FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

for(syte)
Formal Methods in Systems Engineering

Rely-Guarantee Reasoning for Automated Bound Analysis of Concurrent Shared-Memory Programs [3]

Thomas Pani[*,†], Georg Weissenbacher[*], Florian Zuleger[*]

[*] TU Wien, [†] Wolfgang Pauli Institute

Bound analysis

is a **static** program analysis that determines **upper bounds** on a program's **resource usage**.

- Many approaches for **sequential, imperative programs** [GZ'10, ADFG'10, AAGP'11, FH'14, BEFFG'16, CHRS'17, SZV'17, ...].
- We **lift bound analysis** to **concurrent (parameterized) shared-memory programs**.

Resource usage

- *Cost model* assigns each instruction a cost (CPU time, memory, network, ...).
- Here: each control-flow edge has constant cost (back edges: runtime complexity).

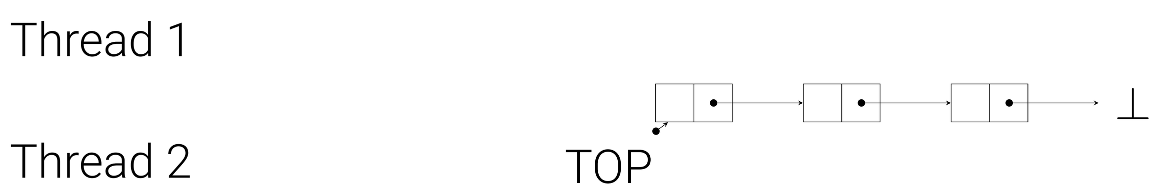
Non-blocking algorithms

Use strong synchronization primitives like compare-and-swap (CAS) to circumvent shortcomings of lock-based concurrency: deadlocks, priority inversion, ...Prominently used in **lock-free data structures**:

- Treiber's stack, Michael-Scott queue (Java: `ConcurrentLinkedQueue`), DGLM queue, ...

Treiber's stack: `push()`

```
n := new Node;
do { o := TOP; n.next = o; }
while (!CAS(TOP, o, n))
```



Implementation pattern for non-blocking algorithms

1. Read the global state .
2. Locally prepare update.
3. Synchronize on global state to make local update globally visible:
   (a) If the global state has not changed since (1), apply the update.
   (b) Otherwise, repeat from (1).

Runtime complexity

- Depends on interference by other threads, i.e., the number of concurrently running threads N.

Analysis of non-blocking algorithms

Manual liveness / bound analysis is hard:

1. Amount of interference affects a thread's complexity:
   - to infer resource bounds on a **single thread**: reason about **unbounded number of threads** N.
2. Fine-grained concurrency: interference may occur anywhere.

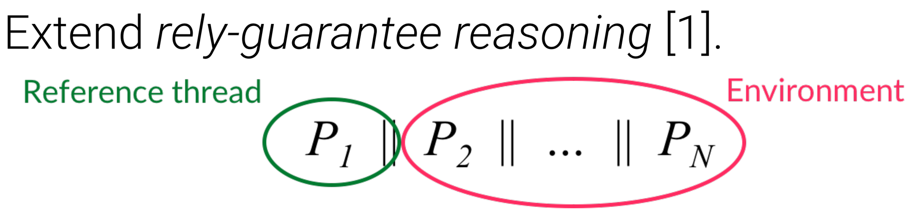
Problem statement

Given an abstract data type with operations `op1`, ..., `opM`:

- Build a general data type client $P = \texttt{op1}() \,[]\, \dots \,[]\, \texttt{opM}()$.
- Compose N concurrent copies of P: $P_1 \parallel \cdots \parallel P_N$.
- For all $N > 0$, compute bounds on P_1 when executed concurrently with $P_2 \parallel \cdots \parallel P_N$.

Unbounded number of threads? Abstract!

Extend *rely-guarantee reasoning* [1].

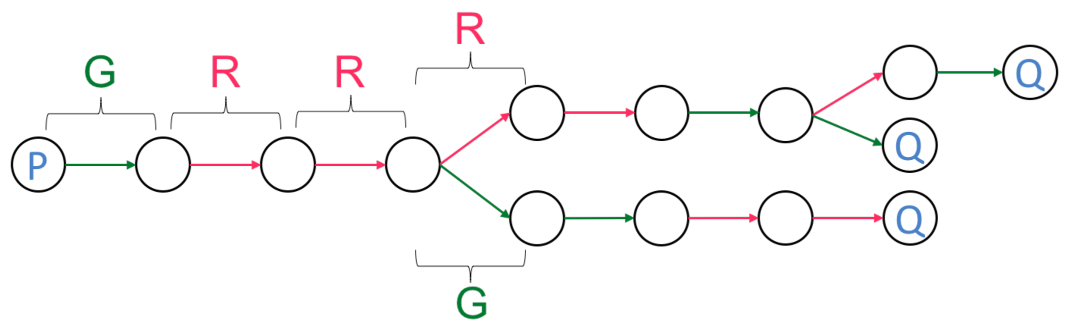


Rely-guarantee (RG) reasoning [1]

Introduced by Jones for safety.
Judgements:

$$\mathcal{R}, \mathcal{G} \vdash \{P\}\ C\ \{Q\}$$

$\mathcal{R}, \mathcal{G}$ are transition relations.



Meaning

If

- the initial state satisfies P, and
- every global state change by another thread is in $\mathcal{R}$

then

- every global state change by C is in $\mathcal{G}$, and
- every final state satisfies Q.

Rely-guarantee is too coarse for bound analysis

RG abstracts

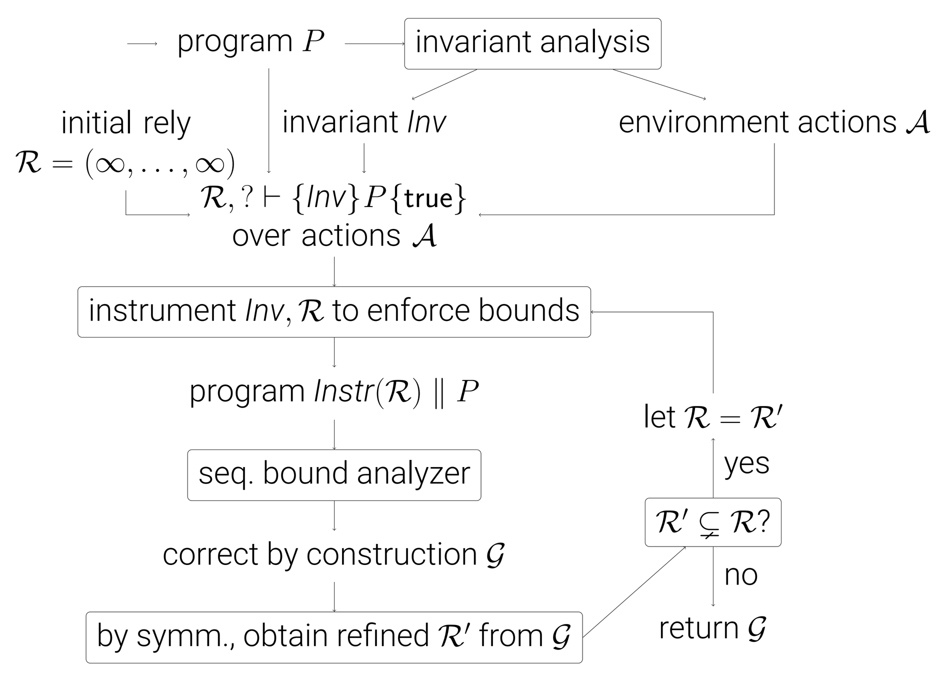
- thread IDs
- the order of environment actions
- **how often** environment actions are executed

Our abstraction

$\mathcal{R}, \mathcal{G}$ are sets of pairs of a transition relation R_i and a bound expression b_i: $\{(R_1, b_1), \dots, (R_m, b_m)\}$. b_i bounds how often R_i can be executed.

RG bound analysis algorithm

Main idea: Reduce RG bound analysis to sequential bound analysis. Iteratively refine environment bounds from local bounds.



Inference rules

Natural extension of Jones' rules:

$$\frac{R + G_1 \vdash \{S_1\}\ P_1\ \{S_1'\} \qquad R + G_2 \vdash \{S_2\}\ P_2\ \{S_2'\}}{R, G_1 + G_2 \vdash \{S_1 \wedge S_2\}\ P_1 \parallel P_2\ \{S_1' \wedge S_2'\}}\ \textsc{Par}$$

$+$ defined on compatible transition relations:

$$\{(R_1, b_1), \dots, (R_m, b_m)\} + \{(R_1, c_1), \dots, (R_m, c_m)\} \stackrel{\text{def}}{=} \{(R_1, b_1 + c_1), \dots, (R_m, b_m + c_m)\}$$

Case studies

- Implemented RG bound analysis in our tool COACHMAN [2].
- First to automatically infer **linear complexity** of well-known concurrent data structures:
  – Treiber's stack, MS queue, DGLM queue

Ongoing & Future Work

Extensions to the bound analysis

- Algorithms where complexity depends on the shape of the data structure (e.g., iterating a list)

Extensions to support other algorithms / protocols

- Distributed algorithms, cache coherence protocols
- Wait-free data structures (guarantee starvation-freedom)
- Data structures where complexity depends on stored data values (sets or counters)
- Other shapes, such as doubly-linked lists or trees

Practical improvements

- Optimize implementation in COACHMAN

Main contributions

1. First extension of RG reasoning to bound analysis.
2. Reduce b.a. of concurrent programs to b.a. of sequential programs.
3. Automatically infer linear complexity of well-known concurrent data structures.

References

[1] C. B. Jones. "Specification and Design of (Parallel) Programs". In: *IFIP Congress*. 1983.
[2] *Coachman*. `https://github.com/thpani/coachman`.
[3] T. Pani, G. Weissenbacher, and F. Zuleger. "Rely-Guarantee Reasoning for Automated Bound Analysis of Lock-Free Algorithms". In: *FMCAD 2018*.

pani@forsyte.at