ALASKA

Antichains for Logic, Automata and Symbolic Kripke structures Analysis[⋆]

M. De Wulf[1], L. Doyen[2], N. Maquet[1⋆⋆] and J.-F. Raskin[1]

[1] Université Libre de Bruxelles (ULB), Belgium
[2] École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

1 Introduction

ALASKA is a verification tool that implements new algorithms based on antichains [5, 7, 6] to efficiently solve the emptiness problem for both alternating finite automata (AFW) and alternating Büchi automata (ABW). Using the well-known translation from LTL to alternating automata, the tool can decide the satisfiability and validity problems for LTL over finite or infinite words. Moreover, ALASKA can solve the model-checking problem for ABW, LTL, AFW and finite-word LTL over symbolic (BDD-encoded) Kripke structures.

While several tools (notably NUSMV [2], and SPIN [17]) have addressed the satisfiability and model-checking problems for LTL [16], ALASKA uses new algorithms that are often more efficient, especially when LTL formulas are large. Moreover, to the best of our knowledge, ALASKA is the first publicly available tool to provide a direct interface to efficient algorithms to decide the emptiness of ABW and AFW.

Given the promising experimental results obtained recently [6], we have decided to polish our prototype and make it available to the research community. Our goal with ALASKA is not to compete with industrial-level tools such as SPIN or NUSMV but rather provide an open and clearly-documented library of antichain-based verification algorithms.

2 Classical and new algorithms

A *linear-time specification* over a set of propositions P is a set of infinite words over $\Sigma = 2^P$. Linear-time specifications can be expressed using LTL formulas or ABW. An LTL formula φ over P defines the set of words $[\![\varphi]\!] = \{w \in \Sigma^\omega \mid w \models \varphi\}$ that satisfy φ. The *satisfiability problem* for LTL asks, given an LTL formula φ, if $[\![\varphi]\!]$ is empty. The *model-checking problem* for LTL asks, given an effective representation of an omega-regular language $\mathcal{L} \subseteq \Sigma^\omega$ (*e.g.*, the set of all computations of a reactive system) and a LTL formula φ, if $\mathcal{L} \subseteq [\![\varphi]\!]$. The language $\mathsf{L}_\mathsf{b}(A)$ of an ABW A is the set of words over which it has an accepting run [15]. The *emptiness problem* for ABW asks, given an

[⋆] This research was supported by the Belgian FNRS grant 2.4530.02 of the FRFC project "Centre Fédéré en Vérification" and by the project "MoVES", an Interuniversity Attraction Poles Programme of the Belgian Federal Government.

[⋆⋆] This author is supported by an FNRS-FRIA grant.

ABW A, if $\mathsf{L_b}(A) = \varnothing$. The *model-checking problem* for ABW asks, given an omega-regular language $\mathcal{L}$ and an ABW A, if $\mathcal{L} \subseteq \mathsf{L_b}(A)$. Note that since ABW are closed under complementation and intersection in polynomial time, the model-checking problem $\mathcal{L} \subseteq \mathsf{L_b}(A)$ reduces in polynomial time to the emptiness problem $\mathcal{L} \cap \overline{\mathsf{L_b}(A)} = \varnothing$. All these problems are PSPACE-COMPLETE.

Due to lack of space, the following focuses mainly on the LTL satisfiability and ABW emptiness problems. Extensions to model-checking and to the finite-word case are rather straightforward.

Classical approaches The link between LTL and omega-regular languages is at the heart of the so-called *automata-theoretic approach* to LTL [22]. Given an LTL formula φ, one constructs a nondeterministic Büchi automaton (NBW) A_φ whose language corresponds exactly to the models of φ, *i.e.* $\mathsf{L_b}(A_\varphi) = [\![\varphi]\!]$. This reduces the satisfiability and model-checking problems for LTL to automata-theoretic questions. This elegant framework has triggered a large body of works (*e.g.* [21, 3, 20, 4, 13, 19, 10, 9, 11, 18, 1, 12, **?**]) that have been implemented in explicit-state model-checking tools such as SPIN [17] and in symbolic-state model-checking tools such as SMV and NUSMV [2]. The translation from LTL to NBW is central to the automata-theoretic approach to model-checking. This construction is however *worst-case exponential*. An explicit translation is required for explicit state model-checking, while in the symbolic approaches [3] the NBW is symbolically encoded using boolean constraints. In [16], Rozier and Vardi have extensively compared several symbolic and explicit tools for satisfiability checking of LTL. According to their experiments, the symbolic approach scales better.

The classical approach to solve ABW emptiness (and therefore LTL satisfiability) is to transform the ABW into an equivalent NBW. The first construction is due to Miyano and Hayashi [14], and many other constructions or variants have been proposed [4, 10, 9, 1]. Again, these constructions can be implemented either explicitly or symbolically.

The antichain approach Given an LTL formula, ALASKA constructs an ABW over the symbolic alphabet $\Sigma = 2^P$ that recognizes the models of the formula. This translation is very fast, as the number of states of the ABW is linear in the size of the formula. This construction is well-known and is an intermediate step in several translators from LTL to explicit NBW [20].

Once the ABW has been constructed, our tool *implicitly* uses the Miyano-Hayashi construction (MH for short) to obtain an equivalent NBW (which is not explicitly computed). This NBW is then explored efficiently in an on-the-fly fashion. ALASKA exploits a *simulation relation* to prune the search towards the most promising states (*i.e.*, minimal for the simulation relation) during the exploration. The crucial point is the that this simulation relation exists by construction for all NBW defined by the Miyano Hayashi construction, and does not need to be computed.

The tools which use explicit translation from LTL to NBW typically spend much effort in minimizing the constructed NBW. The rationale of this approach is that while the size of the NBW is worst-case exponential, it should often be possible to minimize it sufficiently in practice. In contrast, ALASKA systematically explores an NBW which is of exponential size *in all cases* (MH), but does the exploration efficiently by exploiting the special structure of the MH state-space (the simulation relation).

To compute the emptiness of the MH NBW, ALASKA begins by computing the set of reachable accepting states $\mathcal{R}_\alpha \equiv \mathsf{Post}^*(\iota^{\mathsf{MH}}) \cap \alpha^{\mathsf{MH}}$, where ι^{MH} and α^{MH} are respectively the initial and accepting states of MH. It then computes the following fixpoint formula[3]: $\mathcal{F} \equiv \nu X \cdot \mathsf{Post}^*(\mathsf{Post}(X) \cap \mathcal{R}_\alpha)$. Analogously to the Emerson-Lei backward fixpoint formula [8], $\mathcal{F}$ contains exactly those states that are reachable from an accepting state which (1) is reachable from the initial states, and (2) can reach itself by a non-trivial loop. The set $\mathcal{F}$ is thus empty if and only if the NBW is empty.

The computation of the fixpoint $\mathcal{F}$ is done efficiently by ALASKA as follows. The simulation relation that exists by construction on MH is such that ι^{MH} and α^{MH} are both *upward closed sets* for this relation. Also, the Post operation preserves closedness[4] (and so do $\cup$ and $\cap$), which means that *all* the sets of states that appear in the computation of $\mathcal{F}$ are closed sets. ALASKA achieves its performance because the Post operation of a set of states that is closed for a simulation relation is *easier* than for an arbitrary set of states. Indeed, upward closed sets can be canonically represented by a (generally small) number of *minimal* states that are incomparable for the simulation relation (which we call an *antichain*), and all operations can be done on those elements only. ALASKA exploits the fact that antichains are often small in practice by computing the Post operation in the following *semi-symbolic* manner. Given a set of states X symbolically encoded using a BDD, ALASKA computes $\mathsf{Post}(X)$ by first enumerating the antichain elements of X (which we note $\lfloor X \rfloor$) and computing the set $X' = \bigcup_{s \in \lfloor X \rfloor} \mathsf{Post}(\{s\})$. By the simulation relation, we know that $X' = \mathsf{Post}(X)$. Because the input and output of this algorithm are symbolic (X and X' are BDD) but an explicit representation is used internally ($\lfloor X \rfloor$ is an explicit list of states), we call this algorithm semi-symbolic.

Interested readers will find all the details of the algorithms and proofs in [6], along with experimental results comparing the relative performance of an early version of ALASKA and NUSMV for LTL satisfiability and model-checking. More information is available at `http://www.antichains.be`.

3 Implementation

Programming Language

ALASKA is written in Python, except for the BDD package which is written in C. We use the CUDD BDD library, with its PYCUDD Python binding. There is some performance overhead in using Python, but we chose it for enhanced readability and to make the code easy to change. We believe this is especially important in the context of academic research, as we expect other research teams to experiment with the tool, tweak the existing algorithms and add their own.

User Interface

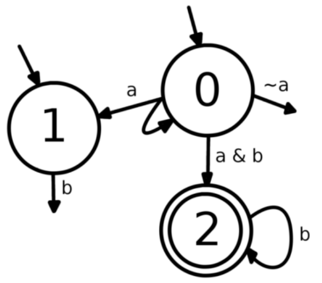
ALASKA is made of two components: a *library* (`alaskalib`) and an *executable script* (`alaska`). The executable script is a simple command-line interface (See Fig. 1) to the

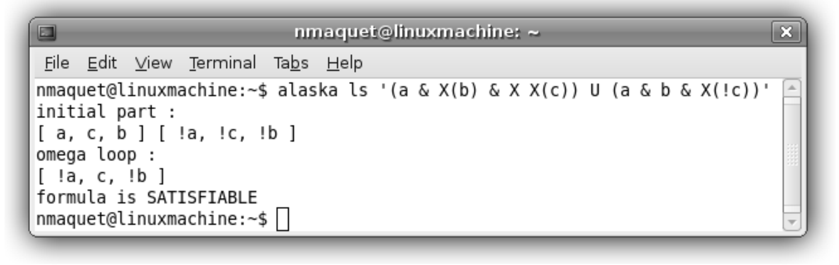
[3] This section details the *forward* algorithm; a *backward* algorithm is also implemented.

[4] There are some details involved, see [6].

algorithms provided with the library. The user interface currently provides access to the following features: finite and infinite-word LTL satisfiability, validity and equivalence checking, AFW and ABW emptiness, and model-checking of specifications expressed with finite or infinite-word LTL, AFW or ABW. Human-readable counter-example generation is available for all the aforementioned features. ALASKA can parse LTL formulas in the SPIN or NUSMV syntax and has a custom syntax for alternating automata (see Fig. 1 for an example). ALASKA uses the NUSMV input syntax for symbolic Kripke structures.

```
automaton_type = "sABW"
locations = [0,1,2]
alphabet = ["a", "b"]
init_constraint = "0 | 1"
accept_condition = [2]
transition_function = {
  0 : "(a & 0 & 1) | (~a) | (a & b & 2)",
  1 : "b", 2 : "b & 2"
}
```





```
nmaquet@linuxmachine:~$ alaska ls '(a & X(b) & X X(c)) U (a & b & X(!c))'
initial part :
[ a, c, b ] [ !a, !c, !b ]
omega loop :
[ !a, c, !b ]
formula is SATISFIABLE
nmaquet@linuxmachine:~$
```

Fig. 1. On the top: example of an ABW encoded in the ALASKA syntax. On the bottom: example of a command-line invocation of ALASKA for LTL satisfiability with counter-example generation.

Library Architecture

As a research tool, we believe that the most important contribution of the ALASKA project is the availability of its source code. As such, we give an overview of its core library components. The ALASKA library is divided into *data* packages[5], *state-space*

data packages	**state-space packages**	**solver packages**
automata	afasubset	afaemptiness
bdd	miyanohayashi	abwemptiness
boolean	kripkemiyanohayashi	ltlsatisfiability
ltl		ltlmodelchecking
nusmv		

Fig. 2. Package structure of ALASKA.

packages and *solver* packages. The data packages contain the data-structures with the associated parsers, pretty-printers and translation modules (e.g., LTL to ABW). The state-space packages provide intuitive abstractions of on-the-fly-explorable *implicit* state-spaces. Finally, the solver packages contain the high-level fixpoint algorithms. Each problem (ABW emptiness, AFA emptiness, LTL satisfiability, etc.) resides in its own module which provides several algorithmic variants (backward, forward, hybrid, etc.). Each solver uses a state-space package to evaluate a fixpoint formula and return the answer, along with a witness or counter-example if appropriate.

The original aspects of ALASKA reside in the state-space packages. They implement the antichain-based techniques which make ALASKA different from existing tools. There are currently three available state-space packages: `afasubset` represents the NFA state-space obtained from an AFA by a powerset construction, `miyanohayashi` represents the NBW state-space obtained from an ABW by the Miyano-Hayashi construction, and `kripkemiyanohayashi` represents the product state-space of a symbolic Kripke structure and a Miyano-Hayashi NBW. Each state-space package provides

package name	**input structure**	**explorable state-space**
`afasubset`	AFA	NFA
`miyanohayashi`	ABW	NBW
`kripkemiyanohayashi`	Kripke, ABW	Kripke $\otimes$ NBW

Fig. 3. Available sate-space packages.

functions for converting between the BDD-encoding of sets of states and the antichain encoding, computing upward/downward closures, converting sets of states and traces to human-readable output, etc. They also each implement the Pre and Post operations in both fully-symbolic (using only BDD) and semi-symbolic (with antichains) variants.

Possible Extensions

The ALASKA library can be used to implement various automata-based algorithms. One possibility of extension would be to mix backward with forward analysis into one algorithm. Also, as sometimes antichains do blowup in size, it might be interesting to have heuristics to detect such blowups in advance and proceed fully-symbolically in that case. For many such purposes, the ALASKA library could be a good starting point.

4 Tool download, examples, and benchmarks

ALASKA is available for download at `http://www.antichains.be`. The tool is available for Linux, Macintosh and Windows (by using Cygwin[6]). For convenience, the tool can also be tested through a web interface, for which a number of examples and benchmarks are provided.

[5] A Python *package* is a directory containing *.py files called *modules*.

[6] Cygwin is an open-source Linux-like environment for Windows. See http://www.cygwin.com

References

1. R. Bloem, A. Cimatti, I. Pill, M. Roveri, and S. Semprini. Symbolic implementation of alternating automata. In *Proceedings of CIAA*, LNCS 4094, pages 208–218. Springer, 2006.
2. A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: A new symbolic model checker. *STTT*, 2(4):410–425, 2000.
3. E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *CAV*, volume 818 of *LNCS*, pages 415–427. Springer, 1994.
4. M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *CAV*, LNCS 1633, pages 249–260. Springer, 1999.
5. M. De Wulf, L. Doyen, T.A. Henzinger, and J-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV*, LNCS 4144, pages 17–30. Springer, 2006.
6. M. De Wulf, L. Doyen, N. Maquet, and J.-F. Raskin. Antichains: Alternative algorithms for LTL satisfiability and model-checking. In *TACAS*, LNCS 4963, pages 63–77. Springer, 2008.
7. L. Doyen and J.-F. Raskin. Improved algorithms for the automata-based approach to model-checking. In *TACAS*, LNCS 4424, pages 451–465. Springer, 2007.
8. E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *LICS*, pages 267–278, 1986.
9. C. Fritz. Constructing Büchi automata from LTL using simulation relations for alternating Büchi automata. In *CIAA*, LNCS 2759, pages 35–48. Springer, 2003.
10. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV*, volume 2102 of *LNCS*, pages 53–65. Springer-Verlag, 2001.
11. M. Hammer, A. Knapp, and S. Merz. Truly on-the-fly LTL model checking. In *TACAS*, volume 3440 of *LNCS*, pages 191–205. Springer, 2005.
12. K. Heljanko, T. A. Junttila, M. Keinänen, M. Lange, and T. Latvala. Bounded model checking for weak alternating büchi automata. In *Proceedings of CAV*, LNCS 4144, pages 95–108. Springer, 2006.
13. C. Löding and W. Thomas. Alternating automata and logics over infinite words. In *IFIP TCS*, pages 521–535, 2000.
14. S. Miyano and T. Hayashi. Alternating finite automata on omega-words. In *CAAP*, pages 195–210, 1984.
15. D. Muller, A. Saoudi, and P. Schnupp. Alternating automata. the weak monadic theory of the tree, and its complexity. In *ICALP*, pages 275–283, 1986.
16. K Rozier and M. Y. Vardi. LTL satisfiability checking. In *14th Int'l SPIN Workshop*, volume 4595 of *LNCS*, pages 149–167. Springer, 2007.
17. T. Ruys and G. Holzmann. Advanced Spin tutorial. In *SPIN*, volume 2989 of *LNCS*, pages 304–305. Springer, 2004.
18. R. Sebastiani, S. Tonetta, and M. Y. Vardi. Symbolic systems, explicit properties: On hybrid approaches for LTL symbolic model checking. In *CAV*, LNCS 3576, pages 350–363, 2005.
19. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *CAV*, volume 1855 of *LNCS*, pages 248–263. Springer, 2000.
20. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *8th Banff Higher Order Workshop*, LNCS 1043, pages 238–266. Springer, 1995.
21. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *LICS*, pages 332–344. IEEE CS, 1986.
22. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.