BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2023
http://excel.fit.vutbr.cz

Efficient Reduction of Finite Automata

Veronika Molnárová

Abstract
A finite state automaton is a mathematical model generally utilized in information technology. While each automaton denotes its language, one language can be represented by an infinite number of different automata. To ensure efficient work with them we want to find the smallest alternative. Reduction is a process used to get an automaton with the same language but fewer states.
In our work, we implemented and optimized three known reduction algorithms, which are the minimization of deterministic automata, the reduction based on a relation of simulation, and the reduction by transformation into a canonical residual automaton. These reductions were tested on a sample set of automata to compare their results.
Moreover, we looked at the possibility of reducing finite state automata using Boolean satisfiability problem (SAT) and quantified Boolean formula (QBF) solvers. We are presenting a set of rules for each solver for generating a clause in conjunctive normal form (CNF), which can precisely represent the given automaton in Boolean algebra. We used this fact to create a new method of nondeterministic automata reduction.

*xmolna08@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The concept of finite state automata (FSAs) was first introduced in 1943 by two neurophysiologists. Since then, this concept was further and further developed to get the basis for the finite state automata we know today. [1]

Reduction of the automata became an everyday part of their usage to guarantee that the work with them will be efficient. Generally, we can speak about two types of automata reductions, which are the reduction of deterministic automata(DFAs) and the reduction of nondeterministic automata(NFAs).

DFAs are restricted to a single transition for each symbol from a state, which can cause their size to be exponential compared to the size of NFAs representing the same language. As we can find the minimal DFA through multiple algorithms, there is not much improvement to be done.

The same algorithms cannot be applied to NFAs without determinizing them first. As determinization would cause the expansion of the automaton, it would be beneficial to reduce automata directly from their nondeterministic version. Such reduction is computationally hard [2] and there are still researched new ways of nondeterministic reduction. We looked at two nondeterministic reductions, in particular, the reduction using the relation of simulation [3] and the reduction through the residual automata [4].

The article [5] presented a new way of looking at FSAs by utilizing the SAT solvers. The main idea is to represent the problem as a clause in CNF, which is then solvable using the solver. Inspired by this idea, we represented the structure of an FSA as a clause and then expanded this idea also onto NFAs using QBF solvers. Altogether, we created an algorithm to use these solvers to find a smaller language-equivalent automaton.

2. Known reduction algorithms

We implemented three variants of automata reduction, which are the minimization of deterministic automata, the reduction based on a relation of simulation, and the reduction by transformation into a canonical residual automaton.

The **minimization of deterministic automata** finds the smallest equivalent DFA to the given one. For this reduction, we examined Hopcroft's and Brzozowski's algorithms.

The **reduction based on a relation of simulation** and the **reduction by transformation into a canonical residual automaton** reduce NFAs directly. Figures 1 and 2 show the results of testing these reductions on a sample set of 3900 abstract regular model-checking automata.

We can observe that residual automata's performance yields the best results in terms of size and runtime. We can notice similarities between the times required by Brzozowski's and residual automata as both algorithms use backward determinization. There are some instances that deviate from the norm since this process can produce automata that are exponentially larger.

3. New reduction utilizing SAT and QBF solvers

SAT and QBF solvers are powerful tools that were optimized to efficiently solve the given problem in form of a clause in CNF. We search for a way to define an automaton as a formula in CNF and utilize these solvers to reduce them.

Three different types of variables:

- transition variables – T_{1a1}, $T_{1b2}, \dots$,
- initial variables – I_1, $I_2, \dots$, and,
- final variables – F_1, $F_2, \dots$,

are used to represent the structure of the automaton each indicating whether the specified transition, initial, or final state exists in the automaton.

Clause for SAT solvers

Firstly we created a set of four rules using which we can define the given DFA in Boolean algebra as a formula in CNF.

Determinism and Completeness

We must fulfill the requirement of having a maximum of one transition for a symbol from each state to create a DFA. We generate each possible pair of transitions from a state and declare that the automaton can only contain one from them. In Figure 3, these clauses are indicated by the yellow background.

If we require the automaton to be complete, we must include clauses forcing the automaton to have at least one of the transitions for a symbol from each state (clauses are indicated by the orange background).

Accepting words

We must provide instances of the words that the automaton accepts and rejects to define the automaton's language. With DFA, we can say that the initial state is always the state 1. Then, we construct every potential path from the state 1 over the input word in the automaton, declare that the ending state must be a final state, and say that at least one of these paths must be present. These clauses are in the DNF because just one of them is necessary for the word to be accepted, and a transformation to CNF is required. In Figure 3, clauses for accepting the word *ab* are indicated by the blue background.

Rejecting words

By negating clauses for accepting words, we can define their rejection. If a path doesn't start in state 1 or end in a final state, or if there are no such transitions over the symbols, we declare that the word is rejected. For example, clauses rejecting the word *ab* are indicated by green color.

Clause for QBF solvers

We utilized quantified variables Q_n for representing states of the automaton instead of creating an exponential number of paths in the automaton. The quantified variables define a binary vector corresponding to the index of the state in the automaton.

Accepting words

As for SAT solvers, we generate clauses for accepting the input word by setting the initial quantified state to the initial variables, the ending state to the final variables, and each pair of states to the transitions through the expected symbol. In Figure 4 we can see these clauses on yellow and blue backgrounds.

Rejecting words

Once again we defined the rejection as a negation of accepting clauses. After the negation, these clauses will be once again in DFA as the rejection must apply to every viable path in the automaton.

Invalid combinations coverage

With a binary vector, we may be generating more combinations for indexes than the actual number of states of the automaton. These combinations must be either accepted ($\forall$) or rejected ($\exists$) depending on the quantifier of the given variables.

Acknowledgements

I would like to thank my supervisor Mr. Ing. Ondřej Lengál, Ph.D for his guidance, advice and patience throughout the whole year.)

References

[1] Amal Dar Aziz, Joe Cackler, and Raylene Yung. Automata theory. online, 2004.

[2] Andreas Malcher. Minimizing finite automata is computationally hard. *Theor. Comput. Sci.*, 327(3):375–390, 2004.

[3] Lucian Ilie, Gonzalo Navarro, and Sheng Yu. *On NFA Reductions*, volume 3113 of *Lecture Notes in Computer Science*, pages 112–124. Springer, Berlin, Heidelberg, 2004.

[4] François Denis, Aurélien Lemay, and Alain Terlutte. Residual finite state automata. In Afonso Ferreira and Horst Reichel, editors, *STACS 2001, 18th Annual Symposium on Theoretical Aspects of Computer Science, Dresden, Germany, February 15-17, 2001, Proceedings*, volume 2010 of *Lecture Notes in Computer Science*, pages 144–157, Berlin, Heidelberg, 2001. Springer.

[5] Anthony W. Lin and Philipp Rümmer. Liveness of randomised parameterised systems under arbitrary schedulers. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, volume 9780 of *Lecture Notes in Computer Science*, pages 112–133. Springer, 2016.