# Advancing the Complementation of Elevator Automata

Author: Ondrej Alexaj
Supervisor: doc. Ing. Ondřej Lengál, Ph.D.
Consultant: Ing. Vojtěch Havlena, Ph.D.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2026

## Why complementation matters

Complementation of nondeterministic $\omega$-automata is a basic ingredient of automata-theoretic verification. In particular,

$$L(S) \subseteq L(A_\varphi) \qquad \Longleftrightarrow \qquad L(S) \cap \overline{L(A_\varphi)} = \emptyset.$$

Thus, inclusion checking reduces to emptiness after complementing the specification automaton. For transition-based Emerson–Lei automata (TELAs), this step is difficult in practice.

## Elevator automata

An automaton is an *elevator automaton* if every strongly connected component (SCC) is either:

- **inherently weak accepting** — every cycle is accepting, or
- **deterministic accepting**.

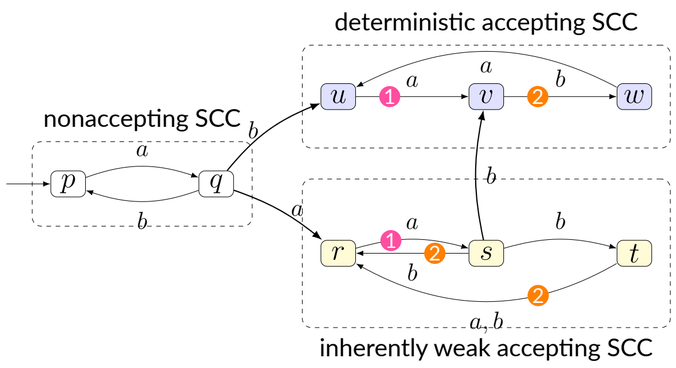


Figure 1. Example of an elevator automaton with acceptance formula $\varphi = Inf(1) \wedge Inf(2)$.

## Deterministic accepting SCCs

The key observation is that once a run enters a deterministic accepting SCC, its future is uniquely determined. Hence, acceptance can be violated *per run*.

For the following acceptance fomrula: $Inf(c_0) \wedge \cdots \wedge Inf(c_k) \wedge Fin(c_{k+1})$, the construction keeps:

- a set $C$ of runs whose long-run behavior has not yet stabilized,
- committed sets $S_0, \ldots, S_k$ for runs guessed to violate $Inf(c_i)$,
- a committed set $C_{k+1}$ for runs guessed to violate $Fin(c_{k+1})$,
- and a breakpoint set $B \subseteq C_{k+1}$.

The control alternates between:

- **waiting mode** ($C \neq \emptyset$): postpone commitment or guess stabilization,
- **checking mode** ($C = \emptyset$): verify the chosen obligation.

## From simple pairs to arbitrary Emerson-Lei formulas

For arbitrary Emerson-Lei formulas, the intuition is the same. Runs in $C$ are still uncommitted, runs in sets $S_c$ are committed to violating $Inf(c)$, and runs in sets $C_c$ are committed to violating $Fin(c)$. All $Fin$-obligations are then checked by one shared breakpoint mechanism in a round-robin order.

The general pattern then is:

- runs in $C$ are still uncommitted,
- runs in sets $S_c$ are committed to violating $Inf(c)$,
- runs in sets $C_c$ are committed to violating $Fin(c)$,
- all $Fin$-obligations are checked by one shared breakpoint mechanism in a round-robin order.

The structure of the formula determines only how runs are committed:

$$\varphi_1 \wedge \varphi_2 \;\Rightarrow\; \text{split runs}, \qquad \varphi_1 \vee \varphi_2 \;\Rightarrow\; \text{copy runs}.$$

## Refinement-based commitment

The main source of blow-up in the general construction is branching in the guessing step.

To reduce this, we use *refinement-based commitment*. Instead of exploring all commitments at once, the construction first follows one preferred guess and refines it only when it is refuted.

Intuitively:

- commit first to a $Fin$-less branch whenever possible,
- if the guess fails, move the affected runs to the postponed sibling branch,
- repeat until a sustainable violating obligation is found.

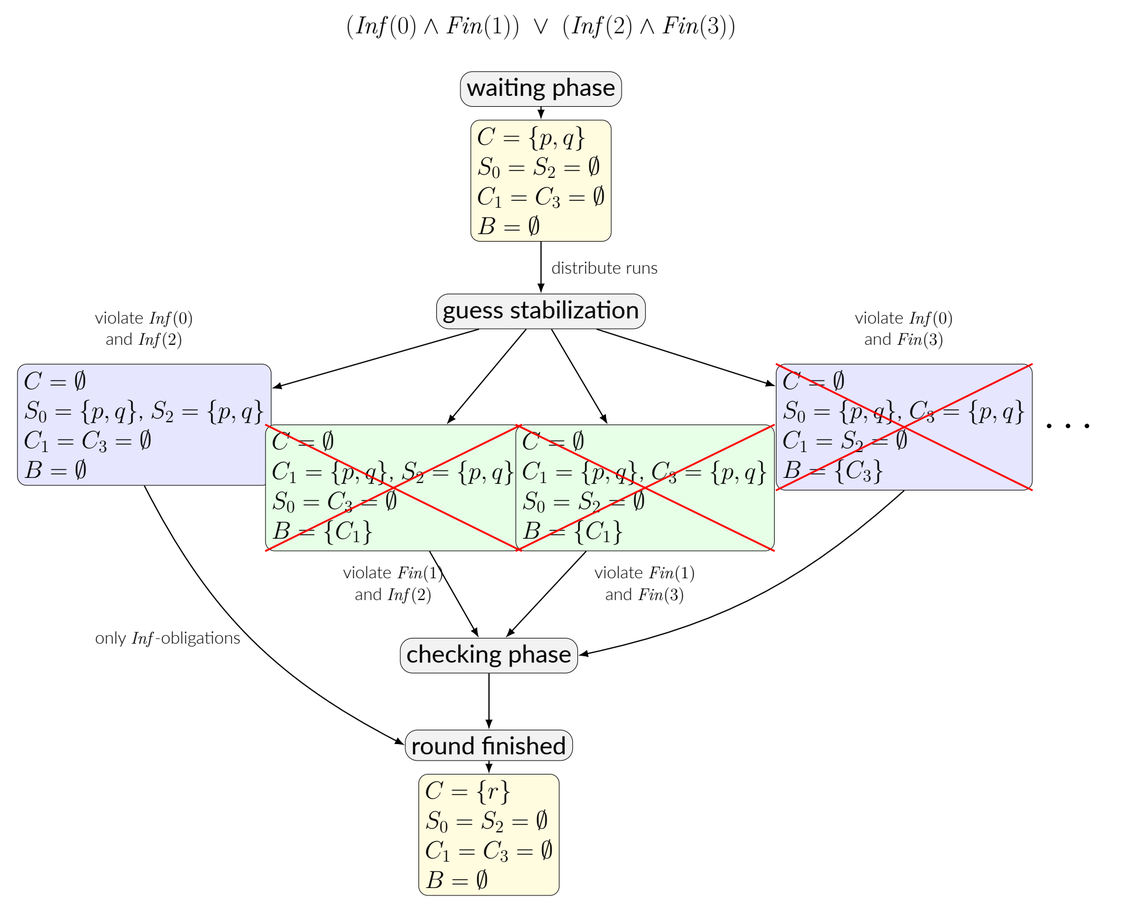


Figure 2. Example of the guessing step for $(Inf(0) \wedge Fin(1)) \vee (Inf(2) \wedge Fin(3))$ and the effect of the refinement-based commitment (RBC).

## Benchmarks

We evaluated the complementation procedure implemented in Kofola against Spot. All experiments used a 120 s timeout.

We considered three benchmark classes:

- **LTL-Lit**: 1,378 existing LTL benchmarks, transformed to elevator automata;
- **GRA**: 315 randomly generated generalized Rabin automata, transformed by a tailored elevatorization;
- **LTL-Rand**: 1,000 random LTL formulas translated to $\omega$-automaton and filtered to already satisfy the elevator property.

These three classes are shown in the plots as ● LTL-Lit, ● GRA, and ● LTL-Rand.

### Kofola vs Spot

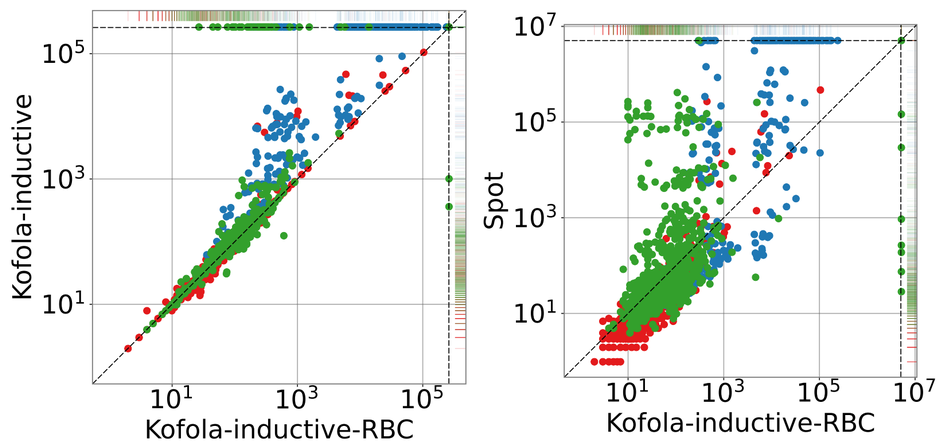


Figure 3. State-space comparison of our implementation and its optimized variant. Kofola-inductive-RBC reduces unnecessary nondeterminism in the complement, We compare Kofola-inductive-RBC against Kofola-inductive (unoptimized variant) and against Spot. Colors denote benchmark classes: ● GRA, ● LTL-Rand, ● LTL-Lit.

Table 1. Complementation statistics. **solved**: successful runs (out of 2,693). **avg**, **med**: state counts. **time**: total runtime (s). **TO**: timeouts. **ERR**: remaining failures.

| tool | solved | max | avg | med | time | TO | ERR |
|---|---|---|---|---|---|---|---|
| Kofola-ind. | 2,447 | 103,380 | 573.83 | 21.00 | 3,090.07 | 220 | 26 |
| Kofola-ind.-RBC | 2,652 | 237,725 | 2,427.55 | 25.00 | 2,729.68 | 15 | 26 |
| Spot | 2,534 | 4,587,580 | 11,136.02 | 15.00 | 1,167.26 | 32 | 127 |

Spot is often faster, but its complements are typically much larger. Our construction, especially the Inductive-FOR variant, produces substantially smaller automata and solves the largest number of instances.


xalexa09@stud.fit.vutbr.cz