The Hanoi Omega-Automata Format

Tomáš Babiak[1], František Blahoudek[1], Alexandre Duret-Lutz[2], Joachim Klein[3], Jan Křetínský[5], David Müller[3], David Parker[4], and Jan Strejček[1]

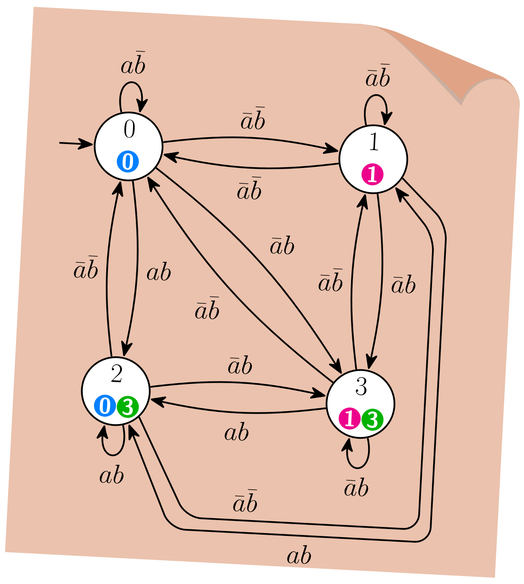
[1]Faculty of Informatics, Masaryk University, Brno, Czech Republic
[2]LRDE, EPITA, Le Kremlin-Bicêtre, France
[3]Technische Universität Dresden, Germany
[4]University of Birmingham, UK
[5]IST Austria

Presented at CAV'15

A Rabin automaton for $\mathbf{GF}\,a \to \mathbf{GF}\,b$

```
HOA: v1
States: 4
Start: 0
AP: 2 "a" "b"
acc-name: Rabin 2
Acceptance: 4 (Fin(0)&Inf(1))|(Fin(2)&Inf(3))
--BODY--
State: 0 {0}
[!0&!1] 1
[0&!1] 0
[!0&1] 3
[0&1] 2
State: 1 {1}
[!0&!1] 1
[0&!1] 0
[!0&1] 3
[0&1] 2
State: 2 {0 3}
[!0&!1] 1
[0&!1] 0
[!0&1] 3
[0&1] 2
State: 3 {1 3}
[!0&!1] 1
[0&!1] 0
[!0&1] 3
[0&1] 2
--END--
```

Atomic propositions are denoted by their indices (0 and 1) in the body.

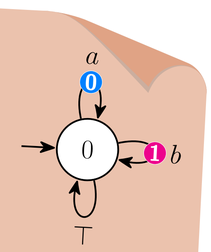


Since this automaton is deterministic and complete, the transition labels can be omitted to shorten the file.

A transition-based Streett automaton for $\mathbf{GF}\,a \to \mathbf{GF}\,b$

A header that starts with a lowercase letter (such as `tool`, `name`, `acc-name`, `properties`...) can be safely ignored without impact on the semantics.

```
HOA: v1
tool: "toolname" "1.2.3"
name: "GF a -> GF b"
States: 1
Start: 0
acc-name: Streett 1
Acceptance: 2 Fin(0) | Inf(1)
AP: 2 "a" "b"
properties: trans-labels explicit-labels
    trans-acc stutter-invariant complete
--BODY--
State: 0
[0] 0 {0}
[1] 0 {1}
[t] 0
--END--
```



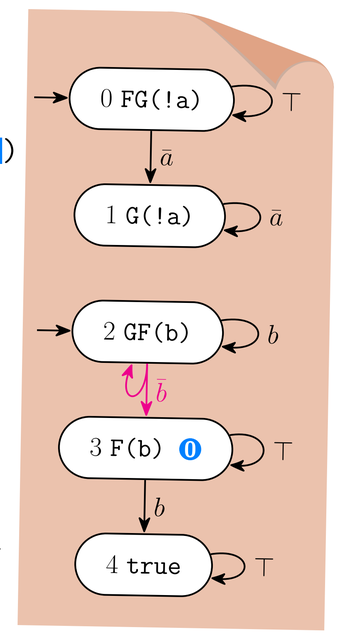
Optional properties can give information about the syntax used in the body (e.g., `explicit-labels` vs. `implicit-labels`), the structure of the automaton (e.g., `deterministic`, `complete`), and the language recognized (e.g., `stutter-invariant`).

An alternating co-Büchi automaton for $\mathbf{GF}\,a \to \mathbf{GF}\,b$

Nondeterministic initial states.

Universal branching is also possible initially. For instance `Start: 2&3`.

```
HOA: v1
States: 5
Start: 0
Start: 2
AP: 2 "a" "b"
acc-name: co-Buchi
Acceptance: 1 Fin(0)
--BODY--
State: 0 "FG(!a)"
[t] 0
[!0] 1
State: 1 "G(!a)"
[!0] 1
State: 2 "GF(b)"
[1] 2
[!1] 2&3
State: 3 "F(b)" {0}
[1] 4
State: 4 "true"
[t] 4
--END--
```



States may be named (for display and debugging).

Open development

The format is developed on GitHub at
`https://github.com/adl/hoaf`
Feel free to make suggestions or report bugs on the issue tracker.

Generic acceptance

`Acceptance:` n acc specifies the acceptance condition using the following grammar:

$$acc \;::=\; \mathtt{f} \mid \mathtt{t} \mid \mathtt{Inf}(s) \mid \mathtt{Inf}(!s) \mid \mathtt{Fin}(s) \mid \mathtt{Fin}(!s) \mid acc\,\&\,acc \mid acc \mid acc \mid (acc)$$

Where s is an accepting set number smaller than n, $!s$ denotes the complement of that set. `Fin` (resp. `Inf`) is satisfied when the set is visited finitely (resp. infinitely) often by a run. For alternating automata all branches of a run-tree have to satisfy the condition.

Known acceptance conditions can be named with the optional `acc-name:` header.

```
acc-name: Buchi
Acceptance: 1 Inf(0)
```

```
acc-name: co-Buchi
Acceptance: 1 Fin(0)
```

```
acc-name: all
Acceptance: 0 t
```

```
acc-name: generalized-Buchi 3
Acceptance: 3 Inf(0)&Inf(1)&Inf(2)
```

```
acc-name: Streett 2
Acceptance: 4 (Fin(0)|Inf(1))&(Fin(2)|Inf(3))
```

```
acc-name: generalized-Rabin 2 3 2
Acceptance: 7 (Fin(0)&Inf(1)&Inf(2)&Inf(3))|(Fin(4)&Inf(5)&Inf(6))
```

```
acc-name: parity min even 5
Acceptance: 5 Inf(0) | (Fin(1) & (Inf(2) | (Fin(3) & Inf(4))))
```

Of course acceptance conditions can be created as needed, they do not require a name.

Tool support

`ltl2dstar 0.5.3`: creates deterministic automata from LTL or Büchi automata
inputs BA, outputs DRA or DSA.

`ltl3ba 1.1.2`: creates automata from LTL
outputs BA, TGBA, or VWAA.

`ltl3dra 0.2.2`: creates deterministic automata from (a subset of) LTL
outputs DRA, TGDRA or MMAA.

`Rabinizer 3`: creates deterministic automata from LTL
outputs DRA, TDRA, GDRA, or TGDRA.

`PRISM 4.3`: probabilistic LTL model checking using deterministic HOA automata; (generalized) Rabin for MDP, any acceptance for CTMC/DTMC; scripts for interfacing with the tools above.

Chatterjee et al. (CAV'13) observed order-of-magnitude speedups replacing Rabin acceptance by generalized Rabin for probabilistic model checking with PRISM.

`Spot 1.99.1`: tool suite for LTL/PSL and automata manipulation
can input/output anything that is not alternating; translates between formats (like never claim or LBTT); has several automata transformations; the tool `ltlcross` can be used to validate translators from LTL/PSL to automata with any acceptance condition.

`jhoafparser/cpphoafparser`: Java and C++ parser libraries with pretty printers, validation, and convenient transformations, to easily develop new consumer tools; `jhoafparser` is used by `PRISM`, `cpphoafparser` is used by `ltl2dstar`.

Up-to-date tool support can be found at
`http://adl.github.io/hoaf/support.html`

If you implement HOA support, tell us so we can list your tool there.

Batch processing

The `--END--` marker allows multiple automata to be chained and be batch-processed by a pipe of several commands.

```
ltl3dra -f 'GFa -> GFb' > aut.hoa
randaut -n -1 -A'Rabin 2' -d.1 a b -H | autfilt -n 10 --intersect=aut.hoa --is-unambiguous -H >result.hoa
```

Generate an infinite number of random "Rabin 2" automata as HOA.

Keep the first 10 that intersect `aut.hoa` and are unambiguous; output them in the HOA format.

Trivia

Work on this format started during the ATVA'13 conference in **Hanoi** (Vietnam). Hence the name.