Empirical Performance Investigation of a Büchi Complementation Construction

Master's Thesis Presentation

Daniel Weibel

Foundations of Dependable Systems Group
Department of Informatics
University of Fribourg
daniel.weibel@unifr.ch

19 August 2015

UNI FR

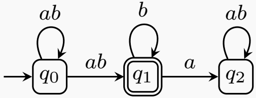
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Outline

1. **Introduction**
2. **Implementation**
3. **Study Setup**
4. **Results**

Basics

Büchi automata



- Finite state automata running on infinite words (ω-words) $\in \Sigma^\omega$
- A word is accepted if it has an accepting run
- A run is accepting if it visits an accepting state infinitely often

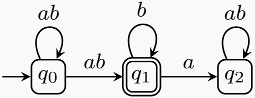
Büchi complementation

The complement of a Büchi automaton A is another Büchi automaton B, such that:

B accepts a word if and only if it is not accepted by A

Basics

Büchi automata



- Finite state automata running on infinite words (ω-words) $\in \Sigma^\omega$
- A word is accepted if it has an accepting run
- A run is accepting if it visits an accepting state infinitely often

Büchi complementation

The complement of a Büchi automaton A is another Büchi automaton B, such that:

B accepts a word if and only if it is not accepted by A

State Complexity of Büchi Complementation

A (n states) $\longrightarrow$ Büchi complementation construction $\longrightarrow$ B (m states)

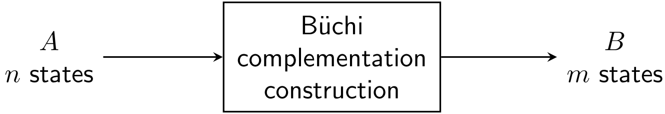
- State complexity: m in relation to n
  - Size of complement in relation to size of input automaton
- Also known as *state growth*, *state blow-up*, or *state explosion*
- Can be **very high**
- Inhibits the application of Büchi complemetation in practice
  - E.g. in automata-theoretic model checking
- The lower the state complexity, the higher the performance of the construction
- Importance to investigate the state complexity of Büchi complementation constructions (see next slides)

State Complexity of Büchi Complementation

A (n states) $\longrightarrow$ Büchi complementation construction $\longrightarrow$ B (m states)

- State complexity: m in relation to n
  - Size of complement in relation to size of input automaton
- Also known as *state growth*, *state blow-up*, or *state explosion*
- Can be **very high**
- Inhibits the application of Büchi complemetation in practice
  - E.g. in automata-theoretic model checking
- The lower the state complexity, the higher the performance of the construction
- Importance to investigate the state complexity of Büchi complementation constructions (see next slides)

State Complexity of Büchi Complementation



- State complexity: m in relation to n
  - Size of complement in relation to size of input automaton
- Also known as *state growth*, *state blow-up*, or *state explosion*
- Can be **very high**
- Inhibits the application of Büchi complemetation in practice
  - E.g. in automata-theoretic model checking
- The lower the state complexity, the higher the performance of the construction
- Importance to investigate the state complexity of Büchi complementation constructions (see next slides)

State Complexity of Büchi Complementation

A n states → Büchi complementation construction → B m states

- State complexity: m in relation to n
  - Size of complement in relation to size of input automaton
- Also known as *state growth*, *state blow-up*, or *state explosion*
- Can be **very high**
- Inhibits the application of Büchi complemetation in practice
  - E.g. in automata-theoretic model checking
- The lower the state complexity, the higher the performance of the construction
- Importance to investigate the state complexity of Büchi complementation constructions (see next slides)

Worst-Case State Complexity

- Every construction has a specific worst-case state complexity
- Maximum number of states that a construction can produce
- Examples:

Complementation construction	Worst-case state complexity	Example value with $n = 15$
[Büchi, 1962]	$2^{2^{O(n)}}$	$1.4 \times 10^{9,864}$
[Piterman, 2007]	$O(n^{2n})$	1.9×10^{35}
[Vardi and Wilke, 2007]	$O((3n)^n)$	6.3×10^{24}
[Schewe, 2009]	$O((0.76n)^n)$	7.1×10^{15}

- Often used to assess the performance or efficiency of a construction, but. . .

Worst-Case State Complexity

- Every construction has a specific worst-case state complexity
- Maximum number of states that a construction can produce
- Examples:

Complementation construction	Worst-case state complexity	Example value with $n = 15$
[Büchi, 1962]	$2^{2^{O(n)}}$	$1.4 \times 10^{9,864}$
[Piterman, 2007]	$O(n^{2n})$	1.9×10^{35}
[Vardi and Wilke, 2007]	$O((3n)^n)$	6.3×10^{24}
[Schewe, 2009]	$O((0.76n)^n)$	7.1×10^{15}

- Often used to assess the performance or efficiency of a construction, but...

Worst-Case State Complexity

- Every construction has a specific worst-case state complexity
- Maximum number of states that a construction can produce
- Examples:

Complementation construction	Worst-case state complexity	Example value with $n = 15$
[Büchi, 1962]	$2^{2^{O(n)}}$	$1.4 \times 10^{9,864}$
[Piterman, 2007]	$O(n^{2n})$	1.9×10^{35}
[Vardi and Wilke, 2007]	$O((3n)^n)$	6.3×10^{24}
[Schewe, 2009]	$O((0.76n)^n)$	7.1×10^{15}

- Often used to assess the performance or efficiency of a construction, but...

Empirical Way to Investigate Performance

- Worst-case state complexity reflects only a **small** aspect of the state complexity of a construction
- From a practical point of view, we are interested in the performance of a construction on **concrete** automata
  - E.g. how does the construction perform on "typical" automata with 15 states?
- Such insights can be gained by **empirical** investigations
- Empirical performance investigation:
  1. **Implement construction**
  2. **Run the implementation on test automata**
  3. **Analyse generated complements**
- Aim of this thesis:
  - Empirically investigate the performance of the **Fribourg construction** (see next slide)

Empirical Way to Investigate Performance

- Worst-case state complexity reflects only a **small** aspect of the state complexity of a construction
- From a practical point of view, we are interested in the performance of a construction on **concrete** automata
    - E.g. how does the construction perform on "typical" automata with 15 states?
- Such insights can be gained by **empirical** investigations
- Empirical performance investigation:
    1. **Implement construction**
    2. **Run the implementation on test automata**
    3. **Analyse generated complements**
- Aim of this thesis:
    - Empirically investigate the performance of the **Fribourg construction** (see next slide)

Empirical Way to Investigate Performance

- Worst-case state complexity reflects only a **small** aspect of the state complexity of a construction
- From a practical point of view, we are interested in the performance of a construction on **concrete** automata
  - E.g. how does the construction perform on "typical" automata with 15 states?
- Such insights can be gained by **empirical** investigations
- Empirical performance investigation:
  1. **Implement construction**
  2. **Run the implementation on test automata**
  3. **Analyse generated complements**
- Aim of this thesis:
  - Empirically investigate the performance of the **Fribourg construction** (see next slide)

Empirical Way to Investigate Performance

- Worst-case state complexity reflects only a **small** aspect of the state complexity of a construction
- From a practical point of view, we are interested in the performance of a construction on **concrete** automata
  - E.g. how does the construction perform on "typical" automata with 15 states?
- Such insights can be gained by **empirical** investigations
- Empirical performance investigation:
  1. **Implement construction**
  2. **Run the implementation on test automata**
  3. **Analyse generated complements**
- Aim of this thesis:
  - Empirically investigate the performance of the **Fribourg construction** (see next slide)

The Fribourg Construction

- Described by [Allred and Ultes-Nitsche, 2014]
- Slice-based complementation construction
  - See main complementation approaches: *Ramsey-based*, *determinisation-based*, *rank-based*, and *slice-based*
- Worst-case state complexity: $O((1.59n)^n)$
- Optimisations:

R2C If input automaton is complete, remove states whose rightmost component is 2-coloured

M1 Merge certain pairs of adjacent components
  - Worst-case state complexity: $O((1.195n)^n)$

M2 Keep only one 2-coloured component in a state
  - Worst-case state complexity: $O((0.86n)^n)$

The Fribourg Construction

- Described by [Allred and Ultes-Nitsche, 2014]
- Slice-based complementation construction
  - See main complementation approaches: *Ramsey-based*, *determinisation-based*, *rank-based*, and *slice-based*
- Worst-case state complexity: $O((1.59n)^n)$
- Optimisations:

R2C If input automaton is complete, remove states whose rightmost component is 2-coloured

M1 Merge certain pairs of adjacent components

- Worst-case state complexity: $O((1.195n)^n)$

M2 Keep only one 2-coloured component in a state

- Worst-case state complexity: $O((0.86n)^n)$

The Fribourg Construction

- Described by [Allred and Ultes-Nitsche, 2014]
- Slice-based complementation construction
  - See main complementation approaches: *Ramsey-based*, *determinisation-based*, *rank-based*, and *slice-based*
- Worst-case state complexity: $O((1.59n)^n)$
- Optimisations:

R2C If input automaton is complete, remove states whose rightmost component is 2-coloured

M1 Merge certain pairs of adjacent components
  - Worst-case state complexity: $O((1.195n)^n)$

M2 Keep only one 2-coloured component in a state
  - Worst-case state complexity: $O((0.86n)^n)$

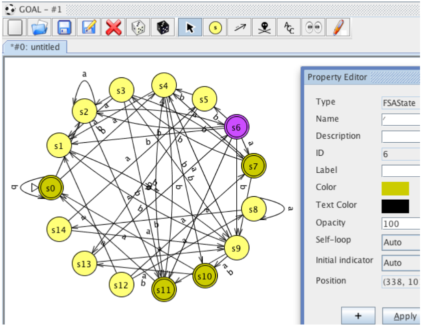
Roadmap

1. Introduction
2. **Implementation**
3. Study Setup
4. Results

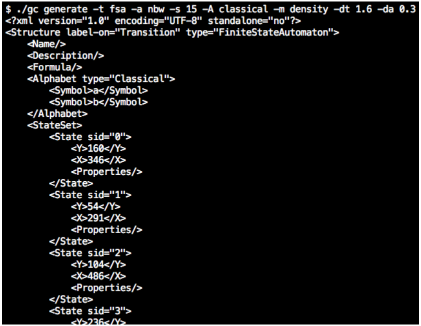
GOAL

- **G**raphical Tool for **O**mega-**A**utomata and **L**ogics
- http://goal.im.ntu.edu.tw/wiki/doku.php
- Allows to create and manipulate ω-automata

Graphical user interface



Command line interface



```
$ ./gc generate -t fsa -a nbw -s 15 -A classical -m density -dt 1.6 -da 0.3
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Structure label-on="Transition" type="FiniteStateAutomaton">
    <Name/>
    <Description/>
    <Formula/>
    <Alphabet type="Classical">
        <Symbol>a</Symbol>
        <Symbol>b</Symbol>
    </Alphabet>
    <StateSet>
        <State sid="0">
            <Y>160</Y>
            <X>346</X>
            <Properties/>
        </State>
        <State sid="1">
            <Y>54</Y>
            <X>291</X>
            <Properties/>
        </State>
        <State sid="2">
            <Y>104</Y>
            <X>486</X>
            <Properties/>
        </State>
        <State sid="3">
            <Y>236</Y>
```

GOAL: Büchi Complementation Constructions

- GOAL contains implementations of several complementation constructions (version 2014–11–17)

GOAL Name	Reference
Ramsey	[Sistla et al., 1985, Sistla et al., 1987]
Safra	[Safra, 1988a, Safra, 1988b]
MS	[Muller and Schupp, 1995]
ModifiedSafra	[Althoff et al., 2006]
Piterman	[Piterman, 2006, Piterman, 2007]
WAA	[Kupferman and Vardi, 1997, Kupferman and Vardi, 2001]
WAPA	[Thomas, 1999]
Rank	[Schewe, 2009]
Slice+P	[Vardi and Wilke, 2007]
Slice	[Kähler and Wilke, 2008]

Fribourg Construction Plugin for GOAL

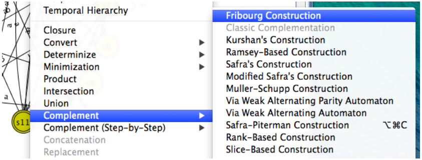
- GOAL is built with the Java Plugin Framework (JPF)[1]
  - Allows to create **plugins** containing **extensions** for pre-defined **extension points**
- Our plugin adds the Fribourg construction to GOAL
  - ... in a fully integrated way
- Download: https://frico.s3.amazonaws.com/goal_plugins/ch.unifr.goal.complement.zip

[1] http://jpf.sourceforge.net/

Fribourg Construction Plugin for GOAL

- GOAL is built with the Java Plugin Framework (JPF)[1]
  - Allows to create **plugins** containing **extensions** for pre-defined **extension points**
- Our plugin adds the Fribourg construction to GOAL
  - . . . in a fully integrated way

Graphical user interface



Command line interface

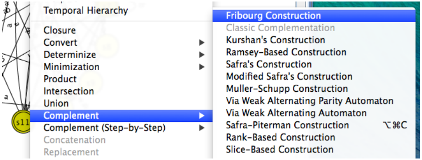
- Download: https://frico.s3.amazonaws.com/goal_plugins/ch.unifr.goal.complement.zip

[1] http://jpf.sourceforge.net/

Fribourg Construction Plugin for GOAL

- GOAL is built with the Java Plugin Framework (JPF)[1]
  - Allows to create **plugins** containing **extensions** for pre-defined **extension points**
- Our plugin adds the Fribourg construction to GOAL
  - . . . in a fully integrated way

Graphical user interface



Command line interface

- Download: https://frico.s3.amazonaws.com/goal_plugins/ch.unifr.goal.complement.zip

[1] http://jpf.sourceforge.net/

GOAL and Plugin Demo

Roadmap

1. Introduction
2. Implementation
3. **Study Setup**
4. Results

Test Data: GOAL Test Set

- Created and used by [Tsai et al., 2011]
- 11,000 automata
  - 15 states
  - Alphabet $\Sigma = \{0, 1\}$
  - 11 transition densities
    - $\mathcal{T} = (1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0)$
  - 10 acceptance densities
    - $\mathcal{A} = (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)$
  - 110 classes at 100 automata for each combination $\mathcal{T} \times \mathcal{A}$
- Analysis
  - 61.8% universal automata
  - 0.6% empty automata
  - 9.0% complete automata
- Download: https://frico.s3.amazonaws.com/test_sets/goal.zip

Test Data: GOAL Test Set

- Created and used by [Tsai et al., 2011]
- 11,000 automata
  - 15 states
  - Alphabet $\Sigma = \{0, 1\}$
  - 11 transition densities
    - $\mathcal{T} = (1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0)$
  - 10 acceptance densities
    - $\mathcal{A} = (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)$
  - 110 classes at 100 automata for each combination $\mathcal{T} \times \mathcal{A}$
- Analysis
  - 61.8% universal automata
  - 0.6% empty automata
  - 9.0% complete automata
- Download: https://frico.s3.amazonaws.com/test_sets/goal.zip

Test Data: GOAL Test Set

- Created and used by [Tsai et al., 2011]
- 11,000 automata
  - 15 states
  - Alphabet $\Sigma = \{0, 1\}$
  - 11 transition densities
    - $\mathcal{T} = (1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0)$
  - 10 acceptance densities
    - $\mathcal{A} = (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)$
  - 110 classes at 100 automata for each combination $\mathcal{T} \times \mathcal{A}$
- Analysis
  - 61.8% universal automata
  - 0.6% empty automata
  - 9.0% complete automata
- Download: https://frico.s3.amazonaws.com/test_sets/goal.zip

Test Data: Michel Test Set

- Four Michel automata
  - **Michel 1:** 3 states, 2 symbols, 5 transitions
  - **Michel 2:** 4 states, 3 symbols, 8 transitions
  - **Michel 3:** 5 states, 4 symbols, 11 transitions
  - **Michel 4:** 6 states, 5 symbols, 14 transitions

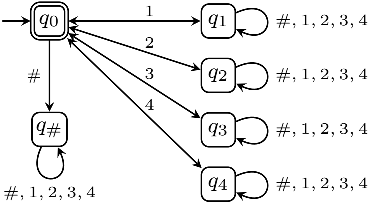
Michel 4:

- Automata used by [Michel, 1988] to prove $n!$ lower bound
- Generally provoke very high state complexity
- Download: https://frico.s3.amazonaws.com/test_sets/michel.zip

Test Data: Michel Test Set

- Four Michel automata
  - **Michel 1:** 3 states, 2 symbols, 5 transitions
  - **Michel 2:** 4 states, 3 symbols, 8 transitions
  - **Michel 3:** 5 states, 4 symbols, 11 transitions
  - **Michel 4:** 6 states, 5 symbols, 14 transitions

Michel 4:

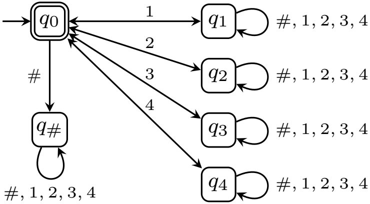


- Automata used by [Michel, 1988] to prove $n!$ lower bound
- Generally provoke very high state complexity
- Download: https://frico.s3.amazonaws.com/test_sets/michel.zip

Test Data: Michel Test Set

- Four Michel automata
  - **Michel 1:** 3 states, 2 symbols, 5 transitions
  - **Michel 2:** 4 states, 3 symbols, 8 transitions
  - **Michel 3:** 5 states, 4 symbols, 11 transitions
  - **Michel 4:** 6 states, 5 symbols, 14 transitions

Michel 4:

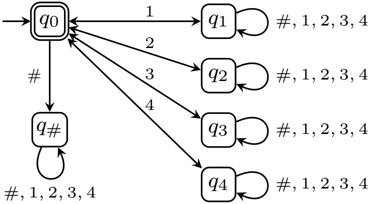


- Automata used by [Michel, 1988] to prove $n!$ lower bound
- Generally provoke very high state complexity
- Download: https://frico.s3.amazonaws.com/test_sets/michel.zip

Test Data: Michel Test Set

- Four Michel automata
  - **Michel 1:** 3 states, 2 symbols, 5 transitions
  - **Michel 2:** 4 states, 3 symbols, 8 transitions
  - **Michel 3:** 5 states, 4 symbols, 11 transitions
  - **Michel 4:** 6 states, 5 symbols, 14 transitions

Michel 4:



- Automata used by [Michel, 1988] to prove $n!$ lower bound
- Generally provoke very high state complexity
- Download: https://frico.s3.amazonaws.com/test_sets/michel.zip

Test Setup

- Internal tests
  - Compare different versions of the Fribourg construction
  - Combinations of optimisations R2C, M1, and M2
  - Further options:
    - C: make input automaton complete
    - R: remove unreachable and dead states from output automaton
- External tests
  - Compare Fribourg construction with other constructions
  - Choose best version of Fribourg construction for each test set
  - Other constructions:
    - Piterman [Piterman, 2006, Piterman, 2007]
    - Rank [Schewe, 2009]
    - Slice [Vardi and Wilke, 2007]

Test Setup

- Internal tests
  - Compare different versions of the Fribourg construction
  - Combinations of optimisations R2C, M1, and M2
  - Further options:
    - C: make input automaton complete
    - R: remove unreachable and dead states from output automaton
- External tests
  - Compare Fribourg construction with other constructions
  - Choose best version of Fribourg construction for each test set
  - Other constructions:
    - Piterman [Piterman, 2006, Piterman, 2007]
    - Rank [Schewe, 2009]
    - Slice [Vardi and Wilke, 2007]

Test Scenarios

	GOAL test set	Michel test set
Internal tests		
External tests		

Test Scenarios

	GOAL test set	Michel test set
Internal tests	– Fribourg – Fribourg+R2C – Fribourg+R2C+C – Fribourg+M1 – Fribourg+M1+R2C – Fribourg+M1+R2C+C – Fribourg+M1+M2 – Fribourg+R	
External tests		

Test Scenarios

	GOAL test set	Michel test set
Internal tests	– Fribourg – Fribourg+R2C – Fribourg+R2C+C – Fribourg+M1 – Fribourg+M1+R2C – Fribourg+M1+R2C+C – Fribourg+M1+M2 – Fribourg+R	– Fribourg – Fribourg+R2C – Fribourg+M1 – Fribourg+M1+M2 – Fribourg+M1+M2+R2C – Fribourg+R
External tests		

Test Scenarios

	GOAL test set	Michel test set
Internal tests	– Fribourg – Fribourg+R2C – Fribourg+R2C+C – Fribourg+M1 – Fribourg+M1+R2C – Fribourg+M1+R2C+C – Fribourg+M1+M2 – Fribourg+R	– Fribourg – Fribourg+R2C – Fribourg+M1 – Fribourg+M1+M2 – Fribourg+M1+M2+R2C – Fribourg+R
External tests	– Fribourg+M1+R2C – Piterman+EQ+RO – Rank+TR+RO – Slice+P+RO+MADJ+EG	

Test Scenarios

	GOAL test set	Michel test set
Internal tests	– Fribourg – Fribourg+R2C – Fribourg+R2C+C – Fribourg+M1 – Fribourg+M1+R2C – Fribourg+M1+R2C+C – Fribourg+M1+M2 – Fribourg+R	– Fribourg – Fribourg+R2C – Fribourg+M1 – Fribourg+M1+M2 – Fribourg+M1+M2+R2C – Fribourg+R
External tests	– Fribourg+M1+R2C – Piterman+EQ+RO – Rank+TR+RO – Slice+P+RO+MADJ+EG	– Fribourg+M1+M2+R2C – Piterman+EQ+RO – Rank+TR+RO – Slice+P+RO+MADJ+EG

Computational Resources

- Shorthand naming for each test scenario:

	GOAL test set	Michel test set
Internal	**IG**	**IM**
External	**EG**	**EM**

- Resource limits per complementation task (only IG and EG)
  - Time: 600 seconds (CPU time)
  - Memory: 1 GB (Java heap)
  - Result analysis based on **effective samples**
    - Automata which have been succesfully complemented by **all** constructions of a test scenario
- Execution environment: HPC cluster UBELIX (hnodes 1–42, jnodes) at the University of Bern[2]

[2] http://ubelix.unibe.ch

Computational Resources

- Shorthand naming for each test scenario:

	GOAL test set	Michel test set
Internal	**IG**	**IM**
External	**EG**	**EM**

- Resource limits per complementation task (only IG and EG)
  - Time: 600 seconds (CPU time)
  - Memory: 1 GB (Java heap)
  - Result analysis based on **effective samples**
    - Automata which have been succesfully complemented by **all** constructions of a test scenario
- Execution environment: HPC cluster UBELIX (hnodes 1–42, jnodes) at the University of Bern[2]

[2] http://ubelix.unibe.ch

Computational Resources

- Shorthand naming for each test scenario:

	GOAL test set	Michel test set
Internal	**IG**	**IM**
External	**EG**	**EM**

- Resource limits per complementation task (only IG and EG)
  - Time: 600 seconds (CPU time)
  - Memory: 1 GB (Java heap)
  - Result analysis based on **effective samples**
    - Automata which have been succesfully complemented by **all** constructions of a test scenario
- Execution environment: HPC cluster UBELIX (hnodes 1–42, jnodes) at the University of Bern[2]

[2] http://ubelix.unibe.ch

Computational Resources

- Shorthand naming for each test scenario:

	GOAL test set	Michel test set
Internal	**IG**	**IM**
External	**EG**	**EM**

- Resource limits per complementation task (only IG and EG)
  - Time: 600 seconds (CPU time)
  - Memory: 1 GB (Java heap)
  - Result analysis based on **effective samples**
    - Automata which have been succesfully complemented by **all** constructions of a test scenario
- Execution environment: HPC cluster UBELIX (hnodes 1–42, jnodes) at the University of Bern[2]

[2] http://ubelix.unibe.ch

Roadmap

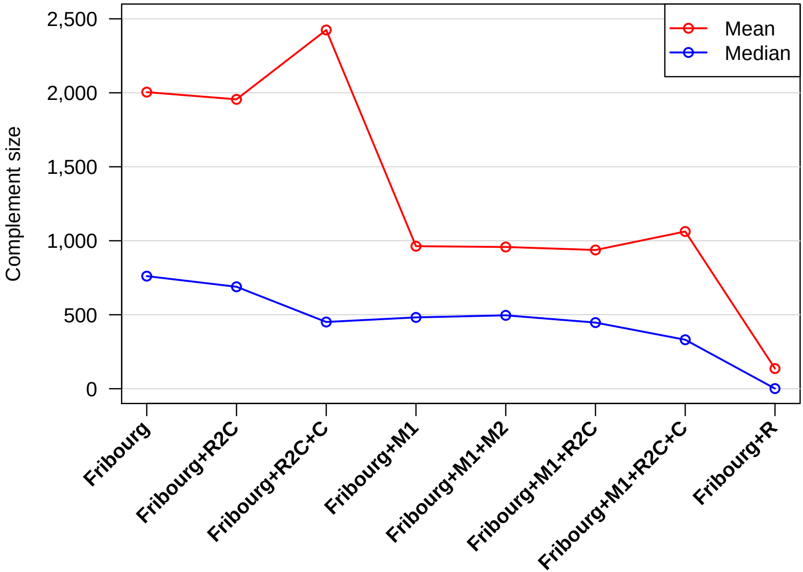
1. Introduction
2. Implementation
3. Study Setup
4. **Results**

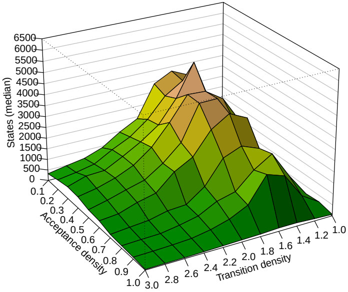
Results: Internal Tests on GOAL Test Set

	GOAL test set	Michel test set
Internal	**IG**	**IM**
External	**EG**	**EM**

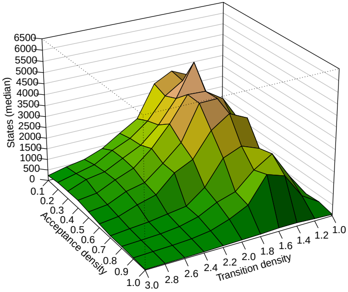
- Fribourg
- Fribourg+R2C
- Fribourg+R2C+C
- Fribourg+M1
- Fribourg+M1+R2C
- Fribourg+M1+R2C+C
- Fribourg+M1+M2
- Fribourg+R

IG: Complement Sizes (10,939 Eff. Samples)

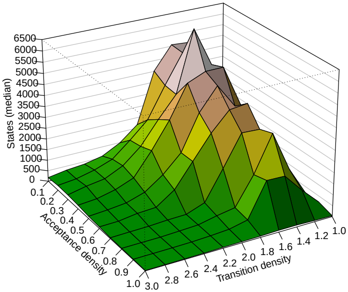




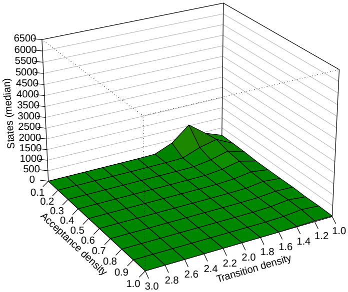
Fribourg



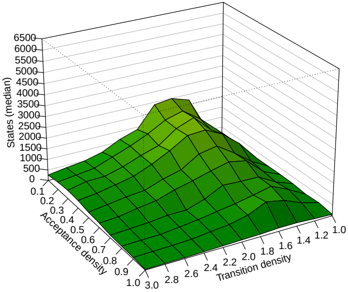
Fribourg+R2C



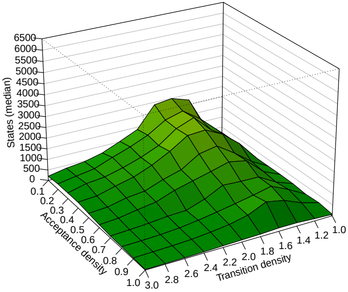
Fribourg+R2C+C



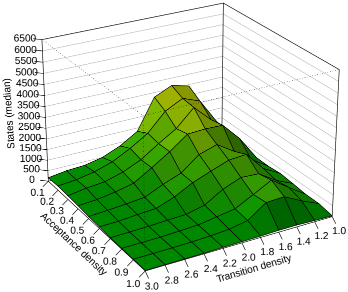
Fribourg+R



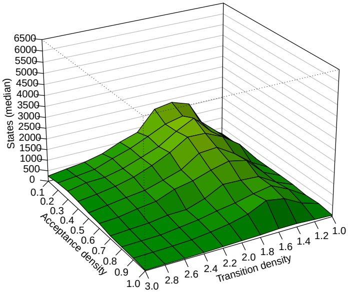
Fribourg+M1



Fribourg+M1+R2C



Fribourg+M1+R2C+C



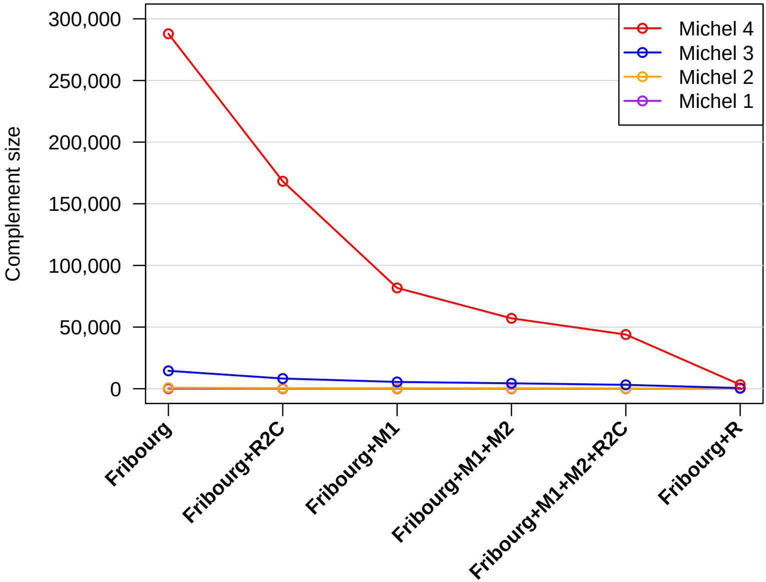
Fribourg+M1+M2

Results: Internal Tests on Michel Test Set

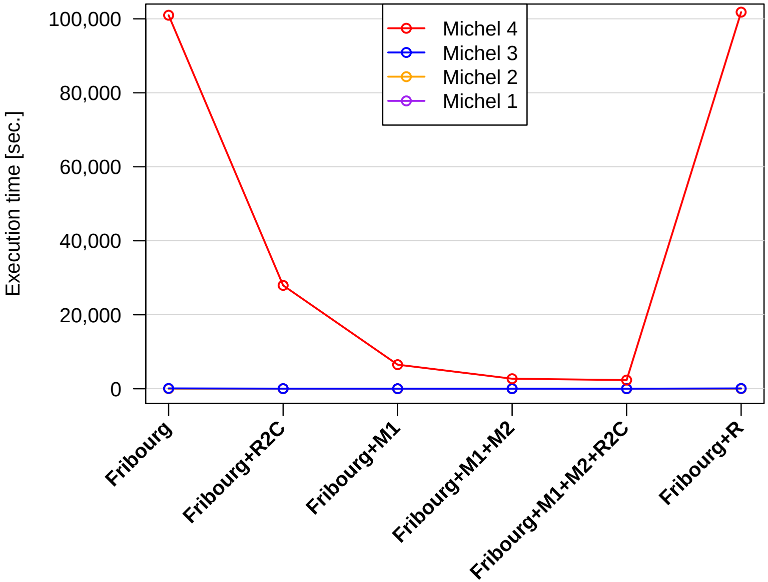
	GOAL test set	Michel test set
Internal	**IG**	**IM**
External	**EG**	**EM**

- Fribourg
- Fribourg+R2C
- Fribourg+M1
- Fribourg+M1+M2
- Fribourg+M1+M2+R2C
- Fribourg+R

IM: Complement Sizes



IM: Execution times

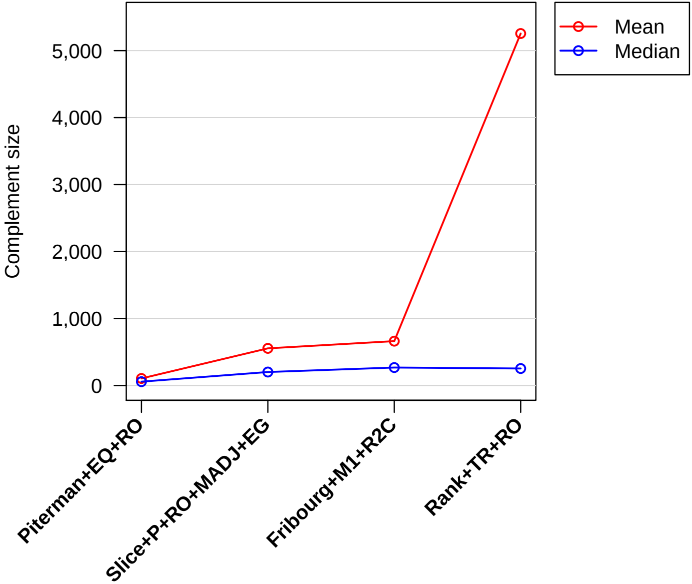


Results: External Tests on GOAL Test Set

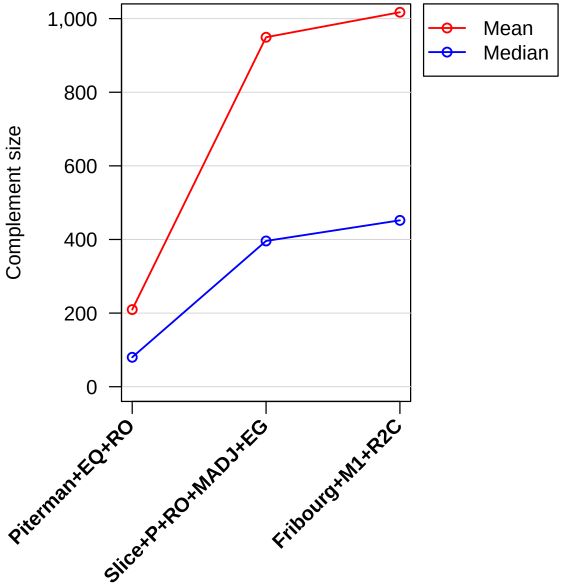
	GOAL test set	Michel test set
Internal	**IG**	**IM**
External	**EG**	**EM**

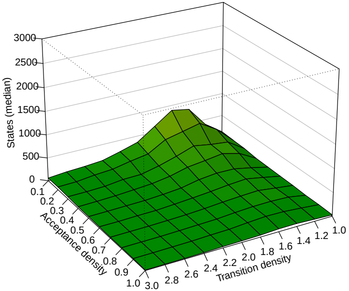
- Fribourg+M1+R2C
- Piterman+EQ+RO
- Rank+TR+RO
- Slice+P+RO+MADJ+EG

EG: Complement Sizes (7,204 Eff. Samples)

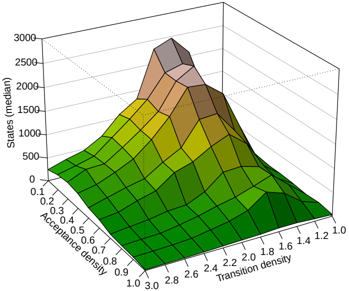


EG: Complement Sizes (10,998 Eff. Samples)

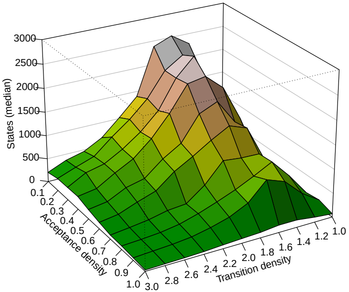




Piterman+EQ+RO



Slice+P+RO+MADJ+EG



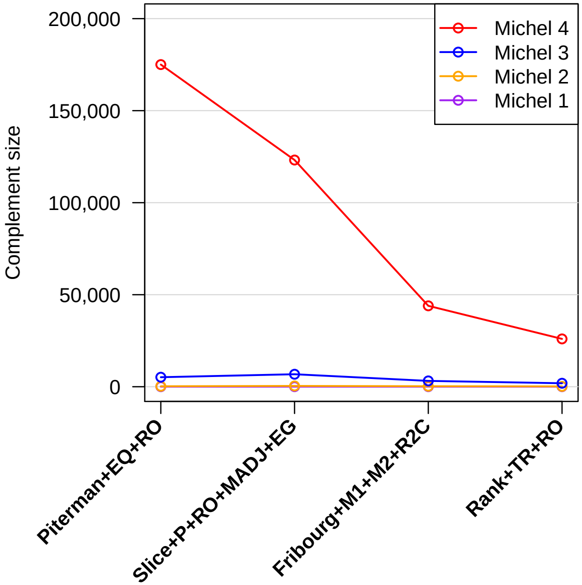
Fribourg+M1+R2C

Results: External Tests on Michel Test Set

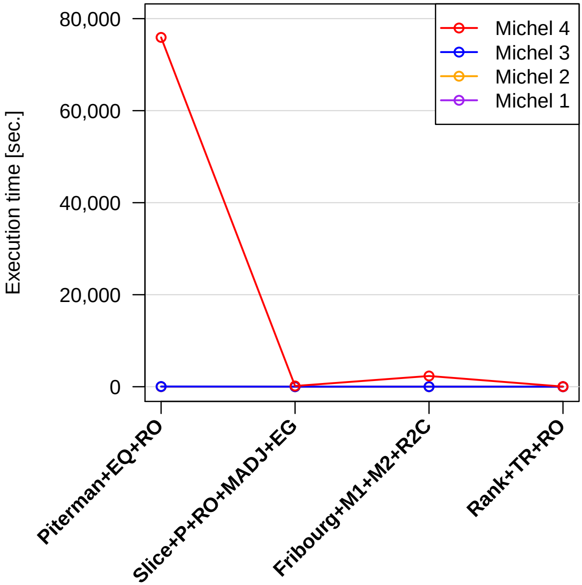
	GOAL test set	Michel test set
Internal	**IG**	**IM**
External	**EG**	**EM**

- Fribourg+M1+M2+R2C
- Piterman+EQ+RO
- Rank+TR+RO
- Slice+P+RO+MADJ+EG

EM: Complement Sizes



EM: Execution times



Conclusions

The End

Thank you very much for listening!

Thank you
for listening!

References I

[Allred and Ultes-Nitsche, 2014] Allred, J. and Ultes-Nitsche, U. (2014).
Complementing büchi automata with a subset-tuple construction.
Technical report, University of Fribourg, Switzerland.

[Althoff et al., 2006] Althoff, C., Thomas, W., and Wallmeier, N. (2006).
Observations on determinization of büchi automata.
In Farré, J., Litovsky, I., and Schmitz, S., editors, *Implementation and Application of Automata*, volume 3845 of *Lecture Notes in Computer Science*, pages 262–272. Springer Berlin Heidelberg.

[Büchi, 1962] Büchi, J. R. (1962).
On a decision method in restricted second order arithmetic.
In *Proc. International Congress on Logic, Method, and Philosophy of Science, 1960*. Stanford University Press.

[Kähler and Wilke, 2008] Kähler, D. and Wilke, T. (2008).
Complementation, disambiguation, and determinization of büchi automata unified.
In Aceto, L., Damgård, I., Goldberg, L., Halldórsson, M., Ingólfsdóttir, A., and Walukiewicz, I., editors, *Automata, Languages and Programming*, volume 5125 of *Lecture Notes in Computer Science*, pages 724–735. Springer Berlin Heidelberg.

[Kupferman and Vardi, 1997] Kupferman, O. and Vardi, M. Y. (1997).
Weak alternating automata are not that weak.
In *Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press.

[Kupferman and Vardi, 2001] Kupferman, O. and Vardi, M. Y. (2001).
Weak alternating automata are not that weak.
ACM Trans. Comput. Logic, 2(3):408–429.

[Michel, 1988] Michel, M. (1988).
Complementation is more difficult with automata on infinite words.
CNET, Paris, 15.

[Muller and Schupp, 1995] Muller, D. E. and Schupp, P. E. (1995).
Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of rabin, mcnaughton and safra.
Theoretical Computer Science, 141(1–2):69 – 107.

[Piterman, 2006] Piterman, N. (2006).
From nondeterministic buchi and streett automata to deterministic parity automata.
In *Logic in Computer Science, 2006 21st Annual IEEE Symposium on*, pages 255–264.

[Piterman, 2007] Piterman, N. (2007).
From nondeterministic buchi and streett automata to deterministic parity automata.
Logical Methods in Computer Science, 3(5):1–21.

[Safra, 1988a] Safra, S. (1988a).
On the complexity of omega-automata.
Journal of Computer and System Science.

[Safra, 1988b] Safra, S. (1988b).
On the complexity of omega-automata.
In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 319–327.

[Schewe, 2009] Schewe, S. (2009).
Büchi complementation made tight.
In *26th International Symposium on Theoretical Aspects of Computer Science-STACS 2009*, pages 661–672.

[Sistla et al., 1985] Sistla, A., Vardi, M., and Wolper, P. (1985).
The complementation problem for büchi automata with applications to temporal logic.
In Brauer, W., editor, *Automata, Languages and Programming*, volume 194 of *Lecture Notes in Computer Science*, pages 465–474. Springer Berlin Heidelberg.

References III

[Sistla et al., 1987] Sistla, A. P., Vardi, M. Y., and Wolper, P. (1987).
The complementation problem for büchi automata with applications to temporal logic.
Theoretical Computer Science, 49(2–3):217 – 237.

[Thomas, 1999] Thomas, W. (1999).
Complementation of büchi automata revisited.
In Karhumäki, J., Maurer, H., Păun, G., and Rozenberg, G., editors, *Jewels are Forever*, pages 109–120. Springer Berlin Heidelberg.

[Tsai et al., 2011] Tsai, M.-H., Fogarty, S., Vardi, M., and Tsay, Y.-K. (2011).
State of büchi complementation.
In Domaratzki, M. and Salomaa, K., editors, *Implementation and Application of Automata*, volume 6482 of *Lecture Notes in Computer Science*, pages 261–271. Springer Berlin Heidelberg.

[Vardi and Wilke, 2007] Vardi, M. Y. and Wilke, T. (2007).
Automata: From logics to algorithms.
In Flum, J., Grädel, E., and Wilke, T., editors, *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 629–736. Amsterdam University Press.