OmAlg – Toolkit for the algebraic theory of omega automata

User Manual

April 1, 2016

Contents

1 Introduction

2 Installation

3 A first example

4 Executables

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

5.4 Example omega semigroups

References

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