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

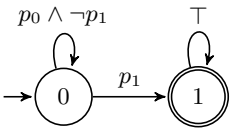
Abstract. We propose a flexible exchange format for ω-automata, as typically used in formal verification, and implement support for it in a range of established tools. Our aim is to simplify the interaction of tools, helping the research community to build upon other people's work. A key feature of the format is the use of very generic acceptance conditions, specified by Boolean combinations of acceptance primitives, rather than being limited to common cases such as Büchi, Streett, or Rabin. Such flexibility in the choice of acceptance conditions can be exploited in applications, for example in probabilistic model checking, and furthermore encourages the development of acceptance-agnostic tools for automata manipulations. The format allows acceptance conditions that are either state-based or transition-based, and also supports alternating automata.

1 Introduction

Finite automata over infinite words, ω-automata, play a crucial role in formal verification. For instance, they are a key component in the automata-theoretic approach to LTL model checking [21], where the property in question is encoded as an ω-automaton. There is a long history of research and ongoing tool development, trying to produce more compact automata in theory and in practice.

Formats to represent ω-automata have mostly been defined in an ad-hoc manner, tailored to their particular tools, setting and scope, and tend to be restricted to a few specific acceptance conditions. For classical Büchi automata,

* T. Babiak, F. Blahoudek, and J. Strejček have been supported by The Czech Science Foundation, grant GBP202/12/G061. J. Klein and D. Müller have been supported by the DFG through the collaborative research centre HAEC (SFB 912), the Excellence Initiative by the German Federal and State Governments (cluster of excellence cfAED and Institutional Strategy), the Graduiertenkolleg QuantLA (1763), and the DFG/NWO-project ROCKS, and the EU-FP-7 grant MEALS (295261). J. Křetínský has been supported in part by the European Research Council (ERC) under grant 267989 (QUAREM), by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award), and by the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement No 291734.



(a) The automaton...

```
2 1
0 1 -1
1 p1
0 & p0 ! p1
-1
1 0 0 -1
1 t
-1
```

(b) ... in LBT's format

```
never { /* p0 U p1 */
T0_init:
  if
  :: ((p1)) ->
        goto accept_all
  :: ((p0) && (!(p1))) ->
        goto T0_init
  fi;
accept_all:
  skip
}
```

(c) ... as a never claim

```
HOA: v1
name: "p0 U p1"
States: 2
Start: 0
AP: 2 "p0" "p1"
acc-name: Buchi
Acceptance: 1 Inf(0)
--BODY--
State: 0
 [1] 1
 [0&!1] 0
State: 1 {0}
 [t] 1
--END--
```

(d) ... in HOA format

Fig. 1: A Büchi automaton for the LTL formula $p_0 \mathsf{U} p_1$ encoded in three formats.

tools often use Spin's never claims [8] (see Fig. 1(c)), or LBT's format [17] (see Fig. 1(b)), which can also represent generalized Büchi automata and which was extended with transition-based acceptance by LBTT [19]. For Rabin and Streett automata, the format of `ltl2dstar` [10] can be used, provided those automata are complete, deterministic, and use state-based acceptance.

The one format that covers most common acceptance conditions (Büchi, generalized Büchi, co-Büchi, Rabin, Streett, etc.) and automata structures (deterministic, non-deterministic, and alternating) is the XML-based Goal File Format (GFF) used internally by the Goal tool [20]. It uses specific encodings for the different acceptance conditions. For instance, there is a special notation to define the sets in each acceptance pair of Rabin conditions. This necessitates changes to the format and its parsers when introducing new acceptance conditions and makes acceptance-agnostic manipulations difficult.

Based on our experience as implementers of tools producing, consuming, and manipulating ω-automata, we have set out to define a common, flexible, and extensible format for representing ω-automata in a uniform way. The result is the *Hanoi Omega-Automata (HOA)* format.[1] A crucial feature is the introduction of a *generic* way to specify the acceptance condition as an arbitrary Boolean formula over the acceptance primitives "infinitely often" and "finitely often", covering the common acceptance conditions discussed so far and more.

Firstly, this approach facilitates the exchange and usage of new acceptance conditions, which can provide important gains in efficiency. For instance, the generalized Rabin condition [13] has led to an orders-of-magnitude speed-up of probabilistic LTL model checking [3, 12]. Secondly, it offers flexibility in the choice of acceptance conditions, which can again be quite beneficial in practice, such as for deterministic Streett and Rabin automata [9], where there is an exponential worst-case size difference in both directions [16].

[1] The discussion about this format started during ATVA'13 in Hanoi, hence the name.

Thirdly, arbitrary Boolean combinations of acceptance conditions can be exploited. For example, building a deterministic automaton for an LTL formula using a product of the automata constructed for its subformulas can be beneficial in practice [9]. But this normally only works when the structure of the formula and acceptance condition are aligned, e.g., conjunctive formulas and a conjunctive acceptance condition such as Streett. With generic acceptance, it becomes possible to compositionally construct automata using disjunction, conjunction, and negation of deterministic automata with unrelated acceptance conditions. For some verification problems, such as probabilistic model checking of LTL in Markov chains, this generic acceptance condition can be used directly for verification.

The HOA format offers flexibility in other respects too. It supports various structural variants of ω-automata such as labels on states or transitions and state-based or transition-based acceptance, and can describe deterministic, non-deterministic, and alternating automata. Despite its generality, the format also contains features that allow a concise and readable representation in special circumstances, such as when dealing with deterministic complete automata, where the number of transitions per state is constant.

We have implemented support for the HOA format in various established tools, as detailed in Section 3, and are already seeing several of the intended benefits. Interaction between existing tools has become significantly easier: they are no longer restricted by the particular format of automata used, but only by the algorithms implemented to work with them. This shortens development time and can bring performance gains, as described above. It also facilitates research into new types of automata; for instance the intermediate co-Büchi alternating automata built by `ltl3ba` can now be exported to an easily-readable format. More generally, we hope to stimulate the development of acceptance-agnostic tools for the automata construction pipeline, e.g., for doing structural transformations such as switching between state- and transition-based acceptance or for reduction algorithms that do not rely on a particular acceptance condition.

2 Main Features of the HOA Format

The HOA format currently supports the following:

- deterministic, non-deterministic, and alternating ω-automata,
- both state-labelled and transition-labelled ω-automata,
- generic acceptance conditions, specified in a uniform and extensible way,
- both state-based and transition-based acceptance.

The format was also designed to:

- be succinct and human-readable,
- be extensible, by allowing additional information to be stored in the headers,
- support streaming, for processing automata in batches.

The full specification of the format and some examples can be found at `http://adl.github.io/hoaf/`. Below, we discuss a few of the most important features.

As seen in Figure 1(d), an automaton is defined in two parts: a header that specifies the characteristics of the automaton, and a body that gives the transition structure, the labels of states or transitions (in square brackets), and the acceptance sets (in curly brackets). Numbers in the body outside any brackets always refer to states. Labels (in square brackets) are Boolean formulas over integers that index the atomic propositions listed in the `AP:` header. Using indices instead of atomic propositions makes it easy to rename an atomic proposition, and allows using arbitrarily long names without bloating the resulting file.

Header lines that start with a capital letter are supposed to affect the semantics of the automaton, while header lines that start with a lower-case letter are only informative. The HOA specification reserves a few header names, but additional headers can be added as needed. This gives an easy and robust way for automata producers to extend the format and emit additional information about the automaton: Consumers that encounter a capitalized header they do not understand should report an error, but can safely ignore a lower-case one.

The `Acceptance:` line specifies the acceptance condition formally. This line has the form "`Acceptance:` n acc", where n gives the number of acceptance sets used, subsequently named $0, \ldots, n-1$, and acc is a formula built according to the following grammar.

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

Above, s denotes one of the acceptance sets. Membership in these sets for states and transitions is defined in the body of the automaton. A run satisfies an acceptance primitive `Inf(`s`)` or `Fin(`s`)` iff it visits the acceptance set s infinitely often or at most finitely often, respectively. The same notations with $!s$ refer to the complement of the set s.[2] A run is accepting if it satisfies the acceptance condition acc. We do not need a negation operator, as negation can be pushed into the acceptance primitives, e.g., $\neg$`Inf(`s`)` is equivalent to `Fin(`s`)`.

In the case of Figure 1(d), there is only one acceptance set, and accepting runs should visit this acceptance set infinitely often. In the body of the automaton, state 1 is marked with `{0}`, meaning that it belongs to the set `0`.

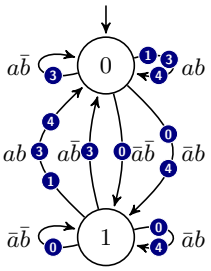
Rabin acceptance with 3 pairs of acceptance sets could be defined as follows:

```
Acceptance: 6 (Fin(0)&Inf(1))|(Fin(2)&Inf(3))|(Fin(4)&Inf(5))
```

Here, a run is accepting if it visits set 0 finitely and set 1 infinitely often, or set 2 finitely and set 3 infinitely often, or analogously for sets 4 and 5.

Figure 2 shows an example of a transition-based generalized deterministic Rabin automaton (such as produced internally by `ltl3dra` before optimizations). Here, acceptance sets are expressed in terms of transitions. As a final example, Figure 3 shows an alternating transition-based co-Büchi automaton, such as those studied by Tauriainen [18]. Alternation is supported by allowing a transition to have multiple destinations. Runs over alternating automata are trees, and in this example a run is accepting iff the only transition in the acceptance

[2] Readers familiar with LTL can interpret `Inf(`s`)`, `Fin(`s`)`, `Inf(!`s`)`, `Fin(!`s`)` as meaning $\mathsf{GF}p_s$, $\mathsf{FG}\neg p_s$, $\mathsf{GF}\neg p_s$, $\mathsf{FG}p_s$, where p_s is the property "belongs to set s".



(a) The automaton...

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

(b) ... and its HOA representation.

Fig. 2: A (non-simplified) transition-based generalized deterministic Rabin automaton for the LTL formula $\mathsf{G}(\mathsf{F}a \wedge \mathsf{XF}b)$.

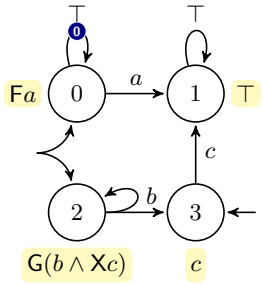
set `0` is visited finitely often in all the branches, as specified by the `Acceptance:` line. This example also demonstrates that states may be named.

In general, most of the tools that are the ultimate consumers of HOA automata, such as model checkers, will employ algorithms restricted to particular acceptance conditions. There are often multiple ways to syntactically structure the acceptance condition. For example, the Rabin acceptance can be expressed with the sets in the pairs swapped or complemented, as in Krishnan et al. [14]. Therefore, we specify canonical expression and acceptance set indices for the common acceptance conditions, and an optional `acc-name:` header line which helps tools to detect acceptance conditions they support. However, as discussed in the introduction, some verification procedures can make direct use of generic acceptance conditions.

3 Application Support

We have implemented support for HOA in a range of tools, with the current status available at `http://adl.github.io/hoaf/support.html`, including links to releases of each tool and a Live CD ISO for easy investigation of them all.

HOA generation. Generating automata in the HOA format is now supported by several tools: `ltl2dstar` [10], which translates LTL to determinis-



(a) The automaton...

```
HOA: v1
name: "(Fa&G(b&Xc))|c"
States: 4
Start: 0&2
Start: 3
acc-name: co-Buchi
Acceptance: 1 Fin(0)
AP: 3 "a" "b" "c"
--BODY--
State: 0 "Fa"
  [t] 0 {0}
  [0] 1
State: 1 "true"
  [t] 1
State: 2 "G(b&Xc)"
  [1] 2&3
State: 3 "c"
  [2] 1
--END--
```

(b) ... and its HOA representation.

Fig. 3: Alternating transition-based co-Büchi automaton for $(\mathsf{F}a \wedge \mathsf{G}(b \wedge \mathsf{X}c)) \vee c$.

tic Rabin or Street automata; `ltl3ba` [1], which generates Büchi automata, transition-based generalized Büchi automata, and very weak alternating co-Büchi automata; `ltl3dra` [2], which converts a fragment of LTL to deterministic Rabin automata, transition-based generalized deterministic Rabin automata, and very weak alternating co-Büchi automata; and `Rabinizer3` [12], which translates LTL into state- and transition-based variants of deterministic Rabin automata and generalized deterministic Rabin automata.

Furthermore, `Spot` [6] offers many tools for generating automata in the HOA format: `ltl2tgba` [5] can translate LTL/PSL into Büchi automata, transition-based generalized Büchi automata or monitors; `randaut` generates random Büchi automata, transition-based generalized Büchi automata or monitors; and finally `dstar2tgba` converts deterministic automata in the `dstar` format into Büchi automata, transition-based generalized Büchi automata or monitors. The `Spot` tool `autfilt` filters, transforms, and converts formats for Büchi automata, generalized Büchi automata, and monitors and supports reading and writing HOA, with `ltldo` wrapping other LTL/PSL-to-automata translators to convert their input and output. This command and the previous one can be used to produce HOA output from existing tools that only output never claims or the LBT format.

HOA parsing. There are two parsers for the HOA format. The first, in C++, is included in `Spot` and is able to read a stream of automata whose format can be either HOA, LBT or never claim. This parser powers the tools `autfilt` and `ltldo` (presented above), and also `ltlcross` [4] (a verifier for LTL translators). At the time of writing, `Spot` does not yet support alternation.

The second is the `jhoafparser` library [11], which provides a Java-based parser. This provides a convenient interface for applications to consume the different elements of the HOA format, taking care of basic sanity checks. The library is accompanied by a command-line tool that checks the well-formedness of an automaton in the HOA format and performs basic manipulations.

HOA import. We have extended the probabilistic model checker `PRISM` [15] to interface with external tools for the conversion from LTL to deterministic automata. This is done using the HOA format and `jhoafparser`. In parallel, we have expanded `PRISM`'s ω-automata verification procedures: Markov chains can now be model checked against generic acceptance conditions, giving producers of deterministic automata full flexibility in terms of acceptance conditions. Markov decision processes can be checked against both generalized or standard Rabin acceptance conditions. As a result, we have successfully interfaced `PRISM` with `Rabinizer3`, `ltl2dstar`, and `ltl3dra`.

4 Conclusion

We have presented a new format for ω-automata that supports generic acceptance conditions, and implemented it in several tools. Besides smoothing the interaction between tools, this representation of acceptance conditions allows a significant flexibility and performance increase, which has already been harnessed in `PRISM`, and encourages tool developers to expand the range of supported acceptance conditions. The HOA format has been developed openly on GitHub, and an issue tracker keeps a public archive of our discussion and decisions. We encourage other tool authors to report issues and suggest improvements.

References

1. T. Babiak, M. Křetínský, V. Řehák, and J. Strejček. LTL to Büchi automata translation: Fast and more deterministic. In *TACAS'12*, vol. 7214 of *LNCS*, pp. 95–109. Springer, 2012.
2. T. Babiak, F. Blahoudek, M. Křetínský, and J. Strejček. Effective translation of LTL to deterministic Rabin automata: Beyond the (F, G)-fragment. In *ATVA'13*, vol. 8172 of *LNCS*, pp. 24–39. Springer, 2013.
3. K. Chatterjee, A. Gaiser, and J. Křetínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In *CAV'13*, vol. 8044 of *LNCS*, pp. 559–575. Springer, 2013.
4. A. Duret-Lutz. Manipulating LTL formulas using Spot 1.0. In *ATVA'13*, vol. 8172 of *LNCS*, pp. 442–445. Springer, 2013.
5. A. Duret-Lutz. LTL translation improvements in Spot 1.0. *International Journal on Critical Computer-Based Systems*, 5(1/2):31–54, Mar. 2014.
6. A. Duret-Lutz and D. Poitrenaud. SPOT: an Extensible Model Checking Library using Transition-based Generalized Büchi Automata. In *MASCOTS'04*, pp. 76–83. IEEE Computer Society Press, 2004.
7. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV'95*, pp. 3–18. Chapman & Hall, 1996.
8. G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.
9. J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. *Theoretical Computer Science*, 363(2):182–195, 2006.
10. J. Klein and C. Baier. On-the-fly stuttering in the construction of deterministic ω-automata. In *CIAA'07*, vol. 4783 of *LNCS*, pp. 51–61. Springer, 2007.
11. J. Klein and D. Müller. The jhoafparser library. `http://automata.tools/hoa/jhoafparser/`, 2015.
12. Z. Komárková and J. Křetínský. Rabinizer 3: Safraless translation of LTL to small deterministic automata. In *ATVA'14*, vol. 8837 of *LNCS*, pp. 235–241. Springer, 2014.
13. J. Křetínský and J. Esparza. Deterministic automata for the (F,G)-fragment of LTL. In *CAV'12*, vol. 7358 of *LNCS*, pp. 7–22. Springer, 2012.
14. S. C. Krishnan, A. Puri, and R. K. Brayton. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *ISAAC'94*, vol. 834 of *LNCS*, pp. 378–386. Springer, 1994.
15. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV'11*, vol. 6806 of *LNCS*, pp. 585–591. Springer, 2011.
16. C. Löding. Optimal bounds for transformations of omega-automata. In *FSTTCS'99*, vol. 1738 of *LNCS*, pp. 97–109. Springer, 1999.
17. M. Rönkkö. LBT: LTL to Büchi conversion. `http://www.tcs.hut.fi/Software/maria/tools/lbt/`, 1999. Implements [7].
18. H. Tauriainen. *Automata and Linear Temporal Logic: Translation with Transition-based Acceptance*. PhD thesis, Helsinki University of Technology, Espoo, Finland, Sept. 2006.
19. H. Tauriainen and K. Heljanko. Testing LTL formula translation into Büchi automata. *International Journal on Software Tools for Technology Transfer*, 4(1): 57–70, 2002.
20. M.-H. Tsai, Y.-K. Tsay, and Y.-S. Hwang. Goal for games, omega-automata, and logics. In *CAV'13*, vol. 8044 of *LNCS*, pp. 883–889. Springer, 2013.

21. M. Y. Vardi. Automata-theoretic model checking revisited. In *VMCAI'07*, vol. 4349 of *LNCS*. Springer, 2007.