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.*

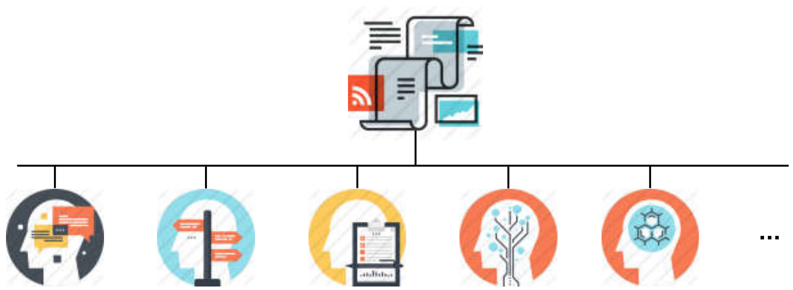
"Resources" could mean:

threads *context switches* *memory writes* *executions* *message channels* ...

Motivation

Reason 1

☞ *Resource-parameterized programs are ubiquitous.*



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 2

☞ *Ensuring their safety is desirable and significant.*

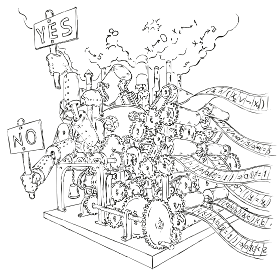
Motivation

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

Tested empirically [ASPLOS'08]

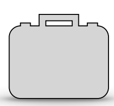
☞ *Most bugs can be exposed with a small number of resources.*

We thus have

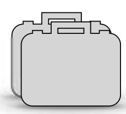
a bug-finding technique

A Sidestep

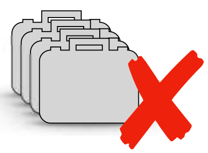
Resource-bounded analysis



start with one resource



iteratively increase resource



until a violation is found

until a pre-specified upper bound

A Sidestep

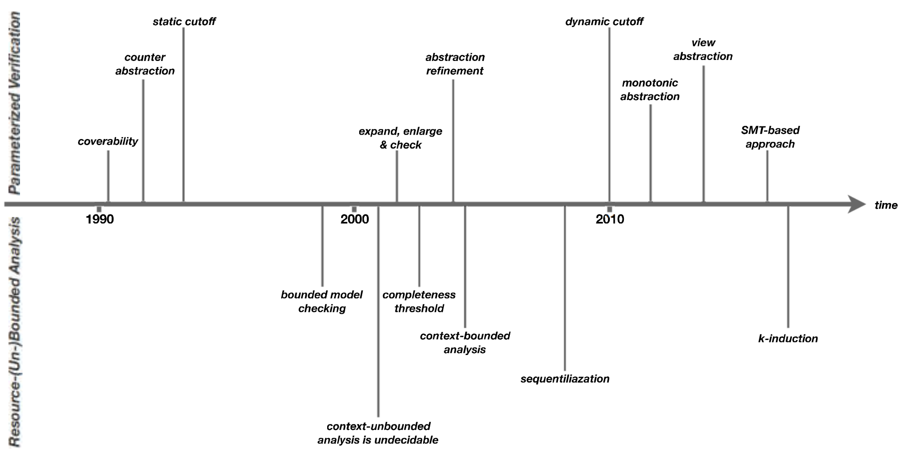
Still, uncertainty remains ...

☞ *beyond the pre-specified bound.*

Beyond the Sidestep

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

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) ...

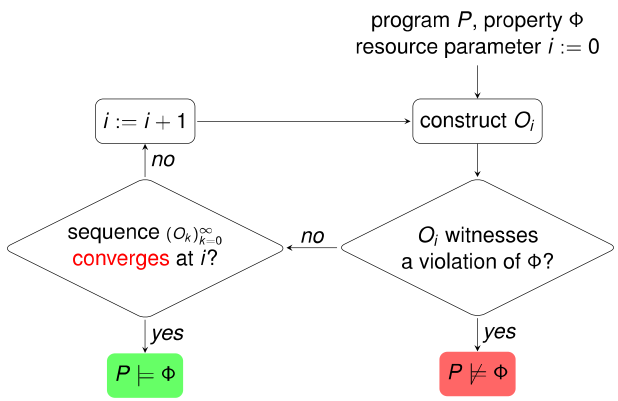
Informally, a sequence of program behaviors O_k observed within k instances of resource.

Examples

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

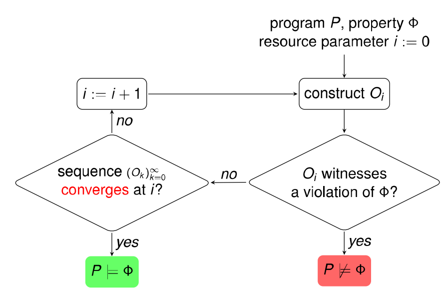
Observation Sequences

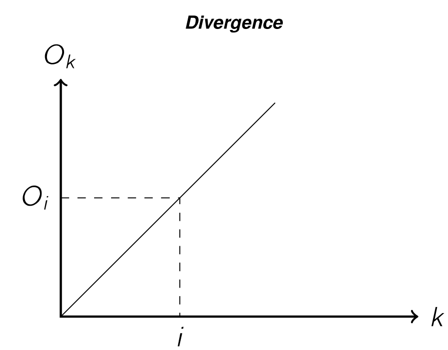
Definition

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

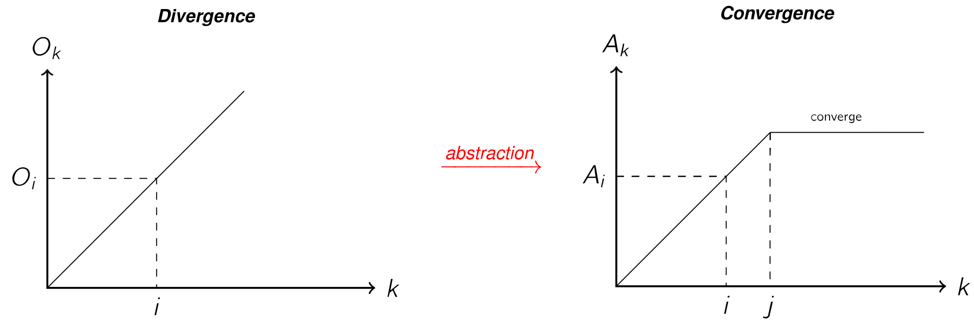
- *for all k, $O_k \subseteq O_{k+1}$, that is monotonicity.*
- *for all k, O_k is computable.*
- *for all k, $O_k \models \Phi$ is decidable, where Φ is a property of interest.*

Convergence Detection is Challenging





Abstraction

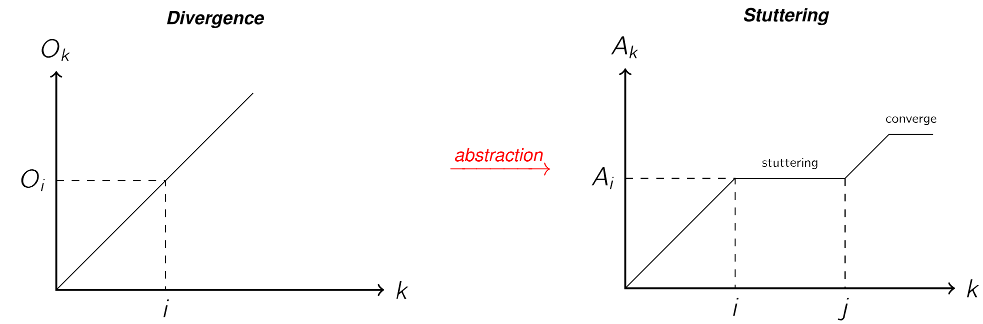


$$A_k := abstraction(O_k)$$

Convergence Property

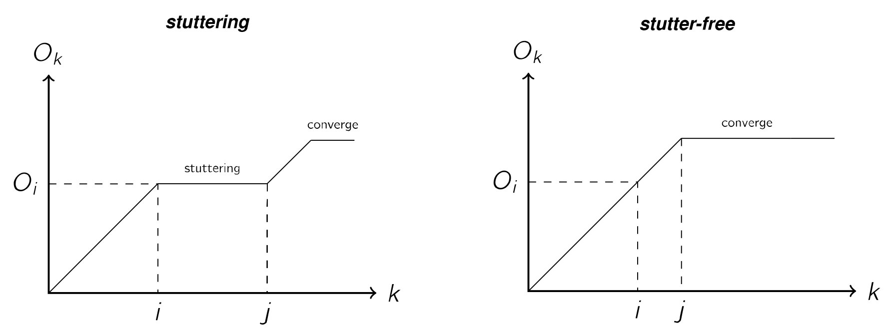
An OS $(O_k)_{k=0}^{\infty}$ over a finite domain always converges.

Stuttering

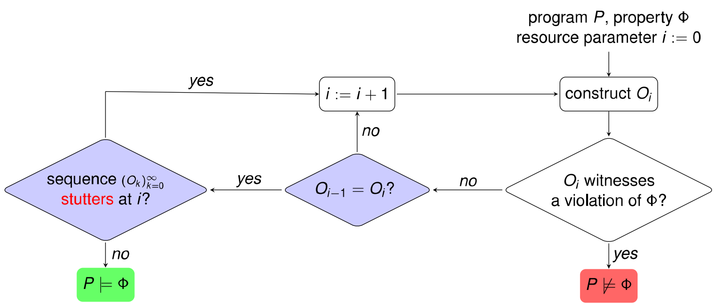


$$A_k := abstraction(O_k)$$

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 $P_i = (Q, \Sigma_i, \Delta_i, q^I)$, $1 \leq i \leq n$, 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.*

Operational Model

Semantics

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

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

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

Operational Model

Executions

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

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

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

resource := contexts

Operational Model

States

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

For instance

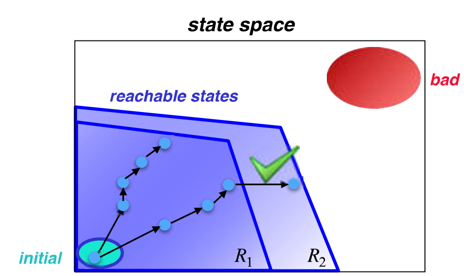
$$q \times \begin{array}{|c|}\hline a_1\\\hline a_2\\\hline \vdots\\\hline a_x\\\hline\end{array} \times \begin{array}{|c|}\hline b_1\\\hline b_2\\\hline \vdots\\\hline b_y\\\hline\end{array} \times \begin{array}{|c|}\hline c_1\\\hline c_2\\\hline \vdots\\\hline c_z\\\hline\end{array}$$

Problem Statement

A state is reachable if ...

$$\underbrace{\bigcirc}_{initial} \longrightarrow \bigcirc \longrightarrow \bigcirc \longrightarrow \bigcirc \longrightarrow \ldots \longrightarrow \bigcirc \longrightarrow \underbrace{\bullet}_{target}$$

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

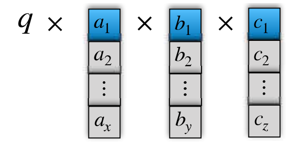
☞ **Reachability of CPDS is undecidable [TOPLAS'00].**

But

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

CUBA using Observation Sequences of Global States

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



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

Example

Shared states:

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

Thread 1:

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

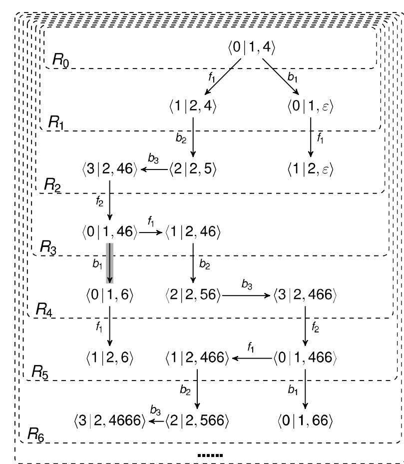
$\Delta_1 = \{ f_1 : (0, 1) \to (1, 2)\ , \ f_2 : (3, 2) \to (0, 1) \}$

Thread 2:

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

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

$q^I = 0$



Example

Shared states:

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

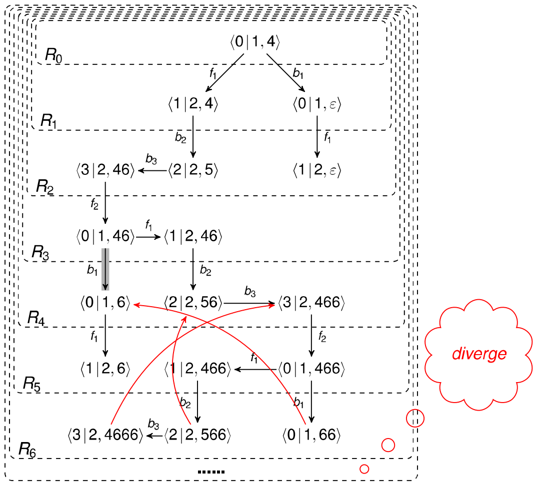
Thread 1:

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

Thread 2:

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

$q^I = 0$



26

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.

27

How to Proceed?

Give up?

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

Convergence Property

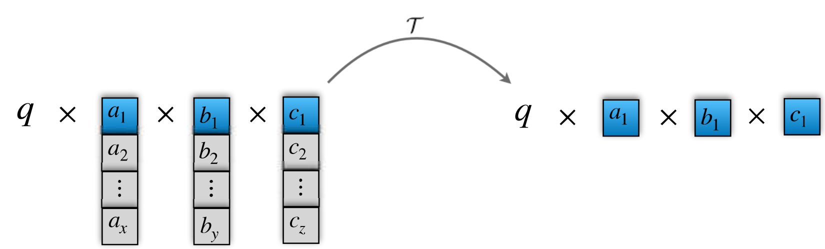
An OS $(O_k)_{k=0}^{\infty}$ over a finite domain always converges.

27

CUBA using Observation Sequences of Visible State

Project global states to a finite domain ...

☞ *by cutting off tails of stacks.*



28

CUBA using Observation Sequences of Visible State

Project global states to a finite domain ...

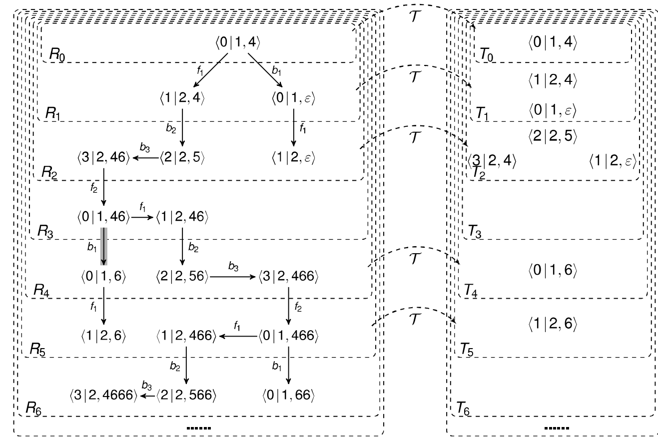
$$q \times a_1 \times b_1 \times c_1$$

Visible states

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

Example Revisited

Can we answer the convergence of visible state sequence easily?



CUBA using Observation Sequences of Visible State

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

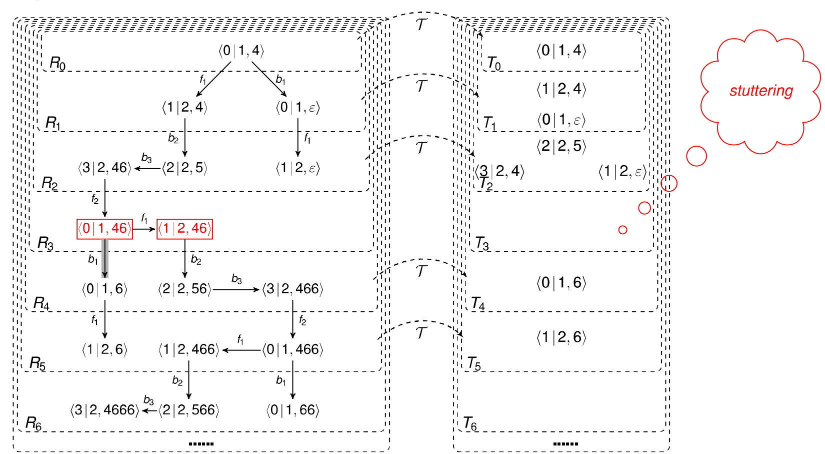
$$q \times a_1 \times b_1 \times c_1$$

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

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

Example Revisited

Not really ...



Example Revisited

But, an observation on generators

Shared states:

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

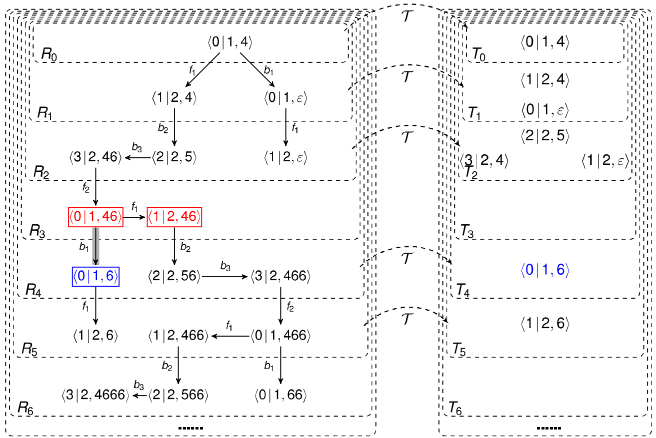
Thread 1:

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

Thread 2:

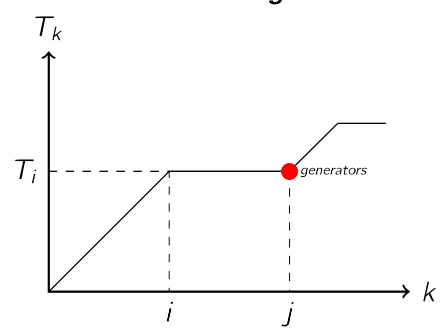
$\Sigma_2 = \{ 4, 5, 6 \}$
$\Delta_2 = \{ b_1 : (0,4) \to (0,\varepsilon)\ ,\ b_2 : (1,4) \to (2,5)\ ,\ b_3 : (2,5) \to (3,46) \}$

$q^I = 0$



Stuttering Detection

Stuttering

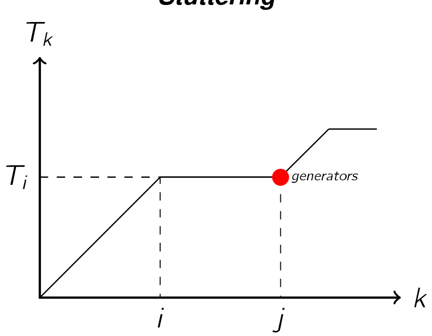


Properties [PLDI'18]

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

Stuttering Detection

Stuttering

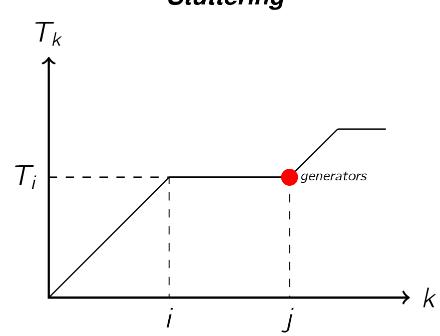


Properties [PLDI'18]

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

Stuttering Detection

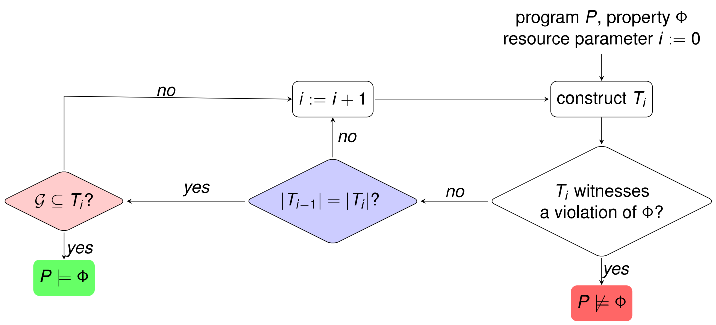
Stuttering



Properties [PLDI'18]

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

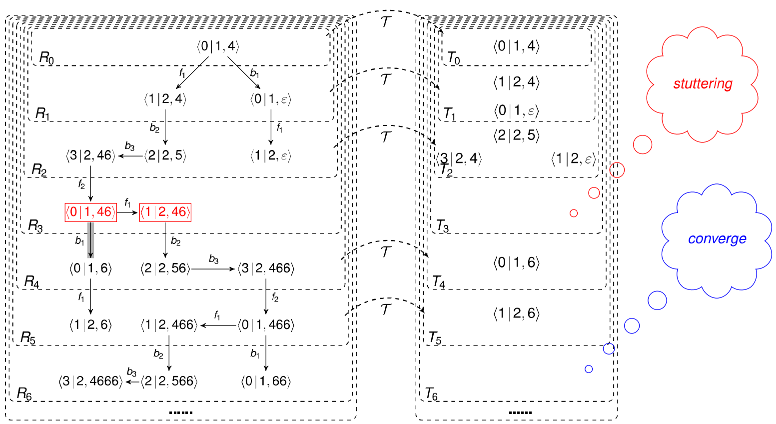
Algorithm



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

Example Revisited

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



Empirical Evaluation

Performance

ID/Program	Prog. Features: *Thread*	Prog. Features: *Safe?*	$(T_k)_{k=0}^{\infty}$: k_{max}	$(T_k)_{k=0}^{\infty}$: *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: *Thread*	Prog. Features: *Safe?*	$(T_k)_{k=0}^{\infty}$: k_{max}	$(T_k)_{k=0}^{\infty}$: *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

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 reachabililty 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

- $(O_k)_{k=0}^{\infty} := R_0, R_1, R_2, \ldots$.
  $\Rightarrow$ *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.*
  $\Rightarrow$ *How to proceed?*

Our Plan: Empirical Evaluation

We will ...

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

Conclusion

We target ...

resource-parameterized programs.

We propose ...

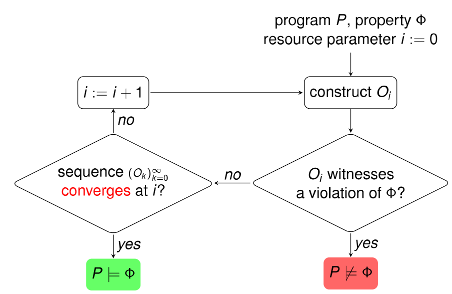
a uniform paradigm of observation sequences.

Outline

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

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

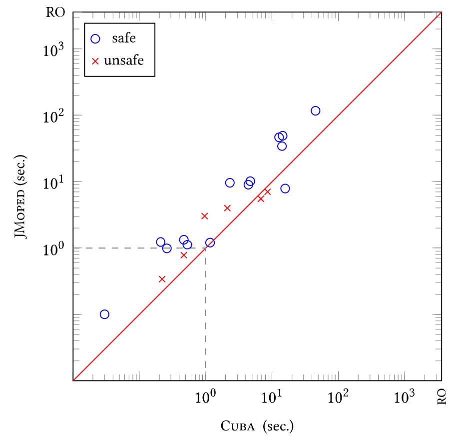
Thank You

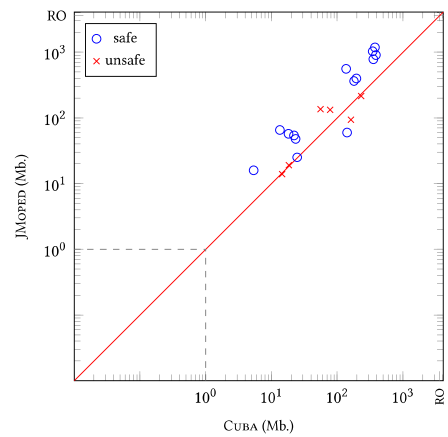
References

- 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

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

Why shared-memory accesses?

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

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

Our Plan: Theory Investigation

Step 3: A decidable subclass

- *We will define a decidable subclass of shared-memory multi-threaded programs.*
  $\Rightarrow$ *What does this mean?*

Our Plan: Theory Investigation

Step 2: Convergence detection

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

Our Plan: Empirical Evaluation

We will

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