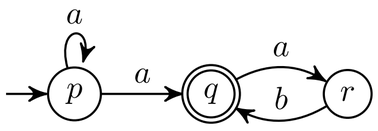
18 - Efficient Complementation of Elevator Automata

Barbora Šmahlíková

Supervisor: Ing. Ondřej Lengál, Ph.D., Consultant: Ing. Vojtěch Havlena, Ph.D.
Published at TACAS'22

Excel@FIT 2022

Büchi Automata



Büchi automaton (BA) accepting infinite words starting with symbol a, followed by a finite number of symbol a and an infinite sequence of symbols ab

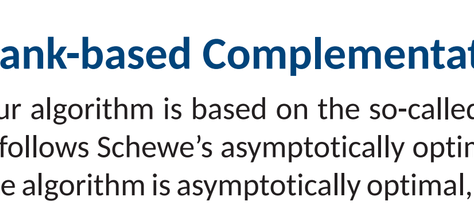
$\mathcal{A} = (Q, \Sigma, \delta, I, F)$ with

- a finite set of states Q,
- an alphabet Σ,
- a transition function $\delta\colon Q \times \Sigma \to 2^Q$,
- a set of initial states $I \subseteq Q$, and
- a set of accepting states $F \subseteq Q$.

Automata over infinite words accepted by looping over some accepting states infinitely often

Elevator Automata

- Large class of Büchi automata occuring often in practice
- Each strongly connected component (SCC) is of one of the following types:
  - **deterministic**,
  - **inherently weak accepting** (accepting state on every cycle),
  - **nonaccepting** (with no accepting state).
- Can be complemented more efficiently than general Büchi automata thanks to their specific structure



An example of elevator automaton with one nonaccepting (N) and one inherently weak accepting (IWA) SCC

Complementing Büchi Automata

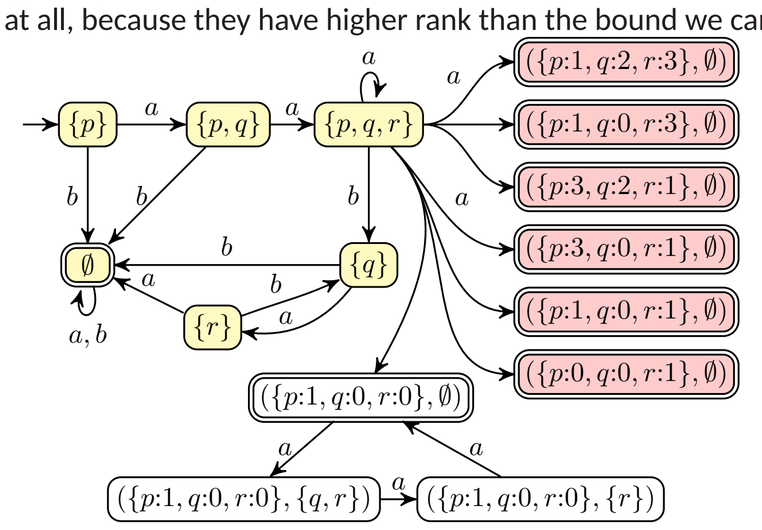
Büchi automata complementation is a crucial operation for decision procedures of various logics, such as the monadic second-order logic S1S or temporal logics ETL and QPTL, as well as for language inclusion, equivalence testing, model checking of temporal properties, or termination analysis of programs. Due to the high space complexity of BA complementation, many different approaches and optimizations have been introduced since the original construction by Büchi was presented in the 1960s. The theoretical lower bound of BA complementation is $(0.76n)^n$.

Rank-based Complementation

Our algorithm is based on the so-called rank-based complementation approach and it follows Schewe's asymptotically optimal complementation algorithm. Even though the algorithm is asymptotically optimal, it can still generate a lot of unnecessary states and transitions. It is therefore desirable to reduce the generated state space as much as possible.

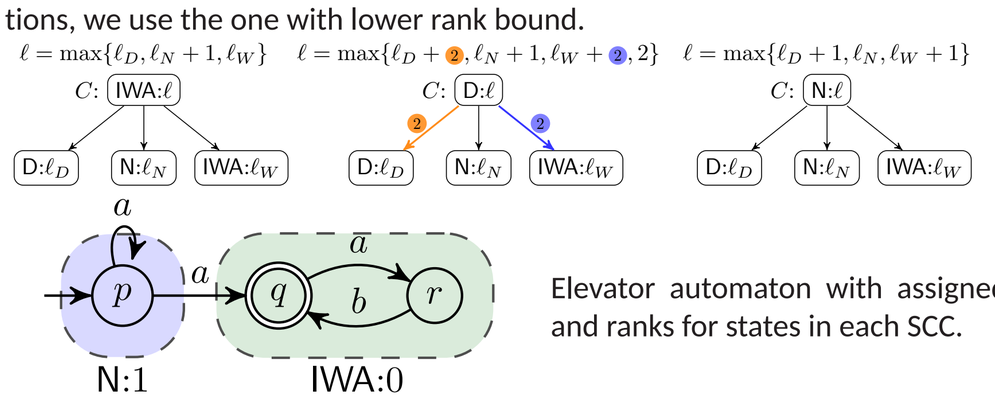
For a given Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, a *level ranking* is a function $f\colon Q\{0, 1, \ldots, 2|Q|\}$ such that $\{f(q_f) \mid q_f \in F\} \subseteq \{0, 2, \ldots, 2|Q|\}$, i.e., f maps all accepting states of $\mathcal{A}$ to even ranks. Given a set of states $S \subseteq Q$, a (level) ranking $f\colon Q\{0, 1, \ldots, 2|Q|\}$ is called *S-tight* if it has an odd rank r, $\{f(s) \mid s \in S\} \supseteq \{1, 3, \ldots, r\}$, and $\{f(q) \mid q \notin S\} = \{0\}$. A ranking is *tight* if it is Q-tight.

The number of successors of a given state in the complement corresponds to the number of possible **tight rankings**, which depends on the factorial of maximum rank. For general BAs, maximum rank for every currently reachable state can be bounded to $2|Q| - 1$. Thanks to the structure of elevator BAs, we can, however, set the bound on the maximum rank more specifically for each state, and thus reduce the number of successors and the generated state space. In the automaton below, which is a complement of the BA shown in the previous pictures, out of 7 possible successors of state $\{p, q, r\}$, only one is generated. The red states and their successors are not generated at all, because they have higher rank than the bound we can get from our algorithm.



Elevator Rules

We can assign type and maximum rank for states in each SCC of an elevator automaton. For terminal SCCs, we assign type IWA and rank 0 to inherently weak accepting SCCs and type D and rank 2 otherwise. For non-terminal SCCs, we use the rules shown below. We can assign the type and rank to an SCC only if they were already assigned to all its successors. Children nodes denote already processed successive SCCs. In particular, a child node of the form k:ℓ_k denotes an aggregate of all siblings of the type k with ℓ_k being the maximum rank of these siblings. The symbols 2 and 2 are interpeted as 0 if all the corresponding edges from the components having rank ℓ_D and ℓ_W, respectively, are deterministic; otherwise they are interpreted as 2. Transitions between two components C_1 and C_2 are deterministic if there are either transitions only to the same SCC or only to other SCCs for a given state and symbol. If there are more options, we use the one with lower rank bound.



Elevator automaton with assigned types and ranks for states in each SCC.

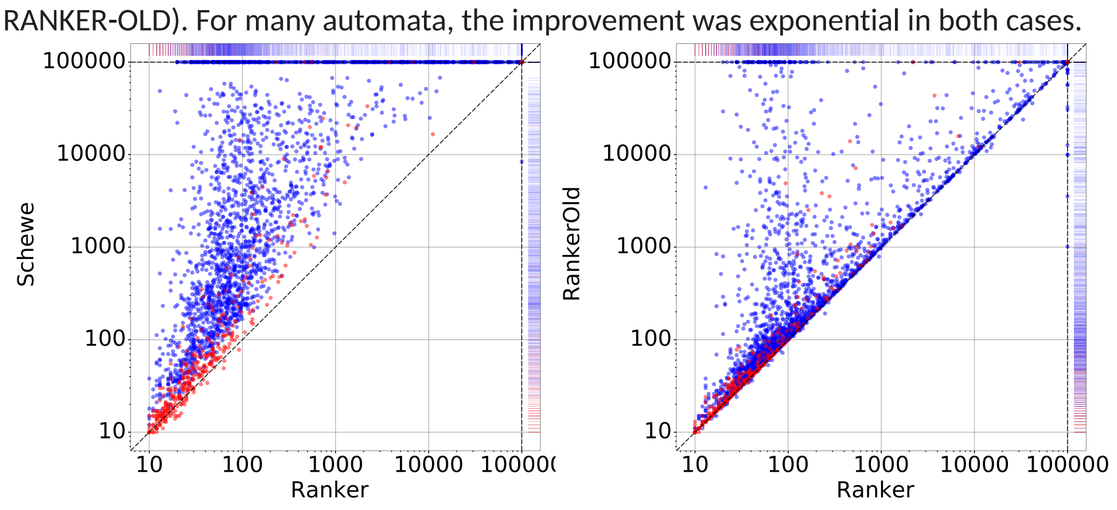
This algorithm can also be extended to general BAs containing at least one deterministic, nonaccepting or inherently weak accepting SCC.

Experiments

The optimization for complementing elevator automata was implemented as an extension of the tool RANKER in C++. We used two datasets for our experiments: random with randomly generated BAs over a two letter alphabet, and LTL with automata over larger alphabets (up to 128 symbols) obtained from LTL formulae. Our benchmark contains 2592 random and 414 LTL BAs. Of all 3006 BAs, 458 were elevator automata. The following scatterplots show comparison with other state-of-the-art tools for BA complementation. Blue data points are from random and red from LTL. Axes are logarithmic.

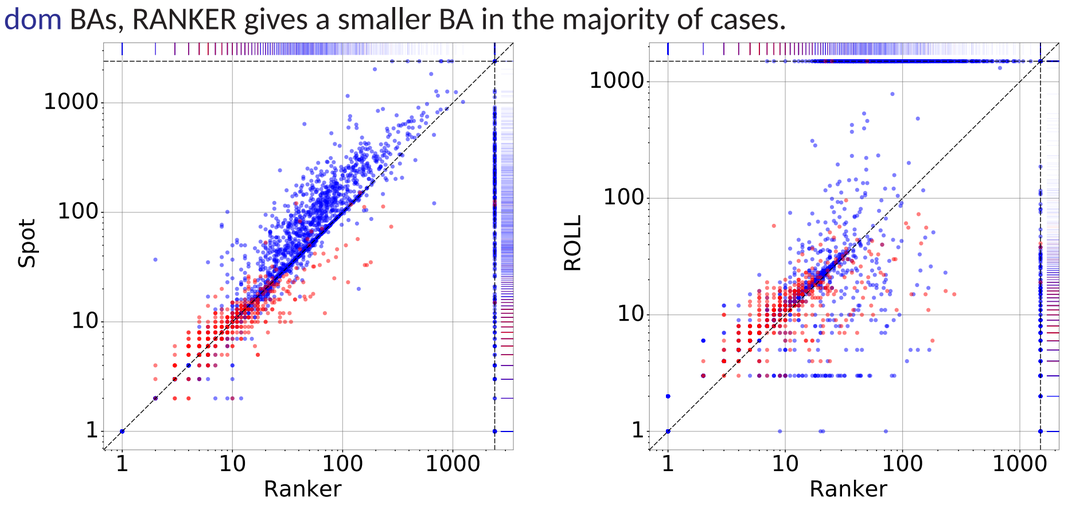
Rank-based complementation

We compared RANKER with other rank-based procedures, namely with Schewe's complementation algorithm, and with the previous version of RANKER (denoted as RANKER-OLD). For many automata, the improvement was exponential in both cases.



Other approaches

We also compared RANKER with other state-of-the-art tools with different complementation approaches, namely GOAL, SPOT, SEMINATOR 2, LTL2DSTAR and ROLL. The scatterplots below show comparison of RANKER with two most competitive tools - SPOT and ROLL. SPOT behaves slightly better on BAs from LTL, but especially for random BAs, RANKER gives a smaller BA in the majority of cases.



Comparison with other state-of-the-art tools

The table below shows comparison of RANKER with other state-of-the-art tools. The values are given for three datasets as all (random : LTL). RANKER gives more wins than losses in comparison with any other tool (i.e., it produces strictly smaller BA more often than a BA with more states).

method	mean	median	wins	losses	mean runtime [s]	median runtime [s]	timeouts
RANKER	3812 (4452 : 207)	79 (93 : 26)			7.83 (8.99 : 1.30)	0.51 (0.84 : 0.04)	279 (276 : 3)
RANKER$_{\text{OLD}}$	7398 (8688 : 358)	141 (197 : 29)	2190 (2011 : 179)	111 (107 : 4)	9.37 (10.73 : 1.99)	0.61 (1.04 : 0.04)	365 (360 : 5)
SCHEWE	4550 (5495 : 665)	439 (774 : 35)	2640 (2315 : 325)	55 (1 : 54)	21.05 (24.28 : 7.80)	6.57 (7.39 : 5.21)	937 (928 : 9)
RANKER	47 (52 : 18)	22 (27 : 10)			7.83 (8.99 : 1.30)	0.51 (0.84 : 0.04)	279 (276 : 3)
PITERMAN	73 (82 : 22)	28 (34 : 14)	1435 (1124 : 311)	416 (360 : 56)	7.29 (7.39 : 6.65)	5.99 (6.04 : 5.62)	14 (12 : 2)
SAFRA	83 (91 : 30)	29 (35 : 17)	1562 (1211 : 351)	387 (350 : 37)	14.11 (15.05 : 8.37)	6.71 (6.92 : 5.79)	172 (158 : 14)
SPOT	75 (85 : 15)	24 (32 : 10)	1087 (936 : 151)	683 (501 : 182)	0.86 (0.99 : 0.06)	0.02 (0.02 : 0.02)	13 (13 : 0)
FRIBOURG	91 (104 : 13)	23 (31 : 9)	1120 (1055 : 65)	601 (376 : 225)	17.79 (19.53 : 7.22)	9.25 (10.15 : 5.48)	81 (80 : 1)
LTL2DSTAR	73 (82 : 21)	28 (34 : 13)	1465 (1195 : 270)	465 (383 : 82)	3.31 (3.84 : 0.11)	0.04 (0.05 : 0.02)	136 (130 : 6)
SEMINATOR 2	79 (91 : 15)	21 (29 : 10)	1266 (1131 : 135)	571 (367 : 204)	9.51 (11.25 : 0.08)	0.22 (0.39 : 0.02)	363 (362 : 1)
ROLL	18 (19 : 14)	10 (9 : 11)	2116 (1858 : 258)	569 (443 : 126)	31.23 (37.85 : 7.28)	8.19 (12.23 : 2.74)	1109 (1106 : 3)