REGAL: A library to randomly and exhaustively generate automata

Julien David, Frédérique Bassino, Cyril Nicaud
Institut Gaspard-Monge, Université Paris-Est, France
Julien.David@univ-mlv.fr

Overview

This poster presents a library in C++ for generating finite deterministic accessible automata.
`http://regal.univ-mlv.fr/`

A few words

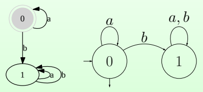
- *REGAL* generates automata with up to hundreds of thousands of states
- The type of both, the states and the alphabet, can be chosen by the user
- Easy to interface with other libraries
- Tools for gathering statistical information
- Around $3,000$ lines of C++ code
- Linux and Windows compatible

The generators

- Several Random generators
  1. Complete accessible DFAs
  2. Minimal DFAs
  3. Strongly connected DFAs
- An efficient exhaustive generator for complete accessible DFAs .

DOT and GasteX

- Straightforward graphical outputs



Examples

Random generation of $10,000$ **automata with** $30,000$ **states and** $\Sigma = \{'0','1'\}$

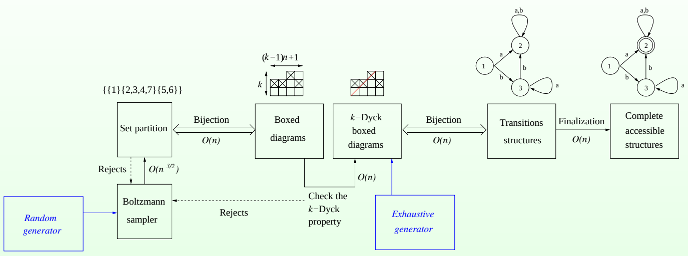
```
DFAAutomaton<int,char> * result;
Alphabet<char> alpha;
alpha.insert('0'); alpha.insert('1');
RandomDFAGenerator<int,char,DFAAutomaton<int,char>> rg(30000, alpha);
for(int counter=0; counter<10000; counter++) result=rg->random();
```

Exhaustive generation of automata with 6 **states and** $\Sigma = \{a, b, c\}$

```
DFAAutomaton<int,char> * result;
Alphabet<char> alpha;
alpha.insert('a'); alpha.insert('b'); alpha.insert('c');
ExhaustiveDFAGenerator<int,char,DFAAutomaton<int,char>> eg(6, alpha);
for(result=eg.first(); result!=eg.end(); result=eg.next()) {}
```

How does this work?

$k := |\Sigma|$
$n := \#$ of states



Uniform random generation[1]:

REGAL uses two bijections to generate the DFAs.

- `Boltzmann Sampler`[2]: algorithm that randomly generates objects according to a probability distribution. We use it to generate set partitions of size kn into n non-empty subsets.

Example: $n = 5$ and $k = 2$
We generate n integers whose sum is $kn = 10$: $3 + 2 + 2 + 2 + 1$.
Each integer corresponds to the size of one such subset.

$$\mathcal{P} = \{\{,,\},\{,\},\{,\},\{,\},\{\}\}$$

Then we fill the partition with a random permutation.

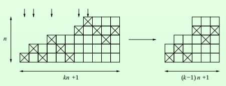
$$\mathcal{P} = \{\{1,3,6\},\{2,5\},\{4,10\},\{7,9\},\{8\}\}$$

- `First Bijection`:
  `From a set partition to a boxed diagram`. Each set partition is transformed into a boxed diagram of height n and width $(k-1)n+1$.

We randomly add an integer to the previous partition:

$$\mathcal{P} = \{\{1,3,6\},\{2,5\},\{4,10\},\{7,9,11\},\{8\}\}$$

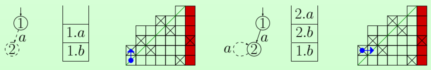
Each value in the set partition is an index of a column in the diagram and the index of its subset indicates the position of the cross in the column. The height of the columns is non-decreasing.



The columns whose indices correspond to the first value in the subsets of the partition are suppressed.

- `Second Bijection`:
  `From a boxed diagram to transition structure`.
  A transition structure is a DFA without its final states.

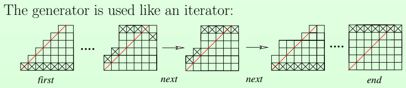
`Example`:



- *REGAL* determines with a probability p, fixed by the user, wether each state is final or not.

Exhaustive generation:

REGAL generates all the k-Dyck boxed diagrams, transforms them into transition structures and generates all possible subsets of final states.
The generator is used like an iterator:



References

[1] F. Bassino, C. Nicaud. Enumeration and random generation of accessible automata, *Theoret. Comput. Sci.*, available at http://www-igm.univ-mlv.fr/~bassino/publi.html, to appear.

[2] P. Duchon, P. Flajolet, G. Louchard, G. Schaeffer, Boltzmann Samplers for the Random Generation of Combinatorial Structures, *Combinatorics, Probability, and Computing*, Special issue on Analysis of Algorithms, 13 (2004), 577–625.

Experimental results for $|\Sigma| = 2$

Enumeration result

Number of `small minimal automata`:

Number of states	Minimal automata
2	24
3	$1,028$
4	$56,014$
5	$3,705,306$
6	$286,717,796$
7	$25,493,886,852$

Testing automata properties

Estimation of the proportion of `co-accessible` automata:

Number of states	10	100	500	1 000	2 000	3 000	4 000
Co-accessibles (%)	96.42	99.70	99.92	99.98	99.98	99.98	100.00

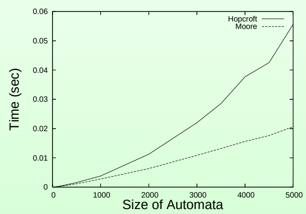
How much, in average, is the size of an automaton reduced by a minimization algorithm?

# of state reduction	*0*	1	2	3	4	5
Proportion of automata (in %)	*85.03*	13.75	1.10	0.11	0.01	0

Proportion of minimal automata

Minimization algorithm comparison

Average time complexity: Moore's vs Hopcroft's



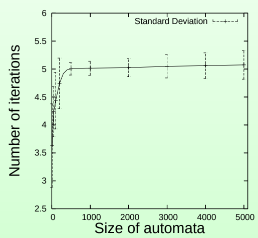
Worst-case complexity:

- Hopcroft : $O(n \log n)$
- Moore : $O(n^2)$

Moore's algorithm seems to be faster on average.

Minimization algorithm study

Number of iterations in Moore's algorithm :



The curve seems to be sub-linear, which could mean that the algorithm is sub-quadratic in the average.

CIAA'07, Prague, July 16-18