The Mealy-machine reduction functions of Spot

Florian Renkin, Philipp Schlehuber-Caissier, Alexandre Duret-Lutz, Adrien Pommellet

The Mealy-Machine Reduction Functions of Spot

Florian Renkin, Philipp Schlehuber-Caissier, Alexandre Duret-Lutz, and Adrien Pommellet

EPITA's Research Laboratory, Le Kremlin-Bicêtre, France

Abstract

We present functions for reducing Mealy machines, initially detailed in our FORTE'22 article. These functions are now integrated into Spot 2.11.2, where they are used as part of the `ltlsynt` tool for reactive synthesis. Of course, since Spot is a library, these functions can also be used on their own, and we provide Python bindings for easy experiments. The reproducible capsule benchmarks these functions on Mealy machines from various sources, and compare them to the MeMin tool.

Keywords: Mealy machines, synthesis, SAT

Metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	Spot 2.11.2
C2	Permanent link to code/repository used for this code version	`https://gitlab.lre.epita.fr/spot/spot`
C3	Permanent link to Reproducible Capsule	`https://codeocean.com/capsule/4358262/tree/v1`
C4	Legal Code License	GPL v3+
C5	Code versioning system used	`git`
C6	Software code languages, tools, and services used	C++17, Python 3.5+
C7	Compilation requirements, operating environments and dependencies	Linux or MacOS with a C++17 compiler and Python 3.5 or later
C8	Link to developer documentation	`https://spot.lre.epita.fr`
C9	Support email for questions	`spot@lrde.epita.fr`

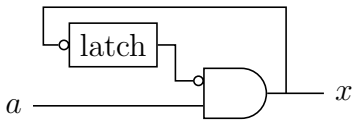

Figure 1: A switching circuit that reads one signal a, and outputs one signal x. The latch is initially 0 and delays the signal by one tick.

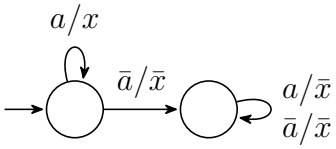

Figure 2: A Mealy machine representing the behavior of the circuit on the left.

1. Motivation and significance

Mealy machines were invented as a model of synchronous reactive circuits, also known as switching circuits. [1] For instance the simple circuit shown on Fig. 1 reads a signal a and outputs a signal x such that x is true until a is false, and x then remains false forever. The behavior of this circuit can be given by the Mealy machine on Fig. 2: this is a finite automaton where edges are labeled by input/output pairs, and that is deterministic with respect to the input. The states of the Mealy machine are used to represent the internal state of the circuit, i.e., the value of all latches. Of course switching circuits and Mealy machines can use multiple input signals, multiple output signals, and multiple latches.

The techniques we discuss here are used to reduce Mealy machines so that they use fewer states. Using fewer states usually means that the circuit generated from the Mealy machine will use fewer latches and gates. Minimizing Mealy machines in presence of *don't care* outputs or destinations has been studied for a long time [2]. The problem is known to be NP-complete [3] and several approaches have been proposed over the time, including: enumerating all possible solutions [2], reducing to other covering problems [4, 5], incrementally reducing machines one state at a time [6], incrementally searching equivalent machines by adding one state at a time and then using counter-examples to refine the machines [7, 8]. MeMin is a tool implementing the strategy of trying to build equivalent machines of increasing size, but using a SAT solver to check their feasibility; it was shown to outperform other approaches [9].

Our original motivation comes from the problem of *reactive LTL synthesis* [10, 11]: build a reactive circuit whose input and output signals are tied by a specification provided as an LTL formula. For instance the circuit of Figure 1 is a possible solution to the specification $a \leftrightarrow \mathsf{F}\, x$ which indicates that x should eventually be true if and only if a was true initially.

Because LTL specifications can be relatively lax, it is possible that for some given input signals and state of the circuit, multiple possible outputs are compatible with the LTL formula. For this reason, our synthesis pipeline

produces a version of Mealy machines called "Incompletely-specified Generalized Mealy Machines" (IGMMs [12]). The generalization is that the output part of an edge can be any arbitrary Boolean function to indicate that the machine is free to output any set of signals that satisfies this function (this is finer than traditional models using *don't care* outputs as it can express some constraints). Reducing such IGMMs can then take advantage of the fact that ultimately, the circuit needs only to settle on a *specialized* version of the machine with a unique set of output signals for each pair of state and input signals.

We have implemented two different reductions procedures. The first one is a heuristic that finds sets of states of the Mealy machine that could be merged if their output would be reduced to a compatible subset. This search is achieved by computing and comparing *signatures* for each state, in a way inspired by how Babiak et al. [13] check for trace inclusion; the reduction is then performed by building a *specialization graph* that will help select one representative for each state.

The second one is a SAT-based minimization procedure that solves the NP-hard problem of finding the minimal Mealy machine that specializes the original one. [12] This second procedure is inspired by an existing tool called MeMin [9]; but MeMin's model of generalized Mealy machines only supports output functions that are cubes (i.e., conjunction of literals), not arbitrary functions. While there are cases in the synthesis competitions where we can really benefit from not being restricted to cubes, the benchmarks we have performed are restricted to cubic outputs, for fairness with MeMin.

2. Software description

Our reduction functions are implemented in Spot [14], a C++ library for LTL formula and ω-automata manipulation. It additionally comes with a set of command-line tools (such as the LTL synthesis tool `ltlsynt`), as well as Python bindings for interactive use, prototyping, and testing.

2.1. Software functionalities

The functions implementing the reductions discussed in our FORTE'22 [12] paper and in greater details in F. Renkin's Ph.D thesis [15]. They are called `reduce_mealy(`*aut*`,`*oa*`)` and `minimize_mealy(`*aut*`)`. The former implements the heuristic-based reduction, while the latter performs SAT-based minimization. When *oa* is false, the `reduce_mealy` simply merges states that are bisimilar (i.e., states that behave identically). When *oa* (output assignment) is true, it merges states that can become bisimilar once restricted to a common set of output signals.

They can both be used as part of our pipeline for LTL synthesis (for instance when running the tool `ltlsynt`, passing option `--simplify=bisim`, `--simplify=bwoa` will cause `reduce_mealy` to be used with *oa* to set to false or true, passing option `--simplify=sat` will cause `minimize_mealy` to be used, and passing option `--simplify=bwoa-sat` will first reduce the Mealy machine before minimizing it).

Additionally, these two functions can be called directly and interactively using the Python bindings. We demonstrate this in the artifact.

2.2. Software architecture

Spot uses a class called `twa_graph` to store an ω-automaton whose structure is stored as a graph. These ω-automata allow the representation of set of infinite words labeled by valuations of Boolean propositions. Because one often want to restrict the sets of infinite runs that are accepted, these automata are equipped with an *acceptance condition* which are Boolean formulas telling which transitions of the automaton may be visited infinitely often or finitely often.

The ω-automaton class in Spot is very flexible and can be extended by attaching *named properties* to it (this is similar to the attribute system of the R programming language). For instance any automaton that declares the `synthesis-output` property (a list of propositions that represent output signals) and whose acceptance condition is *true* (all infinite runs are accepted) can be handled like a Mealy machine. This in turn allows the code for displaying automata to be specialized for this case and separate input and output signals for display.

The two functions `reduce_mealy` and `minimize_mealy` take a `twa_graph` as argument, and then check that this automaton actually represents a Mealy machine before attempting to reduce it.

To find compatible states that can be fused together, the `reduce_mealy` function uses *Binary Decision Diagrams* (BDDs) to encode a signature for each state. The BDD library used by Spot is BuDDy [16].

On the other hand the `minimize_mealy` functions encodes the minimization as a SAT problem, and solves this problem with PicoSAT [17], a SAT-solver chosen for its ease of distribution.

3. Illustrative examples

Our artifact contains a Python notebook demonstrating how to use the above two functions on Mealy machines from our benchmark. Additionally, the notebook shows how to call MeMin on similar machines for comparison.

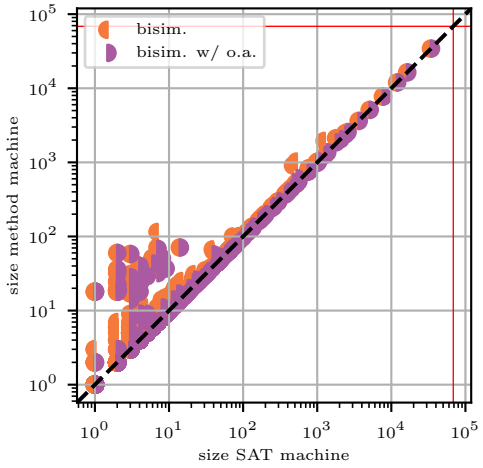

Figure 3: Sizes of the different simulation-based methods, compared to the SAT-based output.

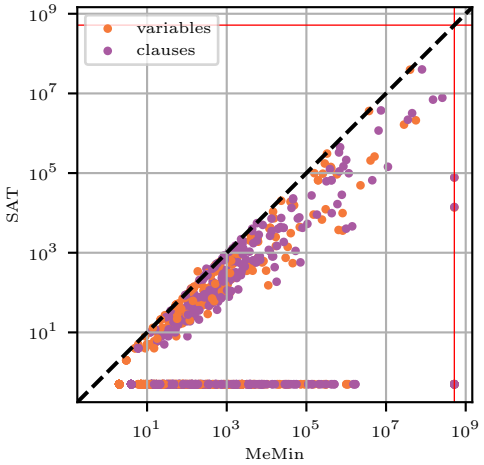

Figure 4: Comparison of the number of variables and clauses in our encoding and in MeMin's

4. Impact

As mentioned earlier, we use these functions in our `ltlsynt` tool for LTL synthesis. Our FORTE'22 paper [12] has shown:

- That our SAT-based encoding generally uses fewer clauses and variables compared to MeMin (Fig. 4).
- That our BDD-based reductions (`reduce_mealy`) are generally much faster than our SAT-based minimization (`minimize_mealy`) (Fig. 5–6).
- That our BDD-based reduction (`reduce_mealy`) with output assignment (bwoa) often produce results that are close to optimal (Fig. 3), and is therefore a good compromise between speed and quality.

Note that Figure 5 differs from its counterpart in our FORTE'22 paper [12] because the latter is based on a development branch of Spot that uses a memory representation of Mealy labels that differs significantly from existing releases, and that has not been merged yet. We have decided to stick to a public release for this original software publication.

5. Future Work

We are currently investigating methods to further reduce the number of variables and clauses in the SAT-based minimization as well as speeding up

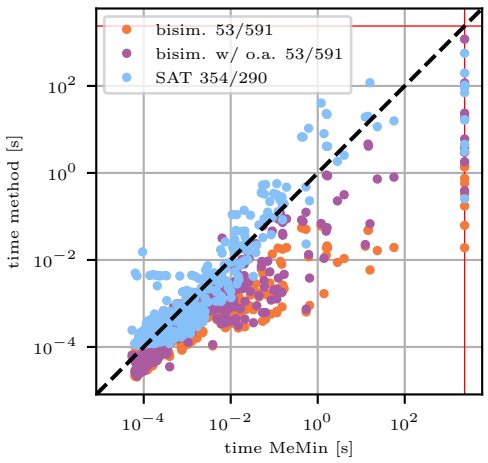

Figure 5: Runtime of our methods compared to the runtime of MeMin.

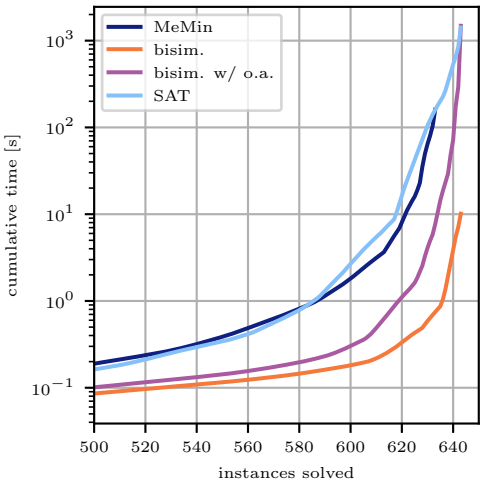

Figure 6: Cactus-plot of the number of cases solved by each methods, with cases sorted for each method by increasing runtime.

the computation by introducing new variables, a technique applicable to both approaches.

References

In addition to the GitLab repository mentioned on first page, each release of Spot is archived at SoftwareHeritage. A copy of the source code is therefore permanently available from `https://archive.softwareheritage.org/swh:1:rel:85d32fb5235cc4606676fe9e4b5f839c21fdbf8f`

References

[1] G. H. Mealy, A method for synthesizing sequential circuits, The Bell System Technical Journal 34 (5) (1955) 1045–1079. doi:10.1002/j.1538-7305.1955.tb03788.x.

[2] M. C. Paull, S. H. Unger, Minimizing the number of states in incompletely specified sequential switching functions, IRE Transactions on Electronic Computers EC-8 (3) (1959) 356–367. doi:10.1109/TEC.1959.5222697.

[3] C. P. Pfleeger, State reduction in incompletely specified finite-state machines, IEEE Transactions on Computers C-22 (12) (1973) 1099–1102. doi:10.1016/j.compeleceng.2006.06.001.

[4] G. Hachtel, J.-K. Rho, F. Somenzi, R. Jacoby, Exact and heuristic algorithms for the minimization of incompletely specified state machines, in: Proceedings of the European Conference on Design Automation, 1991, pp. 184–191. doi:10.1109/EDAC.1991.206387.

[5] T. Kam, T. Villa, R. Brayton, A. Sangiovanni-Vincentelli, A fully implicit algorithm for exact state minimization, in: Proceedings of the 31st Annual Design Automation Conference (DAC'94), Association for Computing Machinery, New York, NY, USA, 1994, p. 684–690. doi:10.1145/196244.196615.
URL https://doi.org/10.1145/196244.196615

[6] A. Alberto, A. Simao, Iterative minimization of partial finite state machines, Central European Journal of Computer Science 3 (2) (2013) 91–103. doi:10.2478/s13537-013-0106-0.

[7] J. Pena, A. Oliveira, A new algorithm for exact reduction of incompletely specified finite state machines, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18 (11) (1999) 1619–1632. doi:10.1109/43.806807.

[8] S. Gören, F. J. Ferguson, On state reduction of incompletely specified finite state machines, Journal of Computers and Electrical Engineering 33 (2007) 58–69. doi:10.1016/j.compeleceng.2006.06.001.

[9] A. Abel, J. Reineke, MeMin: SAT-based exact minimization of incompletely specified Mealy machines, in: Proceedings for the 34th International Conference on Computer-Aided Design (ICCAD'15), IEEE Press, 2015, pp. 94–101. doi:10.1109/ICCAD.2015.7372555.

[10] S. Jacobs, R. Bloem, M. Colange, P. Faymonville, B. Finkbeiner, A. Khalimov, F. Klein, M. Luttenberger, P. J. Meyer, T. Michaud, M. Sakr, S. Sickert, L. Tentrup, A. Walker, The 5th reactive synthesis competition (SYNTCOMP 2018): Benchmarks, participants & results, CoRR abs/1904.07736 (2019).

[11] T. Michaud, M. Colange, Reactive synthesis from LTL specification with Spot, in: Proceedings of the 7th Workshop on Synthesis (SYNT'18), 2018.
URL http://www.lrde.epita.fr/dload/papers/michaud.18.synt.pdf

[12] F. Renkin, P. Schlehuber-Caissier, A. Duret-Lutz, A. Pommellet, Effective reductions of Mealy machines, in: Proceedings of the 42nd International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE'22), Vol. 13273 of Lecture Notes in Computer Science, Springer, 2022, pp. 114–130.

[13] T. Babiak, T. Badie, A. Duret-Lutz, M. Křetínský, J. Strejček, Compositional approach to suspension and other improvements to LTL translation, in: Proceedings of the 20th International SPIN Symposium on Model Checking of Software (SPIN'13), Vol. 7976 of Lecture Notes in Computer Science, Springer, 2013, pp. 81–98. doi:10.1007/978-3-642-39176-7_6.

[14] A. Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. G. Aisse, P. Schlehuber-Caissier, T. Medioni, A. Martin, J. Dubois, C. Gillard, H. Lauko, From Spot 2.0 to Spot 2.10: What's new?, in: Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22), Vol. 13372 of Lecture Notes in Computer Science, Springer, 2022, pp. 174–187. doi:10.1007/978-3-031-13188-2_9.

[15] F. Renkin, Transformations d'ω-automates pour la synthèse de contrôleurs réactifs, Ph.D. thesis, Sorbonne University, Paris, France (Oct. 2022).

[16] J. Lind-Nielsen, BuDDy: Binary Decision Diagram package, Release 2.2 (Nov. 2002).
URL http://www.itu.dk/research/buddy/

[17] A. Biere, PicoSAT essentials., Journal on Satisfiability, Boolean Modeling and Computation 4 (2008) 75–97. doi:10.3233/SAT190039.