Automatic Verification of Multi-threaded Programs by Inference of Rely-Guarantee Specifications

Xuan Bach-Le[1], David Sanan[1], Sun Jun[2], Shang-Wei Lin[1]

[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore

[2]School of Information Systems, Singapore Management University, Singapore

An automated framework for compositional verification of concurrent programs

Concurrent programs

Multiple threads run concurrently with shared resources (e.g. memories, data structures)

Testing is not sufficient, bugs cannot be consistently reproduced

Verification is challenging: space-space explosion of the interleavings

Verification approaches for concurrency

Model checking

- Theories: LTL, CTL, automata,…
- Tools: PAT, SPIN, Java Pathfinder,...
- Pros: decidable, automated
- Cons: hard to scale

Formal Systems

- Theories: CSL, Rely-Guarantee,...
- Tools: CompCert, Iris, Caper,…
- Pros: compositional, scalable, expressive
- Cons: undecidable, semi-automated

Verification approaches for concurrency

Model checking

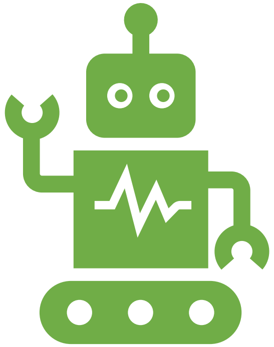
- Theories: LTL, CTL, automata,…
- Tools: PAT, SPIN, Java Pathfinder,...
- Pros: decidable, automated
- Cons: hard to scale

Formal Systems

- Theories: CSL, Rely-Guarantee,...
- Tools: CompCert, Iris, Caper,…
- Pros: compositional, scalable, expressive
- Cons: undecidable, semi-automated

A scalable automated formal system

Inference rules based on Rely-Guarantee technique for compositional reasoning



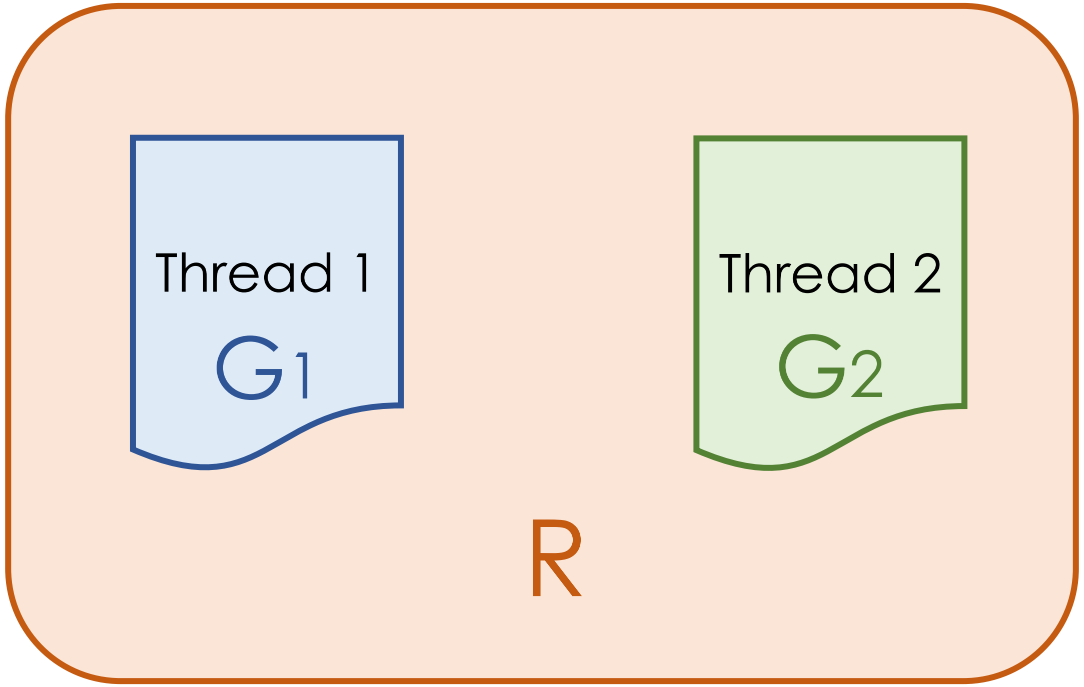
Automated via CEGAR (Counter-Example Guided Abstraction Refinement)

Table of contents

1. Motivation
2. Rely-Guarantee technique
3. Verification framework
4. Evaluation

Rely-Guarantee conditions

- Rely: specs of external environment
- Guarantee: specs of the thread's internal actions



$$R_1 = R \cup G_2$$
$$R_2 = R \cup G_1$$

The Rely of one thread contains the Guarantees of others

Example

$$x = 1 \gg x = 2$$

Guarantee

The thread can update x from 1 into 2

Rely

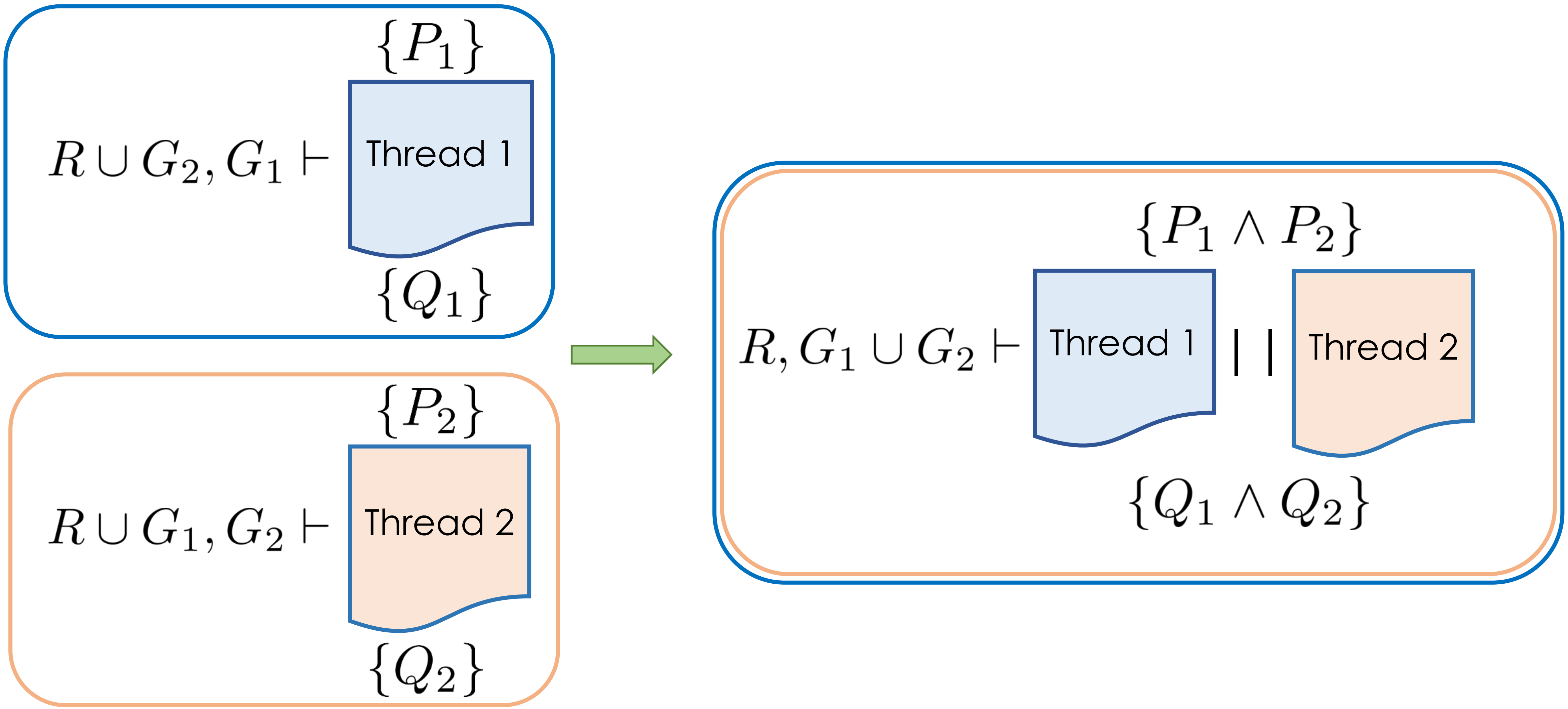
The environment can change x from 1 into 2

Specification

$$R, G \vdash \{P\}c\{Q\}$$

1. Program c with precondition P satisfies Rely R and Guarantee G:
   a) State change satisfy G
   b) State change assume the influence from R
2. Assume c terminates normally. Then Q is the post-condition

Compositional Reasoning



Compositional Reasoning

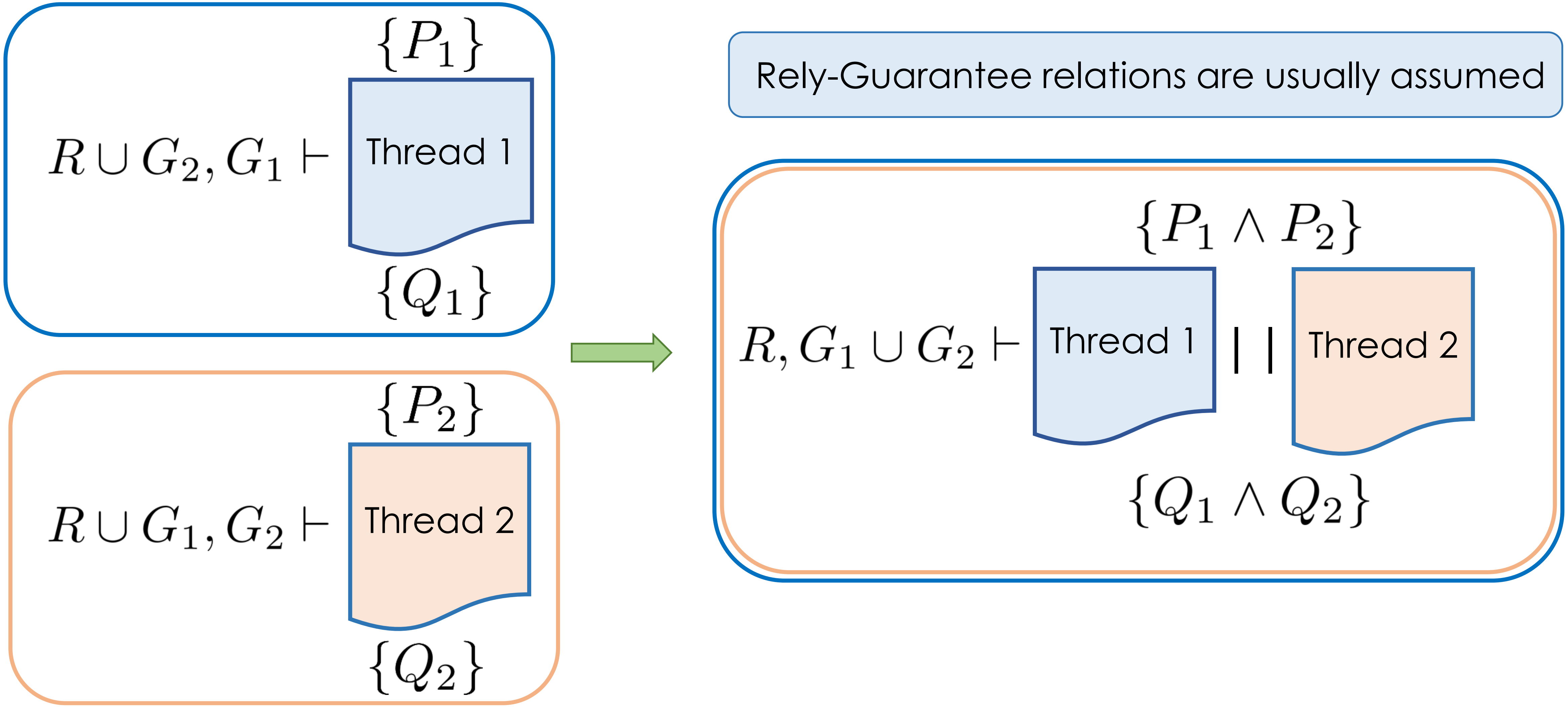
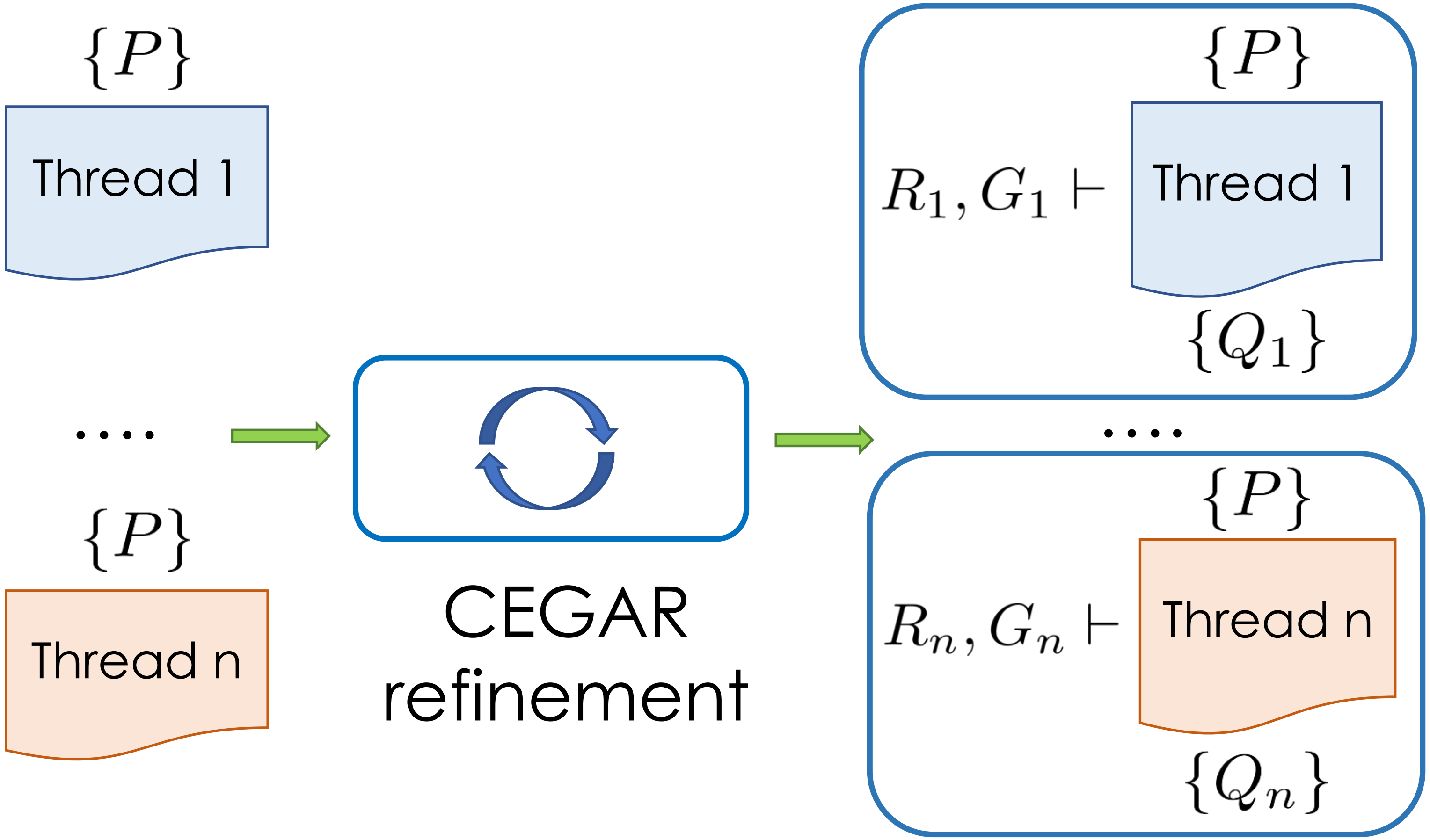


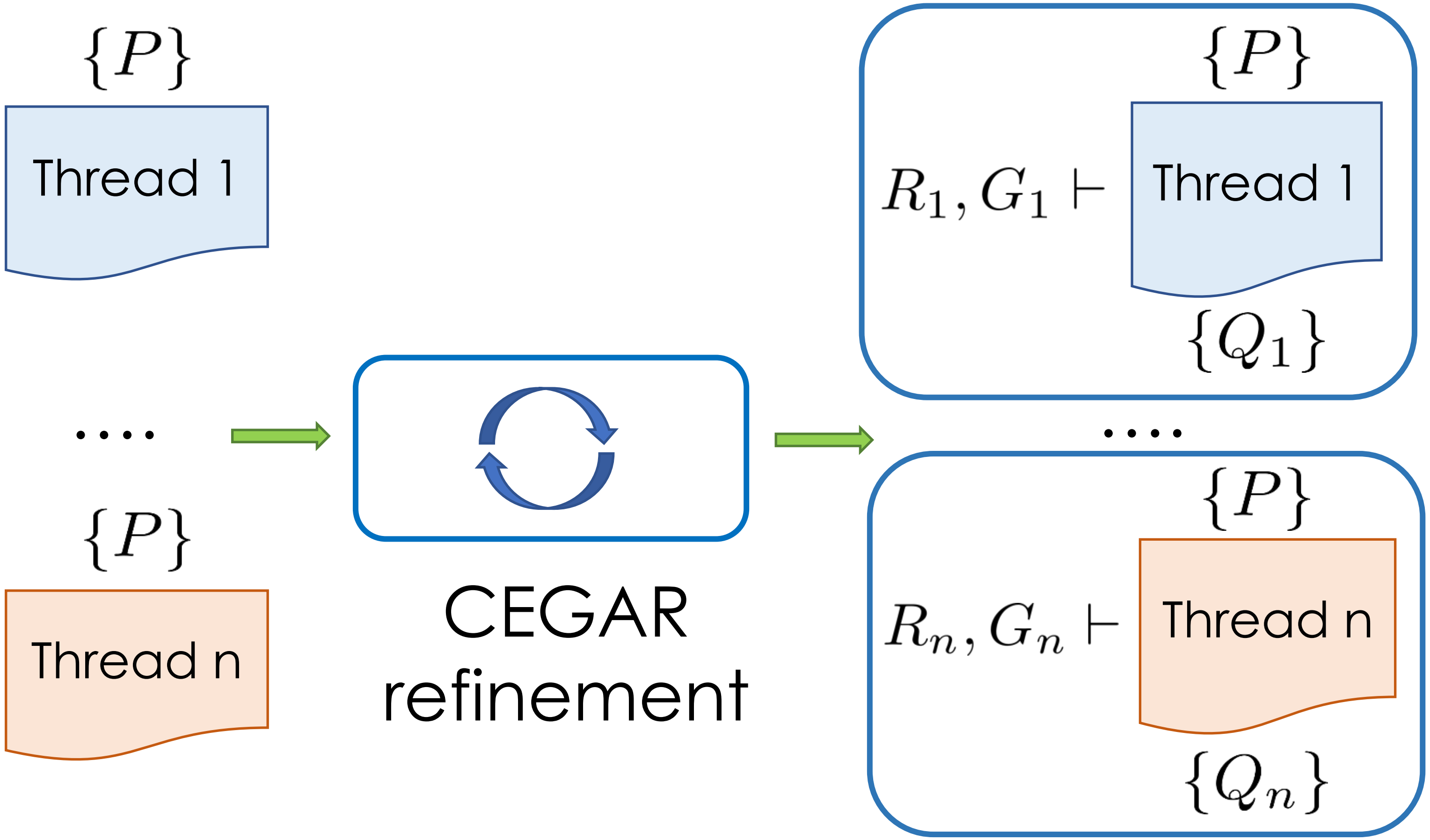
Table of contents

1. Motivation
2. Rely-Guarantee technique
3. Verification framework
4. Evaluation

Overview of the framework



Overview of the framework



Post-condition: $Q_1 \wedge \ldots \wedge Q_n$
We generate the R-G relations instead of assuming ones

Proof construction: High-level

1. For each thread i, generate the local proof L_i
2. Compute the Guarantee G_i from L_i
3. R_i = union of G_j where j <> i

Proof construction: High-level

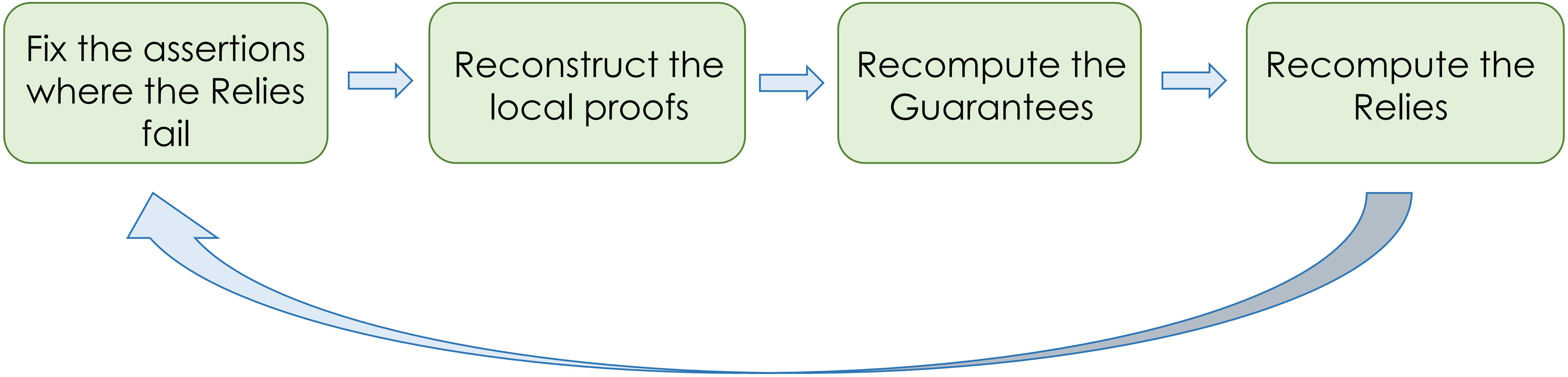
1. For each thread i, generate the local proof L_i
2. Compute the Guarantee G_i from L_i
3. R_i = union of G_j where j <> i

Ri contain other Gj

Ri may not be valid: Li does not satisfy Ri

Refinement via counter examples: High-level

Input: local proofs that fail to satisfy their Relies



Inference rules for construction of local proof and Guarantee relation

A deductive system for constructing/fixing local proof and Guarantee

$$G \lhd \{P\}c\{Q\}$$

1. Program c with precondition P satisfies the Guarantee G
2. If c terminates normally then Q is the post-condition

Checking validity of Relies: Key idea

Transform the validity conditions into equivalent Boolean constraints

UNSAT
valid

SAT
solution is the counter-example

Table of contents

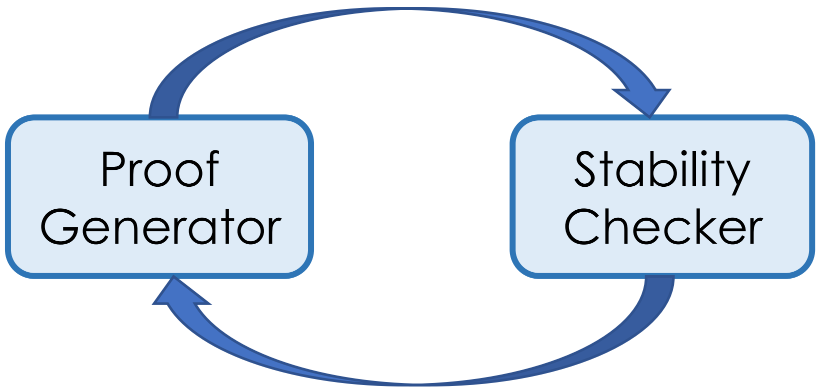
1. Motivation
2. Rely-Guarantee technique
3. Verification framework
4. Evaluation

Implementation

ReGaSol: **Re**ly – **Gua**rantee **Sol**ver

- Java, 4500+ LOC
- 2 main components:

- Construct + fix local proofs
- Compute Guarantees + Relies



- Check validity of Relies
- Generate counter-examples

ReGaSol+ with optimization: parallelization, symmetry reduction, …

Experiment

A small benchmark of 12 programs:

- Standard algorithms for mutex: Peterson, Bakery, Szymanski,…
- Programs with loops

Test against Threader and Lazy-CSeq

Results

No	Name	THREADER	LAZY-CSEQ	REGASOL	REGASOL+
1	peterson	2	**0.92**	1.7	1.22
2	bakery-simpl	2.16	0.8	0.25	**0.17**
3	bakery	61.2	7.07	1.8	**1.1**
4	read-write-lock	0.14	0.59	0.1	**0.11**
5	szymanski	8.02	**2.8**	3.9	2.9
6	time-var-mutex	5.68	0.92	0.13	**0.11**
7	loop1-10-25	0.22	0.71	**0.04**	0.05
8	loop1-100-25	T/O	36.92	**0.03**	0.05
9	loop2-10-20	1.14	0.87	**0.02**	0.03
10	loop2-100-200	T.O	33.92	**0.02**	0.04
11	loop3-10-20	T/O	1.81	**0.17**	**0.17**
12	loop3-100-200	T/O	144.41	**0.17**	0.18

Mutex algorithms:
ReGasol+(**5.59**s) > ReGaSol (**7.78**s) > Lazy-CSeq(**13.1**s) > Threader(**79.2**s)

Loop programs:
ReGaSol(**0.45**s) > ReGaSol+(**0.54**s) > Lazy-CSeq(**218.64**s) > Threader(**T/O**)

Conclusion and future work

An automated framework for verification of concurrent programs

1. Inference rules based on Rely-Guarantee for compositional reasoning
2. CEGAR for refinement
3. ReGaSol and ReGaSol+ with optimizations

Future works

1. Support shared data structures
2. Weakest precondition for completeness

Thank you
☺
Q & A