Implementation of an Algorithm for Büchi complementation

Bachelor Thesis

November 12, 2013

Christian Göttel
christian.goettel@unifr.ch

Department of Informatics
University of Fribourg, Switzerland

Outline

Büchi automata

New slice based complementation algorithm

Implementation

Evaluation

Büchi automata

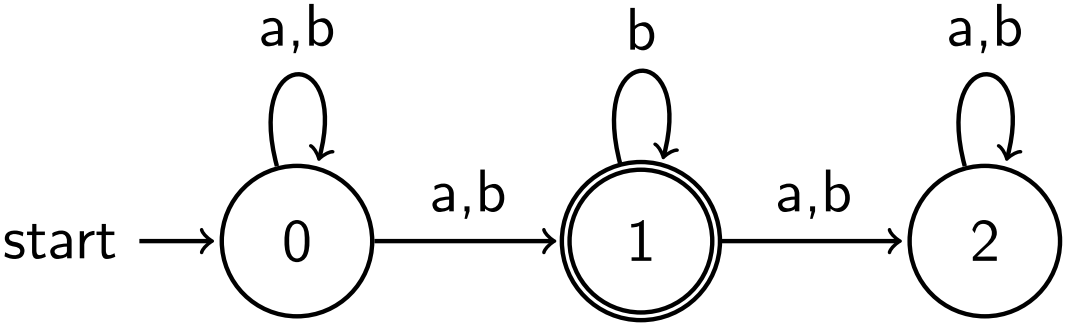
Julius Richard Büchi

Figure: J. R. Büchi 1983 [1]

- 1924 - 1984
- Swiss mathematician
- has influenced theoretical computer science
- inventor of the *Büchi automaton*

[1] http://static.classora.com/files/uploads/images/entries/581092/main.jpg

Büchi automata



Definition

A non-deterministic or deterministic Büchi automaton recognises infinite words of the ω-regular language and is defined by the following 5-tuple: $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$

- Q is the finite set of all states
- Σ is the alphabet
- δ is the transition function $\delta : Q \times \Sigma \rightarrow 2^Q$
- q_0 is the initial state with $q_0 \in Q$
- F is the set of accepting states with $F \subseteq Q$

Büchi automata

Definition

A run sequence ρ is a sequence of visited states q_i during a run of an automaton $\mathcal{A}$ on a word w.

Definition

A Büchi automaton $\mathcal{A}$ accepts an ω-word w if

$$\inf(\rho) \cap F \neq \emptyset, \quad F \subseteq Q$$

is satisfied.

Büchi complementation

My bachelor thesis is based on the paper *"State of Büchi Complementation"* written by Ming-Hsien Tsai, Seth Fogarty, Moshe Y. Vardi and Yih-Kuen Tsay.
Büchi complementation algorithms can be categorised into four different approaches:

- Ramsey-based approach
- Determinization-based approach
- Rank-based approach
- Slice-based approach

Büchi complementation

- Ramsey-based approach
  - $2^{2^{O(n)}}$ (Büchi, 1960)
  - $2^{O(n^2)}$ (Sistla, Vardi and Wolper, 1987)
- Determinization-based approach
  - $2^{O(n \log n)}$ (Safra, 1988)
  - n^{2n} (Piterman, 2007)
- Rank-based approach
  - $2^{O(n \log n)}$ (Klarlund, 1991) (Kupferman and Vardi, 2001)
  - $O\left((0.76n)^n\right)$ (Schewe, 2009)
- Slice-based approach
  - $4\,(3n)^n$ (Kähler and Wilke, 2008)
  - ? (Ultes-Nitsche and Allred, 2013)

The complementation algorithm

New slice-based complementation algorithm:

- Finite part
  - determinization by *modified subset construction*
  - splitting of *mixed sets of states*
  - state removal
- Infinite part
  - determinization by *modified subset construction*
  - splitting of *mixed sets of states*
  - state removal
  - colouring
  - acceptance condition

The complementation algorithm

Definition

A *mixed set of states* is a set of states consisting of accepting and non-accepting states

Definition

Splitting: a *mixed set of states* is split to a non-accepting set of states and an accepting set of states. The non-accepting set of states precedes the accepting set of states.
Example:

$$\{0\} \xrightarrow{\sigma} \{0,1\} \mapsto \{0\},\{1\}$$

The complementation algorithm

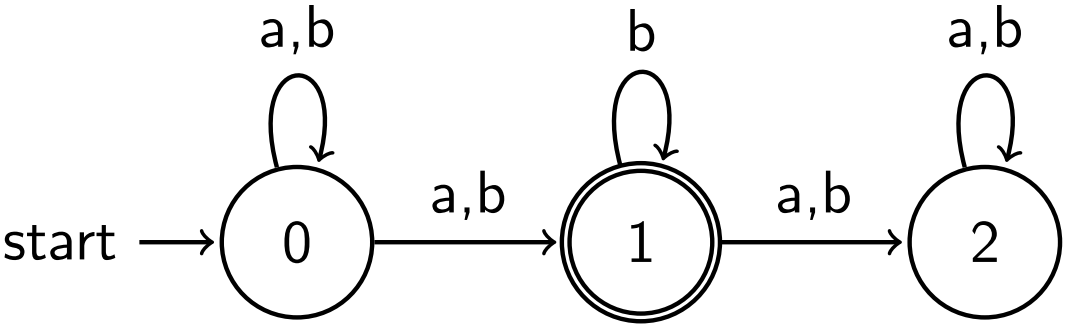
Definition

If a state reappears multiple times we only keep the state of the right most branch in the run tree, because it is the first to have visited an accepting state.
Example:

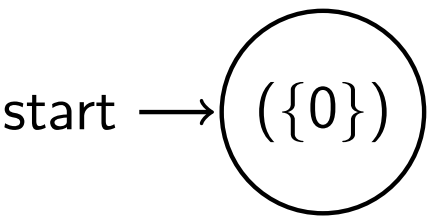
$$(\{0,1\},\{1,2\}) \to (\{0\},\{1,2\})$$

The complementation algorithm

Initial Büchi automaton:

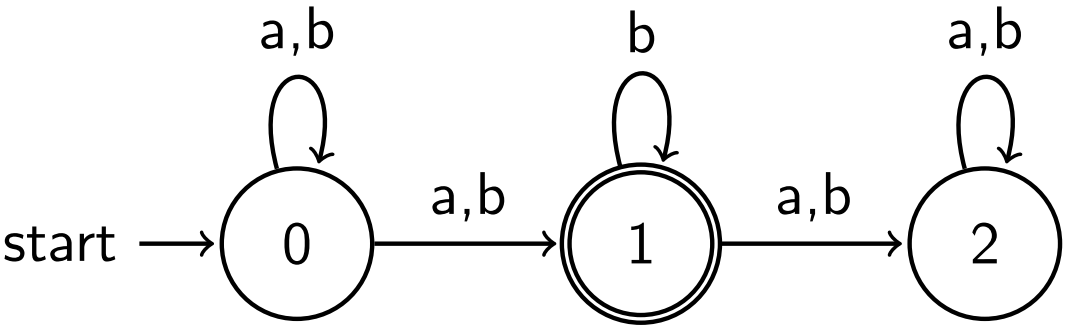


Complement Büchi automaton (finite part):

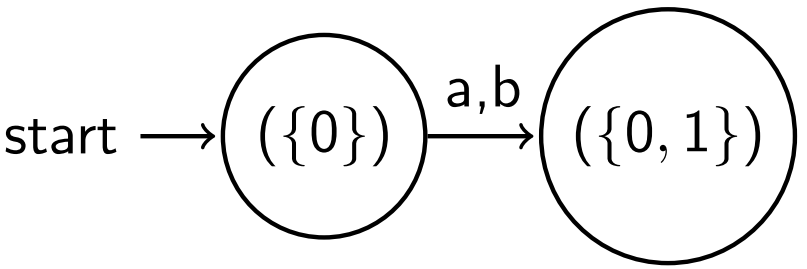


The complementation algorithm

Initial Büchi automaton:

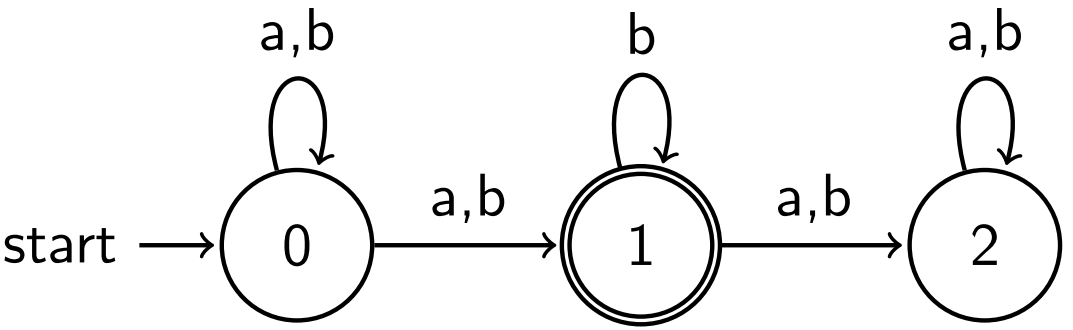


Complement Büchi automaton (finite part):

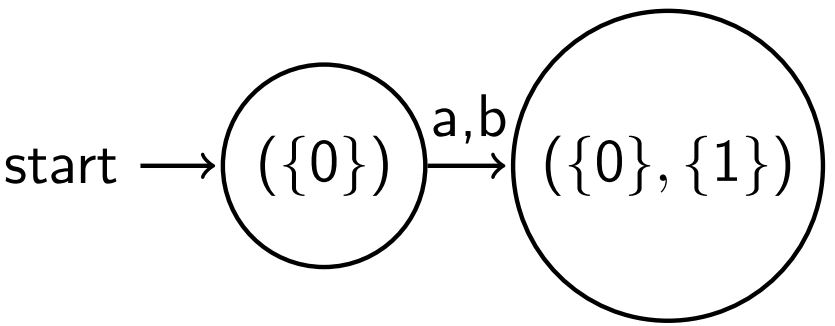


The complementation algorithm

Initial Büchi automaton:

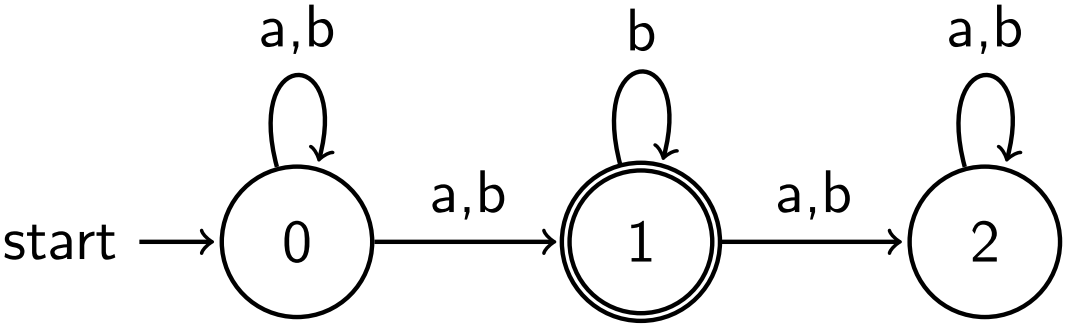


Complement Büchi automaton (finite part):

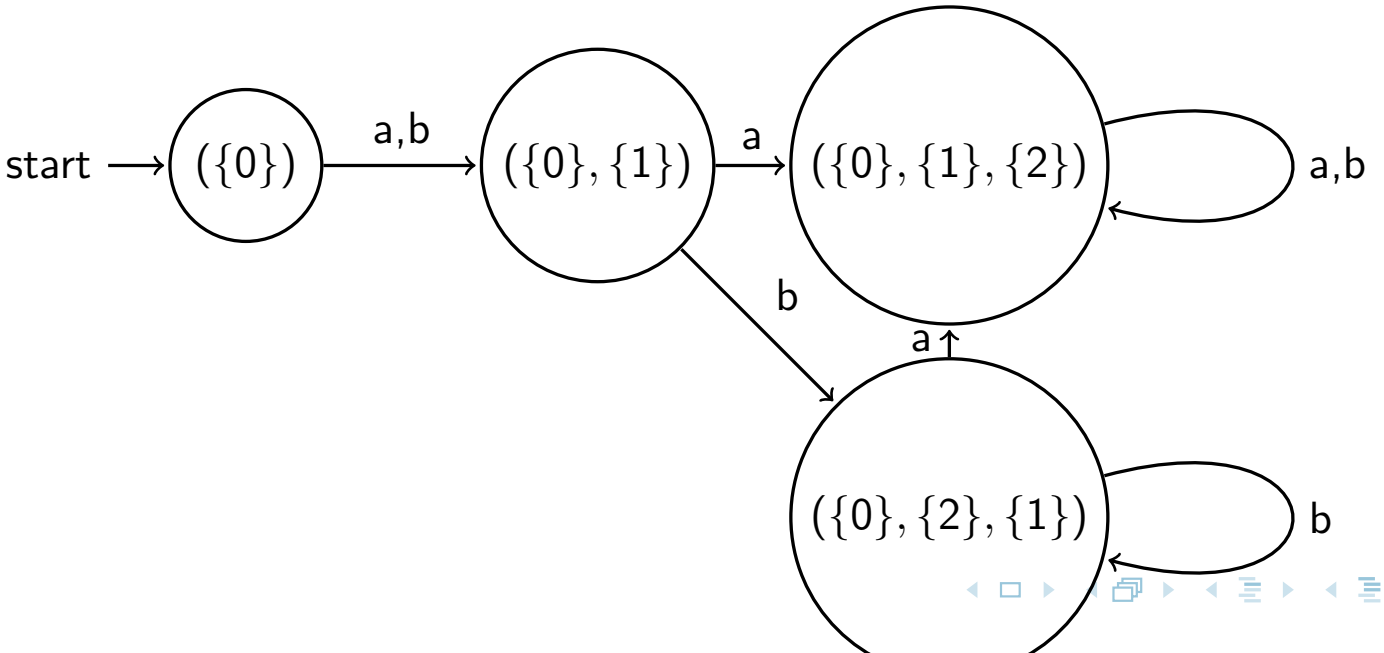


The complementation algorithm

Initial Büchi automaton:



Complement Büchi automaton (finite part):



The complementation algorithm

Colouring of sets of states:

- Finite part
  - $\{\ \}$ set of states in finite part
- Infinite part
  - $\{\ \}$ set of states that has not visited an accepting state
  - $(\ )$ set of states that has visited an accepting state and is on hold
  - $[\ ]$ set of states that has visited an accepting state (discontinued branch)

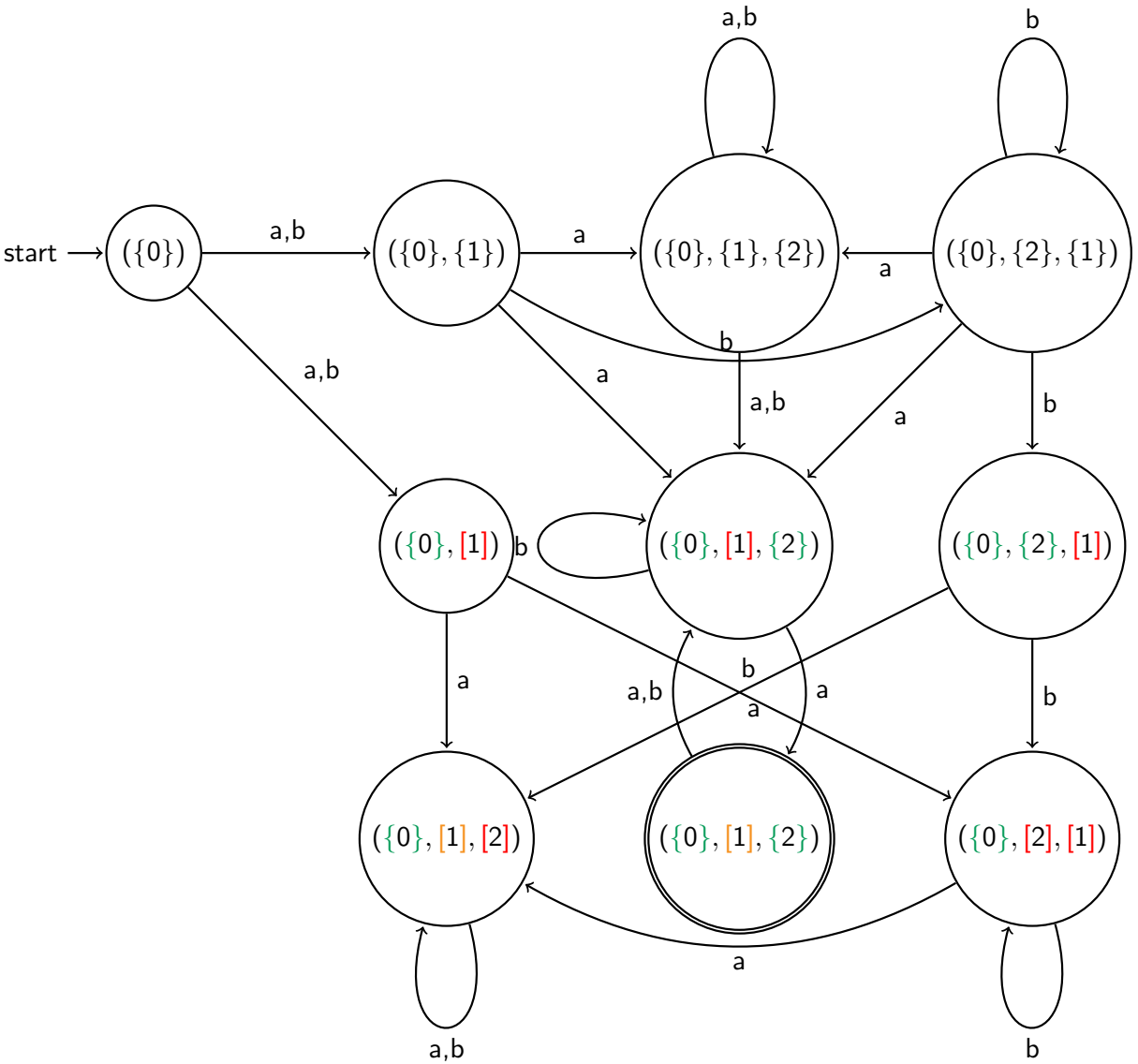
The complementation algorithm

set of states in tuple	finite part	infinite part
{ }	{N},{A}	{N},[A]
{ } (without [ ])		{N},[A]
{ } (with [ ])		{N},(A)
( ) (without [ ])		[N],[A]
( ) (with [ ])		(N),(A)
[ ]		[N],[A]

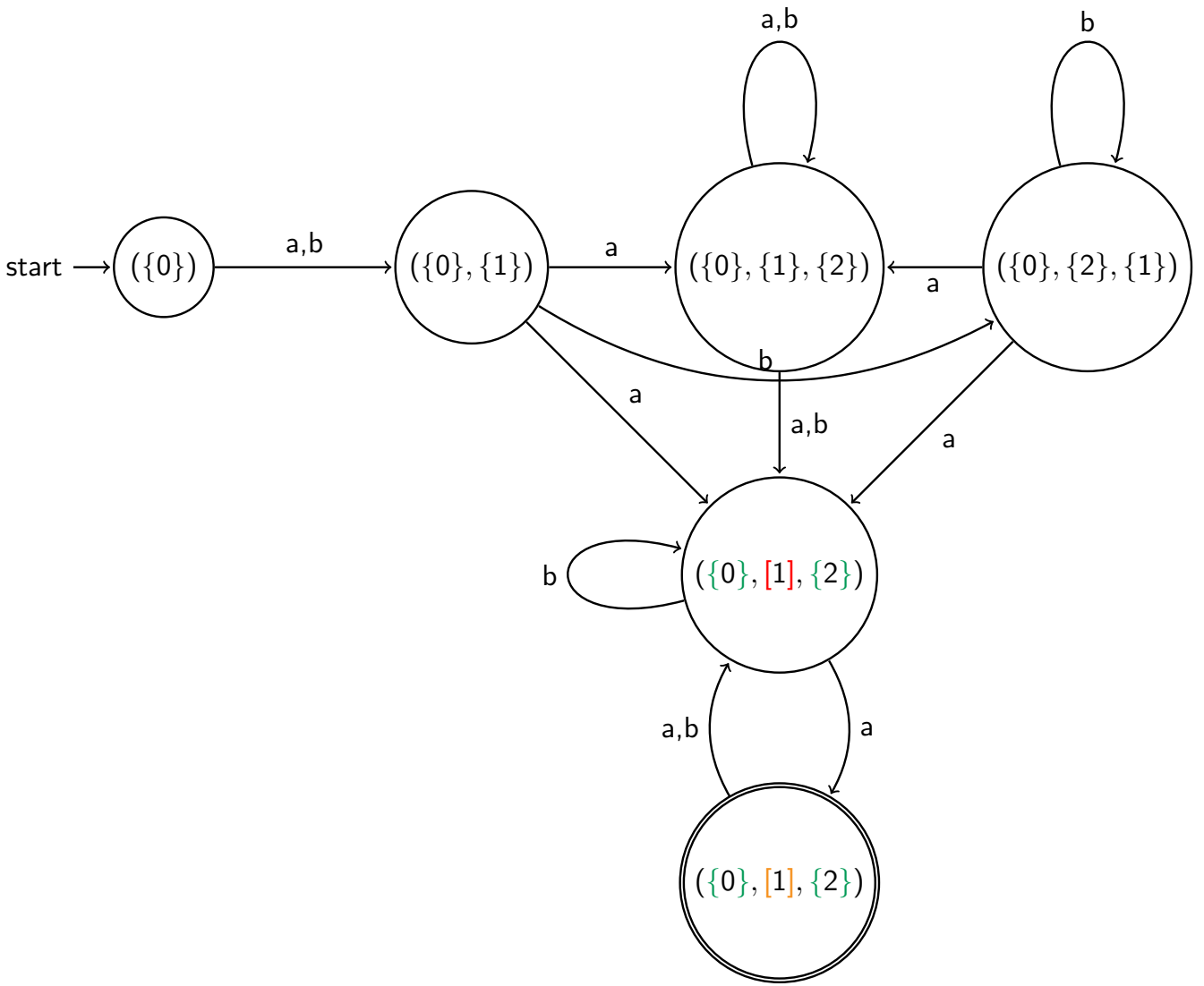
Definition

A state of the complementary Büchi automaton $\bar{\mathcal{A}}$ is accepting if there is no set representing a discontinued branch.

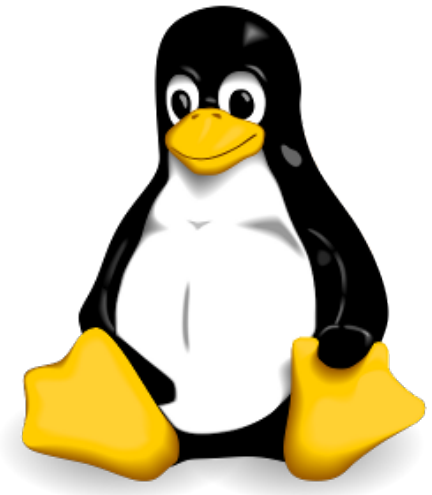
The complementation algorithm

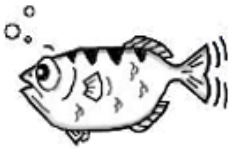


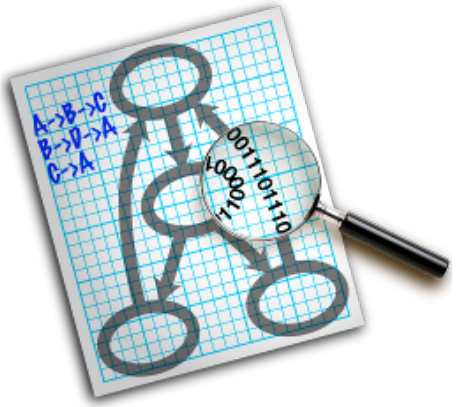
Optimization

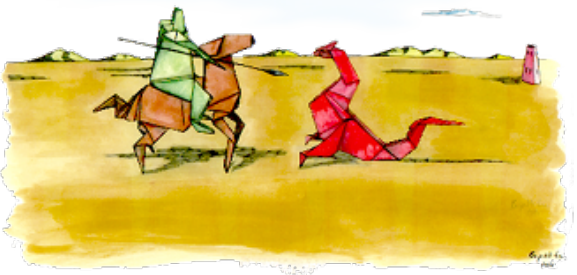


libefa / efatool









libefa / efatool

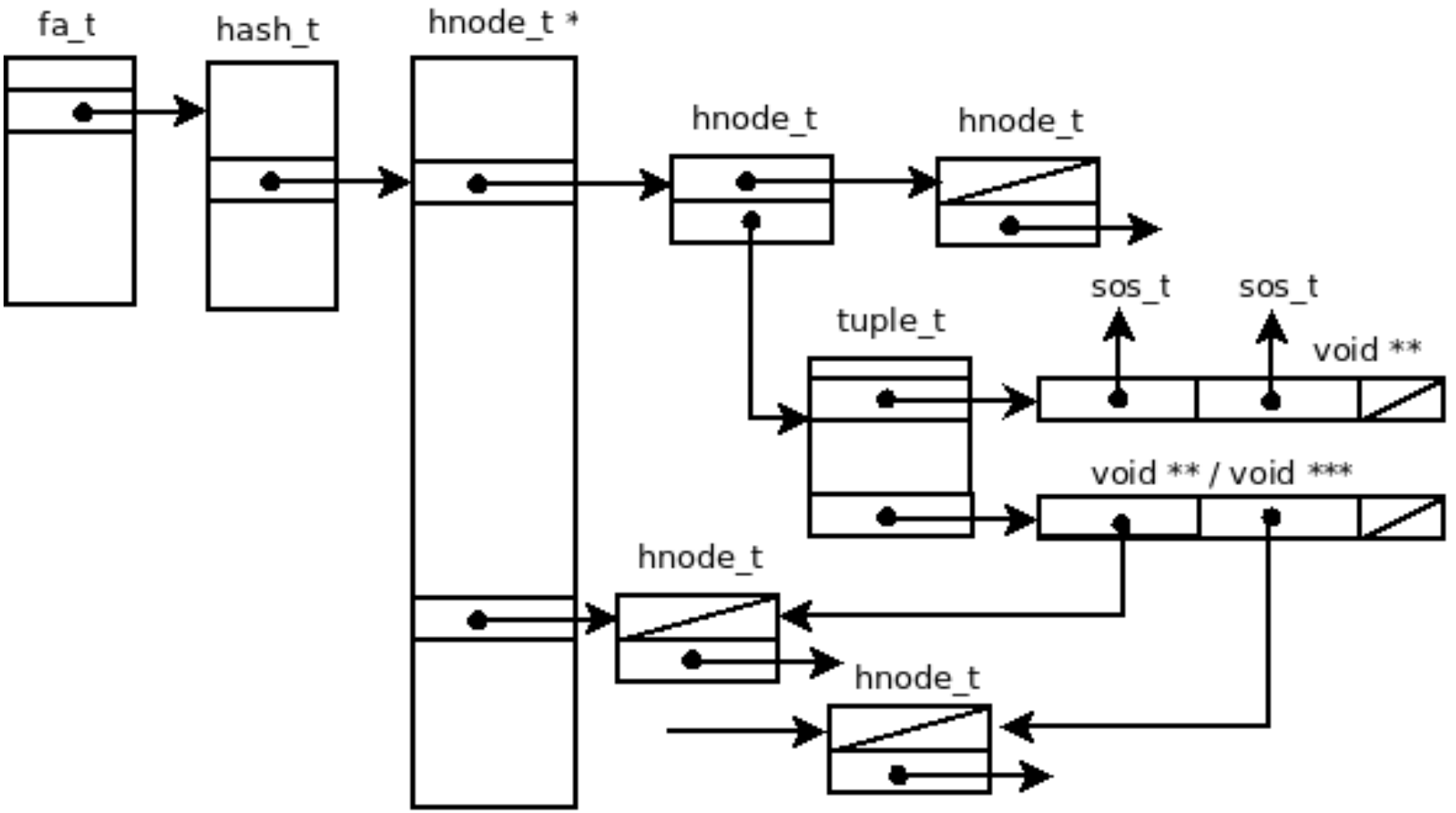
Concept of the program / library:

- read an automaton from a XML file
- store the automaton in convenient data structures
- apply algorithms on the data structures
- gather information for performance measurement / statistics
- write the automton back to a XML file or a DOT file

libefa: FAXML

```
<?xml version="1.0" encoding="utf-8"?>
<faxml version="0.3">
  <automaton name="example1" class="non-deterministic"
             minimal="true" type="buechi" >
    <alphabet>ab</alphabet>
    <states>
      <state initial="true" name="0">
        <transitions>
          <transition label="ab" to="0" />
          <transition label="b" to="1" />
        </transitions>
      </state>
      <state accepting="true" name="1">
        ...
      </state>
    </states>
  </automaton>
</faxml>
```

libefa: Memory diagram



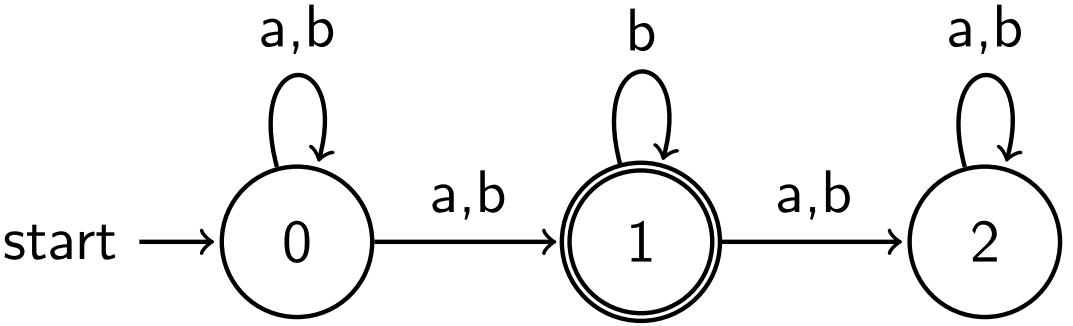
libefa

Implementation of the algorithm:

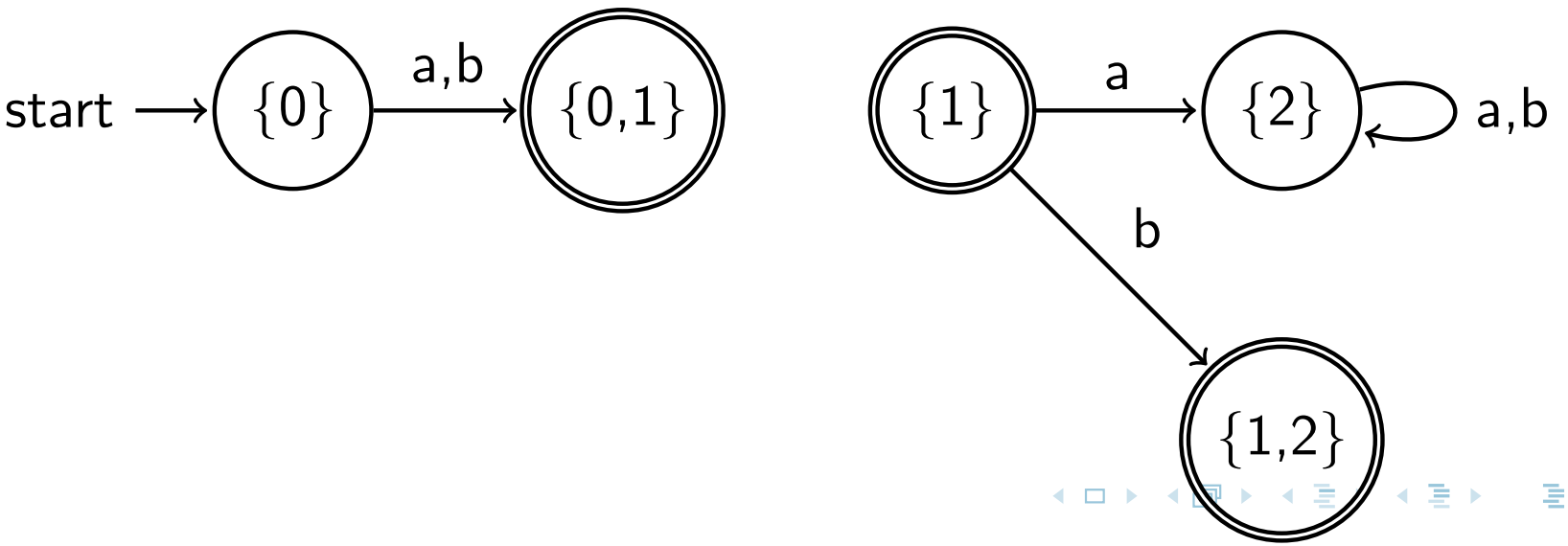
- uses 3 hash tables
  - one hash table for all states
  - one hash table for all sets of states
  - one hash table for all tuples
- sets of states and tuples are put on work lists for modified subset construction to compute successors
- lists of states and sets of states are used to construct sets of states respectively tuples

libefa: Implementation

Initial Büchi automaton:

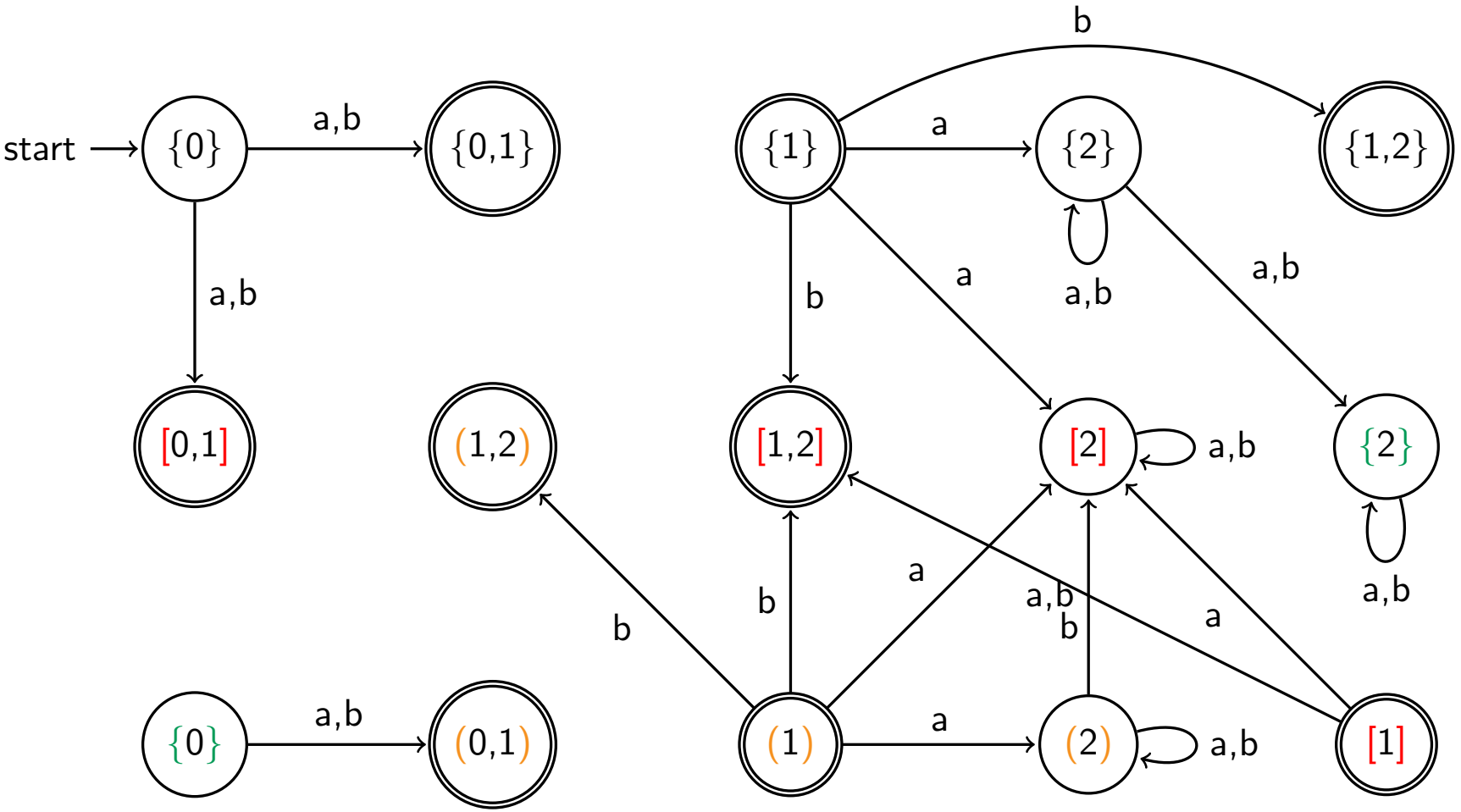


Temporary, disconnected automaton needed to compute successor tuples (finite part):



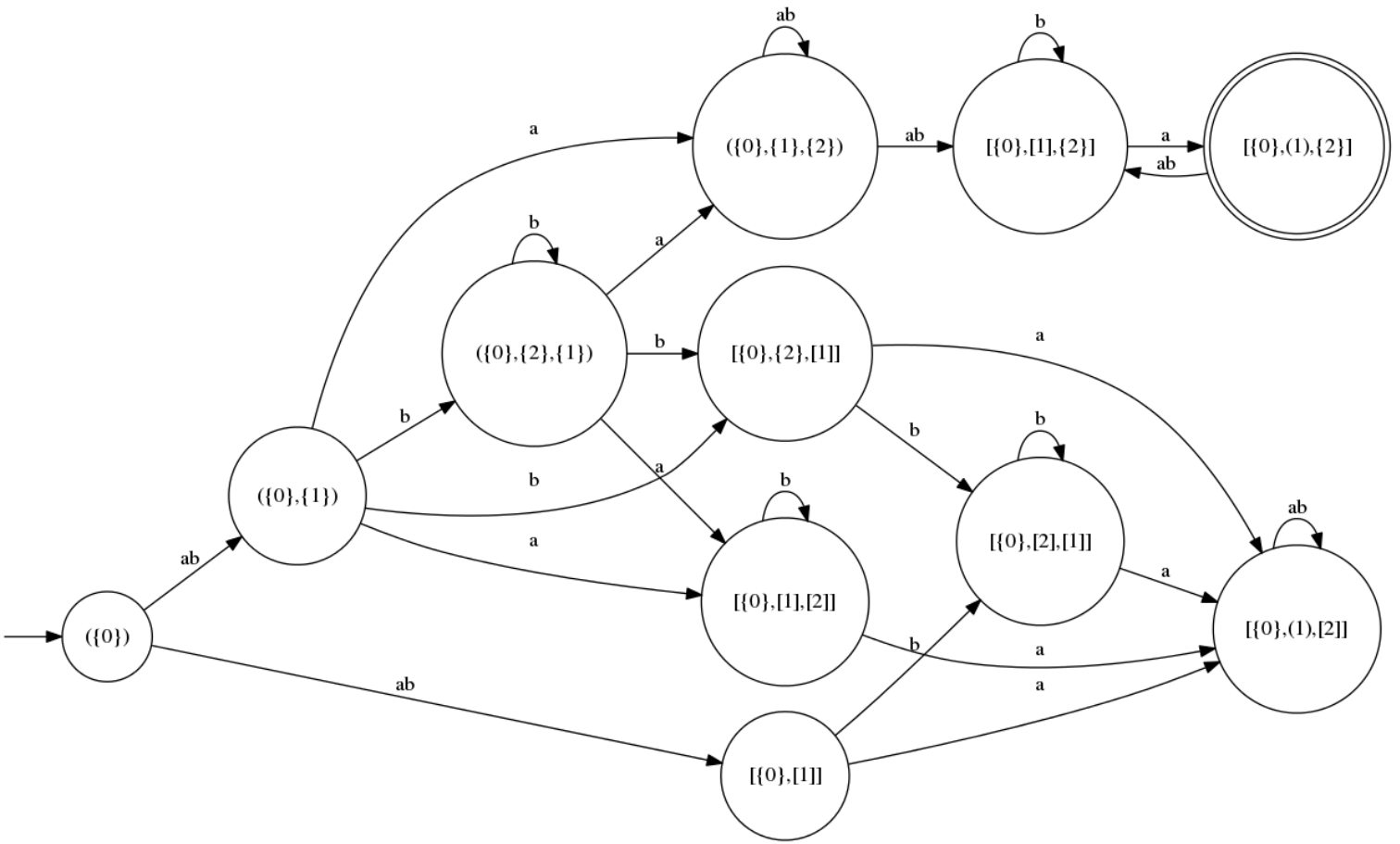
libefa: Implementation

Temporary, disconnected automaton for infinite part:



libefa: Implementation

Complement Büchi automaton:

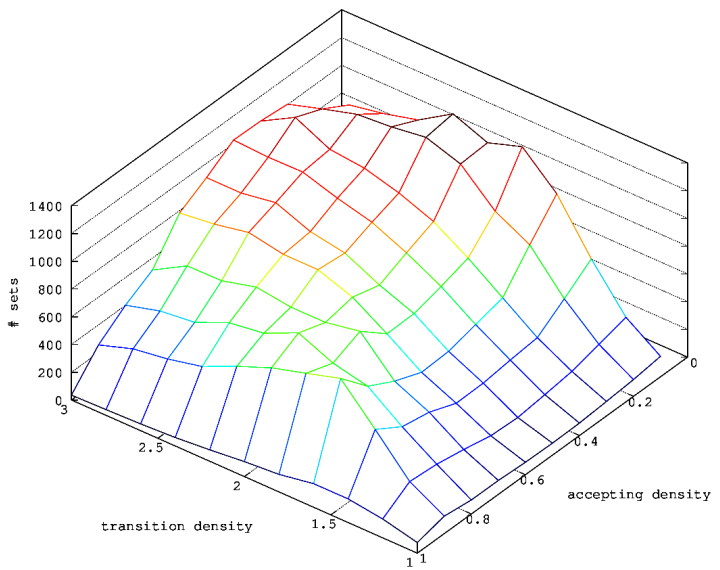


Test set

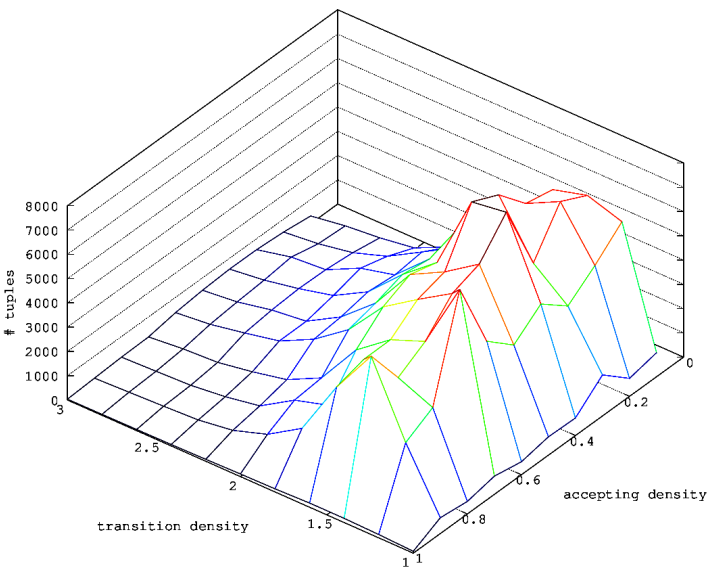
- is part of the paper "*State of Büchi complementation*" of Vardi e.a.
- can be downloaded from `http://goal.im.ntu.edu.tw`
- consists of 11'000 Büchi automata of size 15 and 11'000 Büchi automata of size 20
- Büchi automata have unreachable states
- the 11'000 Büchi automata consist of 11 transition density and 10 acceptance density classes
- each class consists of 100 random Büchi automata

Test set: Memory / Size

Incomplete Büchi automata (size 15):



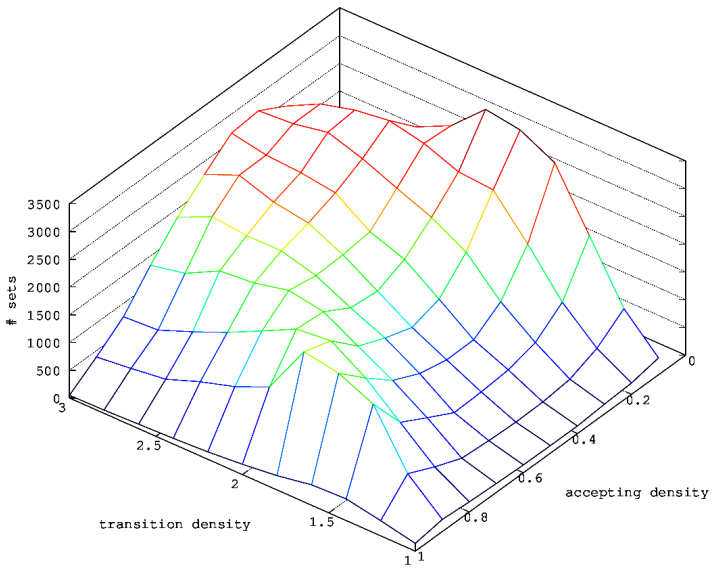
(a) Sets of states



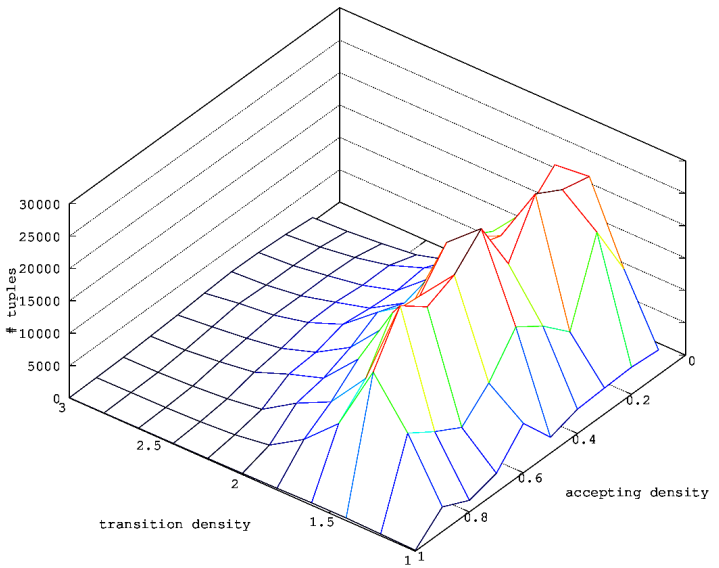
(b) Tuples

Test set: Memory / Size

Incomplete Büchi automata (size 20):



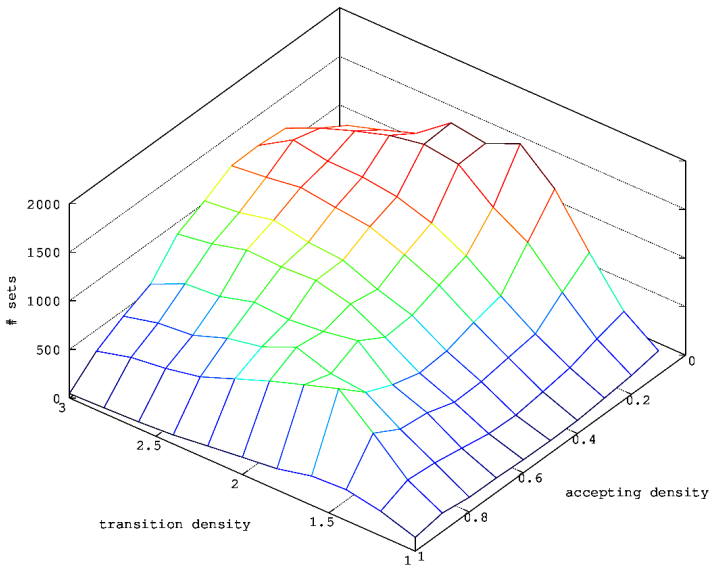
(c) Sets of states



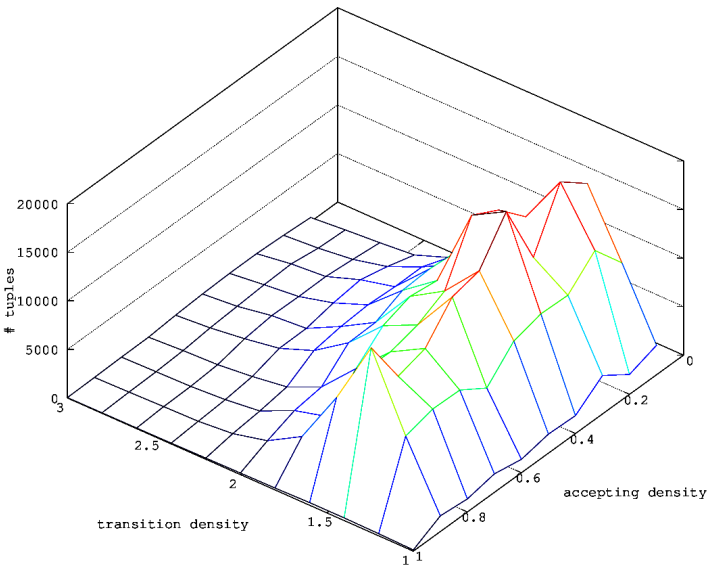
(d) Tuples

Test set: Memory / Size

Complete Büchi automata (size 15):



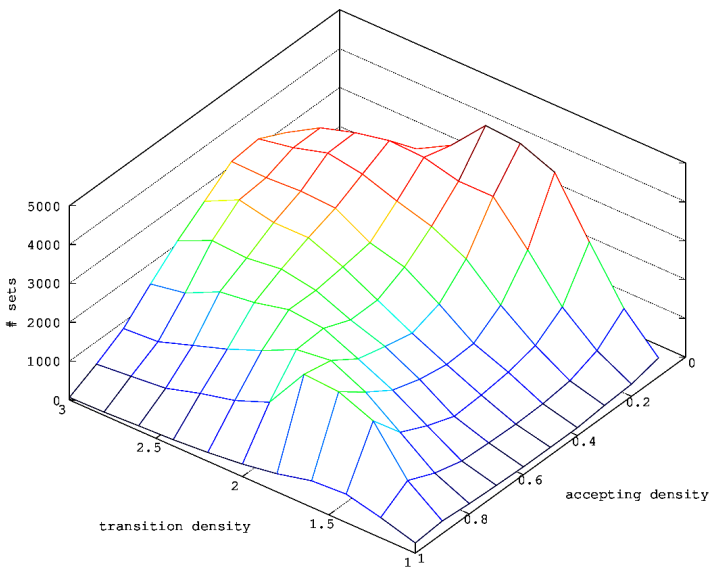
(e) Sets of states



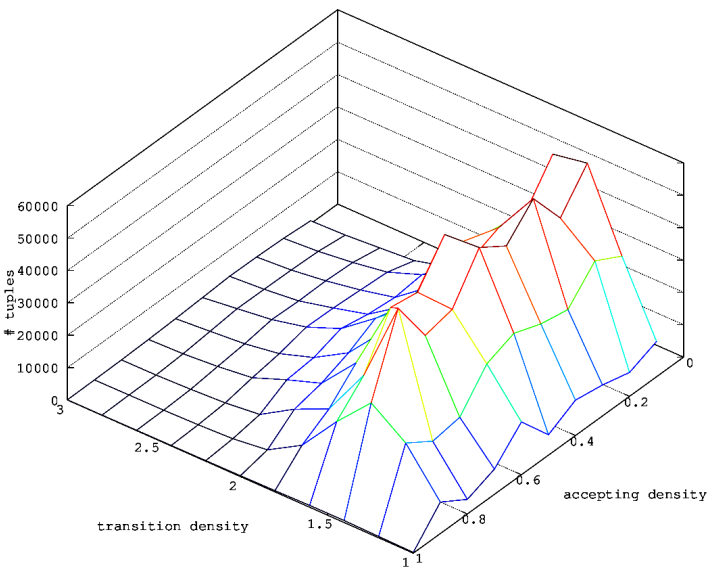
(f) Tuples

Test set: Memory / Size

Complete Büchi automata (size 20):



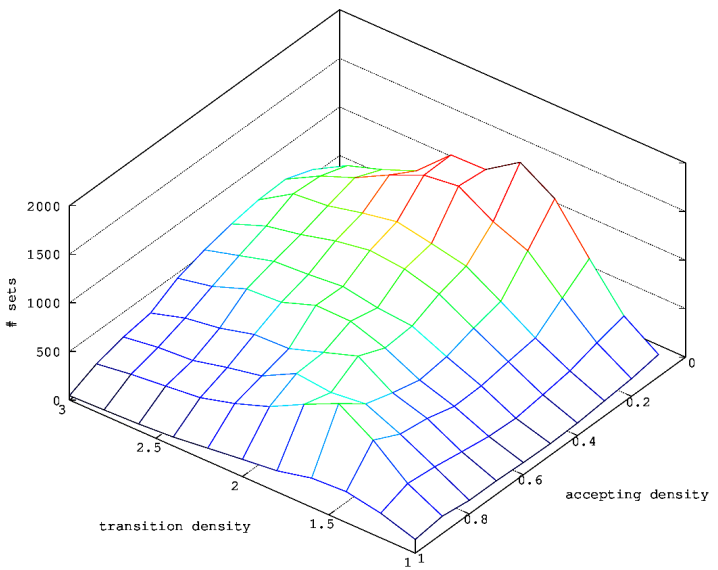
(g) Sets of states



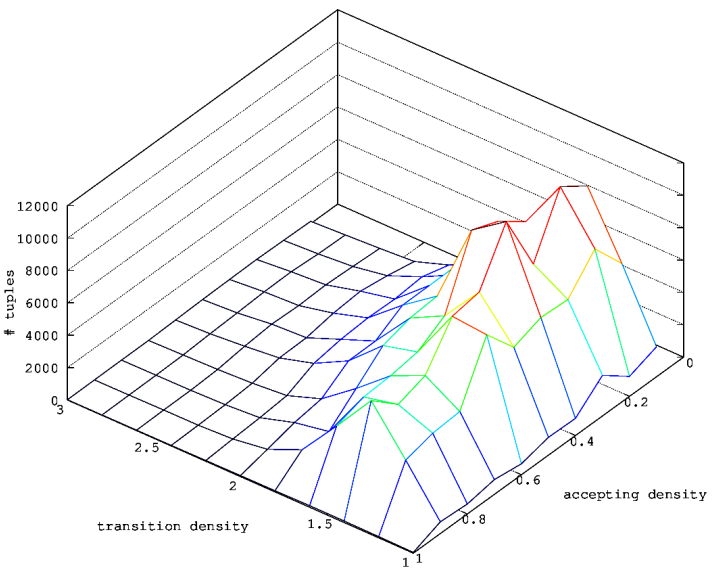
(h) Tuples

Test set: Memory / Size

Complete Büchi automata with optimization (size 15):



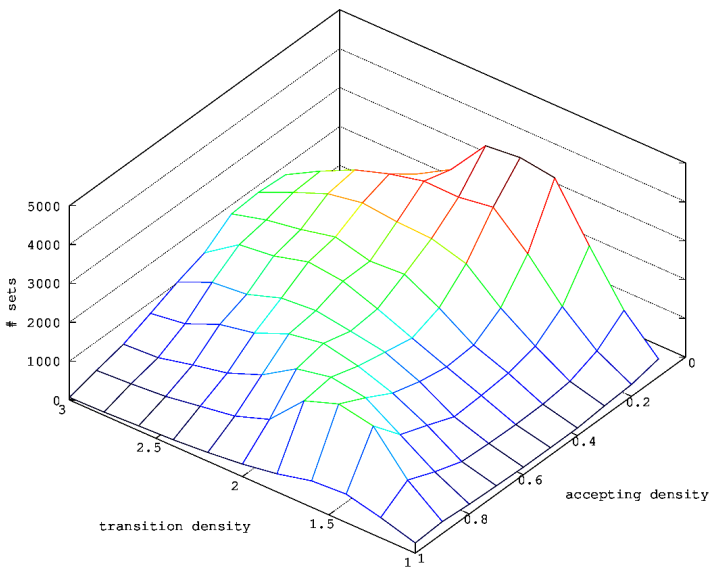
(i) Sets of states



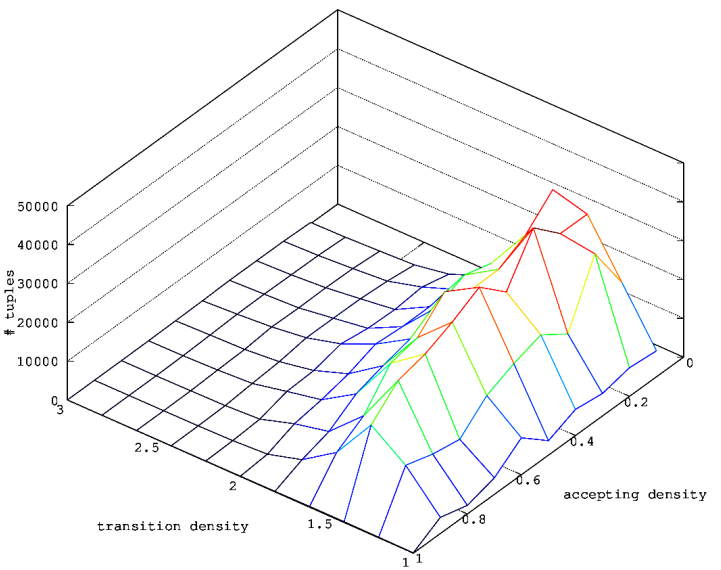
(j) Tuples

Test set: Memory / Size

Complete Büchi automata with optimization (size 20):



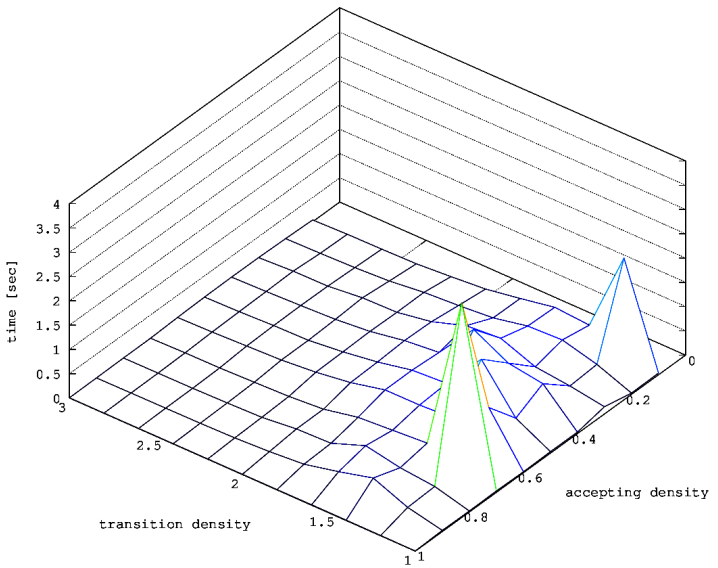
(k) Sets of states



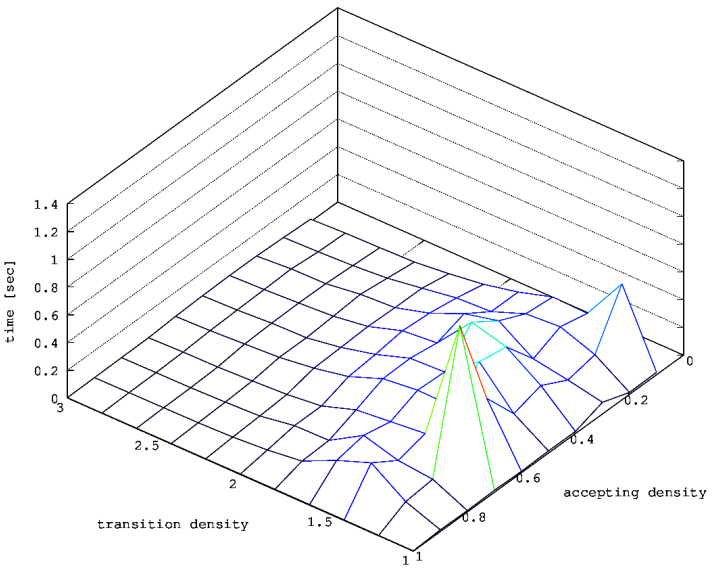
(l) Tuples

Test set: Time

Incomplete Büchi automata (size 15):



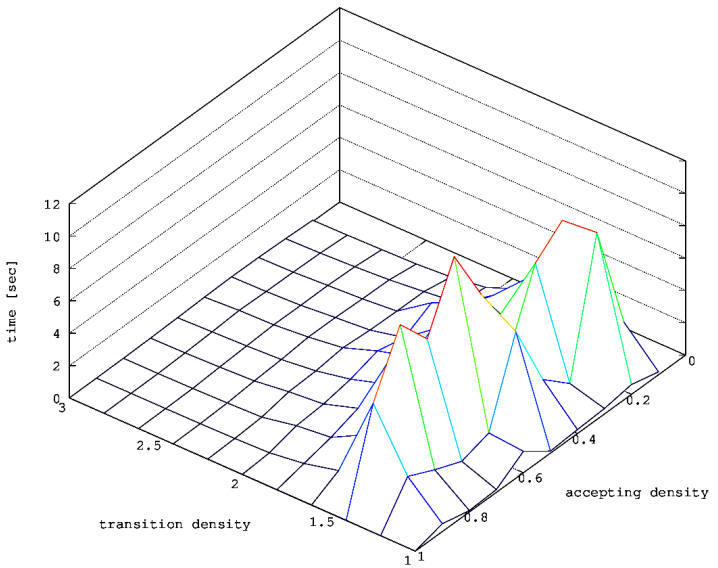
(m) 32-bit



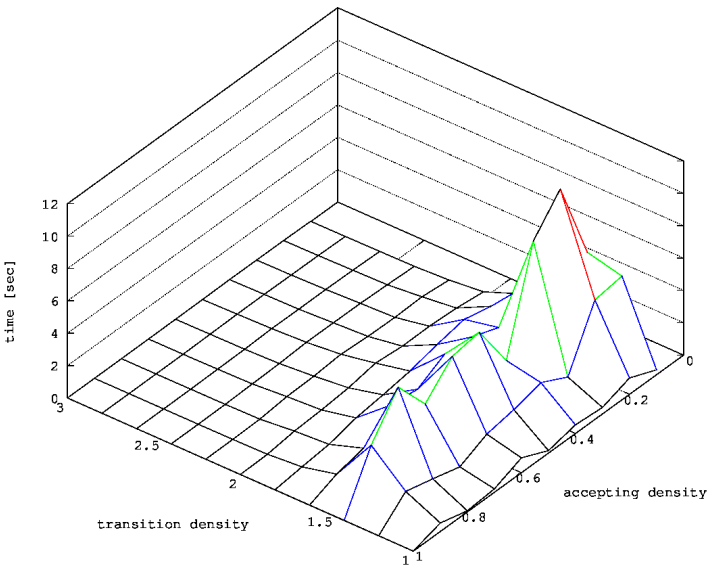
(n) 64-bit

Test set: Time

Incomplete Büchi automata (size 20):



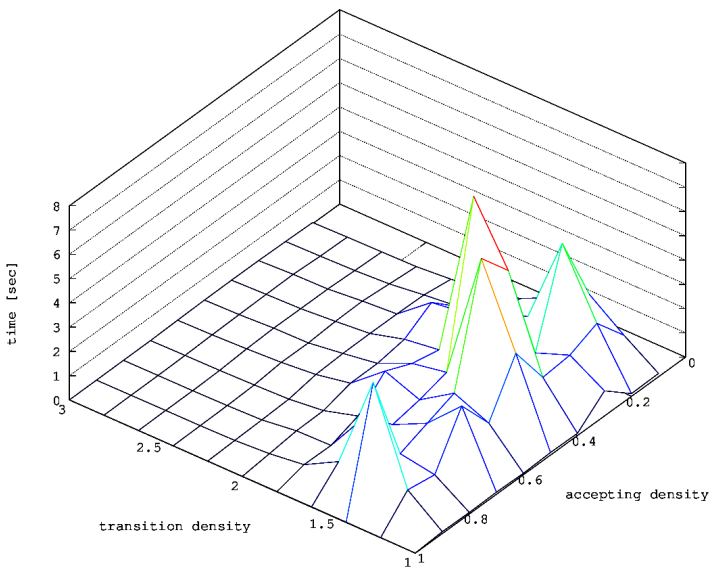
(o) 32-bit



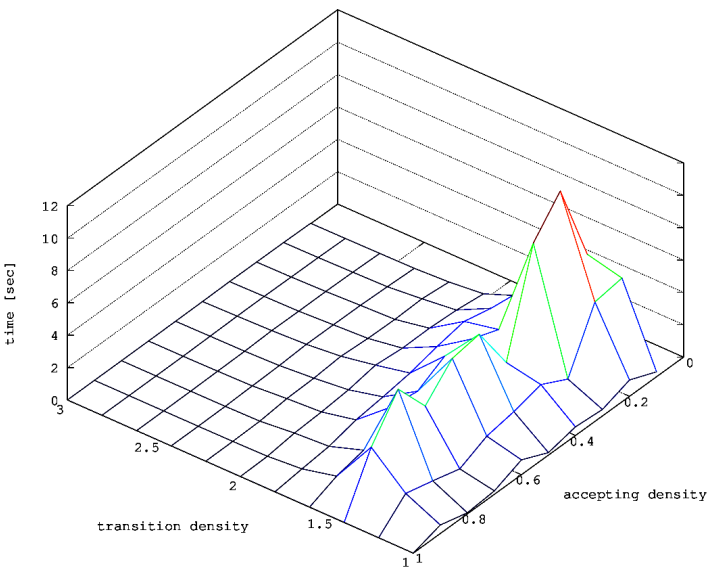
(p) 64-bit

Test set: Time

Complete Büchi automata (size 15):



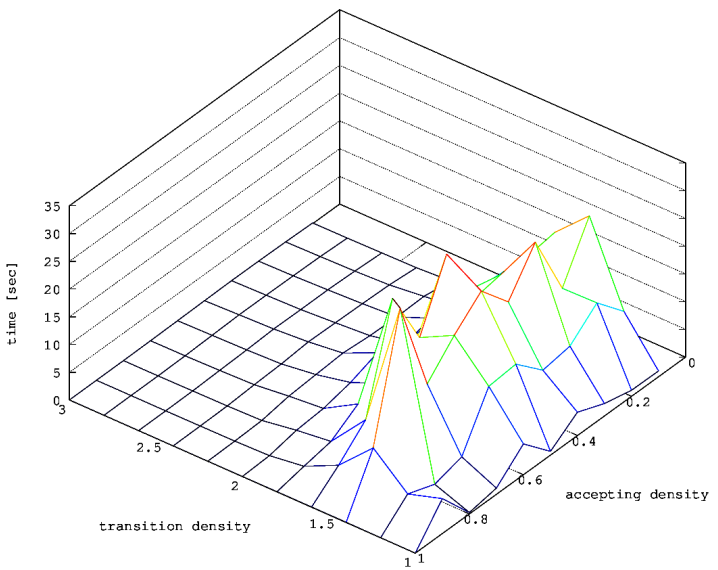
(q) 32-bit



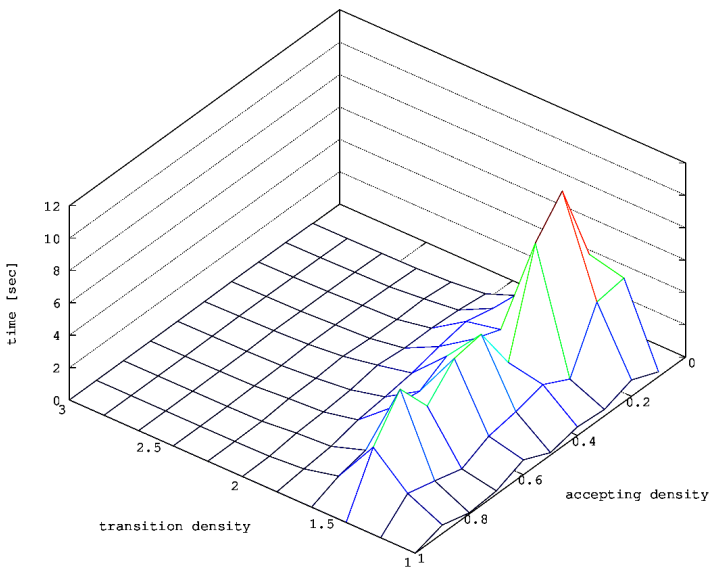
(r) 64-bit

Test set: Time

Complete Büchi automata (size 20):



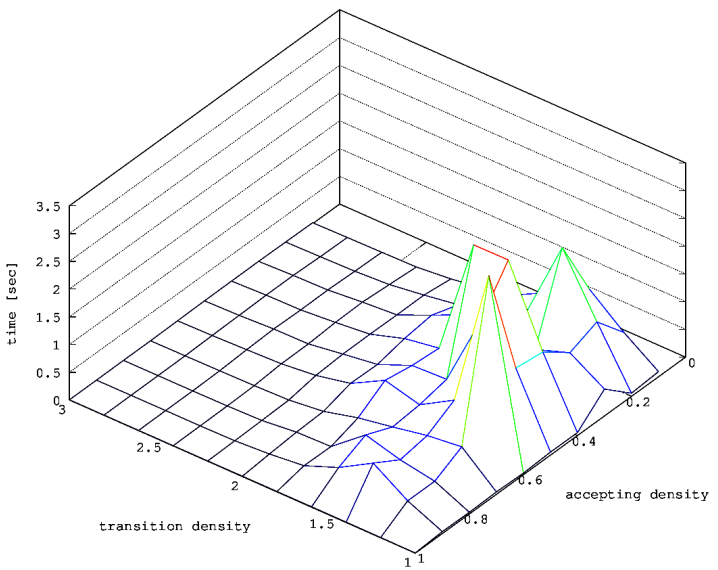
(s) 32-bit



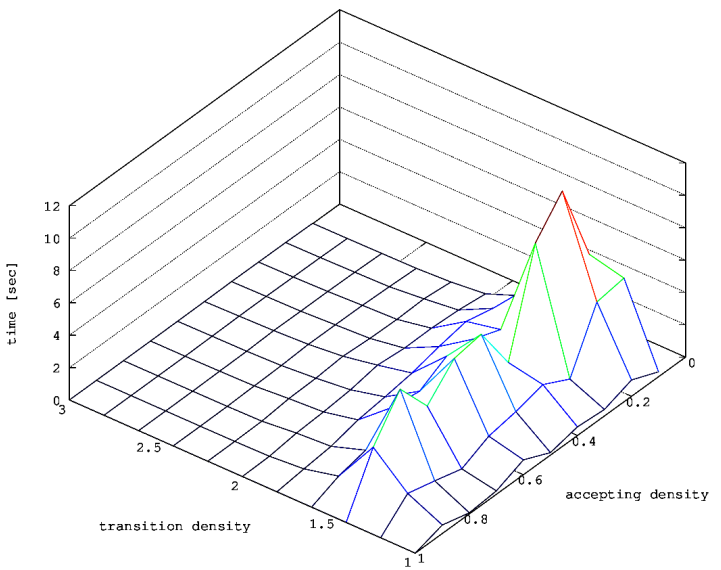
(t) 64-bit

Test set: Time

Complete Büchi automata with optimization (size 15):



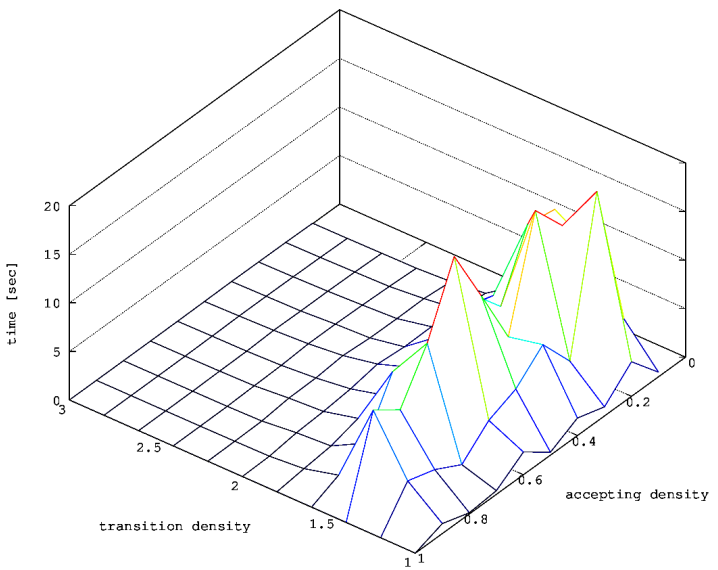
(u) 32-bit



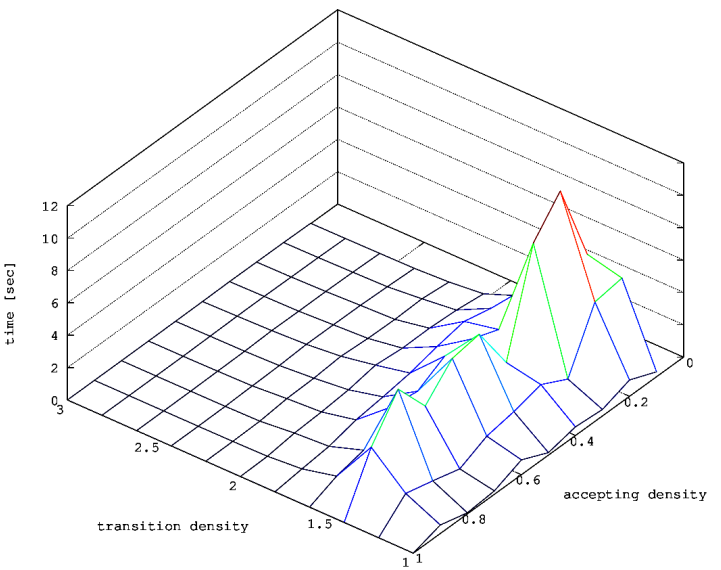
(v) 64-bit

Test set: Time

Complete Büchi automata with optimization (size 20):



(w) 32-bit



(x) 64-bit

Test set: Comparison

Test set with Büchi automata of size 20:

Construction	Eff. Samples	Avg. comp. size	Timeout	OOM
Safra-Piterman+PASE	6'534	62.88	50	70
Rank+PA	6'534	397.70	4'366	0
Slice+PADRM	6'534	465.95	933	0
Slice-U-A	10'990	7'516.66	10	0

Results taken from "*State of Büchi Complementation*". Timeout was set to 10 min.

Complexity

Question:
How could the complexity of the algorithm be approximated?

Idea:
Use a series of Michel's "*state explosion*" Büchi automata, plot their initial size vs complement size and fit the measured values with one of the complexity functions.

Note:
The approximated complexity will *NOT* correspond to the real complexity of the algorithm.

Complexity

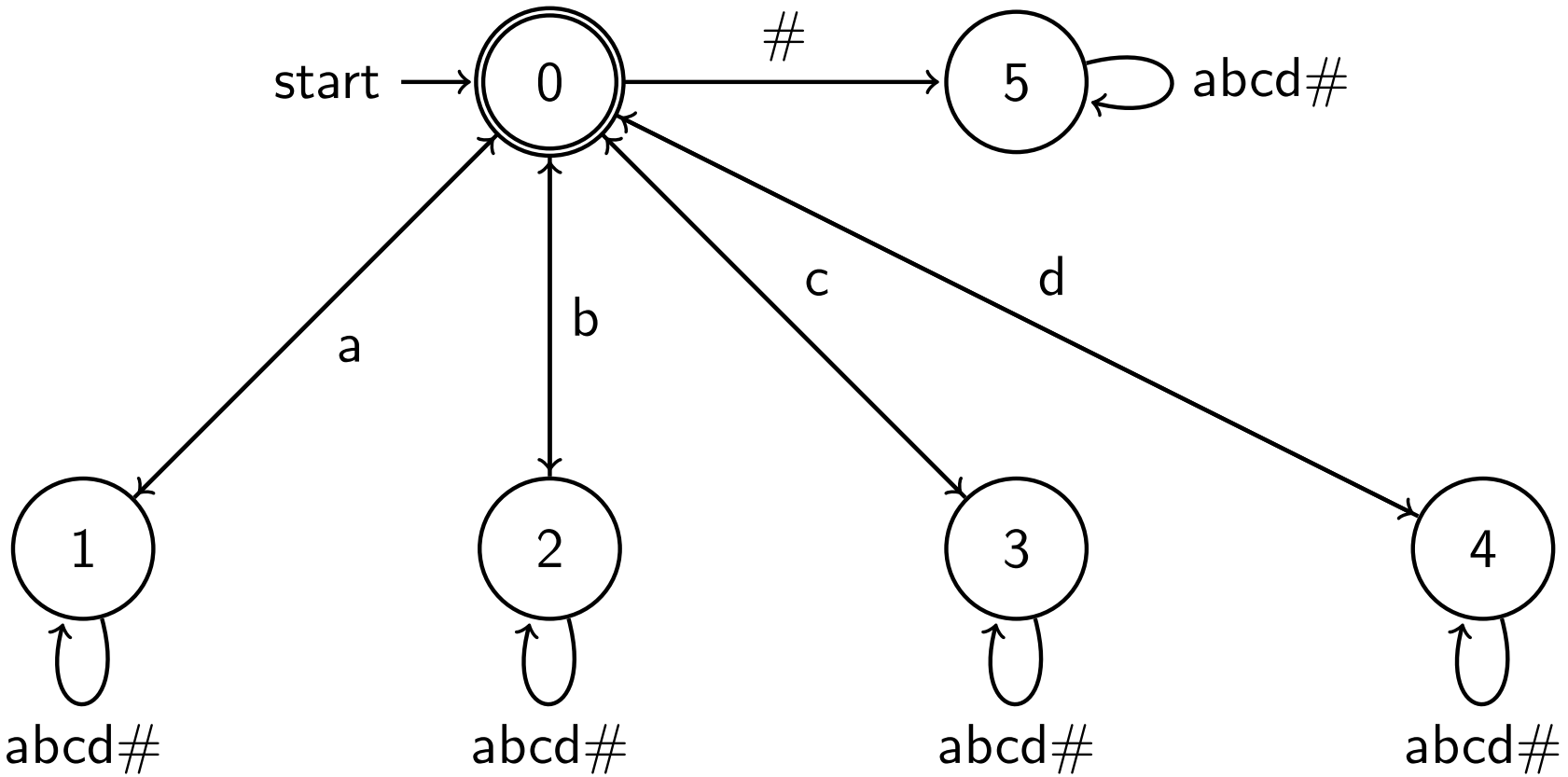
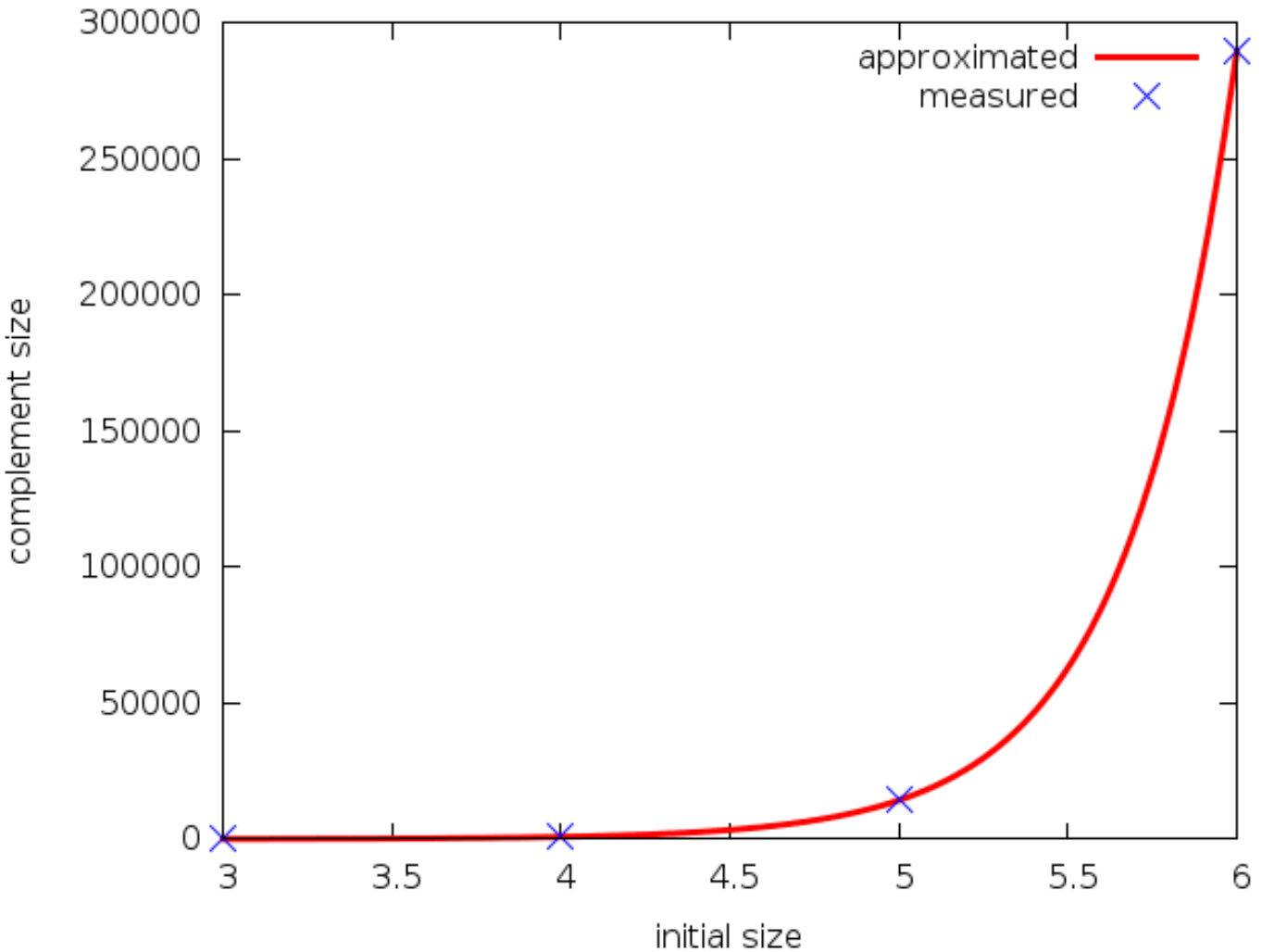


Figure: Michel 4

Complexity


300000
250000
200000
150000
100000
50000
0
complement size
3
3.5
4
4.5
5
5.5
6
initial size
approximated
measured

Complexity

asymptotic standard error

Function	Parameters	ASE
(an)^n $(ax)^x$	$a = 1.36$	0.01 %
$a\,(bx)^x$	$a = 1.10$	1.5 %
	$b = 1.33$	0.25 %

$(0.96n)^n$	upper bound	Friedgut, Kupferman, Vardi[2]
$(0.76n)^n$	lower bound	Schewe[3]
$(1.36n)^n$		Ultes-Nitsche, Allred

[2] Büchi Complementation made tighter

[3] Büchi Complementation made tight

Future work

- fix bug for construction of temporary, disconnected automaton
- add algorithm to intersect automata
- improve code
- implement multithreaded version of algorithm
- add run functionality for automaton
- implement red-black-tree
- finish implementation of algorithm chain