KOFOLA 1.0: A Modular Approach to ω-Regular Complementation and Inclusion Checking

(tool paper)

Ondrej Alexaj[1] Vojtěch Havlena[1] Lukáš Holík[1,2]
Ondřej Lengál[1] Yong Li[3] Nicolas Mazzocchi[4]

[1]Brno University of Technology, Czech Republic
[2]Aalborg University, Denmark
[3]Key Laboratory of System Software (Chinese Academy of Sciences), ISCAS, Beijing, PRC
[4]Slovak University of Technology in Bratislava, Slovakia

CAV'26

A Core Problem in Verification

Language inclusion for Büchi automata (BAs):

$$\mathcal{L}(\mathcal{A}_1) \overset{?}{\subseteq} \mathcal{L}(\mathcal{A}_2)$$

Why does it matter?

- **Model checking**: does the system satisfy the spec?
- **Program termination**: via automata-based reasoning (e.g., ULTIMATE AUTOMIZER)
- **Monitoring**: runtime verification
- **Theorem proving**: deciding logical theories (e.g., S1S, FOL over Sturmian words)

A Core Problem in Verification

Language inclusion for Büchi automata (BAs):

$$\mathcal{L}(\mathcal{A}_1) \overset{?}{\subseteq} \mathcal{L}(\mathcal{A}_2)$$

Why does it matter?

- **Model checking**: does the system satisfy the spec?
- **Program termination**: via automata-based reasoning (e.g., ULTIMATE AUTOMIZER)
- **Monitoring**: runtime verification
- **Theorem proving**: deciding logical theories (e.g., S1S, FOL over Sturmian words)

Complexity

PSpace-complete — remains a **practical bottleneck** in automata-based verification

A Core Problem in Verification

Language inclusion for Büchi automata (BAs):

$$\mathcal{L}(\mathcal{A}_1) \overset{?}{\subseteq} \mathcal{L}(\mathcal{A}_2)$$

Why does it matter?

- **Model checking**: does the system satisfy the spec?
- **Program termination**: via automata-based reasoning (e.g., ULTIMATE AUTOMIZER)
- **Monitoring**: runtime verification
- **Theorem proving**: deciding logical theories (e.g., S1S, FOL over Sturmian words)

Reduction to Emptiness

$$\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$$

$$\Updownarrow$$

$$\mathcal{L}(\mathcal{A}_1) \cap \overline{\mathcal{L}(\mathcal{A}_2)} = \emptyset$$

Two key challenges:

1. Compute $\overline{L(\mathcal{A}_2)}$ compactly
2. Check product emptiness efficiently

Complexity

PSpace-complete — remains a **practical bottleneck** in automata-based verification

What We Address

$$\mathcal{L}(\mathcal{A}_1) \cap \overline{\mathcal{L}(\mathcal{A}_2)} \stackrel{?}{=} \emptyset$$

What We Address

$$\mathcal{L}(\mathcal{A}_1) \cap \overline{\mathcal{L}(\mathcal{A}_2)} \stackrel{?}{=} \emptyset$$

- $\overline{\mathcal{L}(\mathcal{A}_2)}$ — efficient complementation procedure

What We Address

$$\mathcal{L}(\mathcal{A}_1) \cap \overline{\mathcal{L}(\mathcal{A}_2)} \stackrel{?}{=} \emptyset$$

- $\overline{\mathcal{L}(\mathcal{A}_2)}$ — efficient complementation procedure
- $\stackrel{?}{=} \emptyset$ — efficient on-the-fly emptiness checking
  - possibly for a non-Büchi condition, e.g.,

$$\mathit{Fin}(0) \wedge \mathit{Inf}(1) \wedge \ldots \wedge \mathit{Inf}(k) \qquad \text{(Simple Generalized Rabin)}$$

Büchi Complementation

Decomposition-based complementation procedure:

- looks at the strongly connected components (SCCs)
- based on
  - Li, Turrini, Feng, Vardi, Zhang. Divide-and-conquer determinization of Büchi automata based on SCC decomposition. CAV'22. (tool COLA)
  - Havlena, Lengál, Li, Šmahlíková, Turrini. Modular mix-and-match complementation of Büchi automata. TACAS'23.

Büchi Complementation

Decomposition-based complementation procedure:

- looks at the strongly connected components (SCCs)
- based on
  - Li, Turrini, Feng, Vardi, Zhang. Divide-and-conquer determinization of Büchi automata based on SCC decomposition. CAV'22. (tool COLA)
  - Havlena, Lengál, Li, Šmahlíková, Turrini. Modular mix-and-match complementation of Büchi automata. TACAS'23.

Motivation

Do not try to kill a mosquito with a machine gun.

Büchi Complementation

Decomposition-based complementation procedure:

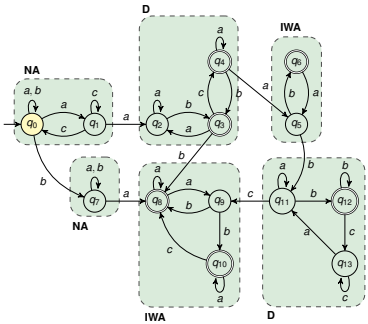
- looks at the strongly connected components (SCCs)
- based on
  - Li, Turrini, Feng, Vardi, Zhang. Divide-and-conquer determinization of Büchi automata based on SCC decomposition. CAV'22. (tool COLA)
  - Havlena, Lengál, Li, Šmahlíková, Turrini. Modular mix-and-match complementation of Büchi automata. TACAS'23.

Motivation

Do not try to kill a mosquito with a machine gun.

Main idea

Complement each SCC with the most suitable procedure ***independently***.



Decomposition-Based Büchi Complementation

1. identify SCCs in the Büchi automaton

Decomposition-Based Büchi Complementation

1. identify SCCs in the Büchi automaton
2. choose a complementation procedure for each SCC C separately:
   - nonaccepting SCCs — simple subset construction ($2^{|C|}$)

Decomposition-Based Büchi Complementation

1. identify SCCs in the Büchi automaton
2. choose a complementation procedure for each SCC C separately:
   - nonaccepting SCCs — simple subset construction ($2^{|C|}$)
   - inherently weak SCCs — Miyano-Hayashi (subset + breakpoint) construction ($3^{|C|}$)

Decomposition-Based Büchi Complementation

1. identify SCCs in the Büchi automaton
2. choose a complementation procedure for each SCC C separately:
   - nonaccepting SCCs — simple subset construction ($2^{|C|}$)
   - inherently weak SCCs — Miyano-Hayashi (subset + breakpoint) construction ($3^{|C|}$)
   - deterministic SCCs — NCSB construction for semi-det. BAs [BlahoudekHSST16] ($4^{|C|}$)

Decomposition-Based Büchi Complementation

1. identify SCCs in the Büchi automaton
2. choose a complementation procedure for each SCC C separately:
   - nonaccepting SCCs — simple subset construction ($2^{|C|}$)
   - inherently weak SCCs — Miyano-Hayashi (subset + breakpoint) construction ($3^{|C|}$)
   - deterministic SCCs — NCSB construction for semi-det. BAs [BlahoudekHSST16] ($4^{|C|}$)
   - initial deterministic SCCs — dualize acceptance condition $\mathit{Inf}(\mathbf{0}) \rightsquigarrow \mathit{Fin}(\mathbf{0})$ ($|C| + 1$)

Decomposition-Based Büchi Complementation

1. identify SCCs in the Büchi automaton
2. choose a complementation procedure for each SCC C separately:
   - nonaccepting SCCs — simple subset construction ($2^{|C|}$)
   - inherently weak SCCs — Miyano-Hayashi (subset + breakpoint) construction ($3^{|C|}$)
   - deterministic SCCs — NCSB construction for semi-det. BAs [BlahoudekHSST16] ($4^{|C|}$)
   - initial deterministic SCCs — dualize acceptance condition $Inf(\mathbf{0}) \rightsquigarrow Fin(\mathbf{0})$ ($|C| + 1$)
   - initial almost deterministic SCCs — subset constr. & dualize acc. cond. ($2^{|C|}$) **NEW!**

Decomposition-Based Büchi Complementation

1. identify SCCs in the Büchi automaton
2. choose a complementation procedure for each SCC C separately:
   - nonaccepting SCCs — simple subset construction ($2^{|C|}$)
   - inherently weak SCCs — Miyano-Hayashi (subset + breakpoint) construction ($3^{|C|}$)
   - deterministic SCCs — NCSB construction for semi-det. BAs [BlahoudekHSST16] ($4^{|C|}$)
   - initial deterministic SCCs — dualize acceptance condition $Inf(\mathbf{0}) \rightsquigarrow Fin(\mathbf{0})$ ($|C|+1$)
   - initial almost deterministic SCCs — subset constr. & dualize acc. cond. ($2^{|C|}$) **NEW!**
   - general SCCs — complementation algorithm for general BAs ($\Omega((0.76n)^n)$)
     - determinization-based [Piterman07]
     - slice-based [Allred, Ultes-Nitsche. LICS'18]

Decomposition-Based Büchi Complementation

1. identify SCCs in the Büchi automaton
2. choose a complementation procedure for each SCC C separately:
   - nonaccepting SCCs — simple subset construction ($2^{|C|}$)
   - inherently weak SCCs — Miyano-Hayashi (subset + breakpoint) construction ($3^{|C|}$)
   - deterministic SCCs — NCSB construction for semi-det. BAs [BlahoudekHSST16] ($4^{|C|}$)
   - initial deterministic SCCs — dualize acceptance condition $Inf(\mathbf{0}) \rightsquigarrow Fin(\mathbf{0})$ ($|C|+1$)
   - initial almost deterministic SCCs — subset constr. & dualize acc. cond. ($2^{|C|}$) **NEW!**

Elevator automata:

[Havlena, Lengál, Šmahlíková. TACAS'22.]

- all SCCs inherently weak or deterministic
- complementation in 4^n
- occur prevalently in practice (98 % of BAs in the inclusion benchmark)

Decomposition-Based Büchi Complementation

1. identify SCCs in the Büchi automaton
2. choose a complementation procedure for each SCC C separately:
   - nonaccepting SCCs — simple subset construction ($2^{|C|}$)
   - inherently weak SCCs — Miyano-Hayashi (subset + breakpoint) construction ($3^{|C|}$)
   - deterministic SCCs — NCSB construction for semi-det. BAs [BlahoudekHSST16] ($4^{|C|}$)
   - initial deterministic SCCs — dualize acceptance condition $Inf(\mathbf{0}) \rightsquigarrow Fin(\mathbf{0})$ ($|C|+1$)
   - initial almost deterministic SCCs — subset constr. & dualize acc. cond. ($2^{|C|}$) **NEW!**
   - general SCCs — complementation algorithm for general BAs ($\Omega((0.76n)^n)$)
     - determinization-based [Piterman07]
     - slice-based [Allred, Ultes-Nitsche. LICS'18]
3. A top-level orchestration algorithm
   - synchronized product construction $\rightsquigarrow$ macrostates
   - output acceptance condition:

$$Fin(\mathbf{0}) \wedge Inf(\mathbf{1}) \wedge \ldots \wedge Inf(\mathbf{k}) \qquad \text{(Simple Generalized Rabin)}$$

On-the-fly Emptiness Check (for Inclusion)

How to efficiently check acceptance for the following condition in $\mathcal{A}_1 \times \overline{\mathcal{A}_2}$?

$$Fin(0) \wedge Inf(1) \wedge \ldots \wedge Inf(k)$$

On-the-fly Emptiness Check (for Inclusion)

How to efficiently check acceptance for the following condition in $\mathcal{A}_1 \times \overline{\mathcal{A}_2}$?

$$Fin(0) \wedge Inf(1) \wedge \ldots \wedge Inf(k)$$

Properties of our algorithm:

- on-the-fly check
  - no need to construct the full product automaton $\mathcal{A}_1 \times \overline{\mathcal{A}_2}$

On-the-fly Emptiness Check (for Inclusion)

How to efficiently check acceptance for the following condition in $\mathcal{A}_1 \times \overline{\mathcal{A}_2}$?

$$\mathit{Fin}(0) \wedge \mathit{Inf}(1) \wedge \ldots \wedge \mathit{Inf}(k)$$

Properties of our algorithm:

- on-the-fly check
  - no need to construct the full product automaton $\mathcal{A}_1 \times \overline{\mathcal{A}_2}$
- maximally lazy procedure
  - if the constructed part of $\mathcal{A}_1 \times \overline{\mathcal{A}_2}$ is sufficient for showing non-emptiness, terminate

On-the-fly Emptiness Check (for Inclusion)

How to efficiently check acceptance for the following condition in $\mathcal{A}_1 \times \overline{\mathcal{A}_2}$?

$$\mathit{Fin}(0) \wedge \mathit{Inf}(1) \wedge \ldots \wedge \mathit{Inf}(k)$$

Properties of our algorithm:

- on-the-fly check
  - no need to construct the full product automaton $\mathcal{A}_1 \times \overline{\mathcal{A}_2}$
- maximally lazy procedure
  - if the constructed part of $\mathcal{A}_1 \times \overline{\mathcal{A}_2}$ is sufficient for showing non-emptiness, terminate

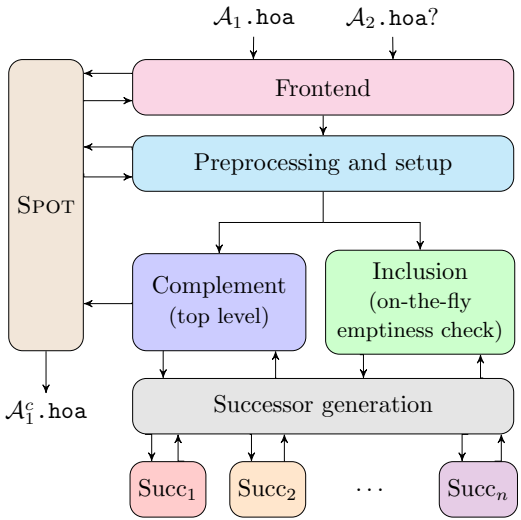
Main idea:

- Modification of [Couvreur. FM'99] (Tarjan-based)
- DFS + when you see a 0-transition, add the target to *Entries*, don't explore further
- When DFS finishes, start another DFS from some state in *Entries*

Architecture

KOFOLA:

- C++ tool
- uses SPOT for
  - loading automata in HOA format
  - pre/post-processing
- polymorphic successor generation through virtual methods
  - each partial complementation procedure implements a high-level interface
  - easy to plug in different procedures



Experiments 1: Complementation

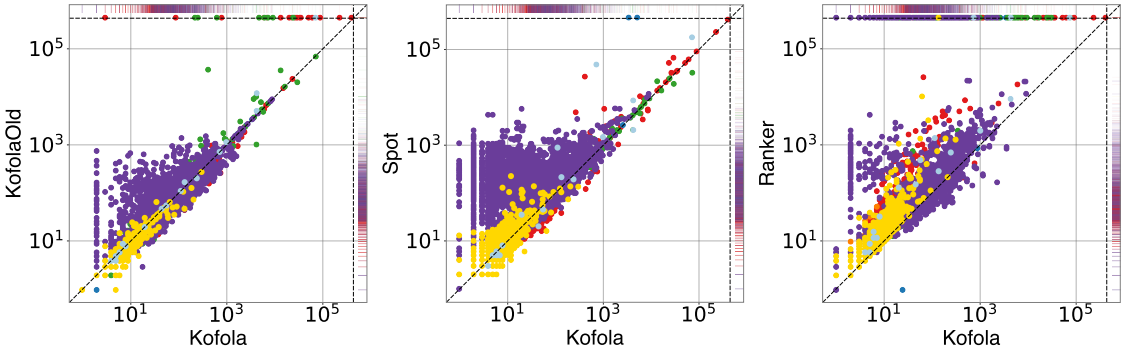
- **Timeout**: 120 s
- **Benchmarks** (31,273 ω-automata):
  - SOBC — random
  - SEMINATOR — from LTL
  - LDBA4LTL — from LTL
  - AUTOMIZER$_{\mathcal{C}}$ — program termination
  - AUTOHYPER$_{\mathcal{C}}$ — HyperLTL verification
  - PECAN$_{\mathcal{C}}$ — FOL over Sturmian words
  - S1S

tool	×	avg	time	unsup
KOFOLA	0	79.24	1,069	0
SPOT	2	115.38	199	0
KOFOLASLICE	4	166.03	1,376	0
KOFOLAOLD	44	42.74	2,471	0
OWL	104	37.39	1,240	507
RANKER	443	45.80	22,817	1,336

×: unsolved cases
avg: average # of states of output
time: total time on solved [s]
unsup: unsupported acc. condition

- **Tools**: (all with default postprocessing, usually ***low***):
  - KOFOLA: determinization-based complementation for general SCCs [Piterman07]
  - KOFOLASLICE: slice-based compl. for general SCCs [AllredUltes-Nitsche18]
  - KOFOLAOLD: version from TACAS'23
  - SPOT, OWL: deteterminization-based
  - RANKER: optimized rank-based complementation [KupfermanVardi01] + 5 papers

Experiments 1: Complementation (# of states)



(a) vs. KOFOLAOLD (b) vs. SPOT (c) vs. RANKER

benchmarks

SOBC SEMINATOR AUTOMIZER$_{\mathcal{C}}$ LDBA4LTL

AUTOHYPER$_{\mathcal{C}}$ PECAN$_{\mathcal{C}}$ S1S

Experiments 2: Inclusion Checking

- **Timeout**: 120 s
- **Benchmarks** (507 BA pairs, $\times 2$):
  - AUTOHYPER$_{\mathcal{I}}$ — HyperLTL verification
  - CONCUR — verif. of concur. systems
  - PECAN$_{\mathcal{I}}$ — FOL over Sturmian words
  - AUTOMIZER$_{\mathcal{I}}$ — program termination

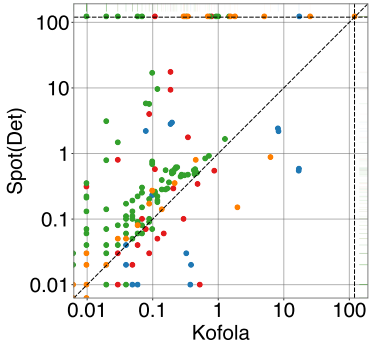
98 % of BAs on the RHS of $\subseteq$ were elevator automata (87 % from total)

tool	×	time	wins	losses
KOFOLA	2	141		
SPOT(DET)	28	122	263	56
FORKLIFT	35	1,273	997	9
SPOT(FORQ)	54	1,134	307	89
RABIT	57	4,300	996	8
BAIT	64	1,981	1,000	5

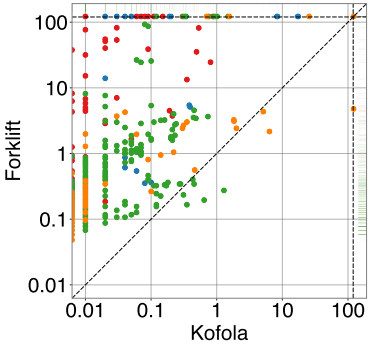
×: unsolved cases
time: total time on solved [s]
wins: # KOFOLA wins, **losses**: # KOFOLA loses

- **Tools**:
  - KOFOLA: slice-based compl. for general SCCs [AllredUltes-Nitsche18]
  - BAIT: well-quasiorders-based [DoveriGantyParoliniRanzato21]
  - FORKLIFT: FORQ-based (family of right quasiorders) [DoveriGantyMazzocchi22]
  - SPOT(DET): deteterminization-based
  - SPOT(FORQ): reimplementation of FORKLIFT in SPOT
  - RABIT: generalized simulations + Ramsey-based [MayrClemente19]

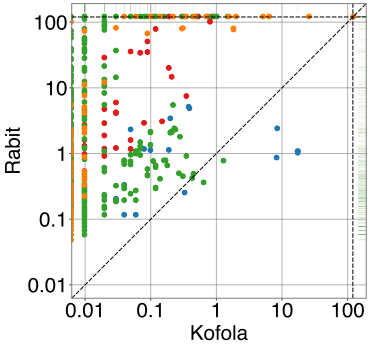
Experiments 2: Inclusion Checking (time)



(a) vs. SPOT



(b) vs. FORKLIFT



(c) vs. RABIT

benchmarks

- AUTOHYPER$_{\mathcal{I}}$
- CONCUR
- PECAN$_{\mathcal{I}}$
- AUTOMIZER$_{\mathcal{I}}$

KOFOLA: Summary

Takeaways:

- SCC-based reasoning
  - essential for dealing with ω-automata in practice

KOFOLA: Summary

Takeaways:

- SCC-based reasoning
  - essential for dealing with ω-automata in practice
- Elevator automata (BAs with SCCs being inherently weak or deterministic)
  - everywhere!
  - working with them much easier

KOFOLA: Summary

Takeaways:

- SCC-based reasoning
  - essential for dealing with ω-automata in practice
- Elevator automata (BAs with SCCs being inherently weak or deterministic)
  - everywhere!
  - working with them much easier

KOFOLA:

- state-of-the-art ω-regular complementation and inclusion checking

KOFOLA: Summary

Takeaways:

- SCC-based reasoning
  - essential for dealing with ω-automata in practice
- Elevator automata (BAs with SCCs being inherently weak or deterministic)
  - everywhere!
  - working with them much easier

KOFOLA:

- state-of-the-art ω-regular complementation and inclusion checking
- plug & play:
  - easy to plug in **YOUR** complementation procedure

KOFOLA: Summary

Takeaways:

- SCC-based reasoning
  - essential for dealing with ω-automata in practice
- Elevator automata (BAs with SCCs being inherently weak or deterministic)
  - everywhere!
  - working with them much easier

KOFOLA:

- state-of-the-art ω-regular complementation and inclusion checking
- plug & play:
  - easy to plug in **YOUR** complementation procedure
- **NEW:**
  - Emerson-Lei elevator automata
    - Alexaj, Havlena, Lengál, Li, Mazzocchi. Complementing Emerson-Lei Elevator Automata. CONCUR'26. (to appear)
    - deterministic SCCs can have Emerson-Lei acceptance condition

KOFOLA: Summary

Takeaways:

- SCC-based reasoning
  - essential for dealing with ω-automata in practice
- Elevator automata (BAs with SCCs being inherently weak or deterministic)
  - everywhere!
  - working with them much easier

KOFOLA:

- state-of-the-art ω-regular complementation and inclusion checking
- plug & play:
  - easy to plug in **YOUR** complementation procedure
- **NEW:**
  - Emerson-Lei elevator automata
    - Alexaj, Havlena, Lengál, Li, Mazzocchi. Complementing Emerson-Lei Elevator Automata. CONCUR'26. (to appear)
    - deterministic SCCs can have Emerson-Lei acceptance condition

Thank you!