Resource-Parameterized Program Analysis Using Observation Sequences

Peizun Liu (reworked by Artem, Janice, Julia)

Ph.D. Proposal

CCIS, Northeastern University

November 26, 2018

Outline

- **Research Problem**
- Observation Sequences Formally
- Application: Context-UnBounded Analysis
- More Applications (Further Work)
- Conclusion and Schedule

Resource-parameterized programs are ubiquitous

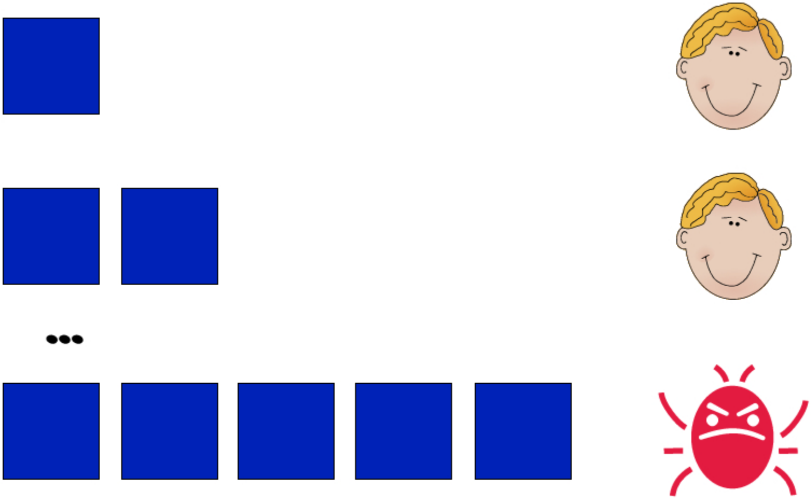
Applications	Resources
Web-servers	threads, message channels
GUI	event queues, threads
Modeling complex systems	computational nodes

Programs must be safe for any number of resources

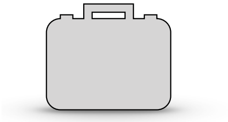
Examples of desired properties:

- 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

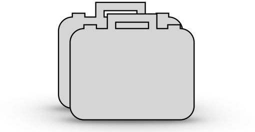
Analysis of resource-parameterized programs is challenging



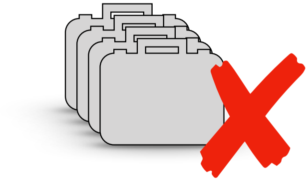
A natural and tested solution: resource-bounded analysis



start with ***one*** *resource*



iteratively increase resource

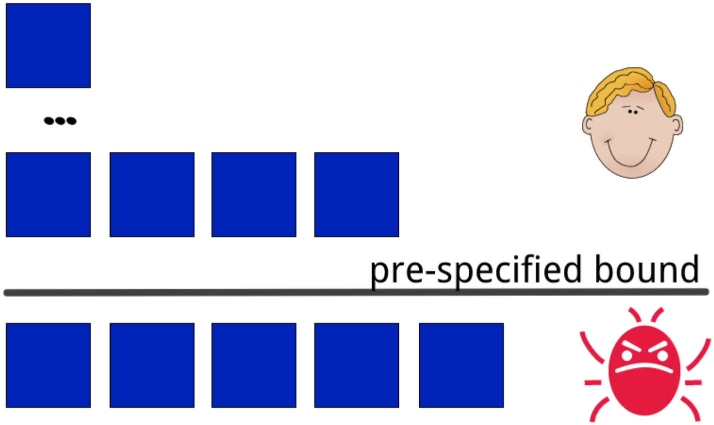


until a violation is found

until a ***pre-specified*** *upper bound*

Most bugs can be exposed with a **small** number of resources [ASPLOS'08]

Still, there might be bugs beyond the pre-specified bound



Can we guarantee unconditional safety?

Resource-unbounded analysis ensures safety

Goal

To provide a resource-unbounded analysis for safety verification.

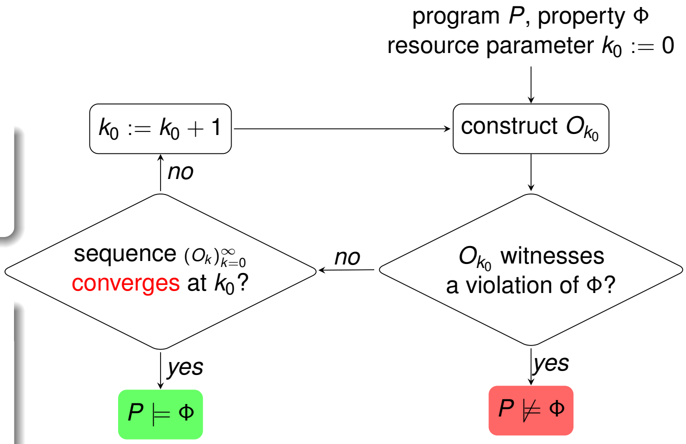
Approach

To lift the resource-bounded bug-finding technique.

Get rid of pre-specified bound via tracking program behaviour

Informally,
observation sequence (OS) is

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



Examples

- O_k := { *reachable program states within k threads*}
- O_k := { *reachable memory locations within k threads*}
- ...

Outline

- Research Problem
- **Observation Sequences Formally**
- Application: Context-UnBounded Analysis
- More Applications (Further Work)
- Conclusion and Schedule

OS-based analysis has to be decidable

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

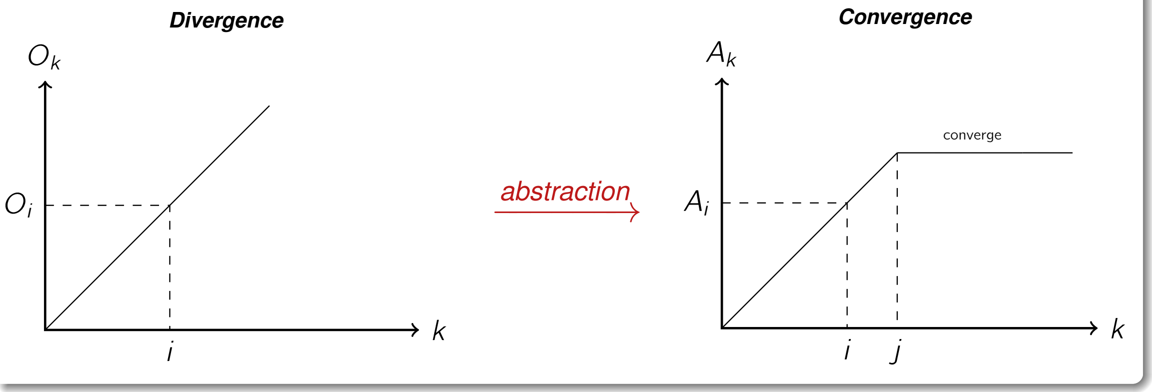
- $\forall\, k\colon\ O_k \subseteq O_{k+1}$ — monotonicity;
- $\forall\, k\colon\ O_k$ is computable;
- $\forall\, k\colon\ O_k \models \Phi$ is decidable, where Φ is a property of interest.

Problem

How to make $(O_k)_{k=0}^{\infty}$ **converge**?

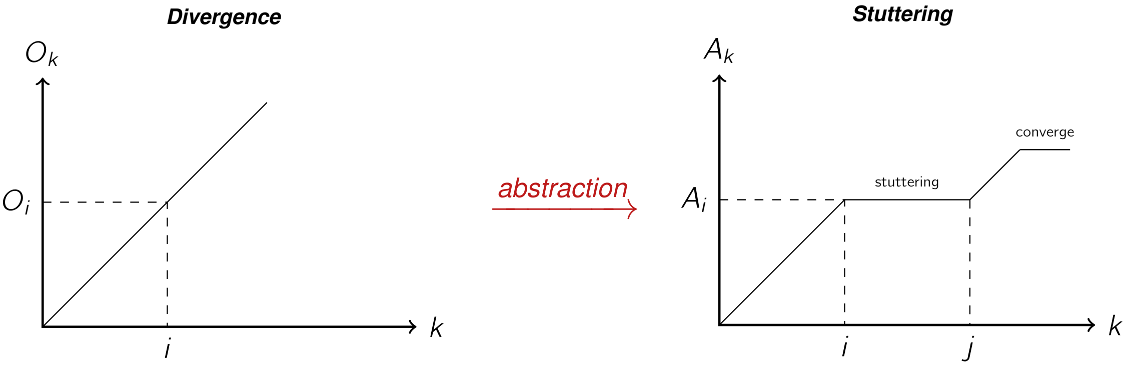
Abstraction of OS converges

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



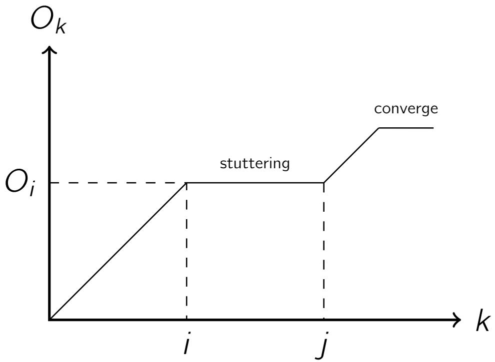
$$A_k := abstraction(O_k)$$

Abstraction of OS might stutter before converging

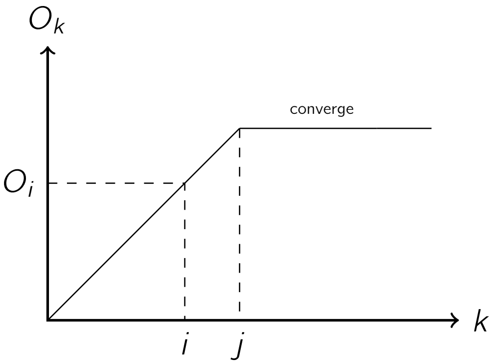


We need to distinguish stuttering from converging

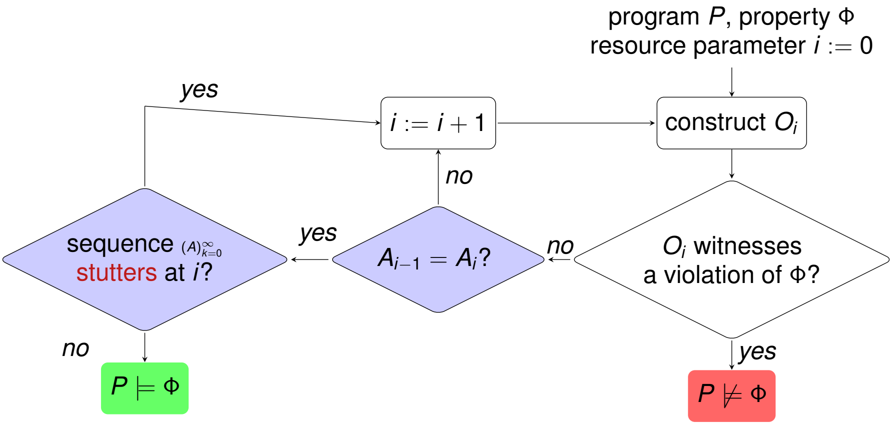
Stuttering



Stutter-free



Abstraction converges and survives stuttering!



Outline

- Research Problem
- Observation Sequences Formally
- **Application: Context-UnBounded Analysis**
- More Applications (Further Work)
- 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.

CPDS can be used to abstract program states

Concurrent Pushdown System (CPDS)

A CPDS P^n is a collection of n PDS $P_i = (Q, \Sigma_i, \Delta_i, q^I)$, $1 \leq i \leq n$, 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 $\Sigma_i^{\leq 2} = \{\omega \in \Sigma_i^* \mid |\omega| \leq 2\}$;
- $q^I \in Q$ is the initial shared state.

CPDS states change on push, pop, overwrite

$$\dfrac{(q,a_1)\to(p,\sigma_1\sigma_2)\in\Delta \qquad q \times \begin{array}{|c|}\hline a_1\\\hline a_2\\\hline \vdots\\\hline a_x\\\hline\end{array}}{\cdots \quad \to \qquad p \times \begin{array}{|c|}\hline \sigma_1\\\hline \sigma_2\\\hline a_2\\\hline \vdots\\\hline a_x\\\hline\end{array}}\ \mathit{push}$$

$$\dfrac{(q,a_1)\to(p,\varepsilon)\in\Delta \qquad q \times \begin{array}{|c|}\hline a_1\\\hline a_2\\\hline \vdots\\\hline a_x\\\hline\end{array}}{\cdots \quad \to \qquad p \times \begin{array}{|c|}\hline a_2\\\hline \vdots\\\hline a_x\\\hline\end{array}}\ \mathit{pop}$$

$$\dfrac{(q,a_1)\to(p,\sigma)\in\Delta \qquad q \times \begin{array}{|c|}\hline a_1\\\hline a_2\\\hline \vdots\\\hline a_x\\\hline\end{array}}{\cdots \quad \to \qquad p \times \begin{array}{|c|}\hline \sigma\\\hline a_2\\\hline \vdots\\\hline a_x\\\hline\end{array}}\ \mathit{overwrite}$$

Program executions are represented with contexts

Thread A : $\underbrace{\bigcirc \to \ldots \to \bigcirc \to}_{\textit{context}}$ $\qquad\qquad\qquad\qquad$ $\underbrace{\bigcirc \to \ldots \to \bigcirc \to}_{\textit{context}}$

Thread B : $\qquad\qquad\qquad\qquad$ $\underbrace{\bigcirc \to \ldots \to \bigcirc \to}_{\textit{context}}$

Thread C : $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\underbrace{\bigcirc \to \ldots \to \bigcirc \to}_{\textit{context}}$

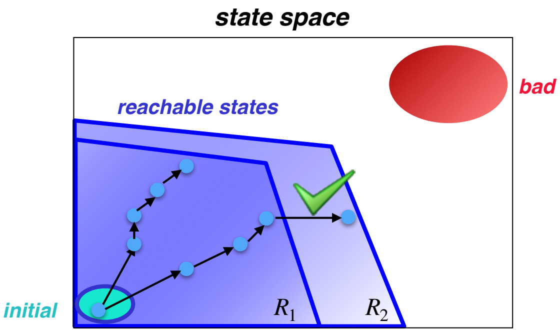
Resources are contexts

Reachable states must not be bad

A state is reachable if

$$\underbrace{\bigcirc}_{\textit{initial}} \longrightarrow \bigcirc \longrightarrow \bigcirc \longrightarrow \bigcirc \longrightarrow \cdots \longrightarrow \bigcirc \longrightarrow \underbrace{\bullet}_{\textit{target}}$$

Safety as reachability



Infinite observations can be reduced to decidable

- Observation R_k – the set of states reachable within k contexts.
- R_k can be infinite [CAV'00], and reachability of CPDS is undecidable [TOPLAS'00].

[TACAS'05]

- R_k can be finitely represented.
- Context-bounded reachability of CPDS is decidable.

We can apply CUBA to sequences of global states

Global state

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

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

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

An example: observation sequences might diverge

Shared states:

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

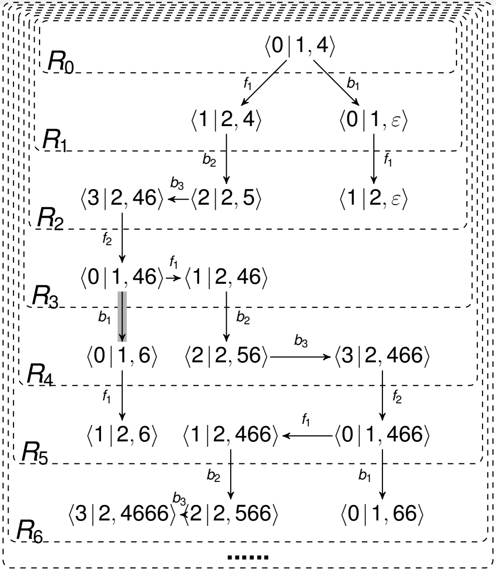
Thread 1:

$$\Sigma_1 = \{ 1, 2 \}$$
$$\Delta_1 = \{ f_1 : (0,1) \to (1,2)\ ,\ f_2 : (3,2) \to (0,1) \}$$

Thread 2:

$$\Sigma_2 = \{ 4, 5, 6 \}$$
$$\Delta_2 = \{ b_1 : (0,4) \to (0,\varepsilon)\ ,\ b_2 : (1,4) \to (2,5)\ ,\ b_3 : (2,5) \to (3,46) \}$$

$$q^I = 0$$



An example: observation sequences might diverge

Shared states:

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

Thread 1:

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

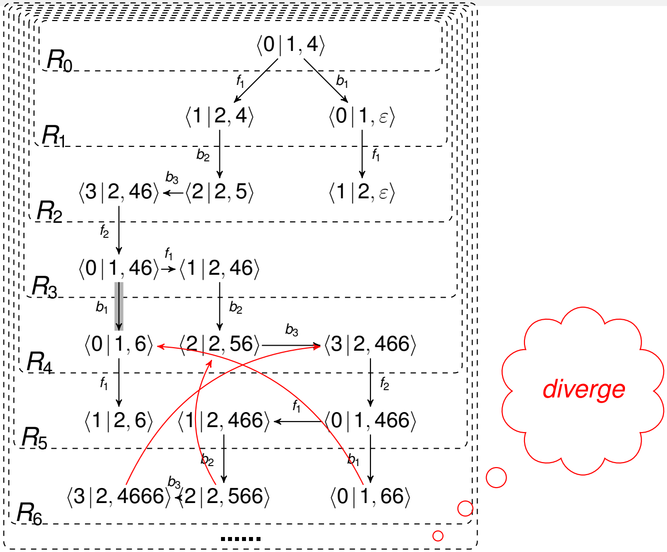
$$\Delta_1 = \{ f_1 : (0,1) \rightarrow (1,2) ,\; f_2 : (3,2) \rightarrow (0,1) \}$$

Thread 2:

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

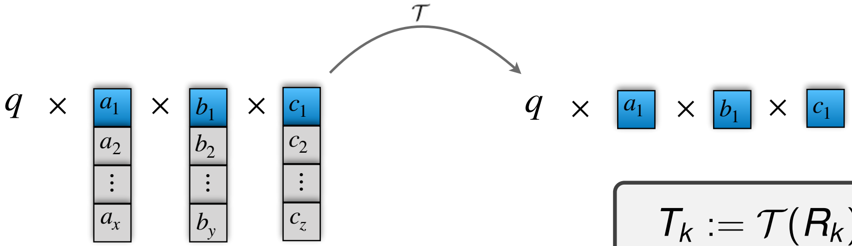
$$\Delta_2 = \{ b_1 : (0,4) \rightarrow (0,\varepsilon) ,\; b_2 : (1,4) \rightarrow (2,5) ,\; b_3 : (2,5) \rightarrow (3,46) \}$$

$$q^I = 0$$



Solution: replace *global state* with *visible state*

Project global states to a finite domain by cutting off tails of stacks: obtain visible states T_i.

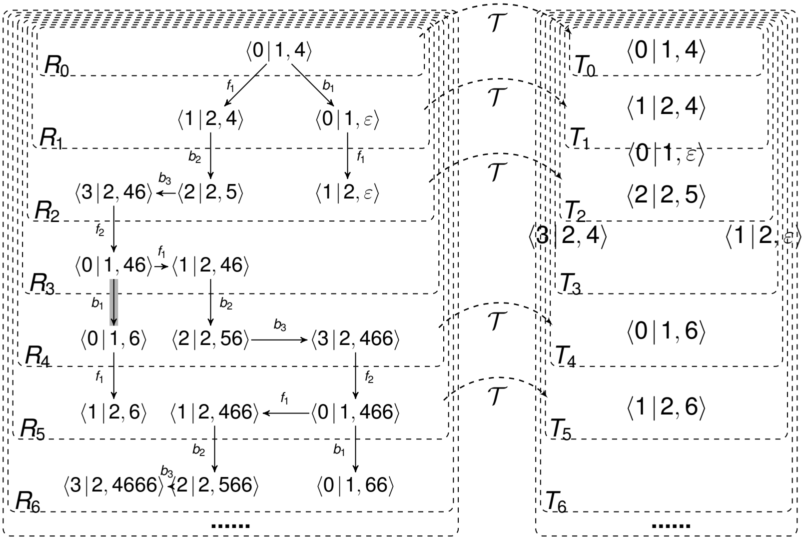


$$\boxed{T_k := \mathcal{T}(R_k)}$$

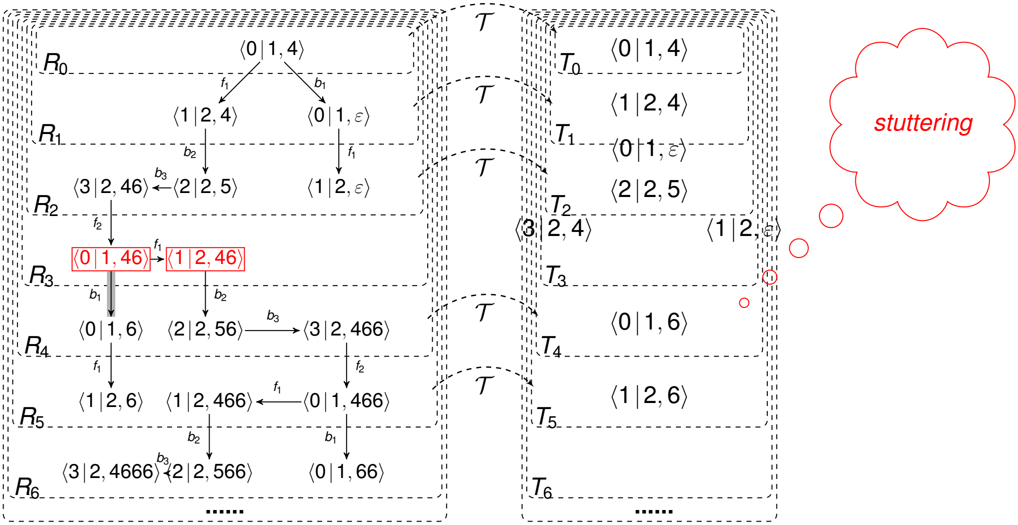
Visible states

- ☞ suffice to express many safety properties, e.g. various assertions, data race, race condition, etc.
- ☞ $(T_k)_{k=0}^{\infty}$ is guaranteed to converge.

Do visible states help in the example?



The problem is stuttering



We can use observation on generators

Shared states:

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

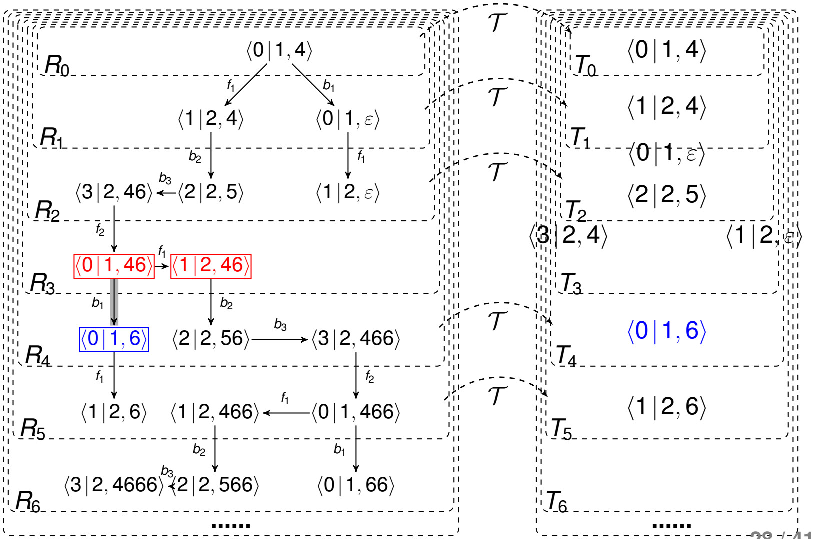
Thread 1:

$$\Sigma_1 = \{ 1, 2 \}$$
$$\Delta_1 = \{ f_1 : (0,1) \rightarrow (1,2)\ ,\ f_2 : (3,2) \rightarrow (0,1) \}$$

Thread 2:

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

$$q^I = 0$$



Generators can be approximated...

Properties [PLDI’18]

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

Generators can be approximated...

Properties [PLDI'18]

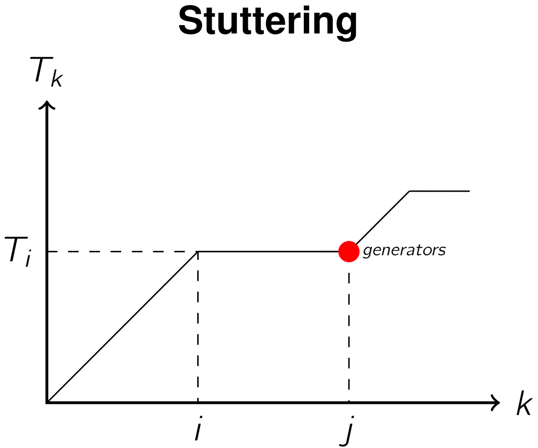
- ☞ generators are of special form $\Rightarrow$ can be statically approximated
- ☞ all generators have been reached $\Rightarrow$ OS converges
- ☞ any overapproximation suffices

Generators can be approximated...

Properties [PLDI'18]

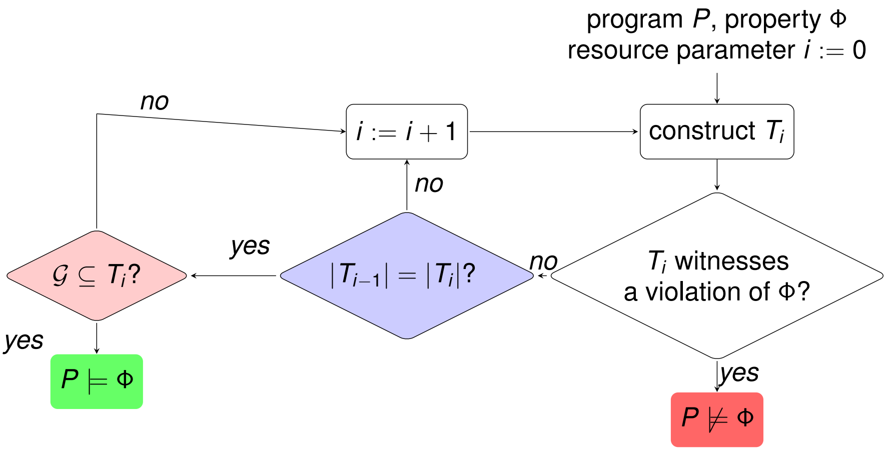
- ☞ generators are of special form $\Rightarrow$ can be statically approximated
- ☞ all generators have been reached $\Rightarrow$ OS converges
- ☞ any overapproximation suffices

Generators help detect stuttering



☞ All generators have been reached
$\Rightarrow$ OS converges

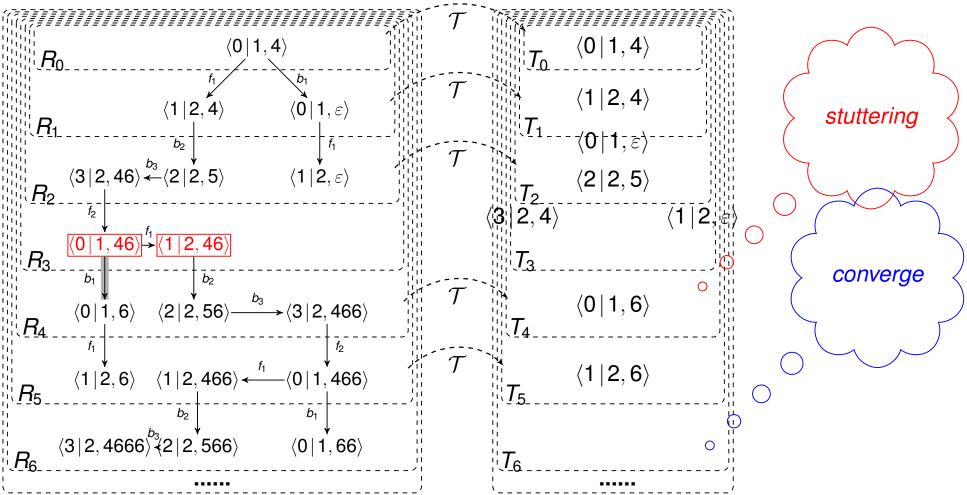
Algorithm relies on generators



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

Example revisited: convergence achieved

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



Performance evaluation shows practical bound on k

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

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

Outline

- Research Problem
- Observation Sequences Formally
- Application: Context-UnBounded Analysis
- **More Applications (Further Work)**
- Conclusion and Schedule

Our technique applies to message passing programs

Step 1: Message queues are challenging:

- generate infinite state space;
- cause undecidability of reachabililty analysis.

Step 2: Bounding message queues are easier

Queue-bounded reachability analysis of message passing programs is often decidable.

Step 3: Our framework is applicable

Target	message-passing programs
Resource	the size of message queues
Observation	the set of reachable program states w.r.t. the size of queue within k.
Analysis	to check the reachability of bad states.

Our Plan: Theory and practise together

First, theory

Step 1: Define observation sequences

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

Step 2: Convergence detection

- Problem: message queues are quite different from contexts.

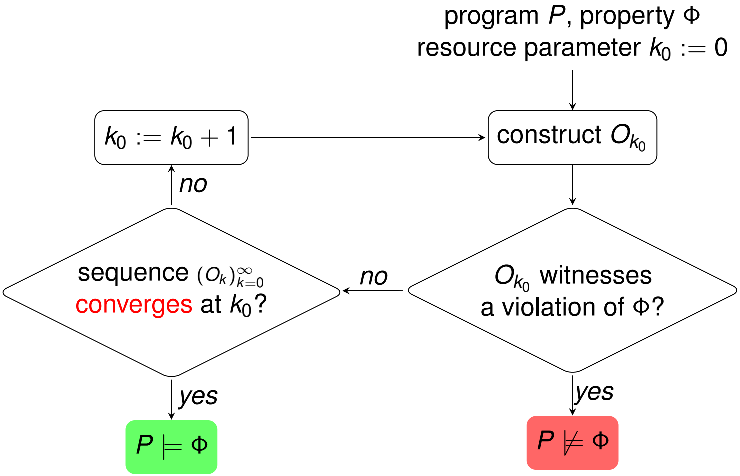
Second, empirical evaluation

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

Outline

- Research Problem
- Observation Sequences Formally
- Application: Context-UnBounded Analysis
- More Applications (Further Work)
- **Conclusion and Schedule**

We proposed a uniform technique of observation sequences for *unbounded* analysis of resource-parameterized programs

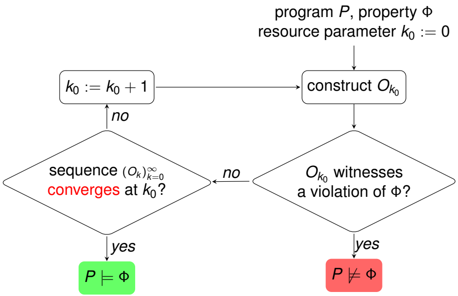


The technique can be applied to different problems

- Context-Unbounded Analysis
- Queue-Parameterized Analysis (work-in-progress)
- ...

We plan future work as follows

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



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)