

OmAlg – Toolkit for the algebraic theory of omega automata

User Manual

April 1, 2016

Contents

1	Introduction	1
2	Installation	1
3	A first example	2
4	Executables	2
4.1	Description of programs	2
4.2	Command line options	2
5	File formats	3
5.1	Automata format	3
5.2	Example automata	5
5.3	Omega semigroup format	5
5.4	Example omega semigroups	7
	Bibliography	8

1 Introduction

Section 2 contains installation instructions. An introductory example on using OmAlg is given in section 3. Then sections 4 and 5 give a detailed description of all the executables, the command line arguments, and the file formats of the package.

2 Installation

- (i) Install CMake if it is not present on your system.
CMake is available at <https://cmake.org/download/>.
- (ii) Download or clone the OmAlg directory from github.
Link: <https://github.com/vanHavel/OmAlg>
- (iii) Open a command line window and navigate to the OmAlg directory. Then run

```
cmake
make
```

3 A first example

4 Executables

4.1 Description of programs

Name	Description	Input	Output
a2os	Transform an omega automaton into equivalent omega semigroups. No support for nondeterministic Muller or Parity automata.	An omega automaton in the OmAlg format(5.1).	An omega semigroup in the OmAlg format(5.3).
oslh	Decide whether the language of an omega semigroup can be recognized by a deterministic Buechi, coBuechi, weak Buechi, E- or A- automaton.	An omega semigroup in the OmAlg format(5.3).	A text describing which kinds of limited automata can accept the language.
os2p	Transform an omega semigroup into equivalent parity automaton.	An omega semigroup in the OmAlg format(5.3).	A deterministic parity automaton in the OmAlg format(5.1).
os2d	Transform an omega semigroup into equivalent det. Buechi automaton(if possible).	An omega semigroup in the OmAlg format(5.3).	A deterministic Buechi automaton in the OmAlg format(5.1).
os2c	Transform an omega semigroup into equivalent det. coBuechi automaton(if possible).	An omega semigroup in the OmAlg format(5.3).	A deterministic coBuechi automaton in the OmAlg format(5.1).
os2w	Transform an omega semigroup into equivalent weak Buechi automaton(if possible).	An omega semigroup in the OmAlg format(5.3).	A deterministic (weak) Buechi automaton in the OmAlg format(5.1).

4.2 Command line options

All of the command line options can be supplied to all the executables.

Short name	Long name	Description
-i	--input-file	Path to the input file containing the program input. If this option is not given, the input is read from stdin and a warning is printed to stderr, unless -s is set.
-o	--output-file	Path to the output file for the program input. If this option is not given, the output is written to stdout and a warning is printed to stderr, unless -s is set.
-h	--help	Print help text.
-s	--suppress-output	If this flag is set, no warnings are written to stderr.

5 File formats

General information

- The file formats were designed to be easy to parse by a machine and humans both, so that one can easily supply simple examples by hand. The downside is that they lack the expressive power of more complex formats like the Hanoi Omega Automata Format[1].
- Whenever a **list of names** has to be given in one of the formats, it is expected to be a comma separated list of names

`<name_1>, <name_2>, ..., <name_n>;`

terminated by a semicolon(;) and a newline. You are allowed to split this list over several lines – in this case only the last line is terminated by a semicolon and all other lines are terminated by a comma(after the last name of that line).

For a name itself one can use lower case and upper case letters, numbers and some special characters like '%' or '@'. The comma(,) and the semicolon(;) character must not be used inside names. In general different types parantheses can be used inside names, but there are some exceptions explained in detail in the following sections. However, it is always safe to use square brackets('[]') and angle brackets('<>').

5.1 Automata format

The text file format used for representing automata has the following structure:

- Acceptance condition
- Determinism
- State names
- Alphabet
- Initial state
- Transitions
- Acceptance component

Each component must be terminated by a semicolon(;) and a newline.
We now describe the format of each component in greater detail.

Acceptance condition

Specifies which acceptance condition should be used by the omega automaton. Possible values are “Buechi”, “coBuechi”, “Muller” and “Parity”.

Determinism

Specifies whether the automaton is deterministic or not. Possible values are “deterministic” and “nondeterministic”.

Of course, every deterministic automaton is also a nondeterministic one. But deterministic automata can be stored with more efficient data structures and handled more efficiently in the algorithms. Therefore, one should always choose “deterministic” if it is known that the input automaton is deterministic.

State names

A list of names which should be used as state labels for the automaton. Must contain at least one name and may not contain duplicates. This automatically specifies the number of states in the automaton. In the case of Muller automata, curly braces may not appear in any state name.

Alphabet

A list of names which corresponds to the alphabet symbols read by the automaton. Must contain at least one name and may not contain duplicates. This automatically specifies the alphabet size.

Initial state

The name of the initial state. Must appear in the list of state names.

Transitions

A list of names representing the transitions of the automaton. Each name must have the format

(source, letter, target)

where source and target occur in the list of state names and letter occurs in the list of alphabet symbols. This codes a transition from the state *source* with letter *letter* to state *target*.

If the automaton was specified to be deterministic, there must be exactly one transition (source, letter, target) for each fixed pair of state *source* and letter *letter*. But the transitions do not have to be given in any exact order.

Acceptance component

The format of this component depends on the acceptance condition written in the first line.

In the case of **Buechi** and **CoBuechi** automata, the acceptance component is a list of names: the list of all final states of the automaton. All names in this list must occur as state names.

In the case of **Parity** automata, the acceptance component is a list of names: the list of the state priorities. Therefore, each of the names must be a nonnegative integer. The list must contain exactly as many elements as the list of state names, and the i -th entry in the priority list corresponds to the priority of the i -th state in the state name list.

In the case of **Muller** automata, the acceptance component is a list of names: the list of all state sets in the Muller automaton's table. Each of the names must be of the format

`{state_1, state_2, ..., state_k}`

where all of the comma separated names inside the curly braces must appear in the list of state names.

5.2 Example automata

5.3 Omega semigroup format

We always store omega semigroups in combination with a morphism ϕ from a finite alphabet Σ to the semigroup S_+ and a set of recognizing elements P . The text file format used for representing omega semigroups $S = (S_+, S_\omega)$, morphism $\phi : \Sigma \rightarrow S_+$ and $P \subseteq S_\omega$ has the following structure:

- S_+ element names
- S_+ multiplication table
- S_ω element names
- Mixed product table
- Omega iteration table
- Alphabet
- Morphism images
- P

Like for automata, each component must be terminated by a semicolon(;) and a newline. We now describe the format of each component in greater detail.

S_+ element names

A list of names which should be used as names for the semigroup elements. Must contain at least one name and may not contain duplicates. This automatically specifies the order of the semigroup S_+ .

S_+ multiplication table

Let n be the number of semigroup elements(specified by the above list) and $s_i, 1 \leq i \leq n$ the i -th semigroup element.

The semigroup multiplication table is stored as a sequence of n lines. The i -th line contains the products of s_i with all the elements s_1, \dots, s_n in order, separated by commas. The last line is terminated by a semicolon, all other lines are simply terminated by a newline(without a final comma).

Every name in the table must occur as an S_+ element name. The operation defined by the table must be associative.

S_ω element names

A list of names which should be used as names for the S_ω elements. Must contain at least one name and may not contain duplicates. This automatically specifies the size of the set S_ω .

Mixed product table

Let n be the number of semigroup elements(specified by the first list) and $s_i, 1 \leq i \leq n$ the i -th semigroup element. Further, let m be the number of S_ω elements and $t_j, 1 \leq j \leq m$ the elements of S_ω .

The mixed product table is stored as a sequence of n lines. The i -th line contains the products of s_i with all the elements t_1, \dots, t_m in order, separated by commas. The last line is terminated by a semicolon, all other lines are simply terminated by a newline(without a final comma).

Every name in the table must occur as an S_ω element name. The operation defined by the table must be compatible with the S_+ semigroup product.

Omega iteration table

Let n be the number of semigroup elements(specified by the first list) and $s_i, 1 \leq i \leq n$ the i -th semigroup element. The omega iteration table is a list of names corresponding to the products $s_1^\omega, \dots, s_n^\omega$. Every name must occur as an S_ω element name, and there must be exactly n names. The operation must conform to the rules

$$(s^n)^\omega = s^\omega, \quad n \in \mathbb{N} \qquad s(ts)^\omega = (st)^\omega$$

Alphabet

A list of names which corresponds to the symbols of the alphabet of the recognized language. Must contain at least one name and may not contain duplicates. This automatically specifies the alphabet size.

Morphism images

A list of names which contains the morphism images of the alphabet symbols, in the order in which they were given in the previous component. Must have the same number of elements as the alphabet size. Every name must occur as an S_+ element name.

P

A list of names of the elements in the recognizing set P . Every name must occur as an S_ω element name.

5.4 Example omega semigroups

References

- [1] Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The hanoi omega-automata format. In *Computer Aided Verification*, pages 479–486. Springer, 2015.