Accelerating Protocol Synthesis and Detecting Unrealizability with Interpretation Reduction

Derek Egolf, Stavros Tripakis
Northeastern University, Boston

https://github.com/egolf-cs/tacas25

egolf.d@northeastern.edu

Key Contributions

- Synthesize symbolic distributed protocols represented in TLA$^+$ [Lamport].
- Improve state of art in TLA$^+$ synthesis (100x).
- Synthesize a lock protocol "from scratch."
- Halt when no solution: *unrealizability*.
- New search space reduction technique: *Interpretation Reduction*.
- Improved counterexample generalization for pruning.

Sketching [Solar-Lezama]

Given an incomplete sketch with "holes," find a correct completion.

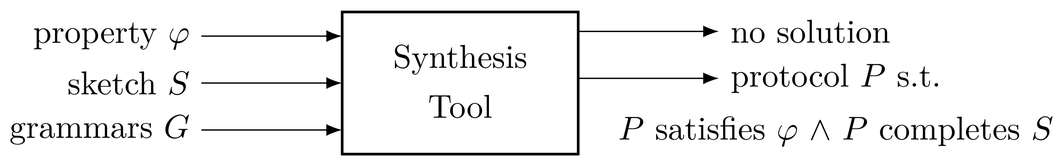
Example sketch:

$Send(src, dst) :=$
$\quad \wedge \; ???_1$
$\quad \wedge \; message' = ???_2$
$\quad \wedge \; has\_lock' = ???_3$
$Receive(src, dst) :=$
$\quad \wedge \; ???_4$
$\quad \wedge \; message' = ???_5$
$\quad \wedge \; has\_lock' = ???_6$

Example completion

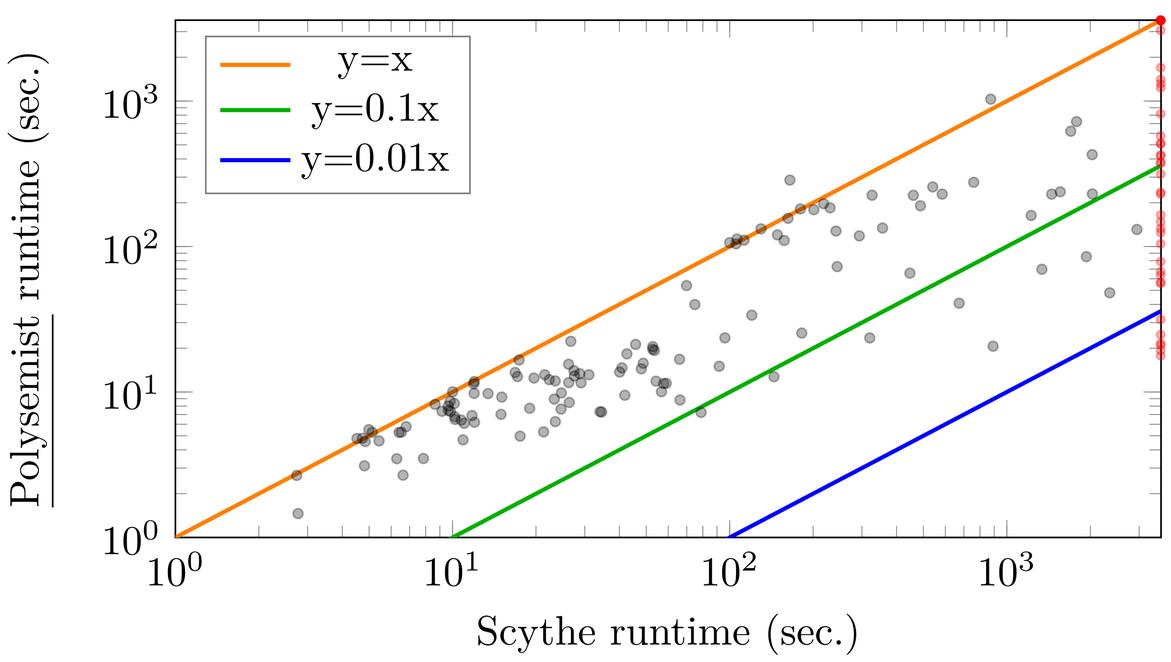
$Send(src, dst) :=$
$\quad \wedge \; has\_lock[src]$
$\quad \wedge \; message' = message \cup \{(src, dst)\}$
$\quad \wedge \; has\_lock' = has\_lock[src \leftarrow false]$
$Receive(src, dst) :=$
$\quad \wedge \; (src, dst) \in message$
$\quad \wedge \; message' = message \setminus \{(src, dst)\}$
$\quad \wedge \; has\_lock' = has\_lock[dst \leftarrow true]$

Problem Statement



Experimental Results

Scythe vs Polysemist Runtime (Realizable Experiments)



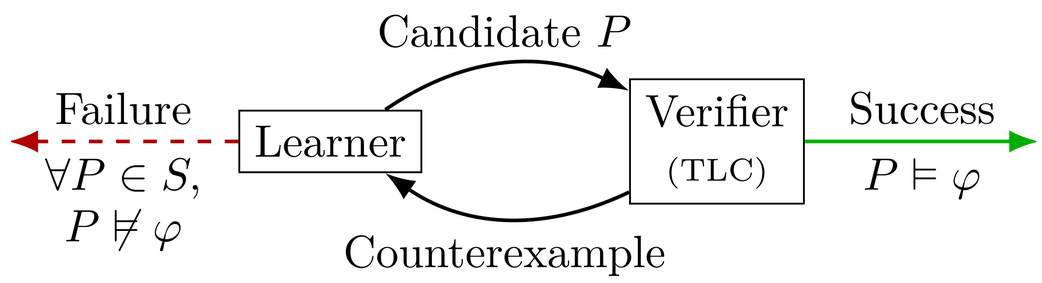
$n = 171$; worse: 11; scythe TO: 47; poly TO 15

Unrealizable Experiments ($n = 123$)

- **Scythe** (old): TO = 107 / HALT = 16
- **Polysemist** (new): TO = 43 / HALT = 80
  - Usually halted in < 60 seconds
  - Did not TO unless Scythe did

Counterexample-Guided Inductive Synthesis (CEGIS)

Our approach uses standard CEGIS technique [Solar-Lezama].



Key Technical Ideas for Learner

- Naive learner: ignore cex, enumerate all protocols
  many expressions; model checking expensive
- Pruning constraints: generalize counterexamples
  discard protocols before model checking
- Equivalence reduction: do not use equiv. sub-expressions
  avoid enumerating protocols in the first place
- Interpretation reduction: coarse, dynamic equivalence relation

Counterexample Generalization

safety cex: $[x \mapsto a] \xrightarrow{A_1} [x \mapsto b]$
Sketch: $(A_1 := ???_1 \wedge \; x' = ???_2)$
Bad Completion: $(A_1 := \text{true} \wedge \; x' = b)$
Bad Completion: $(A_1 := x = a \wedge \; x' = b)$
Good Completion: $(A_1 := x = a \wedge \; x' = a)$
Good Completion: $(A_1 := x \neq a \wedge \; x' = b)$

Pruning Constraint:

$$\pi_{cex} := ???_1([x \mapsto a]) \neq true \vee ???_2([x \mapsto a]) \neq b$$

- In general, many completions violate π_{cex}.
- Checking $P \models \pi_{cex}$ is much cheaper than model checking.
- We generalize deadlock, safety, and liveness violations.
- Prior work [FMCAD'24] uses less exact pruning constraints for deadlock/liveness.

Interpretation Reduction

Absolute Equivalence, e.g: $x + y \equiv y + x$
Interpretation Equivalence, e.g:

$$x + y \equiv_{\mathcal{A}} x + x, \text{ where } \mathcal{A} = \{[x \mapsto 0, y \mapsto 0]\}$$

- $\mathcal{A}$ comes from pruning constraints. E.g.,
  $$\Pi = \{ \; ???_1([x \mapsto 0, y \mapsto 1]) \neq 1 \; \} \rightarrow \mathcal{A} = \{[x \mapsto 0, y \mapsto 1]\}$$
- Suppose we've enumerated y; enumerate $x + y$? No: $y \equiv_{\mathcal{A}} x + y$.
- Avoids enumerating all "super-expressions" of $x + y$, e.g., $x + x + y$.
- Coarse eq. relation makes detecting unrealizability faster.
- [FMCAD'24] uses absolute equivalence for reduction.

Theorem: If $e_1 \equiv_{\mathcal{A}} e_2$ and e_1 enumerated, skipping e_2 does not compromise completeness

References

Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley (2002).
Solar-Lezama, A.: Program Sketching. Int. J. Softw. Tools Technol. Transf. (2013).
Egolf, D., Tripakis, S.: Efficient Synthesis of Distributed Protocols by Sketching. FMCAD (2024).