Modular Mix-and-Match Complementation of Büchi Automata

Vojtěch Havlena[1] Ondřej Lengál[1] Yong Li[2,3]
Barbora Šmahlíková[1] Andrea Turrini[3,4]

[1]Brno University of Technology, Czech Republic
[2]University of Liverpool, UK
[3]Chinese Academy of Sciences, China
[4]Institute of Intelligent Software, China

TACAS'23

Büchi Automata

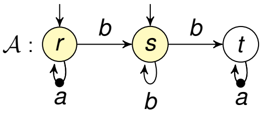
Büchi automata (BAs):

- Automata over infinite words
- $\mathcal{A} = (Q, \delta, I, Acc)$ over Σ
  - Q finite set of states
  - δ transition relation; $\delta \subseteq Q \times \Sigma \times Q$
  - $I \subseteq Q$ initial states
  - $Acc \subseteq \delta$ accepting transitions

Büchi Automata

Büchi automata (BAs):

- Automata over infinite words
- $\mathcal{A} = (Q, \delta, I, Acc)$ over Σ
  - Q finite set of states
  - δ transition relation; $\delta \subseteq Q \times \Sigma \times Q$
  - $I \subseteq Q$ initial states
  - $Acc \subseteq \delta$ accepting transitions
- accept by going infinitely often through accepting transitions



- $r \xrightarrow{a} r \xrightarrow{b} s \xrightarrow{b} t \xrightarrow{a} t \xrightarrow{a} \cdots \quad abba^{\omega} \in \mathcal{L}(\mathcal{A})$
- $L(\mathcal{A}) = (\epsilon + a^{*}bb^{+} + b^{+})a^{\omega}$

- define the class of ω-regular languages
- used in program verification (Ultimate Automizer), linear time MC, probabilistic MC, decision procedures, ...

BA Complementation

Complementation:

- Given $\mathcal{A}$, get a BA $\mathcal{A}^{\complement}$ such that $\mathcal{L}(\mathcal{A}^{\complement}) = \overline{\mathcal{L}(\mathcal{A})}$.

BA Complementation

Complementation:

- Given $\mathcal{A}$, get a BA $\mathcal{A}^{\complement}$ such that $\mathcal{L}(\mathcal{A}^{\complement}) = \overline{\mathcal{L}(\mathcal{A})}$.

Motivation:

- Model checking of linear-time properties

$$\underbrace{S}_{\text{system}} \models \underbrace{\varphi}_{\text{property}} \rightsquigarrow \quad \mathcal{L}(\mathcal{A}_S) \subseteq \mathcal{L}(\mathcal{A}_\varphi) \quad \rightsquigarrow \quad \mathcal{L}(\mathcal{A}_S) \cap \mathcal{L}(\mathcal{A}_\varphi^{\complement}) = \emptyset$$

BA Complementation

Complementation:

- Given $\mathcal{A}$, get a BA $\mathcal{A}^{\complement}$ such that $\mathcal{L}(\mathcal{A}^{\complement}) = \overline{\mathcal{L}(\mathcal{A})}$.

Motivation:

- Model checking of linear-time properties

$$\underbrace{S}_{\text{system}} \models \underbrace{\varphi}_{\text{property}} \rightsquigarrow \quad \mathcal{L}(\mathcal{A}_S) \subseteq \mathcal{L}(\mathcal{A}_\varphi) \quad \rightsquigarrow \quad \mathcal{L}(\mathcal{A}_S) \cap \mathcal{L}(\mathcal{A}_\varphi^{\complement}) = \emptyset$$

- Termination analysis of programs: Ultimate Automizer
  - removing traces with proved termination
  - difference automaton

BA Complementation

Complementation:

- Given $\mathcal{A}$, get a BA $\mathcal{A}^{\complement}$ such that $\mathcal{L}(\mathcal{A}^{\complement}) = \overline{\mathcal{L}(\mathcal{A})}$.

Motivation:

- Model checking of linear-time properties

$$\underbrace{S}_{\text{system}} \models \underbrace{\varphi}_{\text{property}} \rightsquigarrow \quad \mathcal{L}(\mathcal{A}_S) \subseteq \mathcal{L}(\mathcal{A}_\varphi) \quad \rightsquigarrow \quad \mathcal{L}(\mathcal{A}_S) \cap \mathcal{L}(\mathcal{A}_\varphi^{\complement}) = \emptyset$$

- Termination analysis of programs: Ultimate Automizer
  - removing traces with proved termination
  - difference automaton
- Decision procedures: implements negation
  - S1S: MSO over $(\omega, 0, +1)$
  - QPTL: quantified propositional temporal logic
  - FO over Sturmian words

BA Complementation

Complementation:

- Given $\mathcal{A}$, get a BA $\mathcal{A}^{\complement}$ such that $\mathcal{L}(\mathcal{A}^{\complement}) = \overline{\mathcal{L}(\mathcal{A})}$.

Motivation:

- Model checking of linear-time properties

$$\underbrace{S}_{\text{system}} \models \underbrace{\varphi}_{\text{property}} \rightsquigarrow \quad \mathcal{L}(\mathcal{A}_S) \subseteq \mathcal{L}(\mathcal{A}_\varphi) \quad \rightsquigarrow \quad \mathcal{L}(\mathcal{A}_S) \cap \mathcal{L}(\mathcal{A}_\varphi^{\complement}) = \emptyset$$

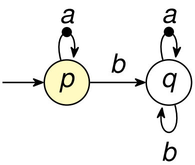
- Termination analysis of programs: Ultimate Automizer
  - removing traces with proved termination
  - difference automaton
- Decision procedures: implements negation
  - S1S: MSO over $(\omega, 0, +1)$
  - QPTL: quantified propositional temporal logic
  - FO over Sturmian words
- Basic operation for inclusion/equivalence checking

BA Complementation

- Notoriously difficult
  - exponential worst-case lower bound $(0.76n)^n$ [Yan'06]

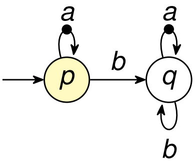
BA Complementation

- Notoriously difficult
  - exponential worst-case lower bound $(0.76n)^n$ [Yan'06]
- Specialized procedures
  - deterministic BAs: $2n$ states

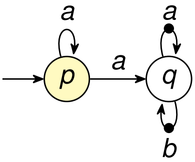


BA Complementation

- Notoriously difficult
  - exponential worst-case lower bound $(0.76n)^n$ [Yan'06]
- Specialized procedures
  - deterministic BAs: $2n$ states

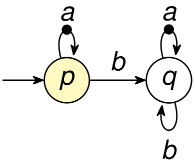


  - inherently weak: $\mathcal{O}(3^n)$

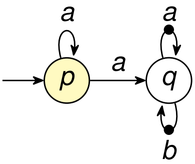


BA Complementation

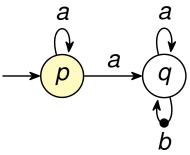
- Notoriously difficult
  - exponential worst-case lower bound $(0.76n)^n$ [Yan'06]
- Specialized procedures
  - deterministic BAs: $2n$ states



  - inherently weak: $\mathcal{O}(3^n)$



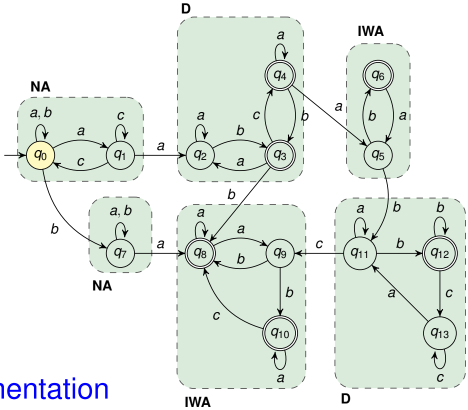
  - semi-deterministic: $\mathcal{O}(4^n)$



BA Complementation

- Elevator automata[1]
  - Inherently weak and deterministic SCCs
  - Upper bound $\mathcal{O}(16^n)$
- Problem: structure on the whole automaton

$\Rightarrow$ decomposition-based complementation



[1] **ElevatorTacas**.

Decomposition-Based Complementation

- Based on decomposition-based determinization[2]
- Decomposition into partition blocks

[2] **LiTFVZ22**.

Decomposition-Based Complementation

- Based on decomposition-based determinization[2]
- Decomposition into partition blocks
- Complementation of each block independently:
  1. Different algorithm for each block based on its properties
  2. Partial algorithm can focus only on one block
  3. More general acceptance condition (ELA) $\Rightarrow$ potentially smaller result

[2] **LiTFVZ22**.

Decomposition-Based Complementation

- Based on decomposition-based determinization[2]
- Decomposition into partition blocks
- Complementation of each block independently:
  1. Different algorithm for each block based on its properties
  2. Partial algorithm can focus only on one block
  3. More general acceptance condition (ELA) $\Rightarrow$ potentially smaller result
- Accepting run eventually stays in one SCC

[2] **LiTFVZ22**.

Decomposition-Based Complementation

1. Decomposition into BAs
   - One BA for each partition block
   - Intersection of all complements

Decomposition-Based Complementation

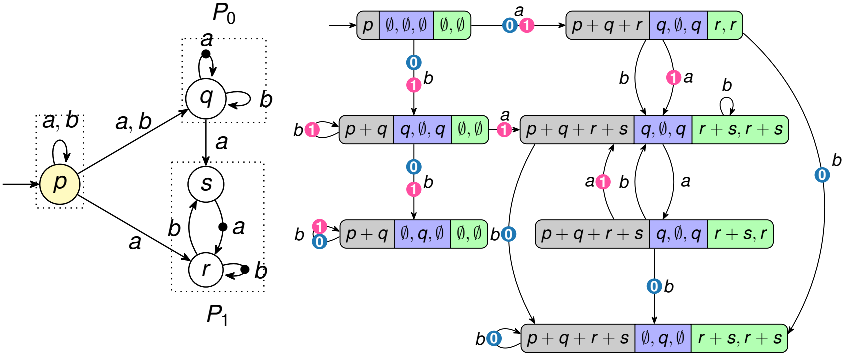
1. Decomposition into BAs
   - One BA for each partition block
   - Intersection of all complements
2. On-the-fly algorithm
   - One complement
   - Macrostates consists of several parts

Synchronous Complementation

- Top-level algorithm
- Orchestrates runs of the different complementation procedures

Synchronous Complementation

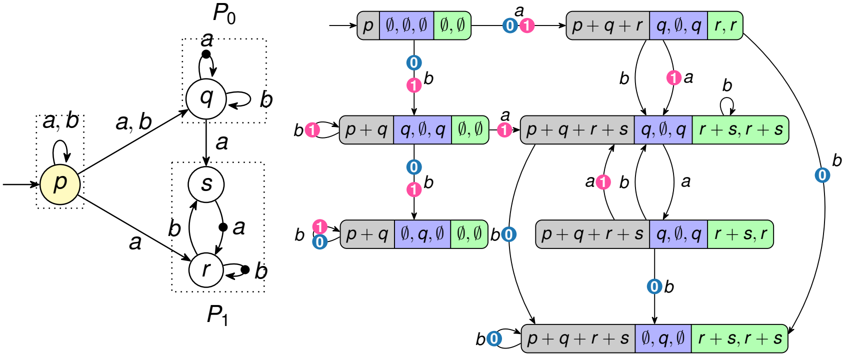
- Top-level algorithm
- Orchestrates runs of the different complementation procedures



Inf(0) ∧ Inf(1)

Synchronous Complementation

- Top-level algorithm
- Orchestrates runs of the different complementation procedures



$\mathrm{Inf}(0) \wedge \mathrm{Inf}(1)$

- Exponentially better upper bound: $\mathcal{O}(16^n) \to \mathcal{O}(4^n)$
  - Same as for semi-deterministic BAs (strict subclass)

Synchronous Complementation

- Works for any Büchi automaton
  - Nonstructured SCCs: rank-based, determinization-based, etc.

Synchronous Complementation

- Works for any Büchi automaton
  - Nonstructured SCCs: rank-based, determinization-based, etc.
- Open framework
  - Flexible algorithm
  - Works for any reasonable complementation algorithm
  - Complementation algorithm for some restricted subclass can be easily pluggen in

Optimizations

- More opportunities for optimizations than determinization
  - Result can be nondeterministic
  - Better upper bounds

Optimizations

- More opportunities for optimizations than determinization
  - Result can be nondeterministic
  - Better upper bounds

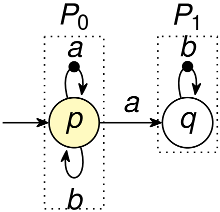
1. Initial deterministic partition blocks
2. Postponed construction
3. Round-robin algorithm
4. Shared breakpoint
5. Simulation pruning

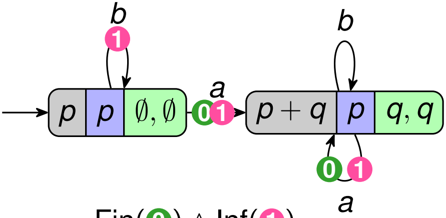
Initial Deterministic Partition Blocks

- Block is deterministic and can be reached only deterministically

Initial Deterministic Partition Blocks

- Block is deterministic and can be reached only deterministically
- Based on complementation of deterministic BAs into co-BAs
- Fin acceptance condition





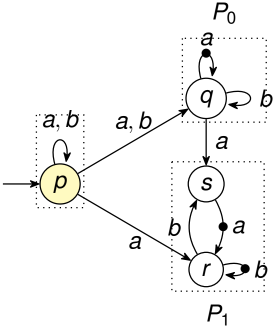
Fin(0) ∧ Inf(1)

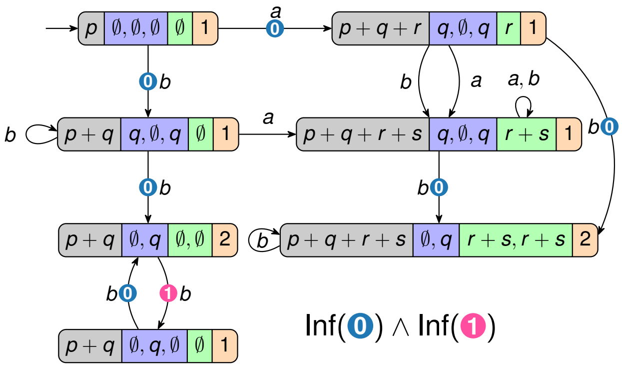
Postponed Construction

- One BA for each partition block
- Intersection of the complements
- Reduction of the intermediate automata
- Does not give better upper bound for elevator BAs

Round-Robin Algorithm

- Combinatorial explosion in a synchronous approach
  - Cartesian product of all successors
- Actively tracks only one partition block, others are passive
- Periodically changes the active algorithm





Shared Breakpoint

- Some partial algorithms use a breakpoint
  - To check whether runs are accepting or not

Shared Breakpoint

- Some partial algorithms use a breakpoint
  - To check whether runs are accepting or not
- Only one breakpoint for all algorithms:
  1. May lead to a smaller complement
  2. Fewer colours (only one for elevator automata)

Simulation Pruning

- Simulation is a relation $\preccurlyeq\ \subseteq Q \times Q$:
  $\forall p, q \in Q\colon p \preccurlyeq q \Longrightarrow \mathcal{L}(\mathcal{A}[p]) \subseteq \mathcal{L}(\mathcal{A}[q])$

Simulation Pruning

- Simulation is a relation $\preccurlyeq \subseteq Q \times Q$:
  $\forall p, q \in Q \colon p \preccurlyeq q \Longrightarrow \mathcal{L}(\mathcal{A}[p]) \subseteq \mathcal{L}(\mathcal{A}[q])$
- We can remove p from a macrostate if there is also q such that
  1. $p \preccurlyeq q$
  2. p is not reachable from q
  3. p is smaller than q in an arbitrary total order over Q

Simulation Pruning

- Simulation is a relation $\preccurlyeq \subseteq Q \times Q$:
  $\forall p, q \in Q \colon p \preccurlyeq q \Longrightarrow \mathcal{L}(\mathcal{A}[p]) \subseteq \mathcal{L}(\mathcal{A}[q])$
- We can remove *p* from a macrostate if there is also *q* such that
  1. $p \preccurlyeq q$
  2. *p* is not reachable from *q*
  3. *p* is smaller than *q* in an arbitrary total order over *Q*
- The behaviour of *p* can be completely simulated by *q*
- More macrostates are mapped to one

Experimental Evaluation

- Tool KOFOLA (C++, built on top of SPOT)
- Comparison with other state-of-the-art tools
  - SPOT, COLA, RANKER, SEMINATOR

Experimental Evaluation

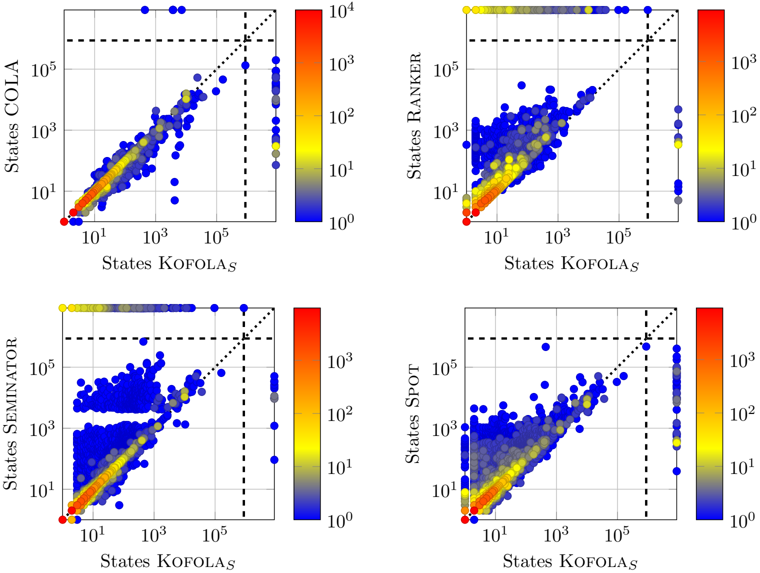
- Tool KOFOLA (C++, built on top of SPOT)
- Comparison with other state-of-the-art tools
  - SPOT, COLA, RANKER, SEMINATOR
- 39 837 BAs
  - Randomly generated
  - From LTL formulae
  - From ULTIMATE AUTOMIZER
  - From PECAN (solver for the first-order logic over Sturmian words)
  - From an S1S solver
  - From LTL to SDBA translation

Experimental Evaluation

tool	solved	unsolved			states			runtime		
		TO		OOM	mean		median	mean		median
KOFOLA$_S$	39,738	89	:	10	76	:	3	0.32	:	0.03
KOFOLA$_P$	39,750	76	:	11	86	:	3	0.41	:	0.03
VBS$_+$	39,834		3		78	:	3	0.05	:	0.01
VBS$_-$	39,834		3		96	:	3	0.05	:	0.01
COLA	39,814	21	:	0	80	:	3	0.17	:	0.02
RANKER	38,837	61	:	939	45	:	4	3.31	:	0.01
SEMINATOR 2	39,026	238	:	573	247	:	3	1.98	:	0.03
SPOT	39,827	8	:	0	160	:	4	0.08	:	0.02

KOFOLA$_S$: synchronous approach
KOFOLA$_P$: postponed approach
VBS$_+$: virtual best solver with Kofola
VBS$_-$: virtual best solver without Kofola

Experimental Evaluation



Conclusion

- Open framework for BA complementation
- Different algorithm for each SCC
- Exponentially better upper bound for elevator automata

Conclusion

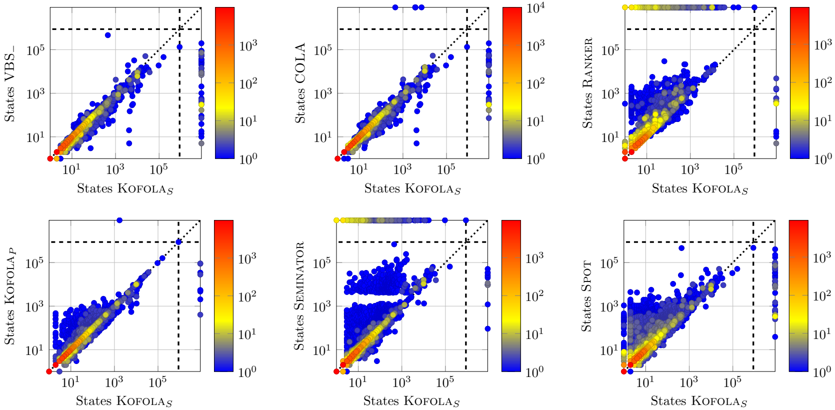
- Open framework for BA complementation
- Different algorithm for each SCC
- Exponentially better upper bound for elevator automata
- Future work
  - Smart ways to choose algorithms based on SCC properties
  - Other algorithms for NACs
  - Language inclusion testing

Conclusion

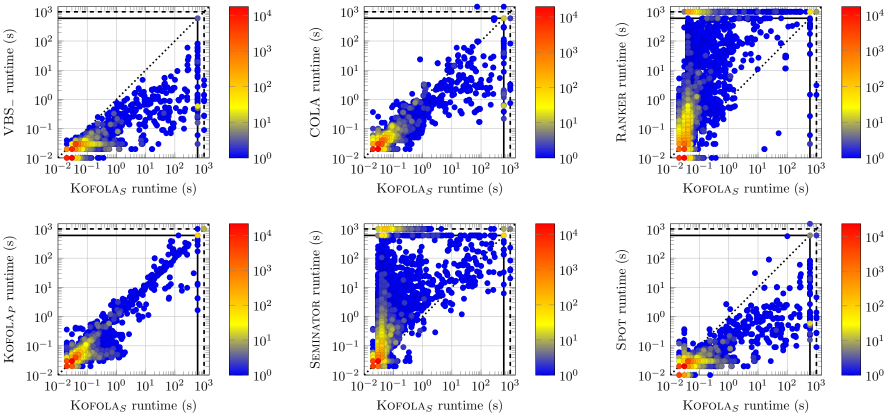
- Open framework for BA complementation
- Different algorithm for each SCC
- Exponentially better upper bound for elevator automata
- Future work
  - Smart ways to choose algorithms based on SCC properties
  - Other algorithms for NACs
  - Language inclusion testing

THANK YOU!

States


States VBS_
States COLA
States RANKER
States KOFOLA$_P$
States SEMINATOR
States SPOT
States KOFOLA$_S$
10^1
10^3
10^5
10^0
10^2
10^4

Runtimes


VBS_ runtime (s)
COLA runtime (s)
RANKER runtime (s)
KOFOLA$_P$ runtime (s)
SEMINATOR runtime (s)
SPOT runtime (s)
KOFOLA$_S$ runtime (s)
10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} 10^{3}
10^{0} 10^{1} 10^{2} 10^{3} 10^{4}