The equivalence problem for programs with mode switching is PSPACE-complete

Rimma Podlovchenko, Dmitry Rusakov and Vladimir Zakharov

Abstract. We study a formal model of imperative sequential programs. In this model programs are viewed as deterministic finite automata whose semantics is defined on Kripke structures. We focus on the equivalence problem for some specific class of programs — programs with mode switching — whose runs can be divided into two stages. In the first stage a program selects an appropriate mode of computation. Several modes may be tried (switched) in turn before making the ultimate choice. Every time when the next mode is put to a test, the program brings data to some predefined state. In the second stage of the run, once a definitive mode is fixed, the final result of computation is produced. The effect of mode switching may be used for automatic generation of opaque predicates, i.e. boolean expressions whose behavior is known a priori. Such predicates provide a very simple and effective means for virus obfuscation; therefore the development of efficient algorithms for the analysis of programs with mode switching is an urgent task in view of designing virus detection tools. We develop a new technique for simulating the behavior of such programs by means of finite automata and demonstrate that the equivalence problem for programs with mode switching is decidable within a polynomial space. By revealing a close relationship between the equivalence problem for this class of programs and the intersection emptiness problem for deterministic finite state automata we show that the former is *PSPACE*-complete.

1. Introduction

The intimate linkage among automata and formal models of programs used for the purpose of translation, verification, optimization, etc. is widely recognized. Since the early 60-th it was found out [4, 5, 9, 21, 29] that common finite state, multi-tape, multi-head, push-down automata give a suitable framework for developing decision procedures/proving undecidability for many program analysis problems originated from software engineering (see [34] for a survey). In this paper we study one of such problems, namely, the equivalence problem, for some specific class of imperative programs with mode switching. The runs of such programs are divided into two stages. In the beginning of a run an

```
program π1                  program π2
read (y1,y2);               read (y1,y2);
x=y1; z=y2;                 x=y1; z=y2;
u=x+z;                      if P(x,z) then {x=z; goto L1}
stop.                                 else {z=z+x; goto L2};
                            L1:  z=y1; x=y2;
                            if P(z,x) then {u=x+z; z=x;
                            x=u-z}
                                      else {u=x-z; z=u+x};
                            stop;
                            L2:  z=y2; x=y1;
                            if P(x,z) then u=2*x-z**2;
                                      else u=z+z;
                            stop.
```

Figure 1: Programs with mode switching

appropriate mode of computation is selected. A program may try (switch) a number of modes in turn before making the final choice. In the second stage, once some mode of computation is chosen, a program starts an ordinary run and yields the final result.

Usually mode switching is achieved by means of constant assignment statements. Two programs π_1 and π_2 in Fig 1. illustrate the concept of mode switching. It is easy to see that both programs are equivalent, i.e. they compute the same function $u = y_1 + y_2$. Boxed statements in these programs play the role of mode switches; when executing these statements the programs bring data to the predefined states that are specific for each mode. Our interest to the equivalence problem for programs with mode switching has both theoretical and practical motivations. In [23, 24] a theory of algebraic models of programs was 1 program introduced for the purpose of designing effective equivalence-checking techniques and complete systems of equivalent transformations of programs. In this theory programs are viewed as deterministic finite automata operating on semigroups or Kripke structures. A series of obtained results [26, 32] show that for many algebraic models of programs it is possible to design efficient (polynomial-time) equivalence-checking procedures. It was found out (see [14, 15, 16, 18]) that decidability and complexity of the equivalence problem for algebraic models of programs depend greatly on some group-theoretic properties of program semantics. That is why it is very important to know how much this or that property of a semigroups or Kripke structures used for the semantics in algebraic models of programs influences the decidability and complexity of the equivalence problem.

In the framework of the theory of algebraic models of programs the semantics of programs with mode switching can be specified in terms of semigroups with right zeroes. In [13, 14] the equivalence problem for finite automata operating on free semigroups with right zeros is considered. A decidability result was obtained by establishing the regularity preserving properties of the set-theoretic and closure operations on topological spaces of functions computed by such automata. In [20, 19] a “hard set” method was successfully applied to the equivalence problem for linear recursive programs with constants. It must be emphasized that in the model of recursive programs constants play the same role as mode switching statements in the propositional model of sequential programs we deal with in this paper. Both equivalence-checking techniques developed in [13, 14, 19] are very much sophisticated; their main deficiency is that they give no means to estimate the complexity of the problem. One of the aims of our paper to is to estimate precisely (as much as possible) the complexity of the equivalence problem for programs with mode switching.

Another topic which involves programs with mode switching is malicious pattern (viruses) detection in software. The classic virus-detection techniques look for the presence of a virus specific sequence of instructions (*virus signature*) inside the programs: if the signature is found, it is highly probably that the program is infected. A new generation of *metamorphic viruses* attempts to evade simple pattern-matching detection by using complex obfuscations: when replicating these viruses change their signatures by applying semantic-preserving transformations (see [1]). The only way to disclose such viruses is to develop efficient equivalence-checking techniques that could cope with the common obfuscating transformations. Thus, in [2] an architecture for detecting malicious patterns in executables is presented that is resilient to some obfuscating transformations (dead-code insertion and code transposition). But the properly resistant obfuscations [3] rely on the existence of opaque predicates whose behavior is known a priori to the obfuscator, but which is difficult for the deobfuscator to deduce. The most simple way to get the opaque predicates is to make the same predicate computable on the same data but in different program points. This may be achieved by bringing data into some fixed state before computing such predicate several times along a run. But this is just the effect of mode switching. When considering the program π_2 depicted in the Fig. 1 one could find that both underlined conditions in the branching statements are opaque predicates obtained with the help of mode switching: one of them is always evaluated to true whereas the other to false. Thus, to detect a metamorphic virus (the program π_2) by its signature (the program π_1) one need an effective equivalence-checking procedure which could cope with mode switching.

In our paper we reveal close relationships between the equivalence problem for programs with mode switching and the Intersection Emptiness Problem (IEP) for deterministic finite state automata (DFAs). The Intersection Emptiness Problem for DFA is that of checking, given a collection of k DFAs $F_1, \dots, F_k$ of size $|F_i| = n$, if their intersection is empty: $\bigcap\limits_{i=1}^{k} L(A_i) \overset{?}{\neq} \emptyset$, where $L(F)$ denotes the language accepted by the automaton F. If the parameter k is a constant then the problem has a polynomial time algorithm, but the general problem, where this parameter can depend on the input size (say, $k = n$), is much harder, known to be *PSPACE*-complete [12].

Some recent papers testify that the IEP would have a substantial impact on many aspects of complexity theory and formal verification of complex systems. In [10] it was proved that integer factoring of an n-bit number is solvable in time $n^{O(1)} \cdot 2^{\varepsilon n}$ for any $\varepsilon > 0$, provided that one can decide the IEP for a family $F_1, \dots, F_k$ of DFAs F_i of size n in time $t(n,k) = n^{\frac{k}{f(k)}+d}$, where $f(\cdot)$ is an unbound function, and $d > 0$ is a constant. Moreover, by assuming that there is a non-uniform circuit that will solve the IEP with size $t(n,k)$, one could deduce $NLOG \neq NP$. In [28] the IEP was used to demonstrate that a great many verification problems of supervisory controllers for discrete-event systems are *PSPACE*-complete. We think that the results of our paper also give a new insight into the IEP.

The paper is organized as follows. In Section 2 we define formally the syntax and the semantics of propositional sequential programs. In Section 3 the model $\mathcal{M}_0$ which captures the semantics of programs with mode switching is introduced. We also reduce the equivalence problem $\pi' \sim_{\mathcal{M}_0} \pi''$ for programs with mode switching to that of checking three characteristic properties of the runs of π', π''. In Section 4 we show that these properties can be verified by constructing a finite number of DFAs and checking their emptiness. The DFAs we use for this purpose are similar to Vectorized Finite State Automata introduced in [11] for natural language processing. A vectorized DFA operates on a tape divided into N tracks and its internal states are vectors $s = \langle v_1, \dots, v_N \rangle$. Some tracks may be synchronized; it is required that the input letters on the synchronized tracks should be unified (in our case this means that the letters on these tracks should be the same). The synchronization of tracks varies along a run of DFA. Vectorized DFAs have the same computation power as common DFAs and the emptiness problem for them is *NLOG*-complete. Since the state space of DFAs we use is exponential of the size of programs to be analyzed, we arrive at the conclusion that the equivalence problem $\pi' \sim_{\mathcal{M}_0} \pi''$ is in *PSPACE*. In Section 5 we reduce the IEP to the equivalence problem for programs mode switching and establish thus the *PSPACE*-completeness of the latter. We conclude with discussing some new research problems caused by the results obtained.

2. Preliminaries

In this section we define the syntax and the semantics of propositional sequential programs.

Fix two finite alphabets $\mathcal{A} = \{a_1, \dots, a_r\}$, $\mathcal{P} = \{p_1, \dots, p_k\}$. The elements of $\mathcal{A}$ are called *basic statements*; they stand for assignment statements in imperative programs. The elements of $\mathcal{P}$ are called *basic predicates*; they stand for elementary built-in relations on program data. Each basic predicate may be evaluated by 0 (false) or 1 (true). A tuple $\langle \delta_1, \dots, \delta_k \rangle$ of truth-values of basic predicates is called a *condition*. The set of all conditions is denoted by $\mathcal{C}$; we write $\Delta_1, \Delta_2, \dots$ for generic elements from $\mathcal{C}$.

Definition 1. *A deterministic* propositional sequential program *(*PSP *for short) is a finite transition system $\pi = \langle V, \mathbf{entry}, \mathbf{exit}, T, B \rangle$, where*

- V *is a non-empty set of* program points*;*
- **entry** *is the* initial point *of the program;*
- **exit** *is the* terminal point *of the program;*
- $T\colon (V - \{\mathbf{exit}\}) \times \mathcal{C} \to V$ *is a (total)* transition function*;*
- $B\colon (V - \{\mathbf{exit}\}) \to \mathcal{A}$ *is a (total)* binding function.

A transition function represents the control flow of a program, whereas a binding function associates with each point some basic statement. By the *size* $|\pi|$ of a program π we mean the cardinality of the set V. Any finite sequence of points $v_1, v_2, \dots, v_n$ such that for every i, $1 \le i < n$, $v_{i+1} = T(v_i, \Delta_i)$ holds for some condition Δ_i, is called a *control path* (or a *trace*) in the PSP π from v_1 to v_n. We say that v_n is *reachable* from v_1 if there exists a trace from v_1 to v_n.

The semantics of PSPs is defined with the help of Kripke structures used in the framework of dynamic logics.

Definition 2. *A* Kripke structure *is a quadruple $M = \langle S, s_0, R, \xi \rangle$, where*

- S *is a non-empty set of* data states*;*
- $s_0 \in S$ *is a distinguished* initial state*;*
- $R : \mathcal{A} \times S \to S$ *is a (total)* updating function*;*
- $\xi : S \to \mathcal{C}$ *is a (total)* evaluation function.

An updating function R gives the interpretation of basic statements: a data state $R(a, s)$ is the result of application of a basic statement a to a data state s. An evaluation function ξ is used for the interpretation of basic predicates: $\xi(s)$ gives a tuple of truth-values for all basic predicates on a data state s. By a *data path* in M we mean any sequence of states $s_1, s_2, \dots, s_k$ such that $s_{i+1} = R(a_i, s_i)$ for some basic statement a_i, $1 \le i < k$.

Let $\pi = \langle V, \mathbf{entry}, \mathbf{exit}, T, B \rangle$ be a PSP and $M = \langle S, s_0, R, \xi \rangle$ be a Kripke structure. A *run* of π on M is a sequence (finite or infinite) of pairs

$$r(\pi, M) = (v_1, s_1), (v_2, s_2), \dots, (v_i, s_i), (v_{i+1}, s_{i+1}), \dots \qquad (1)$$

such that

1. s_0 is the initial state of M, and $v_1 = \mathbf{entry}$;
2. $s_i = R(B(v_i), s_{i-1})$ and $v_{i+1} = T(v_i, \xi(s_i))$ hold for every i, $i \ge 1$;
3. the sequence $r(\pi, M)$ either is infinite (in this case we say that the run *loops* and yields no results), or ends with a pair (v_n, s_n, Δ_n) such that $v_{n+1} = \mathbf{exit}$ (in this case we say that the run *terminates* and gives a result s_n).

We write $\downarrow r(\pi, M)$ to indicate that the run terminates and denote by $[r(\pi, M)]$ its result assuming that the result is undefined when $r(\pi, M)$ loops. It is worth noting that if $v_i = v_j$ and $s_i = s_j$ for some pair of triples in (1) then $r(\pi, M)$ loops. If a point v_i occurs in some triple of (1) then we say that $r(\pi, M)$ *passes via* v_i.

In what follows when referring to a *model of programs* $\mathcal{M}$ we mean the set of all PSPs over fixed alphabets $\mathcal{A}, \mathcal{P}$ whose semantics is specified by the set $\mathcal{M}$ of Kripke structures.

Definition 3. *Given a model of programs $\mathcal{M}$, we say that PSPs π_1 and π_2 are* equivalent *($\pi_1 \sim_{\mathcal{M}} \pi_2$ in symbols) iff $[r(\pi_1, M)] = [r(\pi_2, M)]$ for every $M \in \mathcal{M}$.*

The equivalence problem for a model of programs $\mathcal{M}$ is to check, given a pair of PSPs π_1 and π_2, whether $\pi_1 \sim_{\mathcal{M}} \pi_2$ holds. The complexity of the equivalence problem "$\pi_1 \sim_{\mathcal{M}} \pi_2$?“ depends on the set of structures M which specifies a model of programs. Two examples below illustrate this thesis.

Example 1. *Given a set calA of basic statements, consider a free semigroup $\langle \mathcal{A}, \circ \rangle$ generated by $\mathcal{A}$. The elements of this semigroup may be thought of as finite sequences (words) of basic statements, whereas binary operation $\circ$ may be interpreted as concatenation. The empty sequence lambda stands for the neutral element of the semigroup. Then the equivalence problem for the model of programs $M' = \{\langle calA^*, \lambda, \circ, \xi \rangle \ : \ \xi$ is an evaluation function on $\mathcal{A}$ is decidable in time $O(n \log n)$. In [9, 29]it was demonstrated that the equivalence problem for M' is reducible in linear time to the the equivalence problem for deterministic finite automata; the complexity $O(nlogn)$ of the latter was established in [8].*

Example 2. *Given a set $\mathcal{A} = \{a_1, \dots, a_n, a_1^{-1}, \dots, a_n^{-1}\}$ of basic statements consider Abelean group $\langle S, \circ \rangle$ of rank n generated by the elements from $\mathcal{A}$. The equivalence problem for the model $M' = \{\langle S, e, \circ, \xi \}$: ξ is an evaluation function on S, where e is the unit (neutral) element of S, was studied in [16]. It was shown that this problem is decidable within exponential space when $n = 1$, and it is undecidable when $n \geq 2$.*

Other results on the equivalence problem for some models of programs may be found in [16, 17, 18, 25, 26, 27, 34, 35, 36].

3. Programs with mode switching

In this paper we focus on the equivalence problem for some specific class of programs whose runs can be divided into two stages. In the first stage a program selects an appropriate mode of computation. Several modes may be tried in turn before making the ultimate choice. Every time when the next mode is put to the test, the program brings the data back to the initial state. In the second stage, once a definitive mode is fixed, the final result of computation is generated. In real programs mode switching may be implemented by restart statements or constant assignment statement. In this section we introduce formally the model of such programs in the framework of PSP's syntax and semantics.

We will assume that the set of basic statements $\mathcal{A}$ is partitioned into two subsets $\mathcal{A}_{ord} = \{a^1, \dots, a^k\}$ (*ordinary actions*) and $\mathcal{A}_{mode} = \{b^1, \dots, b^N\}$ (*mode switches*). Those points v in a PSP π that are associated with mode switches (i.e. $B(v) \in \mathcal{A}_{mode}$) are called *switching points*. All other points are called *ordinary points*. We write V_{mode} to denote the set of all switching points of a given program π. Without loss of generality, we will assume that **entry** $\in V_{mode}$.

Two principles are used as the basis for the semantics of PSPs with mode switching:

- each ordinary action a is interpreted according to a current mode of computation, and
- each mode switch b abandons any previous intermediate result of computation and brings data into some distinguished state s_b.

Thus, the model of programs with mode switching is characterized by the set of all Kripke structures $\mathcal{M}_0 = \{M_\xi \ : \ M_\xi = \langle S, s_0, R, \xi \rangle\}$ such that

1. $S = \{\varepsilon\} \cup \mathcal{A}_{mode}\mathcal{A}^*_{ord}$, i.e. S includes the empty string λ and all strings $ba_1a_2 \dots a_n$, where a mode switch $b \in \mathcal{A}_{mode}$ is followed by a string of ordinary actions $a_1a_2 \dots a_n \in \mathcal{A}^*_{ord}$;
2. $s_0 = \lambda$;
3. an updating function R is defined as follows:

$$R(y, s) = \begin{cases} sy, & \text{if } y \in \mathcal{A}_{ord}, \\ y, & \text{if } y \in \mathcal{A}_{mode}. \end{cases}$$

Since the data space S and the interpretation of basic statements R are fixed, each structure $M_\xi \in \mathcal{M}_0$ is completely specified by its evaluation function ξ. PSPs with mode switching π_1 and π_2 are called equivalent ($\pi_1 \sim_{\mathcal{M}_0} \pi_2$ in symbols) iff $[r(\pi_1, M_\xi)] = [r(\pi_2, M_\xi)]$ holds for every structure $M_\xi \in \mathcal{M}_0$. When studying decidability and complexity of the equivalence problem for $\mathcal{M}_0$ we will use the inverse variant of this definition: PSPs π_1 and π_2 are not equivalent iff there exists a structure $M_\xi \in \mathcal{M}$ such that either both runs $r(\pi_1, M_\xi)$ and $r(\pi_2, M_\xi)$ terminate but $[r(\pi_1, M_\xi)] \neq [r(\pi_2, M_\xi)]$, or one of the runs (say, $r(\pi_1, M_\xi)$) terminates, whereas the other (in our case, $r(\pi_2, M_\xi)$) loops.

Given a PSP π, we introduce the *skeleton* G_π of π as a finite directed graph intended for representing the reachability relation between the switching points in π. Formally, $G_\pi = (U, E)$, where $U = V_{mode} \cup \{\textbf{exit}\}$ is the set of vertices and $E = \{\langle v', v'' \rangle \ : \ v', v'' \in U$, and v'' is reachable from v' in π by a trace which does not pass via any switching point other than v' and $v''\}$ is the set of arcs. A *trajectory* is any directed path in G_π (finite or infinite) which begins in the **entry** point. A trajectory reflects a possible scenario of mode selection in the course of some run of π. We say that a run $r(\pi, M_\xi)$ of a PSP π on a structure M_ξ *traverses the skeleton* G_π *along a trajectory* $v_1, v_2, \dots, v_n, \dots$ if $r(\pi, M_\xi)$ passes via switching points $v_1, v_2, \dots, v_n, \dots$ in order. The proposition below follows from the definition of $\mathcal{M}_0$ and states the principal property of PSP's runs on the structures under consideration.

Proposition 1. *If a run $r(\pi, M_\xi)$ of a PSP π traverses the skeleton G_π along a trajectory $v_1, v_2, \ldots, v_i, \ldots, v_j, \ldots$ such that $v_i = v_j$ then $r(\pi, M_\xi)$ loops.*

A trajectory $v_1, v_2, \ldots, v_n$ in a skeleton G_π is called

- *repetition-free* if it does not pass twice via the same vertex;
- *complete* if it is repetition-free and ends in the node $v_n = \mathbf{exit}$.

Thus, a terminating run of π traverses the skeleton G_π only along a complete trajectory, while a looping run of π may traverse G_π along either some repetition-free non-complete trajectory (in this case we say that the run loops on ordinary points of π), or some infinite trajectory (in this case we say that the run loops on switching points of π). By Proposition 1, in order the latter case to happen a run of π should traverse the skeleton along a trajectory $v_1, v_2, \ldots, v_i, v_{i+1}, \ldots$ such that $v_1, v_2, \ldots, v_i$ is a repetition-free trajectory and $v_{i+1} \in \{v_1, v_2, \ldots, v_i\}$.

The proposition below is but a restatement of the equivalence problem $\pi' \sim_{\mathcal{M}_0} \pi''$ in terms of some properties of trajectories in the skeletons of the PSPs.

Proposition 2. *PSPs π' and π'' are not equivalent on $\mathcal{M}_0$ iff there exists a pair of repetition-free trajectories $w' = v'_1, v'_2, \ldots, v'_n$ and $w'' = v''_1, v''_2, \ldots, v''_m$ in the skeletons $G_{\pi'}$ and $G_{\pi''}$ such that for some structure M_ξ the runs $r(\pi', M_\xi)$ and $r(\pi'', M_\xi)$ traverse the skeletons along the trajectories w' and w'' respectively, and meet one of the following requirements:*

R1: both trajectories are complete and $[r(\pi', M_\xi)] \neq r[(\pi'', M_\xi)]$;

R2: one of the trajectories (say, w') is complete, whereas the other (w'') is traversed by the run which loops on ordinary points;

R3: one of the trajectories (say, w') is complete, whereas the other (w'') is traversed by the run which loops on switching points.

This proposition provides a foundation for the following equivalence-checking strategy: given a pair of PSPs, guess a complete trajectory in the skeleton of one PSP and a repetition-free trajectory in the skeleton of the other; then check if there exists some structure M_ξ to satisfy one of the requirements R1–R3 above. Since the number of repetition-free trajectories in the skeletons is finite, the equivalence problem for PSPs is reduced thus to the analysis of trajectories in their skeletons. Next we will show that the latter can be carried out with the help of DFAs.

4. Using DFAs for the equivalence-checking of programs with mode switching

The problem we deal with in this section is as follows: given a pair of repetition-free trajectories w' and w'' in the skeletons of PSPs π' and π'', check if there exists a structure M_ξ which complies with at least one of the requirements R1–R3 in Proposition 2. We demonstrate that to retrieve an appropriate structure M_ξ one could construct the specific DFAs D_1, D_2 and D_3 and check their emptiness: the requirement Ri, i$= 1, 2, 3$, can be satisfied by some M_ξ iff $L(D_i) \neq \emptyset$. First we discuss the key ideas of our construction of the DFAs D_i and briefly describe how they operate. Then we present a detailed description of D_1 and explain what minor modifications should be made to convert D_1 into D_2 and D_3.

When guessing a structure M_ξ which makes it possible to traverse the skeletons of π' and π'' along the trajectories w' and w'' one may rely on the basic property of the semantics of PSPs with mode switchings: each mode switch b abandons the achieved data state s and brings data into the predefined state s_b. Thus, the traversing of each arc $\langle u, v\rangle$ begins with some fixed data state which depends only on a mode switch assigned to the point u. This enables us to try all arcs of the trajectories independently in attempt to find for each arc $\langle u, v\rangle$ a specification $Spec_{uv}$ which provides the reachability of the switching point v from the switching point u in the course of some run. The specification $Spec_{uv}$ imposes constraints on a an evaluation function ξ on some data path b, ba_1, ba_1a_2, …, where $b = B(u)$ and $a_1, a_2, \dots \in \mathcal{M}_{ord}$. We seek to define the specification so that for any structure M_ξ which satisfies $Spec_{uv}$ the run $r(\pi', M_\xi)$ (or $r(\pi'', M_\xi)$ depending on the trajectory $\langle u, v\rangle$ belongs to) when starting from the point u reaches the point v. As soon as the specifications $Spec_{uv}$ are developed for all arcs of the trajectories, we may compose the objective structure M_ξ.

When building up $Spec_{u_1v_1}$ and $Spec_{u_2v_2}$ for a pair of arcs $\langle u_1, v_1\rangle$, $\langle u_2, v_2\rangle$ it should be seen that the specifications are *consistent*, i.e. impose the same constraints on ξ on the same data states. Two cases are possible depending on the mode switches assigned to u_1 and u_2.

1. The switching points u_1 and u_2 are associated with distinct mode switches, i.e. $B(u_1) = b_1 \neq b_2 = B(u_2)$. This implies that the data paths the specifications $Spec_{u_1v_1}$ and $Spec_{u_2v_2}$ refer to are disjoint. Therefore these specifications are always consistent and may be built up independently.

2. The switching points u_1 and u_2 are associated with the same mode switch, i.e. $B(u_1) = B(u_2) = b$. Then the specifications $Spec_{u_1v_1}$ and $Spec_{u_2v_2}$ may refer to the same data path. Therefore, special care must be taken to coordinate (synchronize) the development of such specification. But as soon as the data paths these specifications deal with diverge, they will never refer to any other common data state and the synchronization may be ceased.

Thus, all specifications $Spec_{uv}$ can be built up in parallel provided that some appropriate synchronization is used to ensure their consistency. This parallel synchronized derivation of the specifications can be implemented by some DFA D. The internal states of D keep only track of the following information:

- A tuple $[v_1, \dots, v_N]$ of points in π' and π''. Every element v_i is a point in π' or π'' which is currently achieved in attempt to route a trace from the switching point u to the switching point v for some arc $\langle u, v\rangle$ in the trajectories w' and w''.
- Synchronization table H. It is used to provide the consistency of those specifications that refer to the same data states. The synchronization table may be viewed as a finite set of pairs (e_1, e_2) of arcs. Every such pair when being set into the table H indicates that the corresponding specifications $Spec_1$ and $Spec_2$ need coordination to maintain their consistency.

On each computation step D reads as input (guesses) a tuple of conditions $[\Delta_1, \dots, \Delta_N]$. These conditions give rise to new constraints that should be added to the specifications: each Δ_i is viewed as a possible value of ξ on the currently achieved data state. The synchronization table H is used to check the identity of those constraints that should be added to coordinated specifications. Then DFA D computes the updated tuple of points $[T(v_1, \Delta_1), \dots, T(v_N, \Delta_N)]$ and modifies the synchronization table H by using the transition functions T and the the binding functions B of both PSPs π' and π''. This results in transition of D to the next internal state. To complete the computation step D checks whether the new internal state satisfies some objective condition; if this is the case then D accepts. The acceptance implies the existence of a structure M_ξ such that w', w'' and M_ξ comply with one of the requirements R1–R3 (depending on the objective condition to be checked).

Now we consider a DFA D_1 intended for checking the satisfiability of the requirement R1 and describe this DFA in more detail.

Let $\pi' = \langle V, \mathbf{entry}', \mathbf{exit}, T, B\rangle$ and $\pi' = \langle U, \mathbf{entry}'', \mathbf{exit}, T, B\rangle$. To simplify the notation we will assume that $V \cap U = \{\mathbf{exit}\}$, and both PSPs have the same transition function T and the same binding function B defined on $V \cup U$.

Let $w' = v_1, v_2, \dots, v_n, \mathbf{exit}$ and $w'' = v_{n+1}, v_{n+2}, \dots, v_{n+m}, \mathbf{exit}$ be complete trajectories in the skeletons of π' and π'' respectively.

Let $H(n, m)$ denotes the set of all (unordered) pairs (i, j), $1 \le i, j \le n + m$, $i \ne j$, and $H_0(n, m) = \{(i, j) : \ (i, j) \in H(n, m), B(v_i) = B(v_j)\}$. The set $H_0(n, m)$ indicates all those pairs of switching points in w' and w'' that are associated with the same mode switch.

Then $D_1 = \langle \Sigma, Q, q_0, \mathbf{accept}, \delta\rangle$, where

- $\Sigma = \mathcal{C}^{n+m}$ is the input alphabet;
- $Q = \{\mathbf{accept}, \mathbf{reject}\} \cup (V^n \times U^m \times 2^{H(n,m)})$ is the set of internal states of D_1;
- $q_0 = \langle v_1, \dots, v_n, v_{n+1}, \dots, v_{n+m}, H_0(n, m)\rangle$ is the starting state of D_1;
- **accept** is the accepting state of D_1;
- $\delta : Q \times \Sigma \to Q$ is the (partial) state transition function.

Suppose $q = \langle u_1, \dots, u_{n+m}, H\rangle$ is any internal state of D_1 and $z = \langle \Delta_1, \dots, \Delta_{n+m}\rangle$ is any tuple of conditions. The state transition function δ is defined according to the following rules (we provide each rule with a brief explanation of its intended meaning).

1. If $\Delta_i \ne \Delta_j$ for some pair $(i, j) \in H$ then $\delta(q, z) = \mathbf{reject}$ (*Comm*: the tuple z does not comply with synchronization request; the constraints imposed on ξ are inconsistent);
2. Otherwise, consider the tuple of points $\langle u'_1, \dots, u'_{n+m}\rangle$ such that
$$u'_i = \begin{cases} T(u_i, \Delta_i), & \text{if } u_i \text{ is an ordinary point, or } q = q_0, \\ u_i, & \text{if } u_i \text{ is a switching point and } q \ne q_0, \end{cases} \quad \text{for every}$$
$1 \le i \le n + m$.

   Two cases are possible.

   Case 1: all points u'_i, $1 \le i < n$, and u'_j, $n+1 \le j < n+m$, are switching points, and at least one of the points u'_n and u'_{n+m} is either a switching point, or **exit**. Then

(a) if u'_n or u'_{n+m} is a switching point, or if there exists i, $1 \leq i < n+m$, $i \neq n$, such that u'_i is a switching point, but $u'_i \neq v_{i+1}$ then $\delta(q,z) = \textbf{reject}$ (*Comm*: this means that the automaton D_1 "went of" the trajectory when "laying off" a trace either from v_{n-1} and v_{n+m-1} to **exit**, or from v_i to v_{i+1});

(b) if $u'_i = v_{i+1}$ holds for all $1 \leq i < n+m$, $i \neq n$, and $u'_n = u'_{n+m} = \textbf{exit}$, and $(n, n+m) \in H$ then $\delta(q,z) = \textbf{reject}$ (*Comm*: this means that D_1 built a specification of a structure M_ξ such that $[r(\pi', M_\xi)] = [r(\pi'', M_\xi)]$);

(c) otherwise $\delta(q,z) = \textbf{accept}$ (*Comm*: this means that D_1 built a specification of a structure M_ξ such that both runs $r(\pi', M_\xi)$ and $r(\pi'', M_\xi)]$ traverse the skeletons along the trajectories w' and w'', but $[r(\pi', M_\xi)] \neq [r(\pi'', M_\xi)]$).

Case 2: at least one point u'_i, $1 \leq i < n+m$, $i \neq n$, is an ordinary point, or both u'_n and u'_{n+m} are ordinary points. Then $\delta(q,z) = \langle u'_1, \dots, u'_{n+m}, H' \rangle$, where

$$H' = H - \{(i,j) : \text{ both } u'_i, u'_j \text{ are ordinary points, and } B(u'_i) \neq B(u'_j)\}$$

is a new synchronization table. (*Comm*: this means that the traversing along some arcs in the trajectory is not completed yet; $B(u'_i) \neq B(u'_j)$ implies that data paths in the specifications $Spec_{v_i v_{i+1}}$ and $Spec_{v_j v_{j+1}}$ diverge and the synchronization for these specifications is of no further consequence). □

The DFA D_1 thus defined may be thought of as a one-way finite state machine operating on a tape divided into $n+m$ tracks. On the i-track it simulates some fragment r_i of the runs of PSPs π' and π''. Such fragment $r_i = (v_i^1, s_i^1, \Delta_i^1), \dots, (v_i^{k+1}, s_i^{k+1}, \Delta_i^{k+1})$ begins with a triple whose point v_i^1 is either a switching point or **entry**, and ends with a triple whose point v_i^{k+1} is either a switching point or **exit**. The synchronization tables of D_1 ensure that only those pairs of runs of π' and π'' are simulated that could be performed on the same structure. The accepting conditions allow D_1 to accomplish successfully a simulation of the runs iff the last fragments of these runs yield different results. Thus we arrive at

Proposition 3. *Let w' and w'' be complete trajectories in the skeletons of π' and π'' respectively. Let DFA D_1 be as specified above. Then $L(D_1) \neq \emptyset$ iff there exists a structure M_ξ such that the runs $r(\pi', \xi)$ and $r(\pi'', M_\xi)$ traverse the skeletons along the trajectories w' and w'', and $[r(\pi', M_\xi)] \neq [r(\pi'', M_\xi)]$.*

It is suffice to introduce only some changes in the acceptance and rejection rules (Case 1) to transform a DFA D_1 intended for checking the requirement R1 into the DFAs D_2 and D_3 that would be responsible for the requirements R2 and R3.

To construct D_2 one has to set off in π'' strongly connected components that are free from switching points. Let

$$\begin{aligned} V_{inf} \;=\; & \{v \;:\; \text{there is a cycle in } \pi'' \\ & \text{which contains no switching points and passes via } v\} \end{aligned}$$

Then D_2 operates as follows: after constructing the tuple of points $\langle u'_1, \dots, u'_{n+m} \rangle$ (see item 2 in the description of D_1), it

- rejects (rule 2(b)) if $u'_i = v_{i+1}$ holds for all $1 \leq i < n+m$, $i \neq n$, and $u'_n = \textbf{exit}$, and u'_{n+m} is either a switching point or **exit**,
- accepts (rule 2(c)) $u'_i = v_{i+1}$ holds for all $1 \leq i < n+m$, $i \neq n$, and $u'_n = \textbf{exit}$, and $u'_{n+m} \in V_{inf}$.

It is worthy of notice that when at some computation step D_2 accepts after constructing the tuple $\langle v_2, \dots, v_n, \textbf{exit}, v_{n+2}, \dots, v_{n+m}, u \rangle$ such that $u \in V_{inf}$, this should be interpreted as π'' is presented with an unbound possibility to continue an infinite (looping) run along some cycle on ordinary points.

Similarly to D_2, after constructing the tuple of points $\langle u'_1, \dots, u'_{n+m} \rangle$ D_3 rejects (rule 2(b)) if u'_{n+m} is either **exit**, or a switching point other than $v_{n+1}, \dots, v_{n+m}$, and accepts (rule 2(c)) if $u'_n = \textbf{exit}$ and $u'_{n+m} \in \{v_{n+1}, \dots, v_{n+m}\}$.

Proposition 4. *Let w' be a complete trajectory and w'' be a repetition-free trajectory in the skeletons of π' and π'' respectively. Let DFA D_2 and D_3 be as specified above. Then*

1. *$L(D_2) \neq \emptyset$ iff there exists a structure M_ξ such that the runs $r(\pi', M_\xi)$ and $r(\pi'', M_\xi)$ traverse the skeletons along the trajectories w' and w'', and $r(\pi', M_\xi)$ terminates, whereas $r(\pi'', M_\xi)$ loops on ordinary points of π''.*

2. *$L(D_3) \neq \emptyset$ iff there exists a structure M_ξ such that the runs $r(\pi', M_\xi)$ and $r(\pi'', M_\xi)$ traverse the skeletons along the trajectories w' and w'', and $r(\pi', M_\xi)$ terminates, whereas $r(\pi'', M_\xi)$ loops on switching points of π''.*

Theorem 1. *The equivalence problem $\pi' \sim_{\mathcal{M}_0} \pi''$ for PSPs with mode switching is decidable in polynomial space.*

Proof. Due to Savitch's theorem [30] we can convert any nondeterministic polynomial space algorithm into deterministic one. Therefore it will suffice to design a nondeterministic polynomial space procedure for checking non-equivalence of PSPs with mode switching. It is as follows. Given a pair of PSPs π' and π'' it builds their skeletons $G_{\pi'}$, $G_{\pi''}$ and guesses a complete trajectory w' in one of the skeletons and a repetition-free trajectory in the other. Then it checks the emptiness of DFAs D_1, D_2 and D_3 corresponding to π', π'' and the selected trajectories. Propositions 2–4 guarantee that $\pi' \not\sim_{\mathcal{M}_0} \pi''$ iff for some pair of trajectories one of the DFAs D_i, $i = 1, 2, 3$, is non-empty. As may be seen from the descriptions of DFAs D_i, these automata has $O(2^{poly(n)})$ states, where $n = |\pi'| + |\pi''|$. Hence, their non-emptiness may be certified within a polynomial space. $\square$

5. Complexity issues

The complexity of decision procedure above can not be improved to a large extent.

Theorem 2. *The equivalence problem for PSPs with mode switching is PSPACE-complete.*

Proof. We demonstrate how to build in linear time, given a family of n DFAs F_i, $1 \le i \le n$, of size $|F_i| = n$, a PSP π of size $|\pi| = n^2$ such that the intersection $\bigcap_{i=1}^{n} L(F_i) = \emptyset$ iff π has no terminating runs, i.e. π_0 is equivalent to the empty PSP.

Without loss of generality, we may assume that each DFA F_i operates on binary input alphabet $\{0, 1\}$, i.e. $F_i = \langle Q_i, \delta_i, q_i^0, Q_i' \rangle$, where Q_i is a finite state of states, $\delta_i : Q \times \{0, 1\} \to Q$ is a (total) transition function, $q_i^0 \in Q_i$ is the starting state, and $Q_i' \subseteq Q_i$ is the set of final states. Let $\mathcal{A}_{mode} = \{b\}$, $\mathcal{A}_{ord} = \{a\}$, and $\mathcal{P} = \{p_1, p_2\}$. For every DFA F_i we build the PSP $\pi_i = \langle Q_i \cup \{\mathbf{dead}, \mathbf{entry}_i, \mathbf{exit}_i\}, \mathbf{entry}_i, \mathbf{exit}_i, T_i, B_i \rangle$ whose transition function T_i is defined as follows for every $q \in Q$ and $x \in \{0, 1\}$:

1. $T_i(\mathbf{entry}_i, \langle x, 0 \rangle) = q_i^0$, and $T_i(\mathbf{entry}, \langle x, 1 \rangle) = \begin{cases} \mathbf{exit}_i, \text{ if } q_i^0 \in Q_i', \\ \mathbf{dead}, \text{ otherwise;} \end{cases}$
2. $T_i(q, \langle x, 0 \rangle) = \delta_i(q, x)$, and $T_i(q, \langle x, 1 \rangle) = \begin{cases} \mathbf{exit}_i, \text{ if } \delta_i(q, x) \in Q_i', \\ \mathbf{dead}, \text{ otherwise;} \end{cases}$
3. $T_i(\mathbf{dead}, \Delta) = \mathbf{dead}$.

It is easy to see from the construction of π_i that a binary string $w = x_1 x_2 \dots x_k$, $k \ge 0$, is accepted by F_i iff $r(\pi_i, M_{\xi_w})$ terminates on a structure M_{ξ_w} such that $\xi_w(b(a)^k) = \langle x_k, 1 \rangle$ and $\xi_w(b(a)^i) = \langle x_i, 0 \rangle$ for all $1 \le i < k$.

The PSP π_0 is obtained from the family of PSPs $\pi_1, \dots, \pi_n$ by identifying each point $\mathbf{exit}_i$, $1 \le i \le n-1$, with the point $\mathbf{entry}_{i+1}$. It is easy to verify that a binary string w is accepted by every DFA F_i iff $\downarrow r(\pi_i, M_{\xi_w})$ hold for all PSPs π_i iff $\downarrow r(\pi_0, M_{\xi_w})$.

Finally, consider any PSP π_{empty} such that the terminal point **exit** is unreachable from the initial point entry in π_{empty} (PSPs of this kind are called *empty* PSPs). Clearly, an empty PSP has no terminating runs, and therefore $\bigcap_{i=1}^{n} L(F_i) = \emptyset$ iff $\pi_0 \sim_{\mathcal{M}_0} \pi_{empty}$. Thus the Intersection Emptiness Problem for DFAs, which is known to be PSPACE-complete, is reducible in linear time to the equivalence problem for PSPs with mode switching. $\square$

6. Conclusion and Future Work

We demonstrated that the equivalence problem for programs with mode switching is *PSPACE*-complete. Actually, the semantics of these programs makes it inevitable to face the Intersection Emptiness Problem for DFAs. This gives us a decision procedure which is confined to the manipulations with finite automata but at the expense of fairly large complexity. One could find some analogy between our construction of DFAs D_i and the "parallel stacking" technique introduced in [31]. The results obtained pose a number of open problems and we discuss some of them.

One of the most simple class of programs with undecidable equivalence problem was studied in [22]. The programs from this class are composed of four basic statements $\mathcal{A} = \{a_1, a_2, b_1, b_2\}$. In the framework of PSPs the semantics of such programs is specified by the set of structures $\mathcal{M}_2 = \{M' = \langle \mathbb{N}^2, (0, 0), R', \xi \rangle\}$, where $\mathbb{N} = \{0, 1, 2, \dots\}$, $R'(a_1, (n, m)) = (n+1, m)$, $R'(a_2, (n, m)) = (n, m+1)$, $R'(b_1, (n, m)) = (0, m)$, $R'(b_2, (n, m)) = (n, 0)$. Notice that constant assignment statements b_1, b_2 are but a sort of a (separate) mode switches. In [7, 22] it was proved that two-head finite state automata can be simulated by the PSPs in the model $\mathcal{M}_2$ and this brings the equivalence problem for $\mathcal{M}_2$ into undecidability.

In essence, $\mathcal{M}_2$ captures two main features of the real program semantics: the effect of commutativity of some statements (a_1 and a_2) and the effect of constant assignments (b_1 and b_2). By divorcing these effects from each other we arrive at the model of programs with commutative statements $\mathcal{M}_1$ and the model of programs with mode switching $\mathcal{M}_0$. In [26, 32] it was shown that the equivalence problem for $\mathcal{M}_1$ is decidable within a time $O(n^2 \log n)$. In our paper we proved that the same problem for $\mathcal{M}_0$ is *PSPACE*-complete. In order to make a boarder between decidable and undecidable cases more precise we wonder: what is the complexity of the equivalence problem for the model $\mathcal{M}_{01}$ which may be placed between $\mathcal{M}_2$ and both $\mathcal{M}_0$ and $\mathcal{M}_1$? In contrast to $\mathcal{M}_2$, which involves two separate mode switches b_1 and b_2, we restrict ourselves in $\mathcal{M}_{01}$ only with a joint mode switch b_0 such that $R'(b_1,(n,m)) = (0,0)$. Clearly, $\mathcal{M}_{01}$ is less expressive than $\mathcal{M}_2$. The equivalence problem for $\mathcal{M}_{01}$ was studied in [6], but its complexity is still unknown.

The equivalence-checking algorithms worked out in the framework of this model of programs may be used as the basis for the deobfuscation tools aimed at detecting metamorphic viruses. This could stimulate the development of more practical and efficient equivalence-checking procedures for $\mathcal{M}_0$ than that from Theorem 1. Thus, for example, we wonder, if it is possible to check within a polynomial time the equivalence $\pi' \sim_{\mathcal{M}_0} \pi'$ providing that every mode switch occurs in π', π'' at most k times and k is fixed.

References

[1] M. Christodorescu, S. Jha. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th USENIX Security Symposium (Security'03)*, 2003, p. 169–186.

[2] M. Christodorescu, S. Jha, S.A. Seshia, D. Song, R. E. Bryant. Semantics-aware malware detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (Oakland 2005)*, 2005, (to be published).

[3] Collberg C., Thomborson C., Low D., Manufacturing cheap, resilient and stealthy opaque constructs. *Symposium on Principles of Programming Languages*, 1998, p.184–196.

[4] A.P.Ershov, Theory of program schemata. In *Proc. of IFIP Congress 71*, Ljubljana, 1971, p.93-124.

[5] E.P. Friedman. Equivalence problem for deterministic context-free languages and monadic recursive schemes. *J. Comput. and Syst. Sci.*, **14**, N 3, 1977, p. 344–359.

[6] A.B. Godlevsky. Some special cases of the termination and equivalence problems for automata, 1973, N 4, p. 90-–98 (in Russian)

[7] A.B. Godlevsky. On the one case of special problem of functional equivalence for discrete transducers. Cybernetics, 1974, N 3, p. 32–35 (in Russian)

[8] J.E. Hopcroft, R.M. Karp. A linear algorithm for testing equivalence of finite automata, Technical Report TR 71–114, Cornell University, Computer Science Dep., 1971.

[9] Ianov Iu I., On the equivalence and transformation of program schemes *Communications of the ACM*, 1:10 (1958), 8–12.

[10] G. Karakostas, R.J. Lipton, A. Viglas. On the complexity of intersecting finite state automata and NL versus NP. *Theoretical Computer Science*, **302**, 2003, p. 257–274.

[11] A. Kornai. Vectorized finite state automata. In: *Proceedings of the W1 workshop of the 12th European Conference on Artificial Intelligence*, Budapest, 1996, p. 36–41.

[12] D. Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundation of Computer Science*, IEEE, 1977, p. 254-266.

[13] A.A.Letichevsky. On the equivalence of automata with final states on the free monoids having right zero. *Reports of the Soviet Academy of Science*, **182**, 1968, N 5 (in Russian).

[14] A.A.Letichevsky. Functional equivalence of discrete transducers. *Cybernetics*, 1970, N 2, p. 14–28.

[15] A.A.Letichevsky, Functional equivalence of finite transducers. III, Cybernetics, 1972, N 1, 1—4 (in Russian)

[16] A.A.Letichevsky, On the equivalence of automata over semigroup, Theoretic Cybernetics, 6, 1970, 3—71 (in Russian).

[17] A.A.Letichevsky, Equivalence and optimization of programs. In Programming theory, Part 1, Novosibirsk, 1973, 166-–180 (in Russian).

[18] A.A.Letichevsky, L.B.Smikun, On a class of groups with solvable problem of automata equivalence, Sov. Math. Dokl., 17, 1976, N 2, 341—344.

[19] L.P. Lisovik, Methalinear schemes with constant assignment, Programming and Computer software, N 2, 1985, 29-–38.

[20] L.P. Lisovik. Hard sets method and semilinear reservoir method with applications. *Lecture Notes in Computer Science*, **1099**, 1996, p. 219–231.

[21] D.C.Luckham, D.M.Park, M.S.Paterson, On formalized computer programs, *J. Comput. and Syst. Sci.*, **4**, 1970, N 3, p.220–249.

[22] G.N.Petrosyan, On one basis of statements and predicates for which the emptiness problem is undecidable. *Cybernetics*, 1974, N 5, p.23–28 (in Russian).

[23] R.I.Podlovchenko, The hierarchy of program models, Programming and Computer Software, 1981, N 2, 3—14.

[24] R.I.Podlovchenko, Semigroup program models, Programming and Computer Software, 1981, N 4, 3-–13.

[25] R.I.Podlovchenko, On the decidability of the equivalence problem on a class of program schemata having monotonic and partially commutative statements, Programming and Computer Software, 1990, N 5, 3—12.

[26] R.I.Podlovchenko, V.A.Zakharov, On the polynomial-time algorithm deciding the commutative equivalence of program schemata, *Reports of the Soviet Academy of Science*, **362**, 1998, N 6 (in Russian).

[27] R.I.Podlovchenko, On program schemes with commuting and monotonic statements, Programming and Computer software, N 5, 2003, 46–54.

[28] K. Rohloff, S. Lafortune. On the computational complexity of the verification of modular discrete-event systems. In *Proceedings of IEEE Conference on Decision and Control*, 2002, Las Vegas, NV, Dec., 2002.

[29] J.D.Rutledge, On Ianov's program schemata, *Journal of the ACM*, **11**, 1964, p.1–9.

[30] W.J. Savitch. Relationships between nondeterministic and deterministic space complexities. *Journal of Computer and System Science*, **4**, N 2, 1970, p. 177–192.

[31] L.G.Valiant, The equivalence problem for deterministic finite-turn pushdown automata, *Information and Control*, **25**, 1974, p.123–133.

[32] V.A. Zakharov, An efficient and unified approach to the decidability of equivalence of propositional program schemes. *Lecture Notes in Computer Science*, **1443**, 1998, p. 247–258.

[33] V.A. Zakharov, On the decidability of the equivalence problem for monadic recursive programs, Theoretical Informatics and applications, 34, N 2, 2000, 157-–171.

[34] V.A. Zakharov, The equivalence problem for computational models: decidable and undecidable cases. *Lecture Notes in Computer Science*, **2055**, 2001, 133–153.

[35] V.A. Zakharov, I.M. Zakharyaschev, An equivalence-checking algorithm for polysemantic models of sequential programs, Proceedings of the International Workshop on Program Understanding (14-16 July, Altai Mountains, Russia), 2003, 59-–70.

[36] V.A. Zakharov, I.M. Zakharyaschev, On the equivalence checking problem for a model of programs related with muti-tape automata, Lecture Notes in Computer Science, 3317, 2005, 293-–305