-unbounded analysis*.*

Our Paradigm: Bird's Eye View

Observation sequence (OS) ...

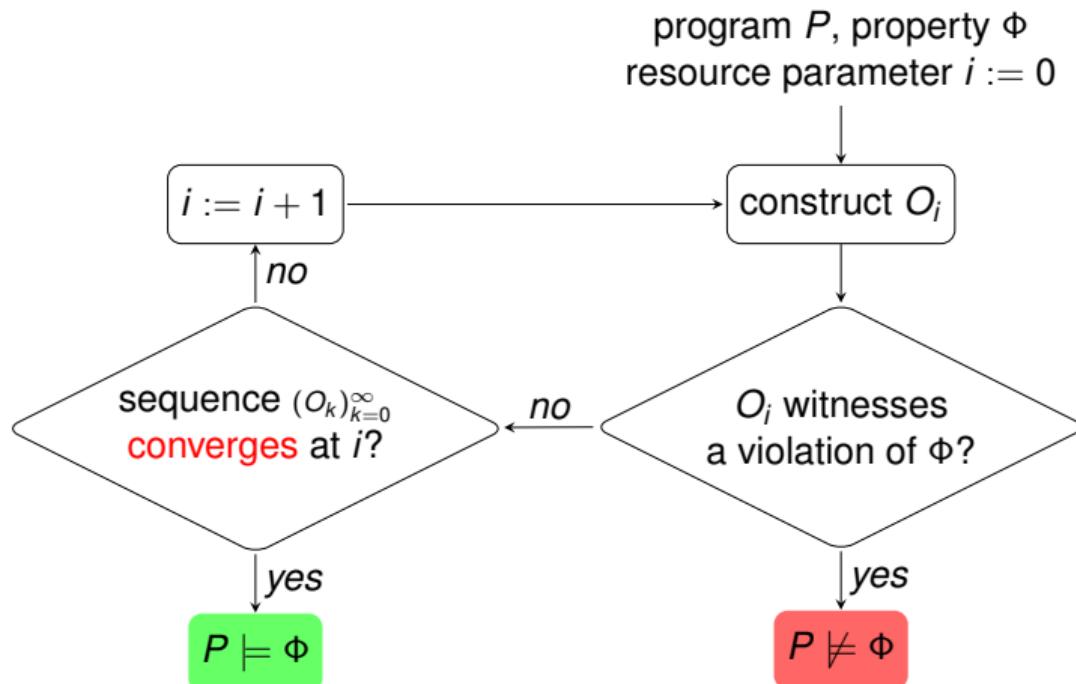
Informally, a sequence of program behaviors O_k observed within k instances of resource.

Examples

- ☞ $O_k := \{ \text{reachable program states within } k \text{ threads} \}$
- ☞ $O_k := \{ \text{reachable program locations within } k \text{ threads} \}$
- ☞ ...

Our Paradigm: Bird's Eye View

Scheme



Outline

- Overview of the Research
- **A Paradigm: Observation Sequences**
- Application: Context-UnBounded Analysis
- Proposed Work: Applications and Beyond
- Conclusion and Schedule

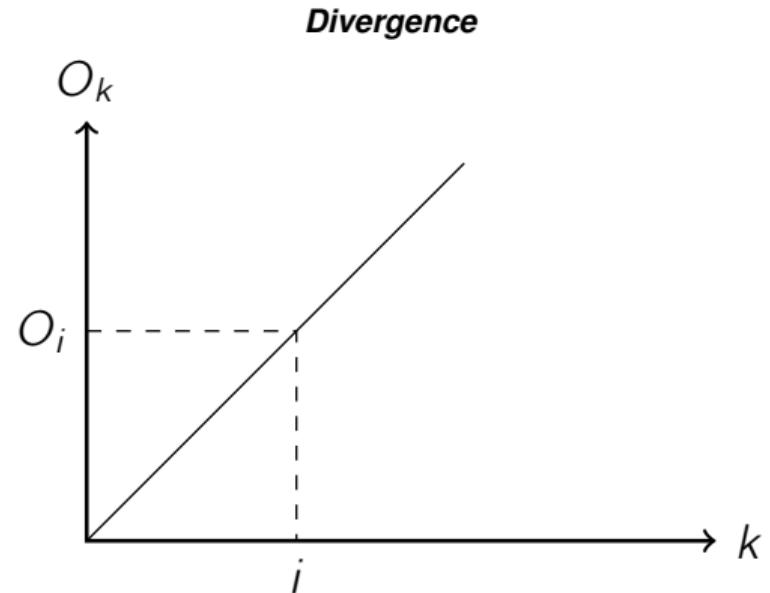
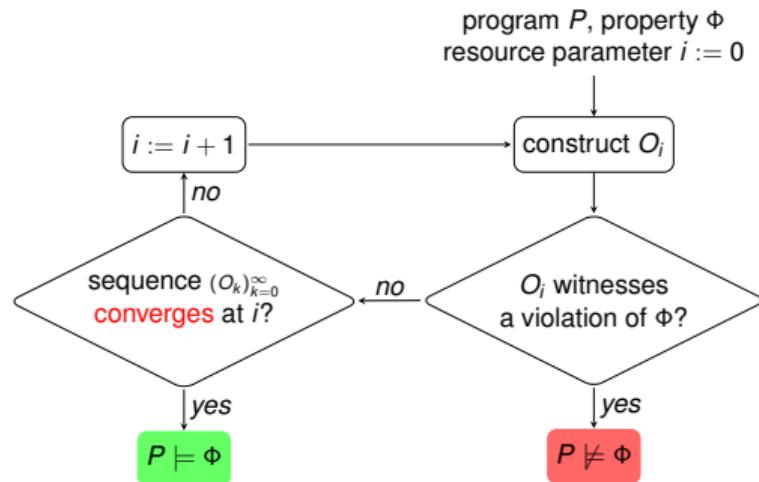
Observation Sequences

Definition

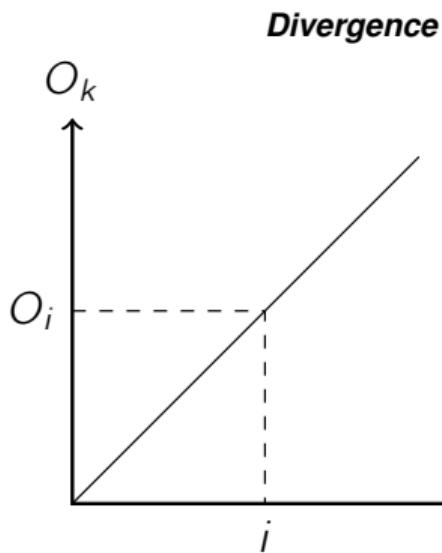
An **observation sequence** is a sequence $(O_k)_{k=0}^{\infty}$ with the following properties:

- for all k , $O_k \subseteq O_{k+1}$, that is *monotonicity*.
- for all k , O_k is *computable*.
- for all k , $O_k \models \Phi$ is *decidable*, where Φ is a property of interest.

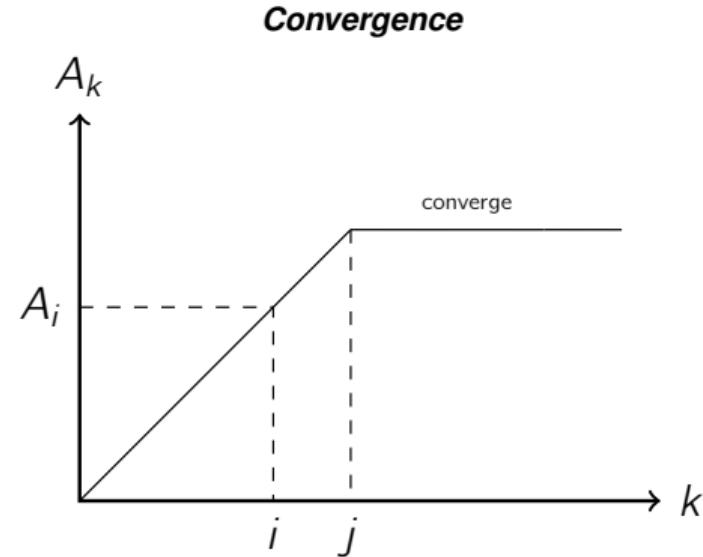
Convergence Detection is Challenging



Abstraction



abstraction \rightarrow

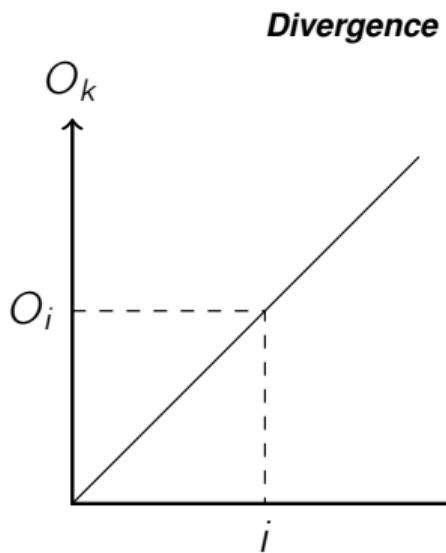


$$A_k := \text{abstraction}(O_k)$$

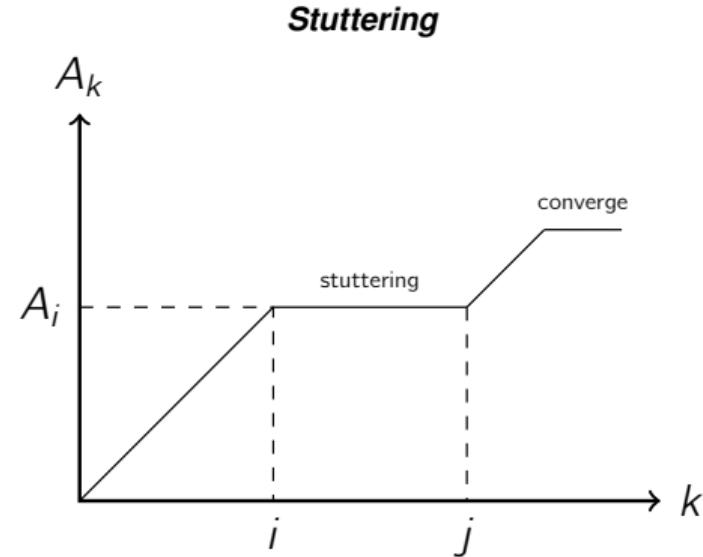
Convergence Property

An OS $(O_k)_{k=0}^{\infty}$ over a **finite** domain always converges.

Stuttering

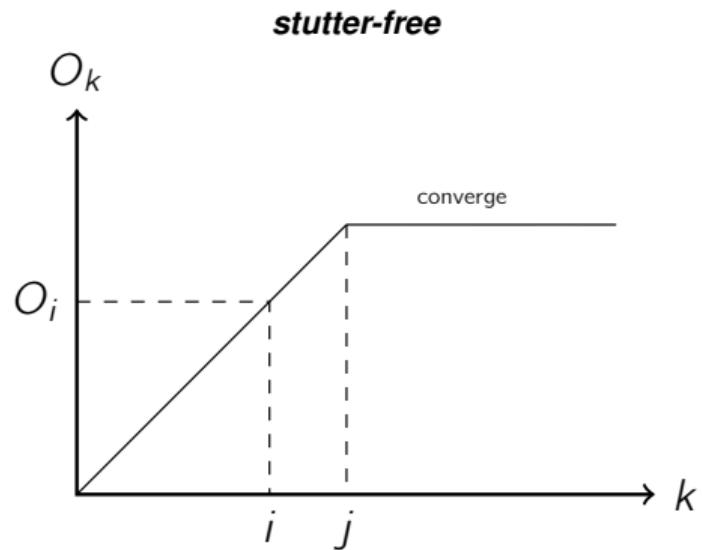
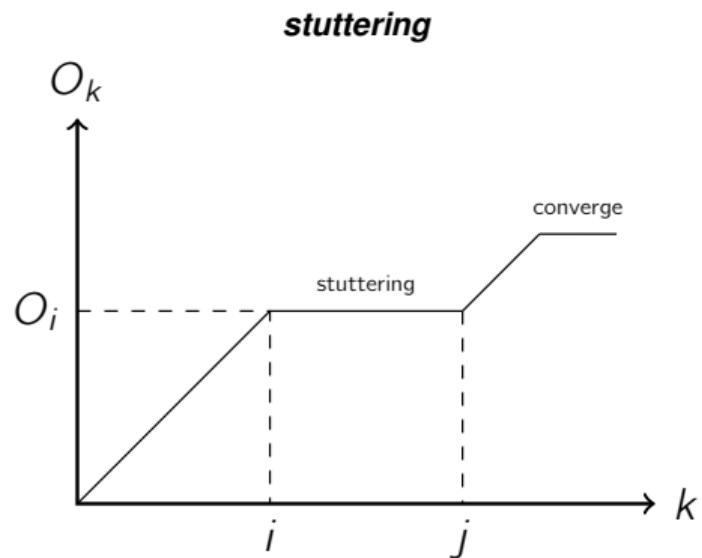


abstraction \rightarrow

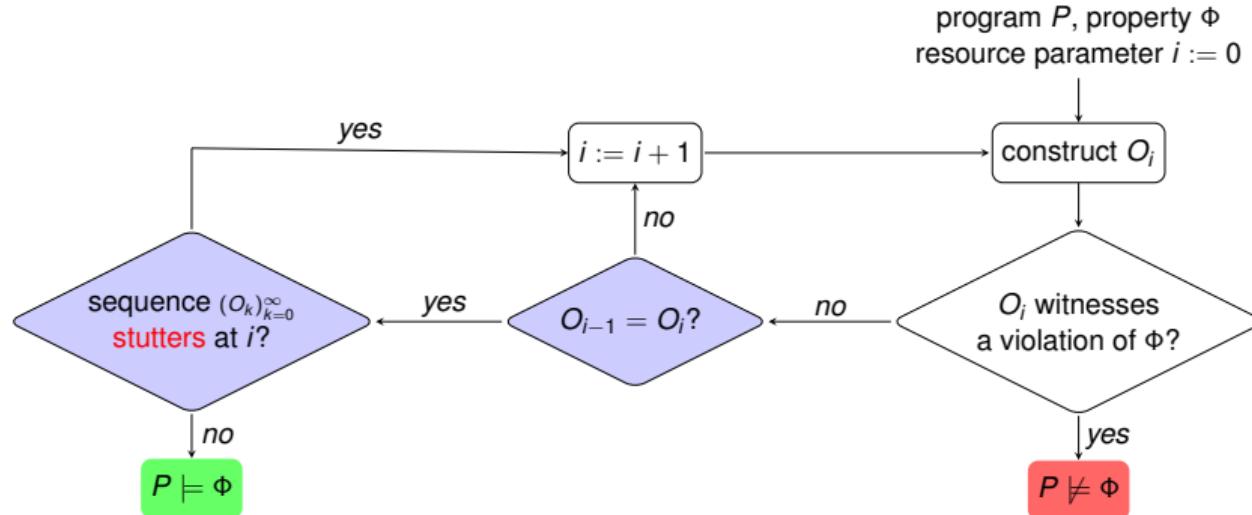


$$A_k := \text{abstraction}(O_k)$$

Convergence



A Refined Scheme



Outline

- Overview of the Research
- A Paradigm: Observation Sequences
- **Application: Context-UnBounded Analysis**
- Proposed Work: Applications and Beyond
- Conclusion and Schedule

Context-UnBounded Analysis (CUBA)

Target is ...

shared-memory multi-threaded recursive programs.

Resource is ...

the number of contexts in the executions.

Observation is ...

the set of reachable program states w.r.t. k contexts.

Analysis is ...

to check the reachability of bad states.

Operational Model

Concurrent Pushdown System (CPDS)

A CPDS P^n is a collection of n PDS $P_i = (Q, \Sigma_i, \Delta_i, q^l)$, $1 \leq i \leq n$, where

- Q is a finite set of **shared** states;
- Σ_i is a finite set of **local** states;
- $\Delta_i \subseteq (Q \times \Sigma_i^{\leq 1}) \times (Q \times \Sigma_i^{\leq 2})$, $\Sigma_i^{\leq 1} = \Sigma_i \cup \{\varepsilon\}$ and $\Sigma_i^{\leq 2} = \{\omega \in \Sigma_i^* \mid |\omega| \leq 2\}$;
- $q^l \in Q$ is the initial shared state.

Operational Model

Semantics

$$\frac{(q, a_1) \rightarrow (p, \sigma_1 \sigma_2) \in \Delta}{\begin{array}{c} p \times \begin{array}{c} \sigma_1 \\ \sigma_2 \\ a_2 \\ \vdots \\ a_x \end{array} \\ \cdots \rightarrow \end{array}} \text{ push}$$

Diagram illustrating a stack structure for push operation:

a_1
a_2
\vdots
a_x

$$\frac{(q, a_1) \rightarrow (p, \varepsilon) \in \Delta}{\begin{array}{c} p \times \begin{array}{c} a_2 \\ \vdots \\ a_x \end{array} \\ \cdots \rightarrow \end{array}} \text{ pop}$$

Diagram illustrating a stack structure for pop operation:

a_1
a_2
\vdots
a_x

$$\frac{(q, a_1) \rightarrow (p, \sigma) \in \Delta}{\begin{array}{c} p \times \begin{array}{c} \sigma \\ a_2 \\ \vdots \\ a_x \end{array} \\ \cdots \rightarrow \end{array}} \text{ overwrite}$$

Diagram illustrating a stack structure for overwrite operation:

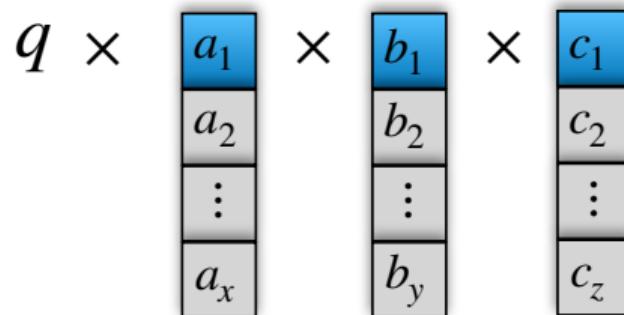
a_1
a_2
\vdots
a_x

Operational Model

States

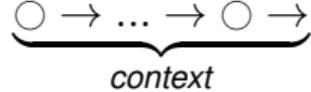
A **global state** of a CPDS is an element of $Q \times \Sigma_1^* \times \dots \times \Sigma_n^*$, written in angle brackets: $\langle q | \omega_1, \dots, \omega_n \rangle$.

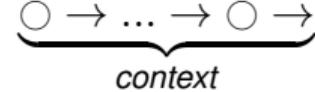
For instance



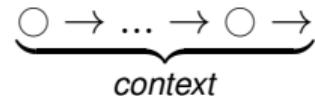
Operational Model

Executions

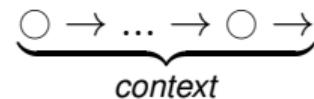
Thread A : 



Thread B :



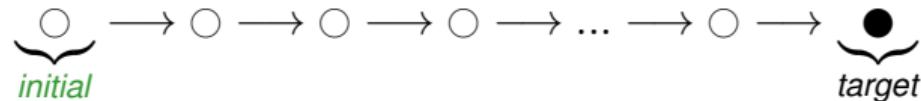
Thread C :



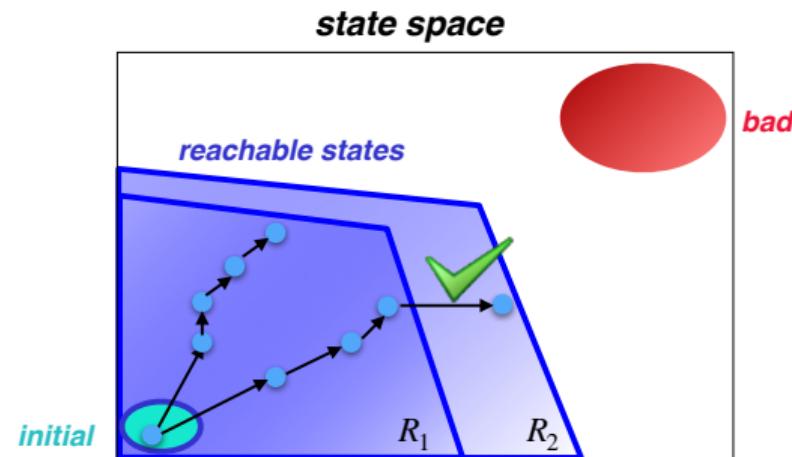
resource := contexts

Problem Statement

A state is reachable if ...



Safety as reachability



Problem Statement

Observation is ...

- ☞ R_k = the set of states reachable within k contexts.

R_k can be *infinite* [CAV'00],
but,

R_k can be *finitely* represented [TACAS'05].

(Un-)Decidability

- ☞ Reachability of CPDS is **undecidable** [TOPLAS'00].

But

- ☞ Context-bounded reachability of CPDS is **decidable** [TACAS'05].

CUBA using Observation Sequences of Global States

- ☞ $(O_k)_{k=0}^{\infty} := R_0, R_1, R_2, \dots$, where each state in R_k is of the form:

$$q \times \begin{array}{c} a_1 \\ a_2 \\ \vdots \\ a_x \end{array} \times \begin{array}{c} b_1 \\ b_2 \\ \vdots \\ b_y \end{array} \times \begin{array}{c} c_1 \\ c_2 \\ \vdots \\ c_z \end{array}$$

$(R_k)_{k=0}^{\infty}$ is defined over an **infinite** domain and **stutter-free** [PLDI'18]

Example

Shared states:

$$Q = \{ 0, 1, 2, 3 \}$$

Thread 1:

$$\Sigma_1 = \{ 1, 2 \}$$

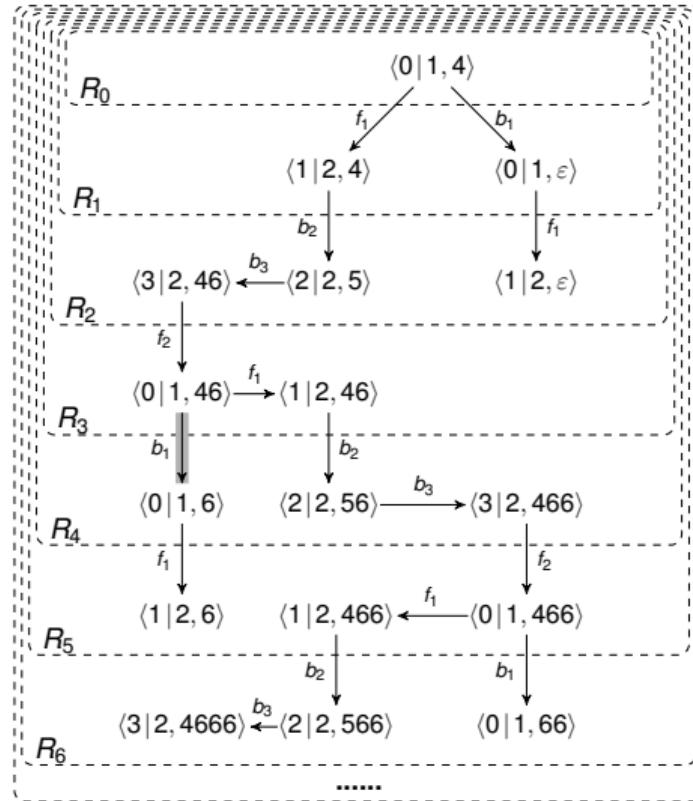
$$\Delta_1 = \{ f_1 : (0, 1) \rightarrow (1, 2), \\ f_2 : (3, 2) \rightarrow (0, 1) \}$$

Thread 2:

$$\Sigma_2 = \{ 4, 5, 6 \}$$

$$\Delta_2 = \{ b_1 : (0, 4) \rightarrow (0, \varepsilon), \\ b_2 : (1, 4) \rightarrow (2, 5), \\ b_3 : (2, 5) \rightarrow (3, 46) \}$$

$$q^I = 0$$



Example

Shared states:

$$Q = \{ 0, 1, 2, 3 \}$$

Thread 1:

$$\Sigma_1 = \{ 1, 2 \}$$

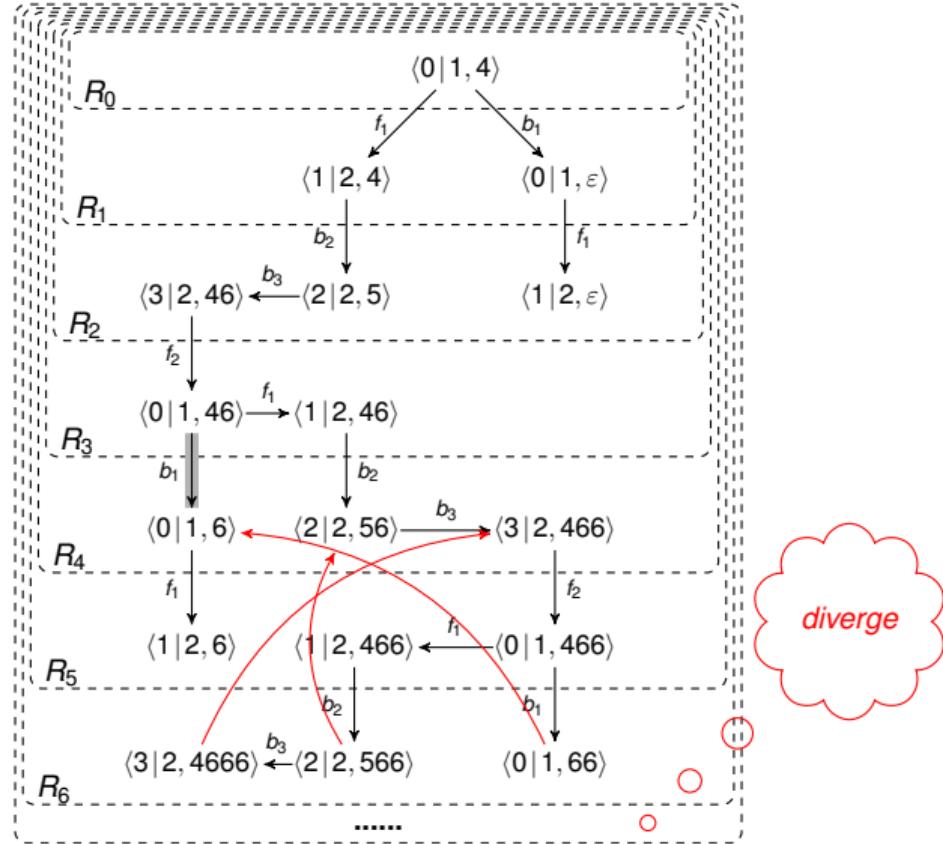
$$\Delta_1 = \{ f_1 : (0, 1) \rightarrow (1, 2), \\ f_2 : (3, 2) \rightarrow (0, 1) \}$$

Thread 2:

$$\Sigma_2 = \{ 4, 5, 6 \}$$

$$\Delta_2 = \{ b_1 : (0, 4) \rightarrow (0, \varepsilon), \\ b_2 : (1, 4) \rightarrow (2, 5), \\ b_3 : (2, 5) \rightarrow (3, 46) \}$$

$$q' = 0$$



How to Proceed?

Give up?

Well, do not give up so quickly. Because we know ...

Convergence Property

*An OS $(O_k)_{k=0}^{\infty}$ over a **finite** domain always converges.*

How to Proceed?

Give up?

Well, do not give up so quickly. Because we know ...

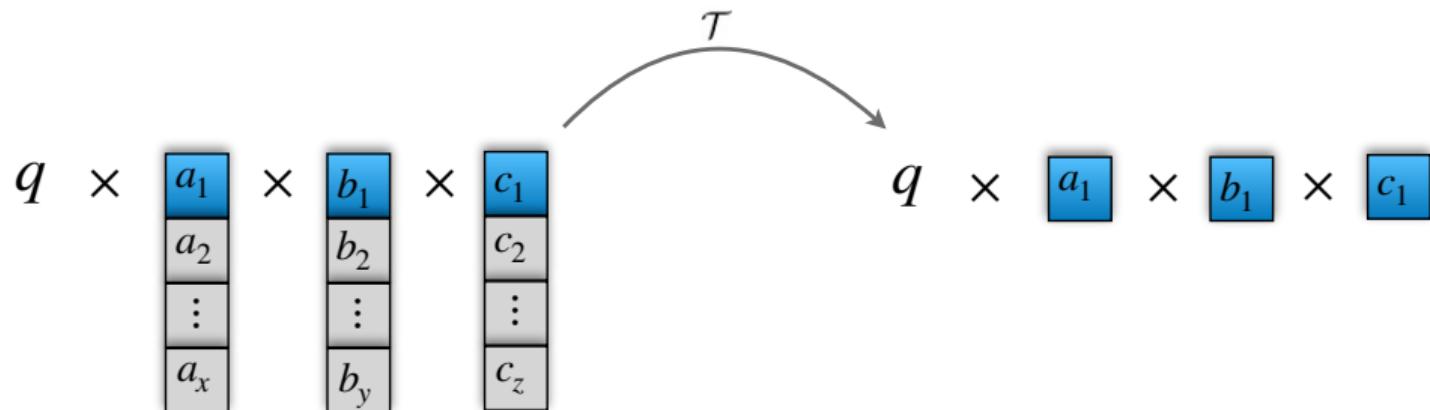
Convergence Property

*An OS $(O_k)_{k=0}^{\infty}$ over a **finite** domain always converges.*

CUBA using Observation Sequences of Visible State

Project global states to a finite domain ...

- ☛ by cutting off tails of stacks.



CUBA using Observation Sequences of Visible State

Project global states to a finite domain ...

$$q \times [a_1] \times [b_1] \times [c_1]$$

Visible states

- ☞ suffice to express many safety properties, e.g. various assertions, data race, race condition, etc.

CUBA using Observation Sequences of Visible State

- ☞ $(O_k)_{k=0}^{\infty} := T_0, T_1, T_2, \dots$, where each state in T_k is of the form:

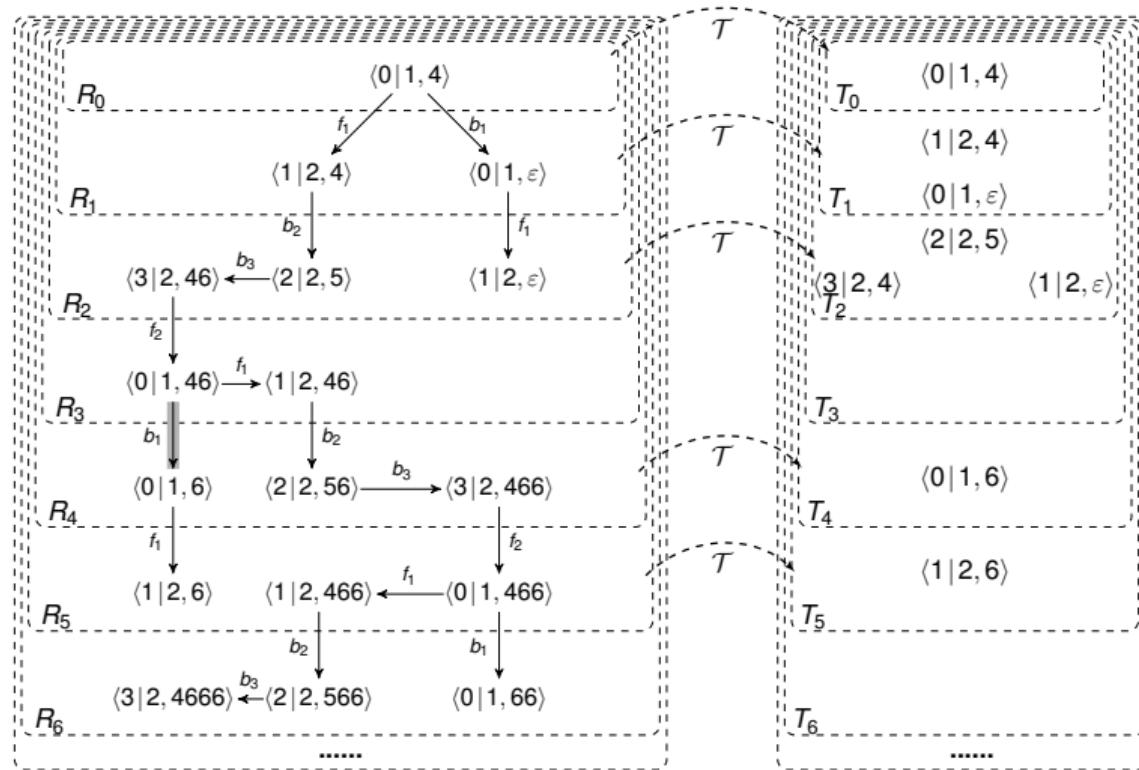
$$q \times \boxed{a_1} \times \boxed{b_1} \times \boxed{c_1}$$

$(T_k)_{k=0}^{\infty}$ is guaranteed to **converge**.

$$T_k := \mathcal{T}(R_k)$$

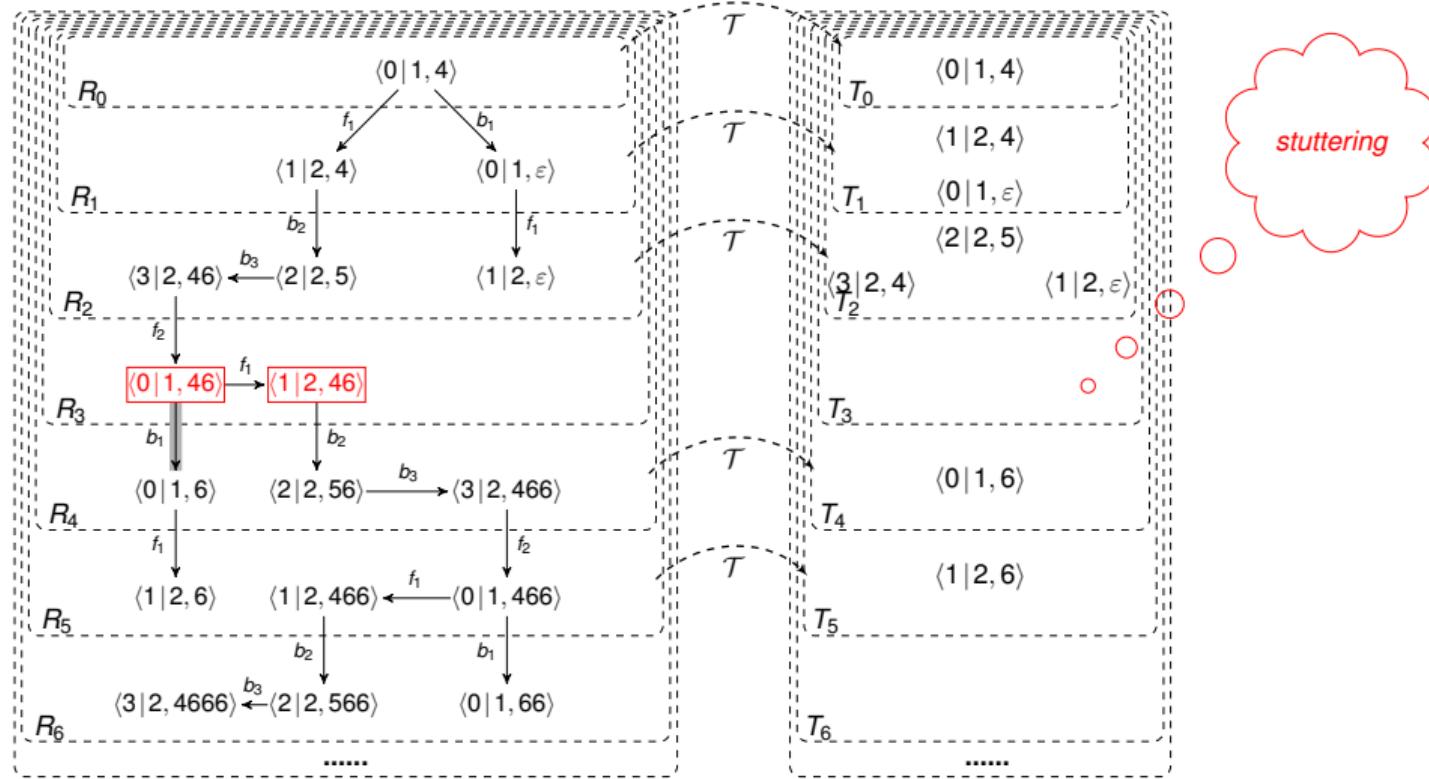
Example Revisited

Can we answer the convergence of visible state sequence easily?



Example Revisited

Not really ...



Example Revisited

But, an observation on generators

Shared states:

$$Q = \{ 0, 1, 2, 3 \}$$

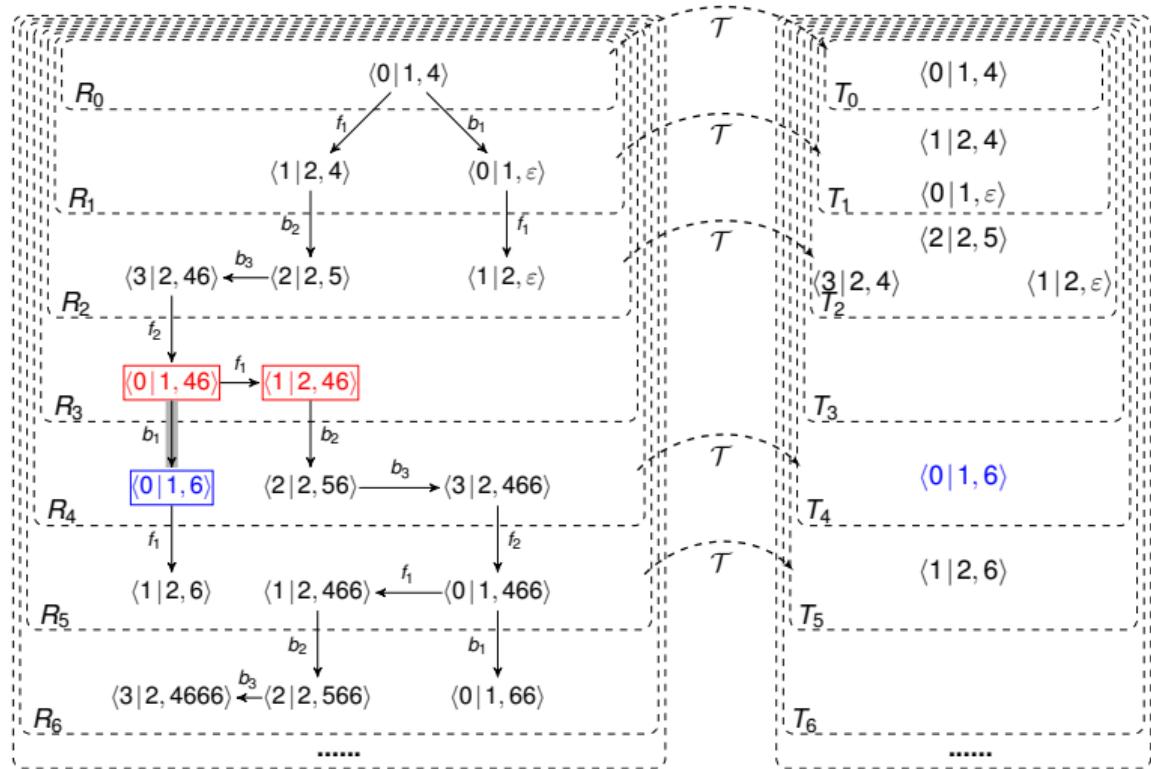
Thread 1:

$$\begin{aligned}\Sigma_1 &= \{ 1, 2 \} \\ \Delta_1 &= \{ f_1 : (0, 1) \rightarrow (1, 2), \\ &\quad f_2 : (3, 2) \rightarrow (0, 1) \}\end{aligned}$$

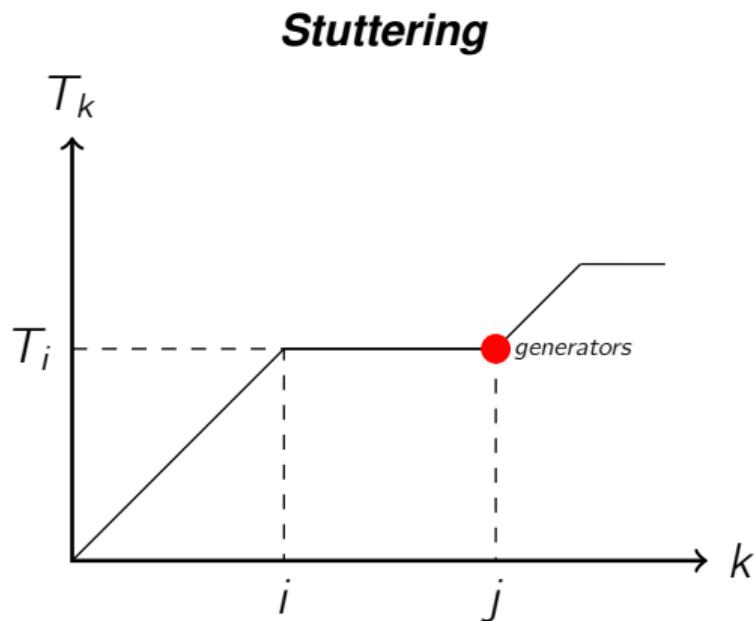
Thread 2:

$$\begin{aligned}\Sigma_2 &= \{ 4, 5, 6 \} \\ \Delta_2 &= \{ b_1 : (0, 4) \rightarrow (0, \varepsilon), \\ &\quad b_2 : (1, 4) \rightarrow (2, 5), \\ &\quad b_3 : (2, 5) \rightarrow (3, 46) \}\end{aligned}$$

$$q' = 0$$



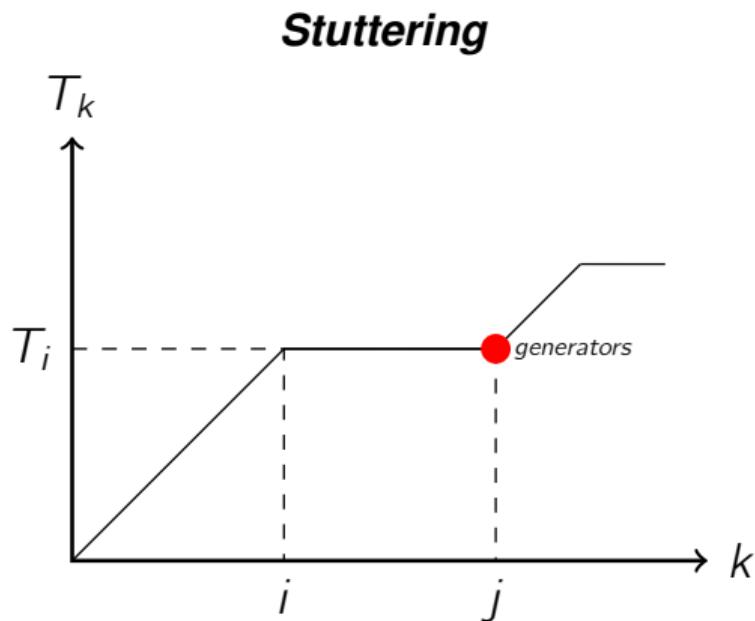
Stuttering Detection



Properties [PLDI'18]

- ☞ generators are of special form
⇒ can be statically approximated
- ☞ all generators have been reached
⇒ OS converges
- ☞ any overapproximation suffices

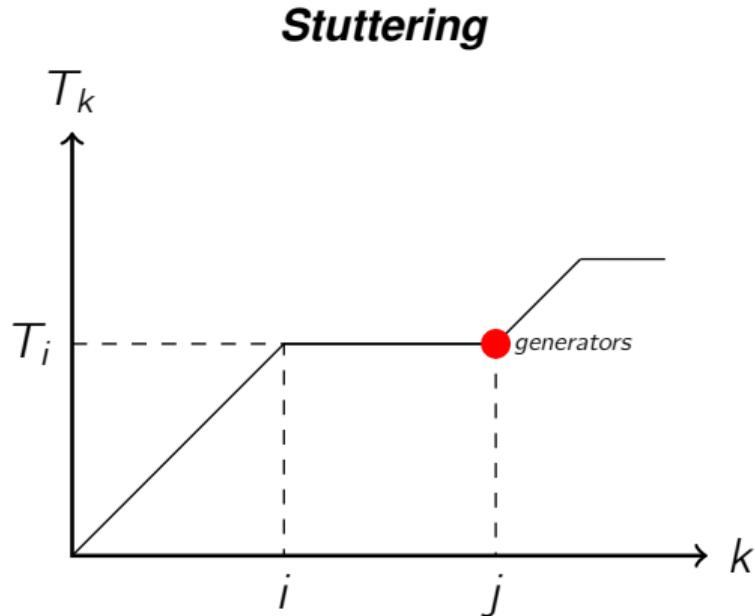
Stuttering Detection



Properties [PLDI'18]

- ☞ generators are of special form
⇒ can be statically approximated
- ☞ all generators have been reached
⇒ OS converges
- ☞ any overapproximation suffices

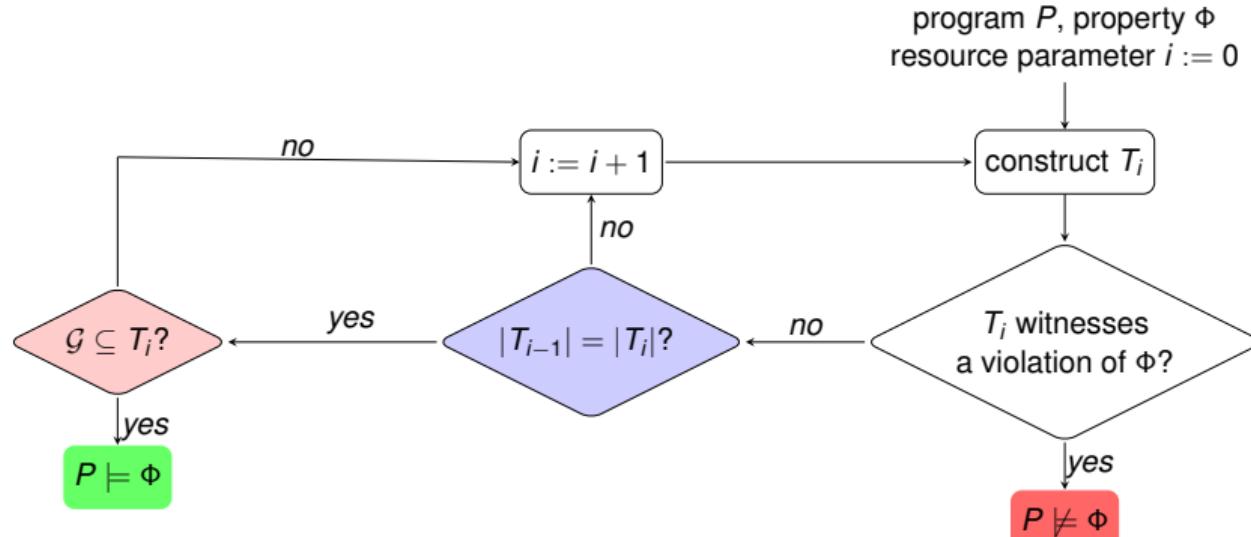
Stuttering Detection



Properties [PLDI'18]

- ☞ generators are of special form
⇒ can be statically approximated
- ☞ all generators have been reached
⇒ OS converges
- ☞ any overapproximation suffices

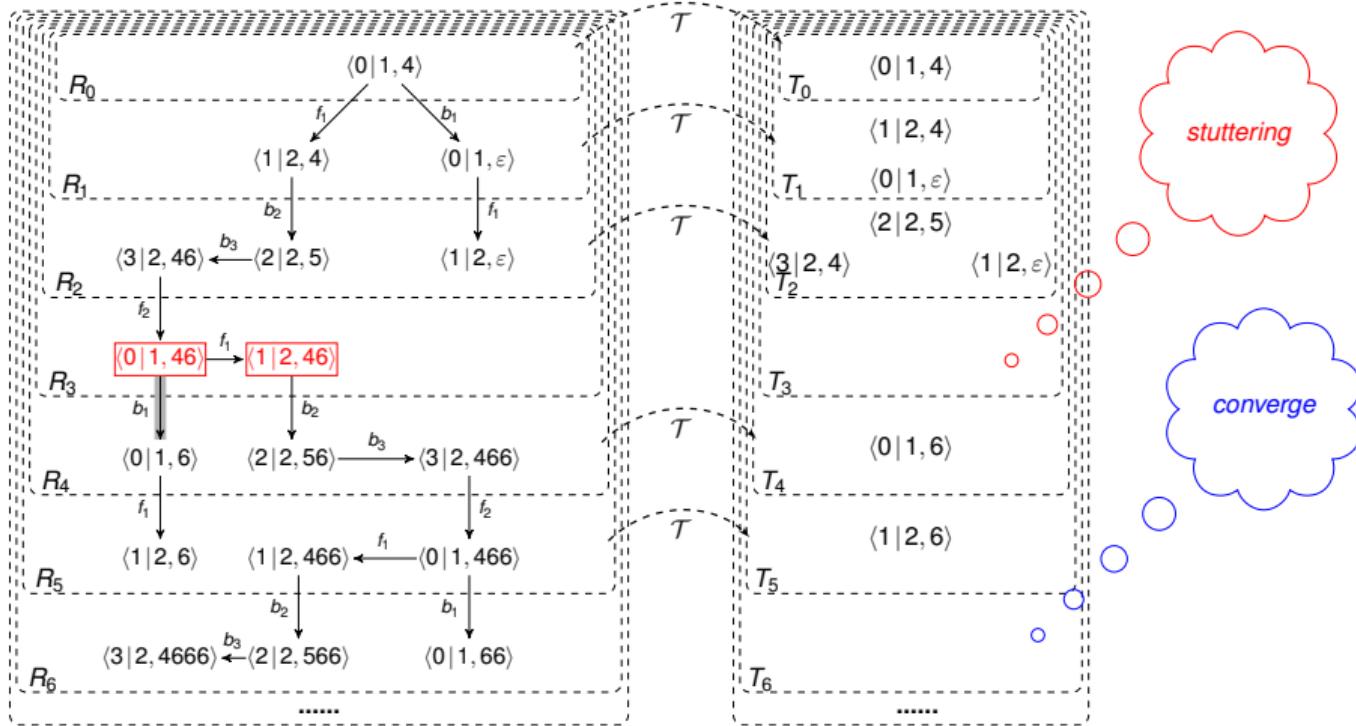
Algorithm



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

Example Revisited

$$\mathcal{G} = \{\langle 0|1, \varepsilon \rangle, \langle 0|1, 6 \rangle\}$$



Empirical Evaluation

Performance

ID/Program	Prog. Features		$(T_k)_{k=0}^{\infty}$	
	Thread	Safe?	k_{max}	Time (sec.)
1/BLUETOOTH-1	1 + 1	\times	6 (4)	0.26
	1 + 2	\times	6 (3)	2.32
	2 + 1	\times	7 (4)	12.76
2/BLUETOOTH-2	1 + 1	\times	6 (4)	0.53
	1 + 2	\times	6 (3)	4.39
	2 + 1	\times	7 (4)	14.21
3/BLUETOOTH-3	1 + 1	\checkmark	6	0.47
	1 + 2	\checkmark	6	4.71
	2 + 1	\checkmark	7	14.46

ID/Program	Prog. Features		$(T_k)_{k=0}^{\infty}$	
	Thread	Safe?	k_{max}	Time (sec.)
4/BST-INSERT	1 + 1	\checkmark	2	1.17
	2 + 1	\checkmark	3	15.84
	2 + 2	\checkmark	4	45.21
5/FILECRAWLER	1 [•] + 2	\checkmark	6	0.03
6/K-INDUCTION	1 + 1	\checkmark	3	0.23
7/PROC-2	2 + 2 [•]	\checkmark	3	0.52
8/STEFAN-1	2	\checkmark	2	1.01
	4	\checkmark	4	16.36
	8	—	≥ 8	—
9/DEKKER	2 [•]	\checkmark	6	0.21

Outline

- Overview of the Research
- A Paradigm: Observation Sequences
- Application: Context-UnBounded Analysis
- **Proposed Work: Applications and Beyond**
- Conclusion and Schedule

Queue-Parameterized Analysis

Target is ...

message-passing programs.

Resource is ...

the size of message queues.

Observation is ...

*the set of **reachable** program states w.r.t. the size of queue within k.*

Analysis is ...

*to check the reachability of **bad** states.*

Motivation

Why message queues? Because they are ...

- ☞ a key *synchronization* mechanism;
- ☞ a key reason to generate *infinite* state space;
- ☞ a key reason to cause *undecidability* of reachability analysis.

Bounding message queues gives us ...

- ☞ an easier problem:
Queue-bounded reachability analysis of message passing programs is often *decidable*.

Our Plan: Theory Investigation

Step 1: Define observation sequences

- ☞ $(O_k)_{k=0}^{\infty} := R_0, R_1, R_2, \dots$
⇒ R_k := the set of reachable states when *message queues* are bounded by k .
- ☞ Projecting R_k to a smaller finite domain ...

Our Plan: Theory Investigation

Step 2: Convergence detection

- ☞ Message queues are quite different from contexts.
⇒ How to proceed?

Our Plan: Empirical Evaluation

We will ...

- ☛ evaluate our approach on an extensive collection of P programs [PLDI'13].

Outline

- Overview of the Research
- A Paradigm: Observation Sequences
- Application: Context-UnBounded Analysis
- Proposed Work: Applications and Beyond
- Conclusion and Schedule

Conclusion

We target ...

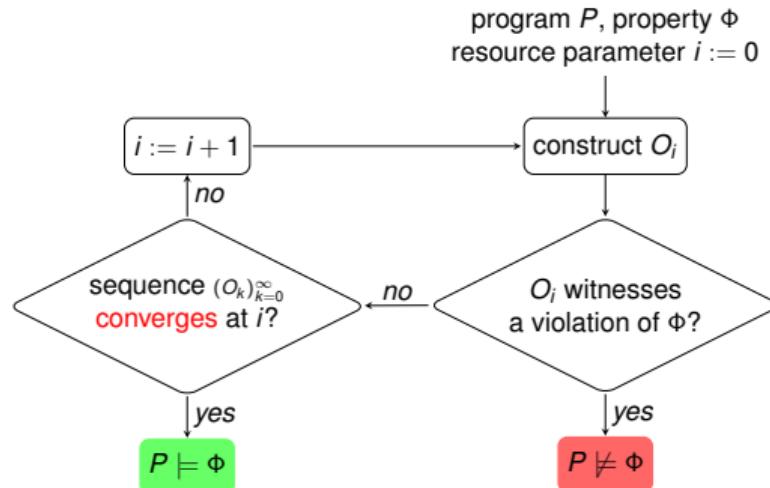
resource-parameterized programs.

We propose ...

a uniform paradigm of observation sequences.

Conclusion

The paradigm ...



can lift the bug-finding technique to resource-unbounded analysis.

Conclusion

We target ..

resource-parameterized programs.

We proposed ...

a uniform paradigm of observation sequences.

The paradigm can lift ...

the bug-finding technique to resource-unbounded analysis.

We applied it ...

to context-unbounded analysis.

We plan to ...

extend it to more applications.

Schedule

	October 2018	Proposal
October 2018 – February 2019		Queue-parameterized analysis
February 2019 – May 2019		More applications
May 2019 – July 2019		Improving the scalability of our tools; writing dissertation
	August 2019	Defense

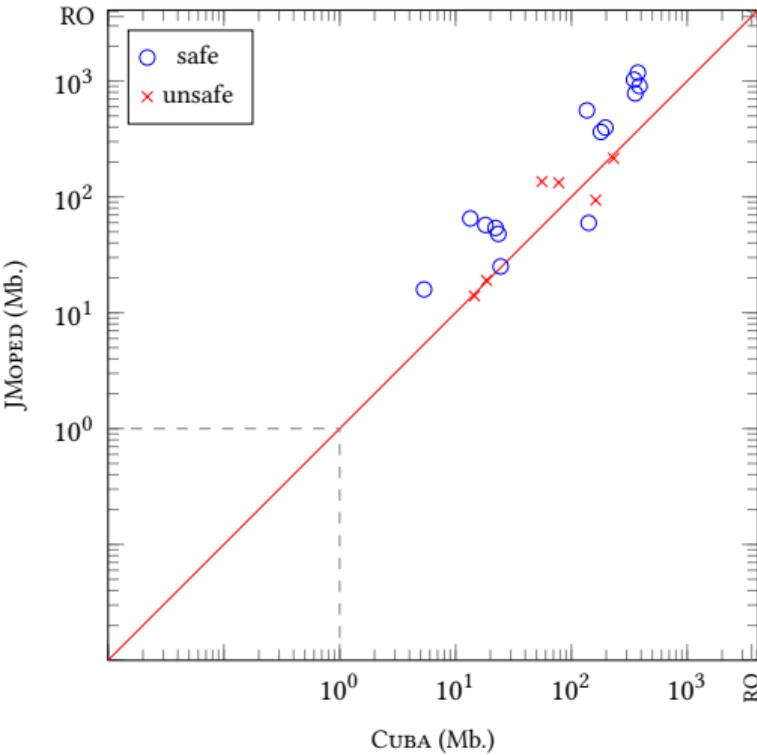
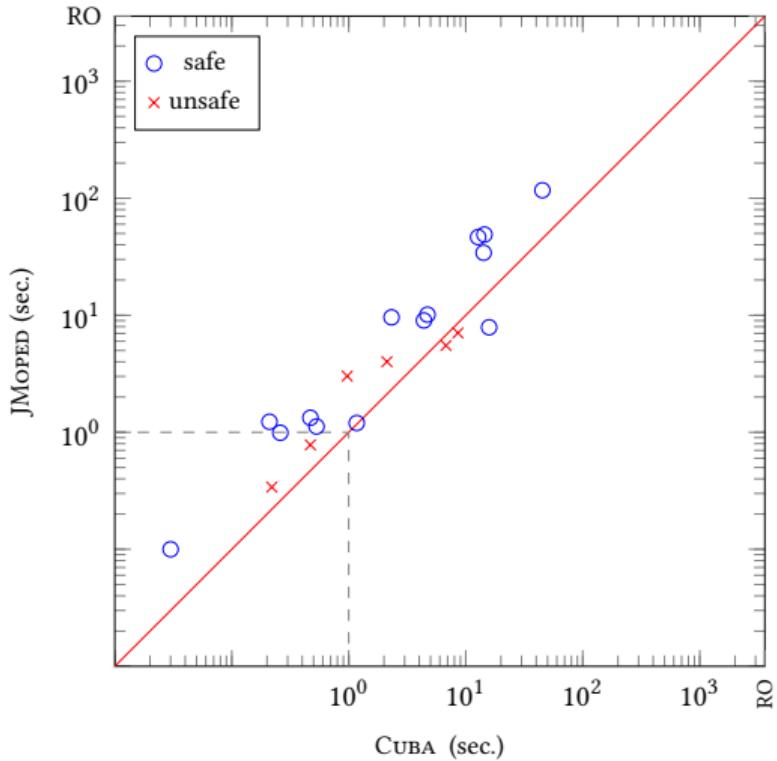
Thank You

References

-  Ramalingam, G.: "Context-sensitive synchronization-sensitive analysis is undecidable." In: ACM Trans. Program. Lang. Syst. (2000)
-  Qadeer, S., Rehof, J.: "Context-bounded model checking of concurrent software." In: TACAS. (2005)
-  Lu, S., Park, S., Seo, E., Zhou, Y.: "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics." In: ASPLOS. (2008)
-  Desai, A., Gupta, V., Jackson, E., Qadeer, S., Rajamani, S., Zufferey, D.: "P: Safe asynchronous event-driven programming." In: PLDI. (2013)
-  Liu, P., Wahl, T.: "CUBA: Interprocedural context-unbounded analysis of concurrent programs." In: PLDI. (2018)

Empirical Evaluation

vs JMOPED [SPIN'08]



Shared-Memory Access-Parameterized Analysis

Target is ...

shared-memory multi-threaded programs.

Resource is ...

the number of shared-memory accesses.

Observation is ...

*the set of **reachable** program states w.r.t. k accesses.*

Verification is ...

*reduced to the reachability of **bad** states.*

Motivation

Why shared-memory accesses?

Motivation

Reason 1

- ☞ Improper shared-memory accesses are a *root cause* of concurrency bugs
 - ⇒ E.g., race condition, data race etc.

Unfortunately,

- ☞ Analysis with unbounded accesses is challenging

Motivation

Reason 2

- ☞ Have an easier problem if bounding the number of accesses
- ⇒ Access-bounded reachability analysis of CPDS is *decidable* [proved]
- ⇒ Many bugs can be exposed with *few* shared-memory accesses [ASPLOS'08]

Our Plan: Theory Investigation

Step 1: Define observation sequences

- ☞ $(O_k)_{k=0}^{\infty} := R_0, R_1, R_2, \dots$
 ⇒ $R_k = \text{the set of states reachable within } k \text{ accesses.}$
- ☞ Projecting R_k to a finite domain ...

Our Plan: Theory Investigation

Step 2: Convergence detection

- ☞ *Based on similar convergence detection used in CUBA, and ...*

Our Plan: Theory Investigation

Step 3: A decidable subclass

- ☞ We will define a decidable subclass of shared-memory multi-threaded programs.
⇒ What does this mean?

Our Plan: Empirical Evaluation

We will

- ☛ evaluate our approach on an extensive collection of shared-memory multi-threaded programs.