BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2024
https://excel.fit.vutbr.cz

Mata: A Fast and Simple Finite Automata Library

David Chocholatý*

Abstract

Mata is a well-engineered, fast, and simple automata library in C++. It is maintainable and understandable. It has a simple architecture allowing a new user, a researcher, to quickly prototype new algorithms and thoroughly optimize the final implementation. Mata targets string constraint solving, reasoning about regular expressions, regular model checking, student projects, and research prototypes. It comes with a large benchmark from string constraint solving, regular model checking, and reasoning about regular expressions.

*xchoch08@stud.fit.vutbr.cz, Faculty of Information Technology, Brno University of Technology

1. Introduction

A new finite automata library Mata is intended to be used in applications where automata languages are manipulated by set operations and queries, presumably in a tight loop where automata are iteratively combined together using the classical as well as special-purpose constructions. Examples are applications like string constraint solving algorithms such as [1, 2, 3, 4, 5, 6, 7], processing of regular expressions [8, 9], regular model checking (e.g., [10, 11, 12, 13, 14, 15, 16]), or decision procedures for logics such as WS1S or quantified Presburger arithmetic [17, 18, 19, 20]. The solved problems are computationally hard. Efficiency is hence a primary concern. An automata library needs flexibility, extensibility, easy access to the low-level data structures, and ideally a low learning curve, which is important when involving students in academic research and utilizing limited resources of small research teams.

Fast and simple are therefore our two main requirements for the library. Mata is therefore built around a data structure for the transition relation of a non-deterministic automaton that is a compromise between simplicity and speed. It represents transitions explicitly, as triples of a sources state, a single symbol, and a target state. It allows to use a data structure specifically tailored for computing post-images of tuples and sets of states in automata algorithms: a source state-indexed array, storing at each index the transitions from that source state in a two layered structure, with the first layer divided and ordered by symbols, and the second layer ordered by target states.

Besides the C++ API, it provides a Python binding for fast prototyping and easy experimenting, for instance using interactive Jupyter notebooks.

That Mata is a good fit for string constraint solving is demonstrated by its central role in the string solver Z3-Noodler, which implements the algorithms of [1, 2], and outperforms the state of the art on many standard benchmarks (see [21, 22] for details).

Out contributions can be summarised as follows:

1. Mata, a fast, simple, and well-engineered automata library, well suited for application in string constraint solving and regex processing, in research and student projects, as well as in industrial applications.
2. An extension of a benchmark of automata problems from string constraint solving, processing regular expressions, regular model checking, and solving arithmetic constraints.
3. A comparison of a representative sample of well-known automata libraries against the above benchmark, demonstrating the superior performance of Mata.

2. Poster Commentary

Mata provides both C++ and Python interfaces for general-purpose finite automata operations as well as some operations specific to string solving domain.

2.1 Distinctive Features

The main distinctive features of Mata are:

- Fast and simple.
- Explicit representation of the transition relation.
- SOTA algorithms to work with nondeterminism.
- Modern development workflow and technologies.
- Easily extensible and modifiable.
- Well-documented, examples, testing infrastructure.
- High-level API with sane defaults, low-level API for maximal optimization.
- Python interface.
- A basis for a modular automata format .mata.

2.2 Supported Operations

Mata supports the following operations:

- Fine-grained modification of NFAs.
- Boolean language operations ($\cap, \cup, \bar{\cdot}$).
- Mintermization to handle large alphabets.
- Antichain-based language inclusion, equivalence, membership, emptiness.
- Determinization, minimization, simulation reduction.
- ϵ-transitions, ϵ-product, ϵ-removal.
- Rich visualization interface.
- Parsing of regexes (from RE2) and .mata format.

2.3 Figuress and Tables Commentary

Figure 1 An example of using the C++ interface for Mata. The code loads automata from a file in the .mata format with bitvectors on transitions, mintermizes them, constructs NFAs from the loaded intermediate representations over the alphabet {a, b, c}, trims and determinizes the NFAs, adds a new transition with a new final state. It then creates a second automaton accepting the word cbba, and optionally concatenates the initial NFA with itself and prints the result in the .mata format, shown in the right-hand side.

Figure 2 The main determinant of Mata is its three-layered data structure `Delta` for the transition relation: an ordered vector indexed by states. For each state, an ordered vector of transitions over symbols, for each symbol, an ordered vector of target states.

Figure 3 The usage of `Delta` in subset construction showing the advantages of `Delta`. `Delta` built for computing a post-image of a set of states. For a set of states S, compute $post(S)$, where you iterate trough all $post(s)$ for $s \in S$. Since these transitions are ordered, it is easy to iterate together. New macrostate transition always inserted at the end of the macrostate `Delta`.

Figure 4 An example of using Python interface for Mata. Mata provides an easy-to-use Python interface, as fast as C++ ($ pip install libmata).

The code loads automata from regular expressions, concatenates them, and displays the trimmed concatenation with conditional formatting.

2.4 Experimental Evaluation

We compared Mata [23] against Vata [24], Brics [25], Awali [26], Automata.net [27], AutomataLib [28], FAdo [29], and Automata.py [30], on a benchmark from string constraint solving, reasoning about regexes, regular model checking, and solving arithmetic formulae. Mata consistently outperforms all other libraries on all benchmarks in all operations. Mata is also the backbone of the efficiency of the SMT solver Z3-Noodler, [21, 22], which outperforms the state of the art on many standard benchmarks.

Cactus plots show cumulative run time. Time axes are logarithmic.
Tables show statistics for the benchmarks. We list the number of timeouts (TO, 60 s), average time on solved instances (Avg), median time over all instances (Med), and standard deviation over solved instances (Std). Best values are in bold, times are in milliseconds unless seconds are explicitly stated. ∼0 means a value close to zero.

3. Conclusions

Mata is not the most general or feature-full library. Other libraries are much more complex and comprehensive, and are more widely applicable. Mata, however, does what it is meant to do better than all the other libraries.

We continue working on Mata's set of features as well as its efficiency. We plan to extend Mata with transducers, add support for registers that could handle, e.g., counting in regular expressions. We believe that the efficiency of the basic data structures can be much improved by focusing on the low-level performance. Custom data structures, specialised memory management, improvement in memory locality, and, generally, the class of optimizations used in BDD packages, could shift Mata's performance much further.

References

[1] František Blahoudek, Yu-Fang Chen, David Chocholatý, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Juraj Síč. Word equations in synergy with regular constraints. In Proc. of FM'23. Springer, 2023.

[2] Yu-Fang Chen, David Chocholatý, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Juraj Síč. Solving string constraints with lengths by stabilization. Proc. ACM Program. Lang., 7(OOPSLA2), oct 2023.

[3] Taolue Chen, Yan Chen, Matthew Hague, Anthony W. Lin, and Zhilin Wu. What is decidable about string constraints with the replaceall function. Proc. of POPL'18, 2018.

[4] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukás Holík, Ahmed Rezine, and Philipp Rümmer. Trau: SMT solver for string constraints. In Proc. of FMCAD'18. IEEE, 2018.

[5] Murphy Berzish, Mitja Kulczynski, Federico Mora, Florin Manea, Joel D. Day, Dirk Nowotka, and Vijay Ganesh. An SMT solver for regular expressions and linear arithmetic over string length. In Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II, volume 12760 of Lecture Notes in Computer Science, pages 289–312. Springer, 2021.

[6] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukáš Holík, Ahmed Rezine, Philipp Rümmer, and Jari Stenman. Norn: An SMT solver for string constraints. In Computer Aided Verification, pages 462–469, Cham, 2015. Springer International Publishing.

[7] Caleb Stanford, Margus Veanes, and Nikolaj S. Bjørner. Symbolic boolean derivatives for efficiently solving extended regular expression constraints. In Proc. of PLDI'21. ACM, 2021.

[8] Arlen Cox and Jason Leasure. Model checking regular language constraints. CoRR, abs/1708.09073, 2017.

[9] Pieter Hooimeijer and Westley Weimer. A decision procedure for subset constraints over regular languages. In PLDI'09. ACM, 2009.

[10] Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In Rajeev Alur and Doron A. Peled, editors, Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings, volume 3114 of Lecture Notes in Computer Science, pages 372–386. Springer, 2004.

[11] Ahmed Bouajjani, Peter Habermehl, Lukáš Holík, Tayssir Touili, and Tomáš Vojnar. Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In Proc. of CIAA'08. Springer, 2008.

[12] Axel Legay. T(O)RMC: A tool for (ω)-regular model checking. In Aarti Gupta and Sharad Malik, editors, Computer Aided Verification, pages 548–551, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[13] Yu-Fang Chen, Chih-Duo Hong, Anthony W. Lin, and Philipp Rümmer. Learning to prove safety over parameterised concurrent systems. In Daryl Stewart and Georg Weissenbacher, editors, 2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017, pages 76–83. IEEE, 2017.

[14] Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large. In Warren A. Hunt and Fabio Somenzi, editors, Computer Aided Verification, pages 223–235, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[15] Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In Alan J. Hu and Moshe Y. Vardi, editors, Computer Aided Verification, pages 88–97, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[16] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena. A survey of regular model checking. In Philippa Gardner and Nobuko Yoshida, editors, CONCUR 2004 - Concurrency Theory, pages 35–48, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[17] J. Richakd Büchi. Weak Second-Order Arithmetic and Finite Automata, pages 398–424. Springer New York, New York, NY, 1990.

[18] Pierre Wolper and Bernard Boigelot. An automata-theoretic approach to Presburger arithmetic constraints (extended abstract). In Alan Mycroft, editor, Proc. of SAS'95, volume 983 of LNCS. Springer, 1995.

[19] Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm.

Mona: Monadic second-order logic in practice. In Proc. of TACAS '95, volume 1019 of LNCS. Springer, 1995.

[20] Bernard Boigelot and Louis Latour. Counting the solutions of Presburger equations without enumerating them. Theoretical Computer Science, 313(1):17–29, 2004. Implementation and Application of Automata.

[21] Yu-Fang Chen, David Chocholatý, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Juraj Síč. Z3-noodler: An automata-based string solver. In Proc. of TACAS'24, LNCS. Springer, 2024.

[22] Yu-Fang Chen, David Chocholatý, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Juraj Síč. Z3-noodler: An automata-based string solver. In Bernd Finkbeiner and Laura Kovács, editors, Tools and Algorithms for the Construction and Analysis of Systems, pages 24–33, Cham, 2024. Springer Nature Switzerland.

[23] David Chocholatý, Tomáš Fiedor, Vojtěch Havlena, Lukáš Holík, Martin Hruška, Ondřej Lengál, and Juraj Síč. Mata: A fast and simple finite automata library. In Bernd Finkbeiner and Laura Kovács, editors, Tools and Algorithms for the Construction and Analysis of Systems, pages 130–151, Cham, 2024. Springer Nature Switzerland.

[24] Ondrej Lengál, Jiří Šimáček, and Tomás Vojnar. VATA: A library for efficient manipulation of non-deterministic tree automata. In Proc. of TACAS'12, volume 7214 of LNCS. Springer, 2012.

[25] Anders Møller et al. Brics automata library.

[26] Sylvain Lombardy, Victor Marsault, and Jacques Sakarovitch. Awali, a library for weighted automata and transducers (version 2.0), 2021. Software available at http://vaucanson-project.org/Awali/2.0/.

[27] Margus Veanes. A .NET automata library.

[28] Malte Isberner, Falk Howar, and Bernhard Steffen. AutomataLib.

[29] André Almeida, Marco Almeida, José Alves, Nelma Moreira, and Rogério Reis. Fado and guitar: Tools for automata manipulation and visualization. In Sebastian Maneth, editor, Implementation and Application of Automata, pages 65–74, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[30] Caleb Evans. Automata, 2023.