Resource-Parameterized Program Analysis using Observation Sequences

Peizun Liu

CCIS, Northeastern University

27th November, 2018

Outline

- Overview of the Research
- A Paradigm: Observation Sequences
- Application 1: Context-UnBounded Analysis
- Application 2: Queue-Parameterized Analysis
- Future Work and Conclusion

Outline

- **Overview of the Research**
- A Paradigm: Observation Sequences
- Application 1: Context-UnBounded Analysis
- Application 2: Queue-Parameterized Analysis
- Future Work and Conclusion

Many modern programs are parameterized over a variable number of discrete resources

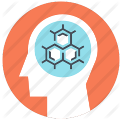
Resources are:

threads

context switches

memory writes

executions



message channels

...

Analysis is hard when programs use an unspecified number of resources

How many...

threads?

context switches?

memory writes?

possible executions?

message channels?

...

...

The safety of such programs for an unspecified number of resource instances is paramount

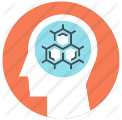
Examples of safety:

threads: deadlock-freedom

context switches: fairness

memory writes: no illegal writes

possible executions: no failing executions



message channels: responsiveness

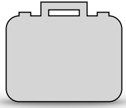
...

...

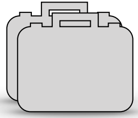
Resource-parameterized programs are ubiquitous

Ensuring their safety is desirable and significant

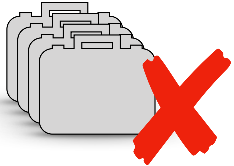
Existing technique: resource-bounded analysis



start with one resource



iteratively increase resource



until a violation is found

until a pre-specified upper bound

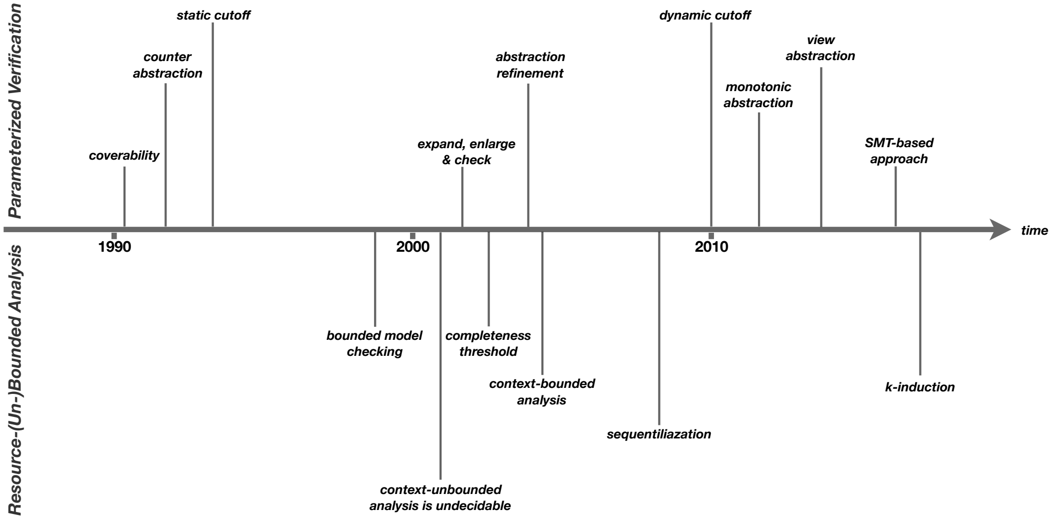
Tested empirically [ASPLOS'08]

Uncertainty still remains beyond the pre-specified bound

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

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

Status of Research



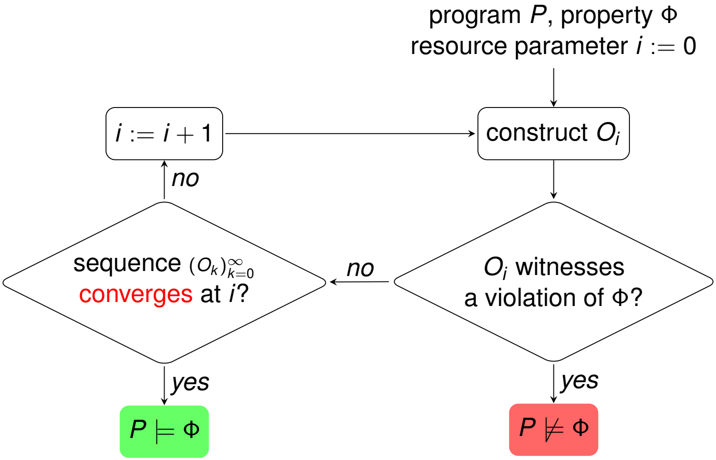
Our goal is to provide a resource-agnostic paradigm, adapting resource-bounded techniques to work in unbounded scenarios

Informally, an observation sequence is a sequence of program behaviors O_k observed within k instances of a resource

Examples

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

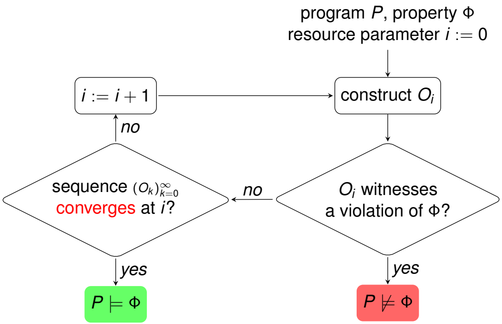
Overview of Scheme

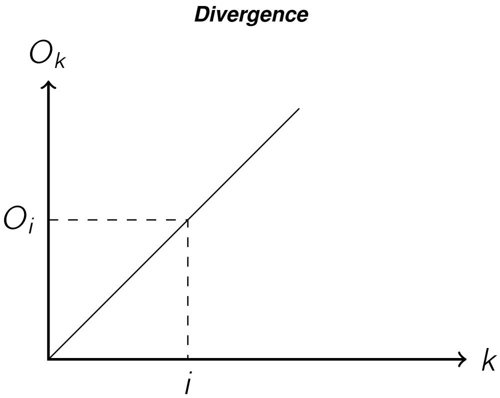


Outline

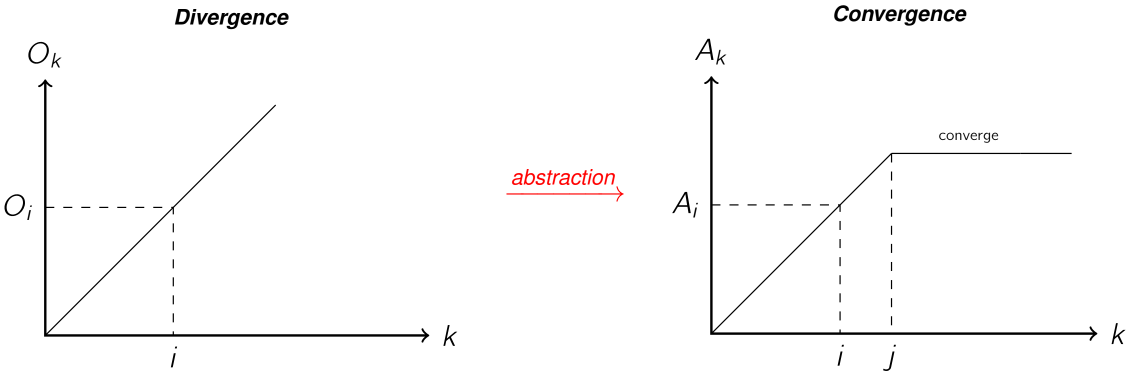
- Overview of the Research
- **A Paradigm: Observation Sequences**
- Application 1: Context-UnBounded Analysis
- Application 2: Queue-Parameterized Analysis
- Future Work and Conclusion

Convergence Detection is Challenging



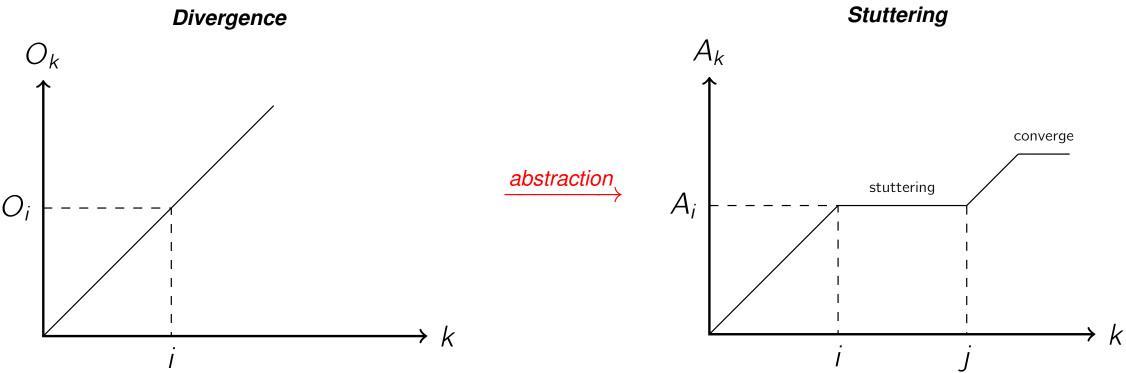


There is no guarantee that an observation sequence will converge, so we add a layer of abstraction



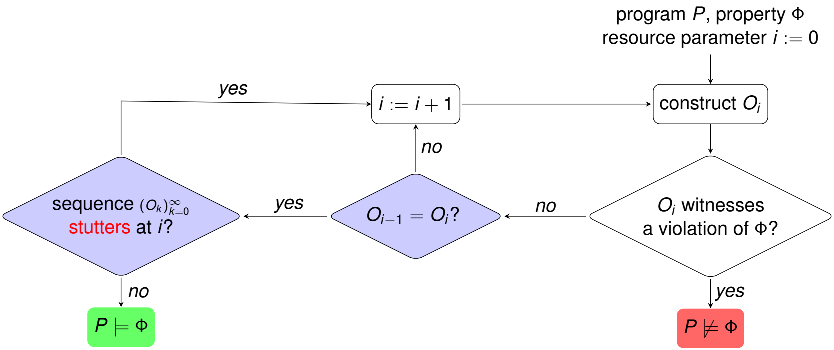
$$A_k := abstraction(O_k)$$

The abstracted sequence isn’t necessarily strictly monotonic, which introduces *stuttering*



$$A_k := abstraction(O_k)$$

We need a refined scheme to deal with stuttering, focused on stutter detection



Outline

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 ...

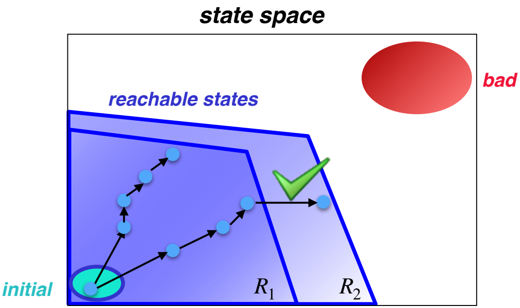
to check the reachability of bad states.

Resource: contexts in the executions

$$
\begin{array}{lllll}
\textit{Thread } A: & \underbrace{\bigcirc \to \ldots \to \bigcirc \to}_{\textit{context}} & & \underbrace{\bigcirc \to \ldots \to \bigcirc \to}_{\textit{context}} & \\
\textit{Thread } B: & & \underbrace{\bigcirc \to \ldots \to \bigcirc \to}_{\textit{context}} & & \\
\textit{Thread } C: & & & & \underbrace{\bigcirc \to \ldots \to \bigcirc \to}_{\textit{context}}
\end{array}
$$

resource := *contexts*

In this application, we define safety as reachability



We are interested in the set R_k of states reachable within k contexts

Our contributions to this area

- CUBA: Interprocedural Context-Unbounded Analysis of Concurrent Programs. In *PLDI, 2018*.
- IJIT: An API for Boolean Program Analysis with Just-in-Time Translation. In *SEFM, 2017*.
- Concolic Unbounded-Thread Reachability via Loop Summaries”. In *ICFEM, 2016*.

Outline

- Overview of the Research
- A Paradigm: Observation Sequences
- Application 1: Context-UnBounded Analysis
- **Application 2: Queue-Parameterized Analysis**
- Future Work and Conclusion

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.

An Application of our Approach: Method Queues

- ☞ *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 contributions to this area

- P: Towards Verified-Safe Asynchronous Event-Driven Programming. In *MAKEUP, 2018*.
- Infinite-State Backward Exploration of Boolean Broadcast Programs. In *FMCAD, 2014*.

Outline

- Overview of the Research
- A Paradigm: Observation Sequences
- Application 1: Context-UnBounded Analysis
- Application 2: Queue-Parameterized Analysis
- **Future Work and Conclusion**

We can analyze shared-memory access-parameterized programs

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.

Analysis is ...

reduced to the reachability of bad states.

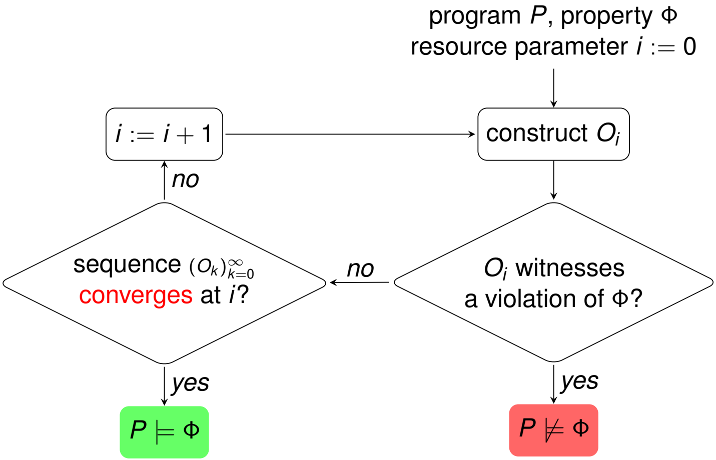
Shared-memory accesses are another interesting application for our method

- ☞ *improper shared-memory accesses are a root cause of concurrency bugs*
  $\Rightarrow$ *E.g., race condition, data race etc.*
- ☞ *analysis with unbounded accesses is challenging*

Bounding shared-memory accesses gives us ...

- ☞ *an easier problem:*
  Access-bounded reachability analysis of CPDS is decidable [proved]

We propose a general technique for resource-parameterized program analysis



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)