BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

ABSTRACTION FOR FINITE AUTOMATA IN PROGRAM ANALYSIS

ABSTRAKCE PRO KONEČNÉ AUTOMATY V ANALÝZE PROGRAMŮ

PROJECT PRACTICE 1
PROJEKTOVÁ PRAXE 1

AUTHOR AUTOR PRÁCE	DÁVID BOLVANSKÝ
SUPERVISOR VEDOUCÍ PRÁCE	Mgr. LUKÁŠ HOLÍK, Ph.D.

BRNO 2017

Abstract

In this work we analyze the problem of computing intersection of automata and deciding its emptiness via computing lengths of words in the language of the automaton. In order to achieve this, we implemented our algoritm to compute the lengths. Using lengths we create aritmetic formula of lengths and then we decide about its satisfiability using SMT solver.

Abstrakt

V tejto práci skúmame problém prieniku automatov a následného rozhodovania o prázdnosti prieniku pomocou počítania dĺžiek slov v jazyku rozpoznávaného automatom. Za týmto účeľom sme vytvorili algoritmus na počítanie dĺžok. Pomocou dĺžok vytvoríme aritmetickú formu dĺžok a následne rozhodneme o jej splniteľnosti pomocou SMT solvera.

Keywords

automata, abstraction, lengths of words, intersection, powerset construction, approximation

Klíčová slova

automat, abstrakcia, dĺžky slov, prienik, determinizácia, aproximácia

Reference

BOLVANSKÝ, Dávid. *Abstraction for Finite Automata in Program Analysis.* Brno, 2017. Project Practice 1. Brno University of Technology, Faculty of Information Technology. Supervisor Holík Lukáš.

Abstraction for Finite Automata in Program Analysis

Declaration

Hereby I declare that this work was prepared as an original author's work under the supervision of Mgr. Lukáš Holík, Ph.D. All the relevant information sources, which were used during preparation of this work, are properly cited and included in the list of references.

. . . .
Dávid Bolvanský
February 2, 2017

Contents

Chapter 1

Introduction

Automata appear almost everywhere in computer science. Although classical finite automata are quite simple, many variants and extensions have seen wide applicability. They are heavily used in logic design, natural language processing, systems analysis, model checking, text processing, etc.

We concentrate on their applications in verification and analysis of computing systems and in decision procedures of logics. More particularly, we are interested in analysis of string manipulating programs, language theoretic model checking, decision procedures for WS1S [5] and Presburger [7], abstract regular model checking, forest automata based verification of pointer programs [2]. Among some interesting automata problems belong boolean operations. By these we mean operations with automata like computing intersection, union, complement and deciding emptiness of the result language. Most of automata algorithms are based on some form of product construction.

1.1 Emptiness of Intersection of Automata

In this work, we focus on the simplest problem of these, the problem of computing intersection of automata and deciding its emptiness.

The main idea is optimization of the product construction using information about the possible lengths of strings in the language. So what is our problem? Basically, we need to verify the following formula:

$$L(A) \cap L(B) = \emptyset$$

We started to work with basic version of this optimization. It aims to compute the possible lengths of words in the language. We create artimetic formulas describing of lengths of words in languges of automata and then any SMT solver can decide about satisfiability of the conjunction of both formulas and then, whether intersection is empty or not. If it is not satisfiable, we are sure that the intersection is empty. Otherwise, we cannot declare anything and we have to perform the clasical product construction to find out more.

As the first step, we implemented an algoritm which does what we mentioned above - computes the lengths of words in a regular language of an automaton. We plan to test it more and focus to make it more efficient. However, there are ways to improve it even more. Particularly, we could compute Parikh images [4] for every state, written as $\mathcal{P}(q)$, and prune out all pairs (q, q') in the product construction, which do not satisfy $SAT(\mathcal{P}(q) \wedge \mathcal{P}(q'))$. We have an improvement even if the conjuction is satisfiable (with previous basic method we don't know anything) because information about Parikh images of individual states allows us to prune the product construction.

1.2 Similar Works

Our construction is similar as the approach described in other work [3]. Possibly another useful work [4] describes the computation of the lengths of strings or of a Parikh image in a regular / context free language more efficiently than via determinization of one letter automaton. Recently published work [1] metions an abstraction in which a Petri net markings are allowed to carry rational values and the transitions can work with fractional parts of tokens. If the target marking is not coverable in the abstract system, then it is not coverable in the classical sense. A heuristic very effective in combination with the classical backward algorithm.

Chapter 2

Preliminaries

In this section, we recapitulate basic relevant automata constructions.

Definition 1 *A **finite automaton** is a 5-couple $(Q, \Sigma, \Delta, q_0, F)$, where:*

- *Q is a finite set called the states,*
- *Σ is a finite set called the alphabet,*
- *$\Delta : Q \times \Sigma_\epsilon \longrightarrow \mathcal{P}(Q)$ is the transition function, where $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\mathcal{P}(Q)$ is the power set of Q,*
- *$q_0 \in Q$ is the start state, and*
- *$F \subseteq Q$ is set of accept states.*

Every nondetermistic finite automation has an equivalent deterministic finite automata which can be computed using the following construction.

Definition 2 *Let Σ^* denote the set of all strings over Σ. Every subset $L \subseteq \Sigma^*$ is a language over Σ.*

Definition 3 *Let $M = (Q, \Sigma, \Delta, q_0, F)$ be a finite automaton and let $w = w_1, w_2, \ldots, w_n$ be a string where each w_i is a member of the alphabet Σ. Then M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:*

1. *$r_0 = q_0$*
2. *$\Delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and*
3. *$r_n \in F$.*

Condition 1 says that the machine starts in the start state. Condition 2 says tha the machine goes from state to state according the transition function. Condition 3 says that the machine accepts its input if it ends up in an accept state. We say that M recognizes language A if $= \{w \mid M \text{ accepts } w\}$.

Definition 4 *A language is called a regular language if some finite automaton recognizes it.*

Definition 5 ***Powerset construction** or **subset construction** is a standard method for converting a nondeterministic finite automaton (NFA) into a deterministic finite automaton (DFA) which recognizes the same formal language.*

Let $N = (Q, \Sigma, \Delta, q_0, F)$ be the NFA recognizing some language L. We construct a DFA $M = (Q', \Sigma', \Delta', q'_0, F')$ with the same language, where:

- $Q' = 2^Q$
- $\Sigma' = \Sigma$,
- Δ'*: For $R \in Q'$ and $a \in \Sigma$ let $\Delta'(R, a) = \{q \in Q \mid q \in \Delta(r, a)$ for some $r \in R\}$,*
- $q'_0 = \{q_0\}$,
- $F' = \{R \in Q' \mid R$ *contains an accept state of* $M\}$.

Definition 6 *Let w be a string over Σ. The length of w, written as $|w|$, is defined as follows:*

- *if $w = \epsilon$, then $|w| = 0$,*
- *if $w = a_1 \ldots a_n$, then $|w| = n$, for some $n \geq 1$, and $a_i \in \Sigma$ for all $i = 1, \ldots, n$.*

Definition 7 *The automaton operation* ***union*** *combines two automata $M_1 = (Q_1, \Sigma, \Delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \Delta_2, q_2, F_2)$ into an automaton $M' = (Q', \Sigma', \Delta', q'_0, F')$ such that $L(M') = (M_1) \cup (M_2)$ and its components are defined as follows:*

- *Q' is the Cartesian product of Q_1 and Q_2, written as $Q_1 \times Q_2$,*
- *Σ' is Σ, since we assume that both M_1 and M_2 have the same input alphabet Σ,*
- *Δ': for each $(r_1, r_2) \in Q'$ and each $a \in \Sigma$, let $\Delta((r_1, r_2), a) = (\Delta_1(r_1, a), \Delta_2(r_2, a))$,*
- *q'_0 is a pair (q_1, q_2),*
- *F' is $(F_1 \times Q_2) \cup (F_2 \times Q_1)$.*

Definition 8 *The automaton operation* ***intersection*** *combines two automata $M_1 = (Q_1, \Sigma, \Delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \Delta_2, q_2, F_2)$ into an automaton $M' = (Q', \Sigma', \Delta', q'_0, F')$ such that $L(M') = (M_1) \cap (M_2)$ and its components are defined as follows:*

- *Q' is the Cartesian product of Q_1 and Q_2, wrriten as $Q_1 \times Q_2$,*
- *Σ' is Σ, since we assume that both M_1 and M_2 have the same input alphabet Σ,*
- *Δ': for each $(r_1, r_2) \in Q'$ and each $a \in \Sigma$, let $\Delta((r_1, r_2), a) = (\Delta_1(r_1, a), \Delta_2(r_2, a))$,*
- *q'_0 is a pair (q_1, q_2),*
- *F' is $F_1 \times F_2$.*

Definition 9 *The* **complement** *of a language L (with respect to an alphabet Σ such that Σ^* contains L) is $\Sigma^* \setminus L$.*

$$\overline{L} = \Sigma^* \setminus L$$

Definition 10 *An automaton operation* ***complement*** *computes an automaton M' of a deterministic automaton M, where $M = (Q, \Sigma, \Delta, q_0, F)$ and $L(M') = \Sigma^* \setminus L(M)$, which is defined as: $M' = (Q, \Sigma, \Delta, q_0, Q \setminus F)$*

Definition 11 *A one letter automaton is an automaton with cardinality of its input alphabet equaling one, written as:* $|\Sigma| = 1$.

Definition 12 *Brzozowski's algorithm for DFA minimization uses the powerset construction, twice, to compute the smallest deterministic automaton of the given automaton. It converts the input DFA into an NFA for the reverse language, by reversing all its arrows and exchanging the roles of initial and accepting states, converts the NFA back into a DFA using the powerset construction, and then repeats the process.*

Our aim is to decide about this problem:

$$L(A) \cap L(B) = \emptyset$$

Definition 13 *An emptiness test of the language of the automaton analyzes the reachability of any final state of the product of automata from the initial state.*

Chapter 3

Approximating the Emptiness Test by the Word Lengths Computation

As we previously metioned, we are intested in the following problem:

We introduced our notation - $\Phi(A)$, which represents an aritmetic formula of lenghts of words in the language of automaton A.

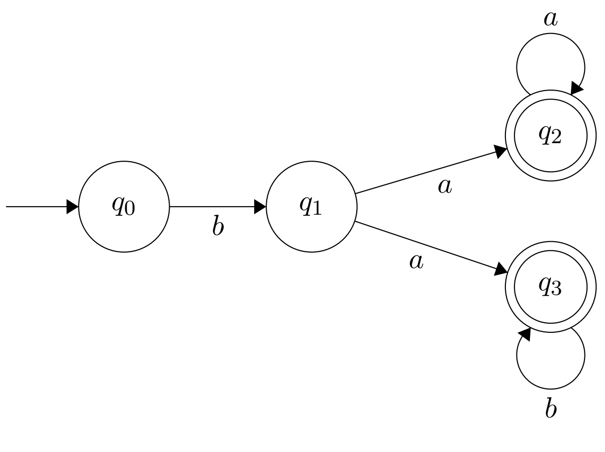
$$\parallel \Phi(A) \parallel = \{|w| \mid w \in L\}$$

Also, we introduced another notation - $\Phi_q(A)$, representing an aritmetic formula for every final state of automaton A.

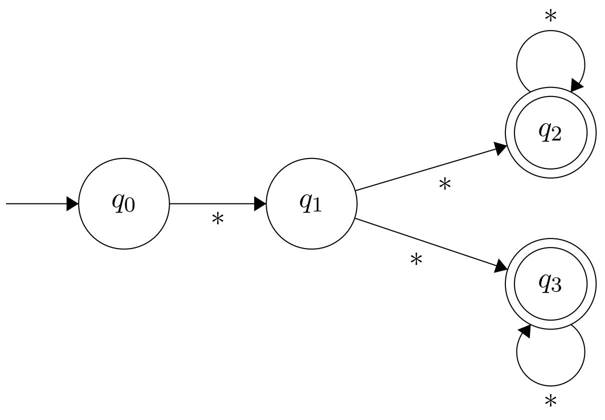
$$\Phi(A) = \bigvee_{q \in F} \Phi_q(A)$$

We have two automata A and B. We compute $\Phi(A)$ and $\Phi(B)$ and the we check $SAT(\Phi(A) \wedge \Phi(B))$.

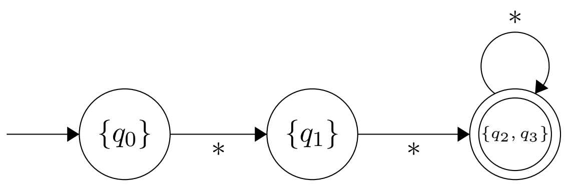
We will explain the process of computing $\Phi(A)$ below. At first, we have a nondeterministic finite automaton (NFA) - A.



We have to convert it to a one letter NFA - A', which has a different language but it is known that $\Phi(A) = \Phi(A')$. Below you can see an example result of the conversion.

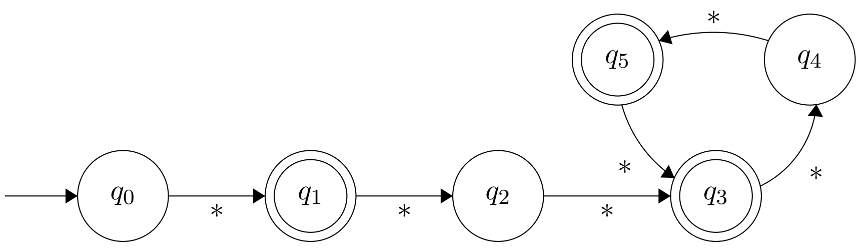


Next, we determinize our one letter NFA A' to get a one letter deterministic finite automata (DFA) A'' using the powerset construction.



All one letter deterministic finite automata have handle and loop parts.

To illustrate how we create the formula from the one letter automaton, we choose a different example below.



As we mentioned, it has two parts - the handle and the loop. Therefore, for our example, states q_0, q_1, q_2, q_3 and transitions between them are in the handle, states q_3, q_4, q_5 and transitions between them are in the loop.

We mark the length of handle as n, the length of the part from the initial state to any state as v, the length of the loop as k, and l represents cycling in the loop. With the following algoritm we can determine the lengths of words in the language of the automaton. For every $q \in Q$ of an automaton A we obtain $\Phi_q(A)$ as follows:

$$\Phi_q(A) = \begin{cases} |d| = v & \text{if } q \text{ in handle, } v < n \\ |d| = v + k * l & \text{if } q \text{ in loop, } v \geq n \end{cases}$$

Let us describe our implementation in detail. The input of our algorithm is a one letter deterministic finite automaton and the output is pthe formula of lengths of words of the language of this automaton. The implemementation in following pseudocode presented below works with the automanon using the following data structures. There is aut, which is structure which holds states of an automaton as an array of pointers, the initial state is represeted as the first item in this array. We have a record called state which contains truth value whether this state is final or not, and pointer to the next state, if any. We have two counters - to count the length of the part from the initial state to any state and to count the length of the loop. Also, we use an array to save states in the loop. After we find any final state, we put it to this array and we put all following states here too. We check if there is any state in this array twice - which means an infinite cycling and therefore, no loop. We count the length of the part from the initial state to any final state. Then we count the length of the loop and finally we print the formula of the lengths of the words in the language of the automata.

```
first = aut.states[0];
for state in aut.states:
    loop_states = {};
    v = 0, loop = 1; s = first;
    is_loop = true, in_loop = false;

    while (true) {
        if (loop_states.contains(s) && s == state)) {
            break;
        }

        if (s == NULL) {
            is_loop = false;
            break;
        }

        if (s == state) {
            in_loop = true;
            if (s.is_final) {
                print("d = %d", v);
            }
        }

        if (in_loop) {
             if (loop_states.contains(s)) {
                is_loop = false;
                break;
            }
            loop_states.add(s);
            loop++;
        } else {
            v++;
        }

        s = s.next;
    }

    if (is_loop) print("d = %d + %d * l", v, loop);
```

With computed lengths and especially, the final artimetic formula, we can safely aproximate the emptiness test. SMT solver processes our aritmetic formula $SAT(\Phi(A) \wedge \Phi(B))$ and we are finally able to decide about the emptiness of intersection of automata by checking if the formula is satisfiable or not. If it is not, then intersection is empty, otherwise we don't know anything useful (as typical fact for safe approximation). We implemented our algoritm in Java. We used BRICS library [1] for automaton operations like determinization. As a SMT solver, we chose open source solver called Alt-Ergo [2].

[1]http://www.brics.dk/automaton/

[2]http://alt-ergo.lri.fr/

Chapter 4

Conclusion

Our aim has been to implement basic version of algorithm to decide about emptiness of automata intersection. As we see, in the basic version, we have useful information about the emptiness only if the intersection is satisfiable - if it is not, we know nothing. As we mentioned in the Introduction, possibly the next work to solve this problem more efficiently would be based on tightly interconnection of product construction with the lengths of words and / or Parikh images.

Bibliography

[1] Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. Approaching the coverability problem continuously. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 480–496, 2016.

[2] Lukás Holík, Martin Hruska, Ondrej Lengál, Adam Rogalewicz, and Tomás Vojnar. Counterexample validation and interpolation-based refinement for forest automata. In *Verification, Model Checking, and Abstract Interpretation - 18th International Conference, VMCAI 2017, Paris, France, January 15-17, 2017, Proceedings*, pages 288–309, 2017.

[3] Lukás Holík, Malte Isberner, and Bengt Jonsson. Mediator synthesis in a component algebra with data. In *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015. Proceedings*, volume 9360 of *Lecture Notes in Computer Science*, pages 238–259. Springer, 2015.

[4] Eryk Kopczynski and Anthony Widjaja To. Parikh images of grammars: Complexity and applications. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 80–89, 2010.

[5] Mukesh Kumar and Kamlesh Dutta. Detecting wormhole attack on data aggregation in hierarchical WSN. *IJISP*, 11(1):35–51, 2017.

[6] Michael Sipser. *Introduction to the theory of computation.* PWS Publishing Company, 1997.

[7] Makoto Tatsuta, Quang Loc Le, and Wei-Ngan Chin. Decision procedure for separation logic with inductive definitions and presburger arithmetic. In *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, pages 423–443, 2016.