EFFICIENT ALGORITHMS FOR FINITE AUTOMATA

Martin Hruška
Bachelor Degree Programme (3), FIT BUT
E-mail: xhrusk16@stud.fit.vutbr.cz

Supervised by: Ondřej Lengál
E-mail: ilengal@fit.vutbr.cz

Abstract: Nondeterministic finite automata (NFA) are used in many areas of computer science, including, but not limited to, formal verification, or the design of digital circuits. Their advantages over deterministic finite automata (DFA) is that they may be exponentially conciser. However, this advantage may be lost if a naïve approach to some operations, in particular checking language inclusion, which performs explicit determinization of the NFA, is taken. Recently, several new techniques for this problem that avoid explicit determinization (using the so-called antichains or bisimulation up to congruence) have been proposed. The main goal of this paper is to describe these techniques which are being implemented as the new extension for the VATA library.

Keywords: finite automata, formal verification, language inclusion, bisimulation up to congruence, antichains, simulation

1 INTRODUCTION

A finite automaton (FA) is a computation model with applications in different branches of computer science, e.g., compiler design, formal verification, the design of digital circuits or natural language processing. In formal verification alone are its uses abundant, e.g., in model checking of safety temporal properties, abstract regular model checking, static analysis, or decision procedures of some logics, such as Presburger arithmetic or weak monadic second-order theory of one successor (WS1S).

Many of the mentioned applications need to perform certain expensive operations on FA, such as checking universality of an FA (i.e., checking whether it accepts any word over a given alphabet), or checking language inclusion of a pair of FA (i.e., testing whether the language of one FA is a subset of the language of the second FA). The classical (so called *textbook*) approach is based on complementation of the language of an FA. Complementation is easy for *deterministic* FA (DFA)—just swapping accepting and non-accepting states—but a hard problem for *nondeterministic* FA (NFA), which need to be determinized first (this may lead to an exponential explosion in the number of the states of the automaton). Both mentioned operations over NFA are PSPACE-complete problems.

Recently, there has been a considerable advance in techniques for dealing with these problems. The new techniques are either based on the so-called *antichains* [1, 2] or the so-called *bisimulation up to congruence* [3]. In general, those techniques do not need an explicit construction of the complement automaton. They only construct a sub-automaton which is sufficient for either proving that the universality or inclusion hold, or finding a counterexample.

Unfortunately, currently there is no efficient implementation of a general NFA library that would use the state-of-the-art algorithms for the mentioned operations on automata. The closest implementation is VATA [4], a general library for nondeterministic finite *tree* automata, which can be used even for NFA (being modelled as unary tree automata) but with not the optimal performance given by its overhead that comes with the ability to handle much richer structures.

The goal of this work is two-fold: (i) extending VATA with an NFA module implementing basic

operations on NFA, such as union, intersection, or checking language inclusion, and (ii) an efficient design and implementation of checking language inclusion of NFA using bisimulation up to congruence (which is missing in VATA for tree automata).

2 PRELIMINARIES

We call a finite set of symbols Σ an *alphabet*. A *word* w over Σ of *length* n is a finite sequence of symbols $w = a_1 \dots a_n$, where $\forall 1 \leq i \leq n \,.\, a_i \in \Sigma$. A *nondeterministic finite automaton* (NFA) is a quintuple $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of states, Σ is an alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation (we use $p \xrightarrow{a} q$ to denote that $(p,a,q) \in \delta$), $I \subseteq Q$ is a set of initial states and $F \subseteq Q$ is a set of final states. A *deterministic finite automaton* (DFA) is a special case of an NFA, where δ is a partial function $\delta : Q \times \Sigma \to Q$ and $|I| \leq 1$. A *run* of an NFA $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ from a state q over a word $w = a_1 \dots a_n$ is a sequence $r = q_0 \dots q_n$, where $\forall 0 \leq i \leq n \;.\; q_i \in Q$ such that $q_0 = q$ and $(q_i, a_{i+1}, q_{i+1}) \in \delta$. The run r is called *accepting* iff $q_n \in F$. The *language* of state $q \in Q$ is defined as $L_{\mathcal{A}}(q) = \{w \in \Sigma^* \mid \textit{there exists an accepting run of } \mathcal{A} \textit{ from } q \textit{ over } w\}$, while the language of a set of states $R \subseteq Q$ is defined as $L_{\mathcal{A}}(R) = \bigcup_{q \in R} L_{\mathcal{A}}(q)$. The language of an NFA $\mathcal{A}$ is defined as $L_{\mathcal{A}} = L_{\mathcal{A}}(I)$.

3 CHECKING LANGUAGE INCLUSION OF NFA

Given a pair of automata $\mathcal{A}$ and $\mathcal{B}$, the *textbook* algorithm for checking inclusion of their languages $L_{\mathcal{A}} \subseteq L_{\mathcal{B}}$ works by first determinizing $\mathcal{B}$ (yielding the DFA $\mathcal{B}_{det}$), complementing it ($\overline{\mathcal{B}_{det}}$) and constructing the NFA $\mathcal{A} \times \overline{\mathcal{B}_{det}}$ accepting the intersection of $L_{\mathcal{A}}$ and $L_{\overline{\mathcal{B}_{det}}}$ and checking whether its language is nonempty. Any accepting run in this automaton may serve as a witness that the inclusion between $\mathcal{A}$ and $\mathcal{B}$ does not hold. Some recently introduced approaches avoid the explicit construction of $\overline{\mathcal{B}_{det}}$ and the related state explosion in many practical cases.

3.1 ANTICHAINS

We define an antichain and simulation before describing the algorithm itself. Given a partially ordered set Y, an *antichain* is a set $X \subseteq Y$ such that all elements of X are incomparable. A forward *simulation* on an NFA $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ is a relation $\preceq \subseteq Q \times Q$ such that if $p \preceq r$ then (i) $p \in F \Rightarrow r \in F$ and (ii) for every transition $p \xrightarrow{a} p'$, there exists a transition $r \xrightarrow{a} r'$ such that $p' \preceq r'$. Note that simulation implies language inclusion, i.e., $p \preceq q \Rightarrow L_{\mathcal{A}}(p) \subseteq L_{\mathcal{A}}(q)$.

The antichains algorithm [1] starts searching for a final state of automaton $\mathcal{A} \times \overline{\mathcal{B}_{det}}$ while pruning out states which are not necessary to explore. $\mathcal{A}$ is explored nondeterministically and $\mathcal{B}$ is gradually determinized, so the algorithm explores pairs (p,P) where $p \in Q_1$ and $P \subseteq Q_2$. The antichains algorithm derives new states along the product automaton transitions and inserts them to the set of visited pairs X. X keeps only minimal elements with respect to the ordering given by $(r,R) \sqsubseteq (p,P)$ iff $r = p \wedge R \subseteq P$. If there is generated a pair (p,P) and there is $(r,R) \in X$ such that $(r,R) \sqsubseteq (p,P)$, we can skip (p,P) and not insert it to X for further search.

An improvement of the antichains algorithm using simulation [2] is based on the follow optimization. We can stop search from a pair (p,P) if either (a) there exists some already visited pair (r,R) such that $p \preceq r \wedge \forall r' \in R \, \exists \, p' \in P : r' \preceq p'$, or (b) there is $p' \in P$ such that $p \preceq p'$.

3.2 BISIMULATION UP TO CONGRUENCE

Another approach to checking language inclusion of NFA is based on bisimulation up to congruence. This technique was originally developed for checking equivalence of languages of automata but it can also be used for checking language inclusion, based on the observation that $L_{\mathcal{A}} \cup L_{\mathcal{B}} = L_{\mathcal{B}} \Leftrightarrow L_{\mathcal{A}} \subseteq L_{\mathcal{B}}$. This approach is based on the computation of a *congruence closure* $c(R)$ for

some binary relation on states of the determinized automaton $R \subseteq 2^Q \times 2^Q$ defined as a relation $c(R) = (r \cup s \cup t \cup u \cup id)^w(R)$, where $id(R) = R$, $r(R) = \{(X,X) \mid X \subseteq Q\}$, $s(R) = \{(Y,X) \mid XRY\}$, $t(R) = \{(X,Z) \mid \exists Y \subseteq Q,\ XRYRZ\}$, $u(R) = \{(X_1 \cup X_2, Y_1 \cup Y_2) \mid X_1 R Y_1 \wedge X_2 R Y_2\}$.

The congruence algorithm works on a familiar principle as the antichains algorithm. It starts building $\mathcal{A}_{det}$ and $\mathcal{B}_{det}$ and checks if macrostates in generated pairs are both final or not. The optimization used is based on computing congruence closure of the set of already visited pairs of macrostates. If the generated pair is in this congruence closure it can be skipped and not further processed.

Comparison of mentioned approaches to the checking language inclusion could be seen in Figure 1.

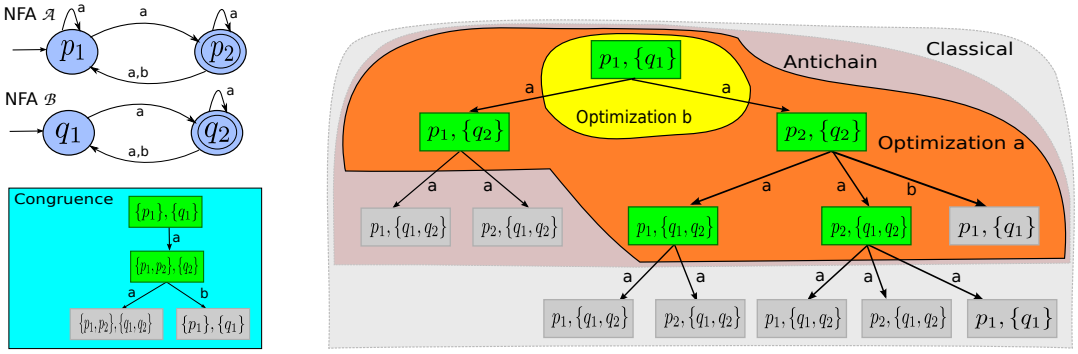

Figure 1: The picture is based on an example from [2]. It shows checking language inclusion between NFA $\mathcal{A}$ and NFA $\mathcal{B}$. The labeled areas correspond to mentioned approaches. The antichain algorithm reduces number of generated states compared with the classical one, e.g., $(p_2,\{q_1,q_2\})$ is not further searched because $(p_2,\{q_2\}) \sqsubseteq (p_2,\{q_1,q_2\})$. The optimization a and b are improvements of the antichain algorithm using simulation. The congruence algorithm also reduces number of generated states, so $(\{p_1,p_2\},\{q_1,q_2\})$ is not further searched because it is in congruence closure of visited states.

4 CONCLUSION

This paper briefly describes new approaches to checking language inclusion of NFA. These algorithms are being implemented in the new extension for finite automata manipulation of VATA library.

ACKNOWLEDGEMENT

This work was supported by the Czech Science Foundation within the project No. P103/10/0306.

REFERENCES

[1] M. De Wulf, L. Doyen, T.A. Henzinger and J.-F. Raskin Antichains: A New Algorithm for Checking Universality of Finite Automata. CAV 2006. Springer-Verlag (2006).

[2] Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, Richard Mayr and Tomáš Vojnar. When Simulation Meets Antichains: On Checking Language Inclusion of Nondeterministic Finite (Tree) Automata. TACAS 2010. Springer-Verlag (2010).

[3] Filippo Bonchi and Damien Pous. Checking NFA Equivalence with Bisimulations up to Congruence. POPL 2013. ACM (2013).

[4] Ondřej Lengál, Jiří Šimáček and Tomáš Vojnar. VATA: A Library for Efficient Manipulation of Non-deterministic Tree Automata. TACAS 2012. Springer-Verlag (2012).