Observations on Determinization of Büchi Automata

Christoph Schulte Althoff Wolfgang Thomas
Nico Wallmeier

Chair of Computer Science 7
RWTH Aachen, Germany

Conference on Implementation and Application of Automata, 2005

Outline

1. Introduction
   - Motivation
   - Automata
2. Algorithms
   - Muller-Schupp
   - opt. Muller-Schupp
   - Safra
3. Implementation
   - OmegaDet
   - Observations

Motivation

- Determinization is necessary for specification of infinite games

McNaughton's Theorem

A nondeterministic Büchi automaton can be converted into a deterministic Muller automaton.

- In most constructions target automaton: deterministic Rabin automaton instead of Muller automaton
- Lower bound of $2^{O(n \log n)}$ for the number of states
- Safra and Muller-Schupp match this bound

Def. Büchi and Rabin automata

Büchi automata

Büchi automaton $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$

- Q Finite set of states
- Σ Input alphabet
- q_0 Initial state
- $\Delta \subseteq Q \times \Sigma \times Q$, transition relation
- F Set of final states, $F \subseteq Q$

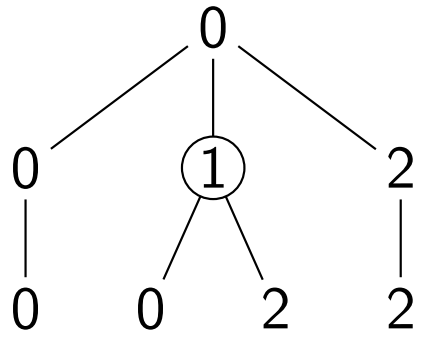
Deterministic Rabin automata

Deterministic Rabin automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Omega)$

- $\delta : Q \times \Sigma \to Q$, transition function
- $\Omega = ((E_1, F_1), \ldots, (E_k, F_k))$ list of "accepting pairs"

Derivation of Muller-Schupp trees

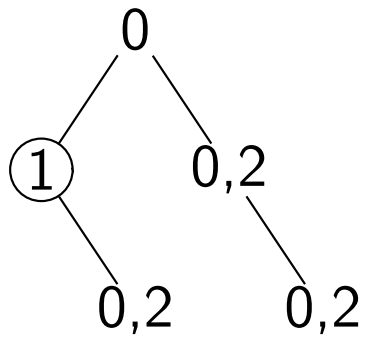
- Büchi automaton: computation tree on given input
- Partitioning the sons of a vertex into two classes
  - At most binary branching
- Keep only leftmost occurrences of a state on the respective level
  - Bounded width of tree
- Compress path segments between branching points
  - Strictly binary tree
  - Finite number of trees



prefix of a computation tree

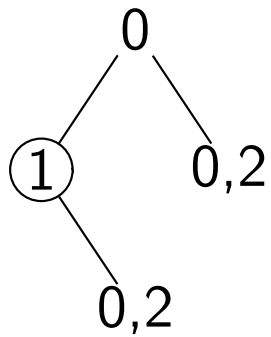
Derivation of Muller-Schupp trees

- Büchi automaton: computation tree on given input
- Partitioning the sons of a vertex into two classes
  - At most binary branching
- Keep only leftmost occurrences of a state on the respective level
  - Bounded width of tree
- Compress path segments between branching points
  - Strictly binary tree
  - Finite number of trees



Derivation of Muller-Schupp trees

- Büchi automaton: computation tree on given input
- Partitioning the sons of a vertex into two classes
  - At most binary branching
- Keep only leftmost occurrences of a state on the respective level
  - Bounded width of tree
- Compress path segments between branching points
  - Strictly binary tree
  - Finite number of trees



Derivation of Muller-Schupp trees

- Büchi automaton: computation tree on given input
- Partitioning the sons of a vertex into two classes
  - At most binary branching
- Keep only leftmost occurrences of a state on the respective level
  - Bounded width of tree
- Compress path segments between branching points
  - Strictly binary tree
  - Finite number of trees

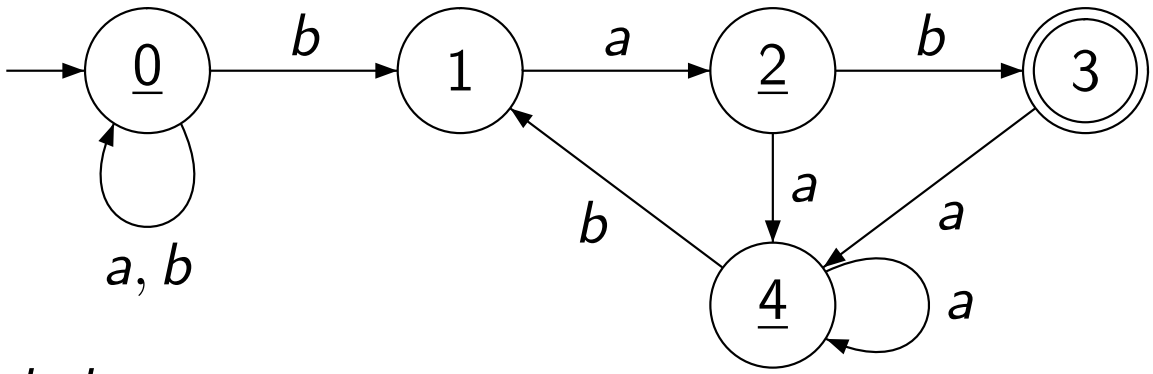
Muller-Schupp trees

- Vertex represents a path segment:
  - Named with positive natural number
  - Labelled by a subset of Q
  - Has color from the set $\{\text{red}, \text{yellow}, \text{green}\}$
- Coloring:

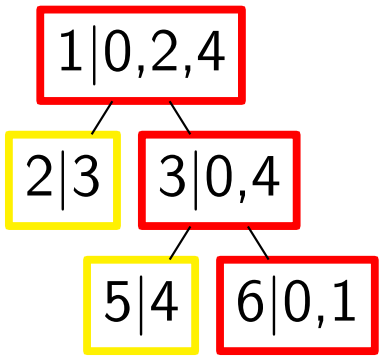
red	Has no final state
yellow	Has final state but did not receive this state in the last step
green	Received final state in the last step

- Set of states at a parent node: disjoint union of the two sons

Update of Muller-Schupp trees

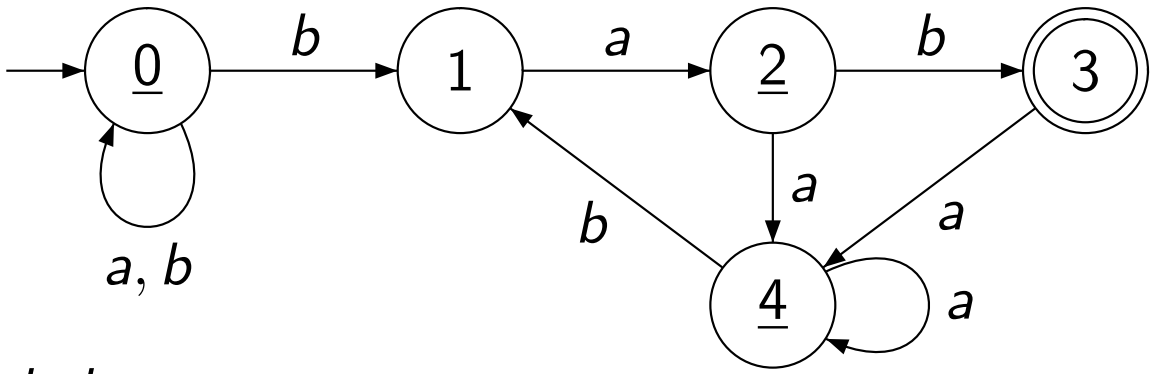


Word: *bababab*
Next letter: *a*

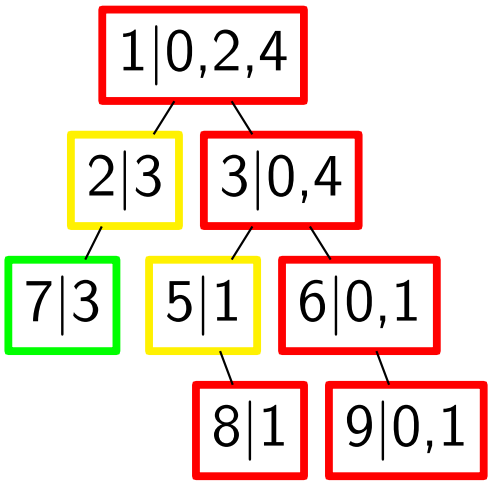


1. Copy given tree t, changing all colors green to yellow.
2. Apply subset construction to each leaf, add left and right son, color these sons green and red.
3. Keep only leftmost occurrence of each state.
4. Delete the vertices which are only on paths leading to leaves whose value is the empty set.

Update of Muller-Schupp trees

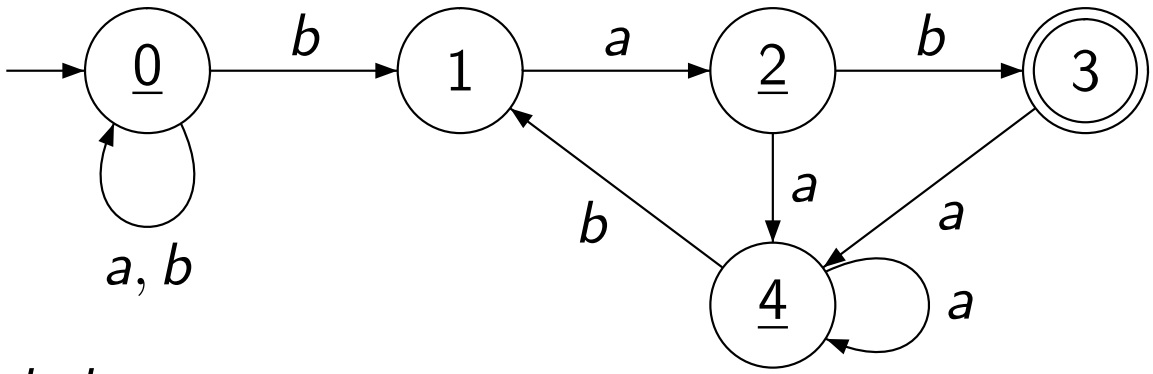


Word: *bababab*
Next letter: *a*

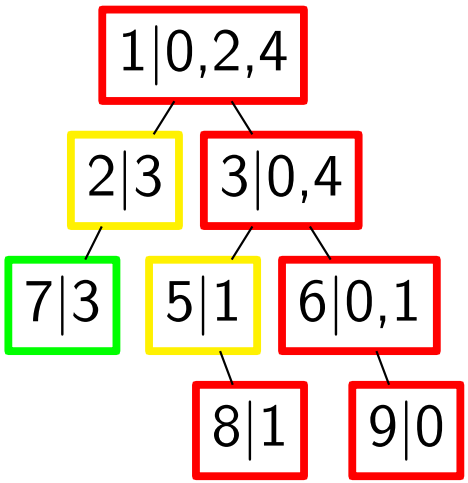


1. Copy given tree t, changing all colors green to yellow.
2. Apply subset construction to each leaf, add left and right son, color these sons green and red.
3. Keep only leftmost occurrence of each state.
4. Delete the vertices which are only on paths leading to leaves whose value is the empty set.

Update of Muller-Schupp trees

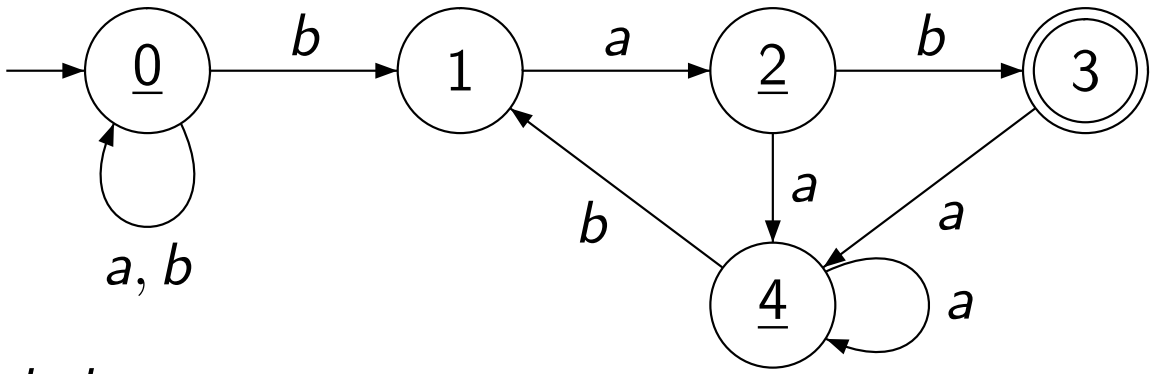


Word: *bababab*
Next letter: *a*

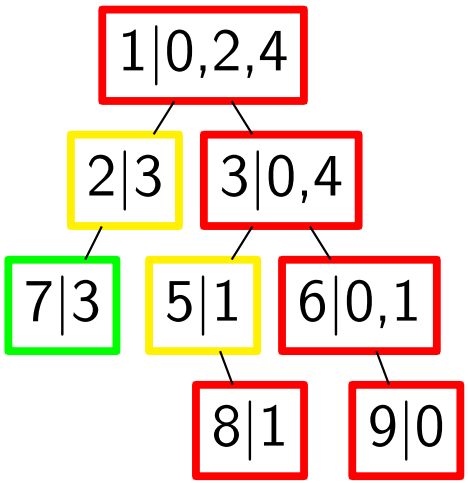


1. Copy given tree t, changing all colors green to yellow.
2. Apply subset construction to each leaf, add left and right son, color these sons green and red.
3. Keep only leftmost occurrence of each state.
4. Delete the vertices which are only on paths leading to leaves whose value is the empty set.

Update of Muller-Schupp trees

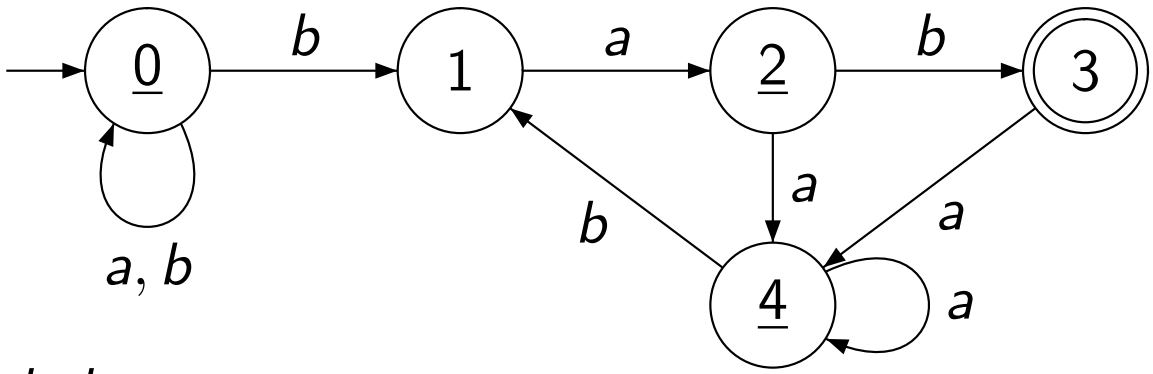


Word: *bababab*
Next letter: *a*

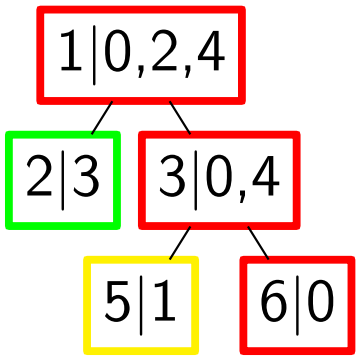


3. Keep only leftmost occurrence of each state.
4. Delete the vertices which are only on paths leading to leaves whose value is the empty set.
5. As long as there exists a vertex of degree one merge this vertex with its successor, inheriting the color green if this successor was colored green or yellow.

Update of Muller-Schupp trees

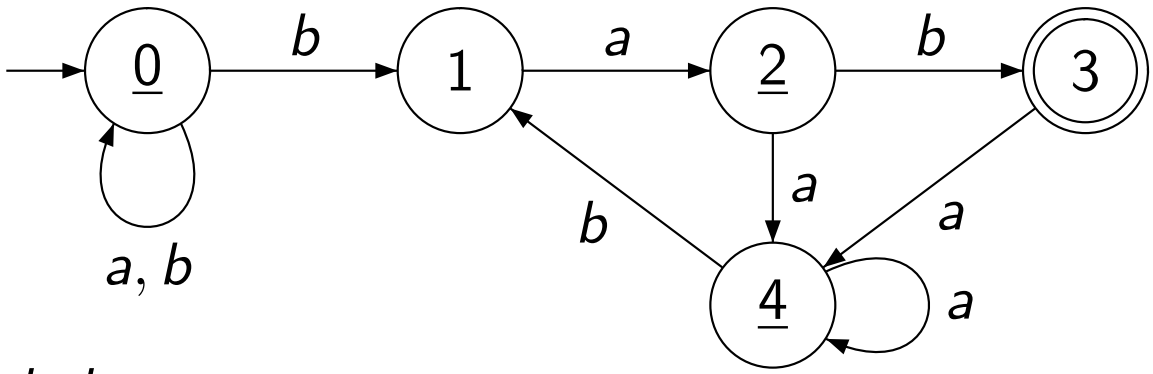


Word: *bababab*
Next letter: *a*

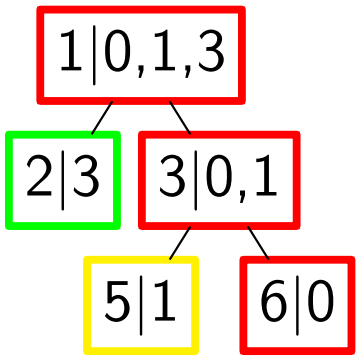


5. As long as there exists a vertex of degree one merge this vertex with its successor, inheriting the color green if this successor was colored green or yellow.
6. Proceeding from the leaves, label each parent by the union of the two state sets from the labelling of the two sons.

Update of Muller-Schupp trees



Word: *bababab*
Next letter: *a*



5. As long as there exists a vertex of degree one merge this vertex with its successor, inheriting the color green if this successor was colored green or yellow.
6. Proceeding from the leaves, label each parent by the union of the two state sets from the labelling of the two sons.

Result automaton according to Muller-Schupp

Rabin automaton

- States: reachable Muller-Schupp trees
- Input alphabet: unchanged
- Transition function: According to the update of Muller-Schupp trees
- Acceptance condition of pairs (E_i, F_i) for each vertex name
  - E_i contains those trees where i is missing
  - F_i has those trees where i occurs colored green

Optimized update of Muller-Schupp tree

Idea

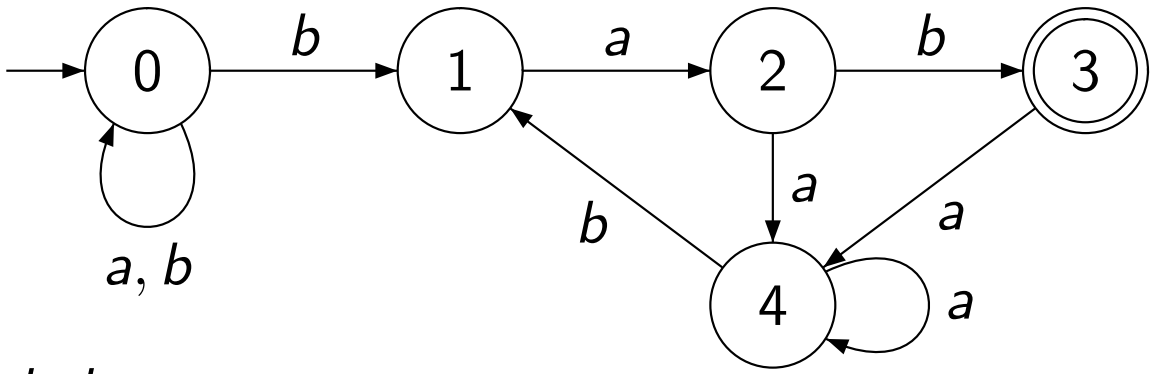
Create sons only if they are really needed

1. Copy the given tree t, changing all colors green to yellow
2. Apply the subset construction (via letter a) to each leaf.
3. Keep only the leftmost occurrence of each state.
4. Only for leaves which contain final as well as non-final states add left and right son carrying the reached final, respectively non-final states; color these sons green and red, respectively.
5. Color all leaves containing only final states green.
6. Items 4., 5. and 6. of original algorithm

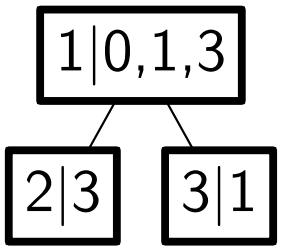
Safra trees

- Safra trees are more succinct
- More than binary branching may occur
- Different color policy: if it is ensured that from all states a final state is visited a vertex is marked green
- In a Muller-Schupp tree the intermediate vertices with non-final state-sets amount to a binary encoding of the Safra trees.
- However, by the different color policy we do not have in general:
  $|$ Safra automaton $| \leq |$ Muller-Schupp automaton $|$

Update of Safra trees

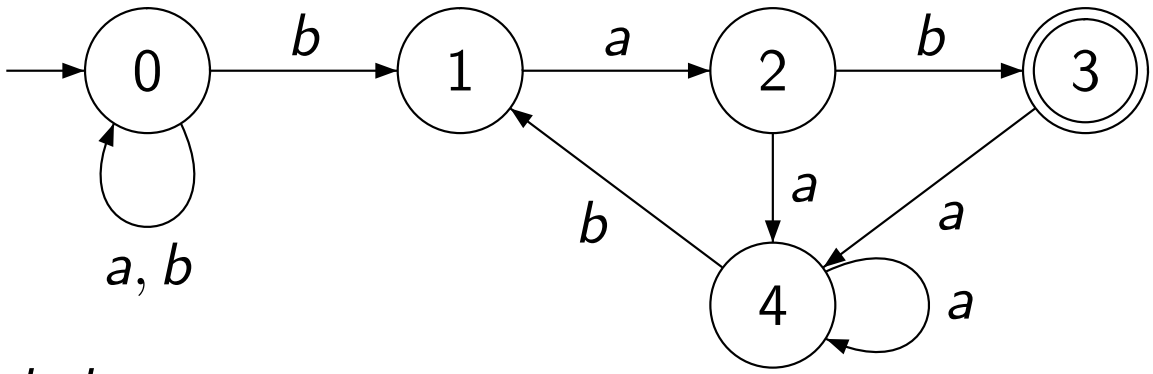


Word: *bababab*
Next letter: *a*

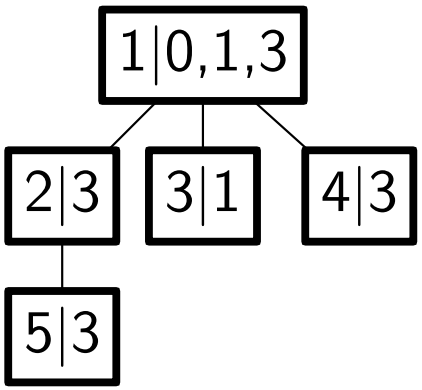


1. Unmark all nodes.
2. For each node copy the final states as its new youngest son.
3. For each node compute the powerset and replace the label.
4. Keep only the leftmost occurrence of each state under brothers.

Update of Safra trees

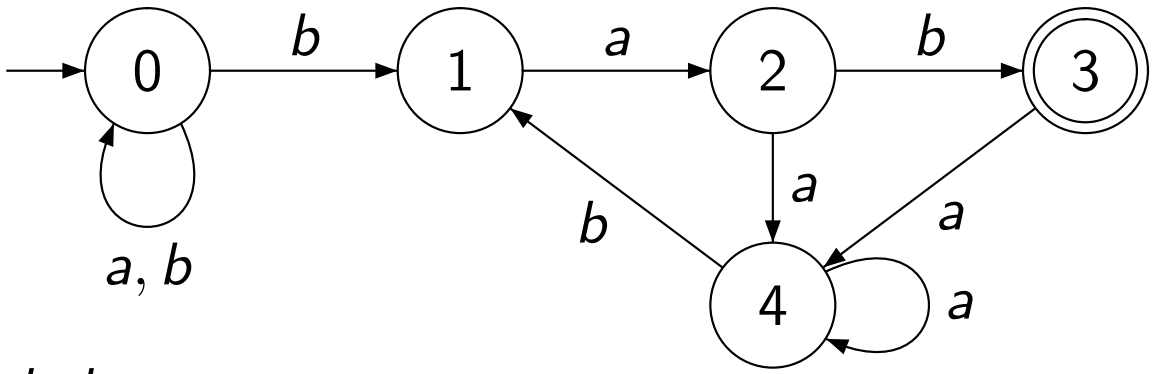


Word: *bababab*
Next letter: *a*

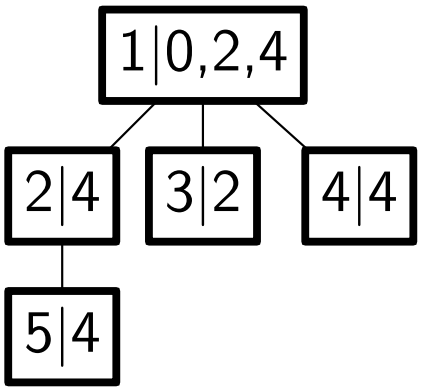


1. Unmark all nodes.
2. For each node copy the final states as its new youngest son.
3. For each node compute the powerset and replace the label.
4. Keep only the leftmost occurrence of each state under brothers.

Update of Safra trees

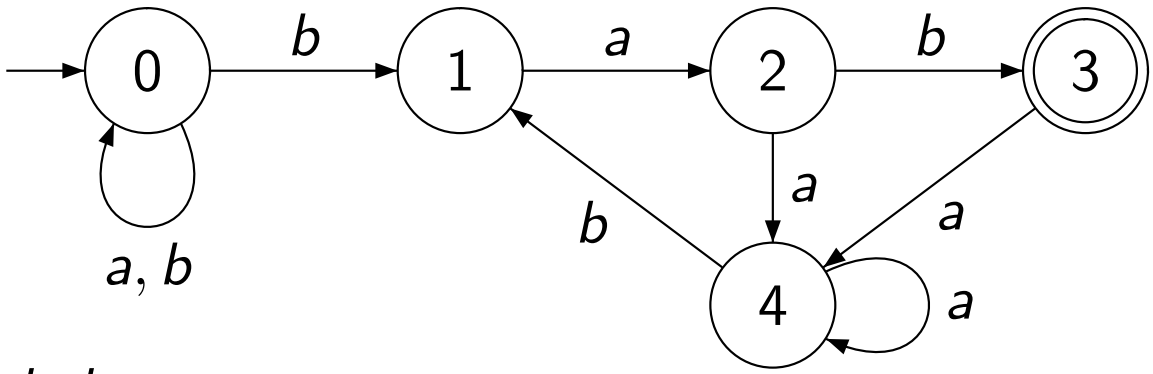


Word: *bababab*
Next letter: *a*

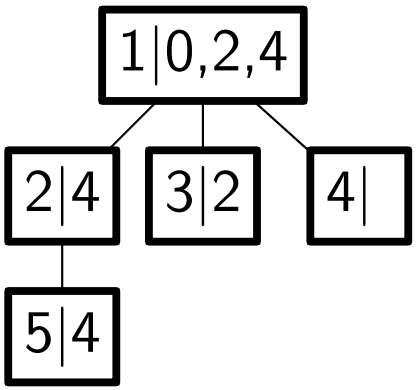


1. Unmark all nodes.
2. For each node copy the final states as its new youngest son.
3. For each node compute the powerset and replace the label.
4. Keep only the leftmost occurrence of each state under brothers.

Update of Safra trees

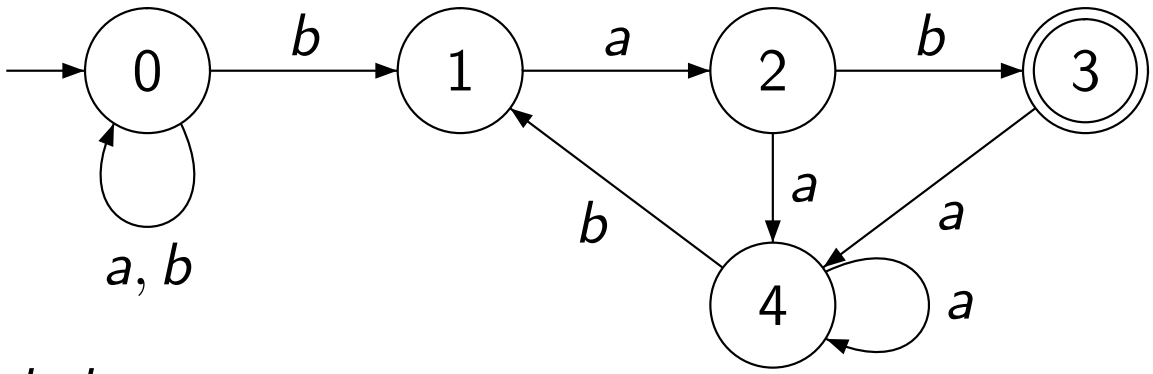


Word: *bababab*
Next letter: *a*

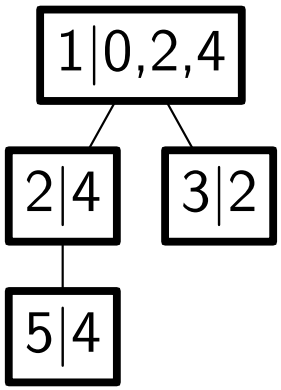


2. For each node copy the final states as its new youngest son.
3. For each node compute the powerset and replace the label.
4. Keep only the leftmost occurrence of each state under brothers.
5. Remove all nodes with empty labels.

Update of Safra trees

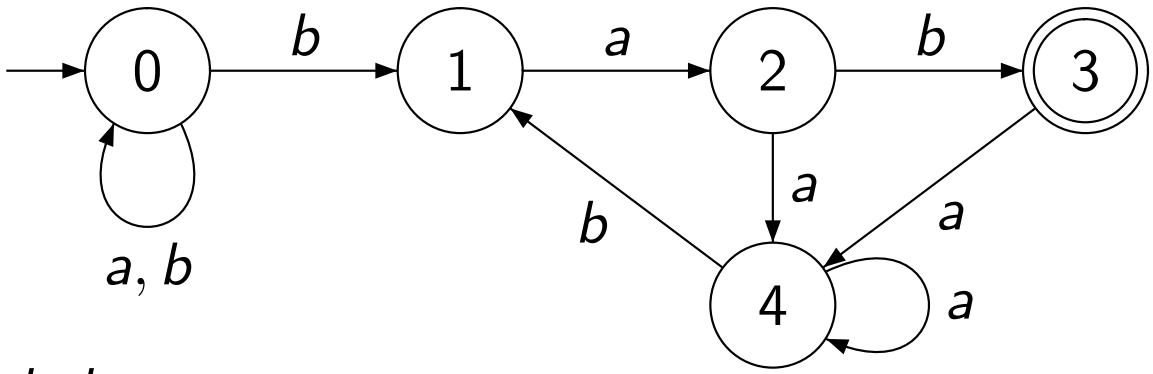


Word: *bababab*
Next letter: *a*

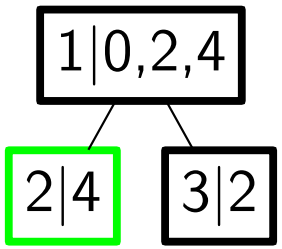


3. For each node compute the powerset and replace the label.
4. Keep only the leftmost occurrence of each state under brothers.
5. Remove all nodes with empty labels.
6. For every node whose label is equal to the union of the labels of its sons, remove all the descendants and color it green.

Update of Safra trees



Word: *bababab*
Next letter: *a*



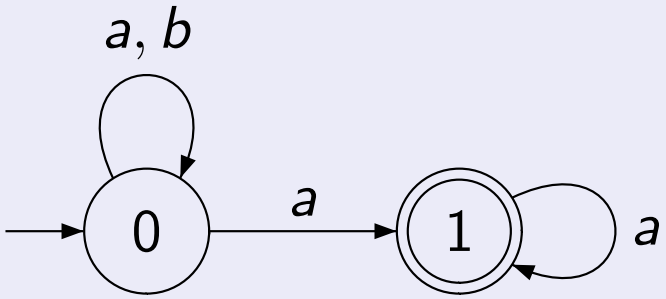
3. For each node compute the powerset and replace the label.
4. Keep only the leftmost occurrence of each state under brothers.
5. Remove all nodes with empty labels.
6. For every node whose label is equal to the union of the labels of its sons, remove all the descendants and color it green.

Input format

OmegaDet

`http://www-i7.informatik.rwth-aachen.de/d/research/omegadet.html`

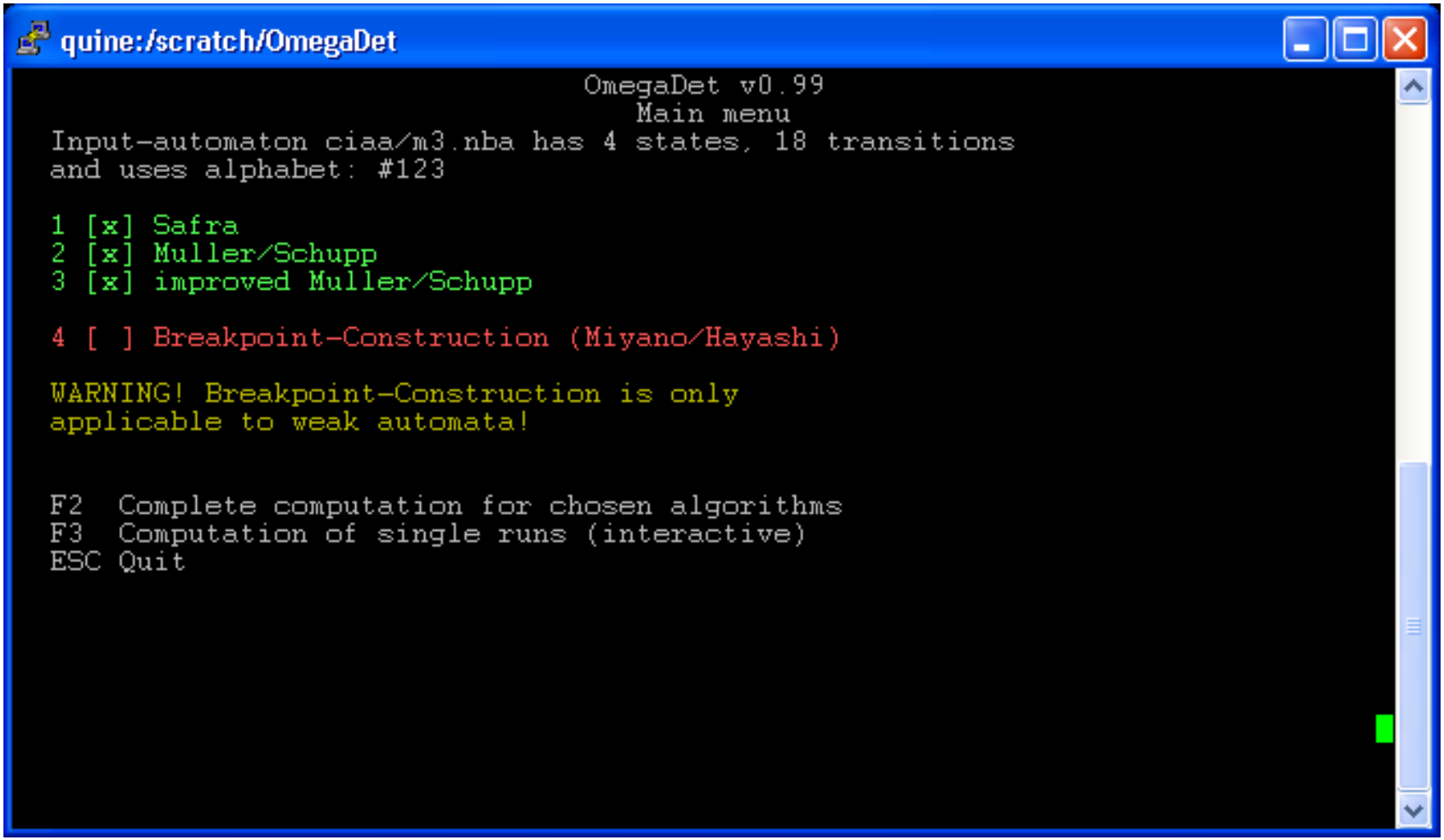
Input format




```
2
ab
1
0 a 0
0 b 0
0 a 1
1 a 1
```

- Number of states
- Alphabet
- Final states
- Transitions

Main menu


quine:/scratch/OmegaDet
OmegaDet v0.99
Main menu
Input-automaton ciaa/m3.nba has 4 states, 18 transitions
and uses alphabet: #123
1 [x] Safra
2 [x] Muller/Schupp
3 [x] improved Muller/Schupp
4 [ ] Breakpoint-Construction (Miyano/Hayashi)
WARNING! Breakpoint-Construction is only
applicable to weak automata!
F2 Complete computation for chosen algorithms
F3 Computation of single runs (interactive)
ESC Quit

Interactive mode

quine:/scratch/OmegaDet

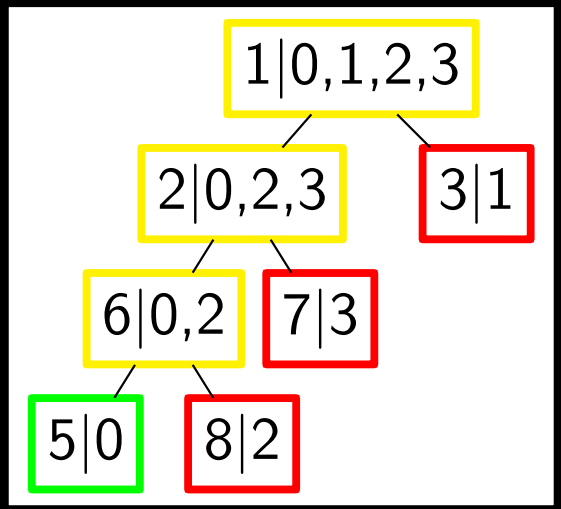
```
Input-automaton: ciaa/m3.nba with alphabet: #123
States after reading word: 112321221333212###2
ESC Quit     F2 Save current computations     F3 Reset

--------------------------------------------------------------------------------

Safra
[1|0,1,2,3]
 +-> [2|0,2]
 +-> [3|1]

Muller/Schupp
[1|0,1,2,3]0
 +-> [2|0,2,3]0
 |    +-> [6|0,2]0
 |    |    +-> [5|0]+
 |    |    +-> [8|2]-
 |    +-> [7|3]-
 +-> [3|1]-

improved Muller/Schupp
[1|0,1,2,3]0
 +-> [2|0,2,3]0
 |    +-> [6|0,2]0
 |    |    +-> [4|0]+
 |    |    +-> [5|2]-
 |    +-> [7|3]-
 +-> [3|1]-
```



Resulting automaton

```
Deterministic Rabin automaton
according to Safra:
4 States:
s0:
    [1|0]
s1: a
    [1|0,1]
s2: aa
    [1|0,1]
     +-> [2|1]
s3: aaa
    [1|0,1]
     +-> [2|1]!
```

- Number of states
- List of trees

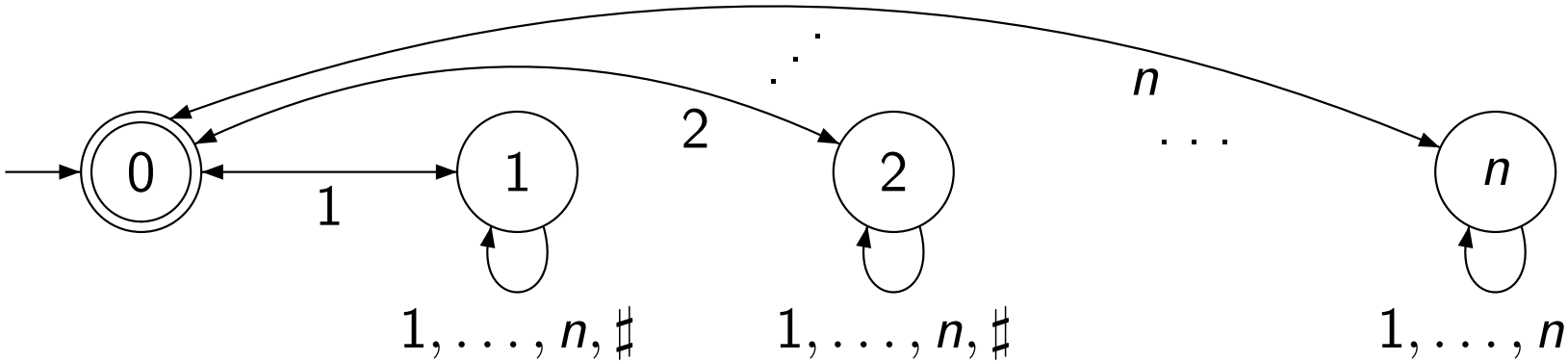
```
Transition table:
     a   b
s0   s1  s0
s1   s2  s0
s2   s3  s0
s3   s3  s0

Acceptance pairs:
for vertex 2 (sizes 2,1):
({s0,s1},{s3})

Overall: 1 pair with non-empty
acceptance set
```

- Transition table
- List of acceptance pairs

Example: worst case automata $\mathcal{M}_n$



	Safra		Muller-Schupp		Opt. Muller-Schupp	
	States	Pairs	States	Pairs	States	Pairs
$\mathcal{M}_1$	7	1	9	5	9	5
$\mathcal{M}_2$	33	2	4,058	8	262	7
$\mathcal{M}_3$	385	5	4,823,543	11	23,225	9
$\mathcal{M}_4$	13,601	7	memory exceeded		3,656,802	11
$\mathcal{M}_5$	1,059,057	9	memory exceeded		memory exceeded	

Conclusion

- Determinization procedures of Safra and Muller-Schupp
- Improvement of the Muller-Schupp algorithm
- Superiority of the Safra procedure for building small automata

Still needs more investigation:

- "Preprocessing": Analysis
- "Postprocessing": Reduction or minimization