The Equivalence Problem for Computational Models: Decidable and Undecidable Cases

Vladimir A. Zakharov

Faculty of Computational Mathematics and Cybernetics,
Moscow State University, Moscow, RU-119899, Russia
zakh@cs.msu.su

Abstract. This paper presents a survey of fundamental concepts and main results in studying the equivalence problem for computer programs. We introduce some of the most-used models of computer programs, give a brief overview of the attempts to refine the boarder between decidable and undecidable cases of the equivalence problem for these models, and discuss the techniques for proving the decidability of the equivalence problem.

Informally, the equivalence problem is to find out whether two given programs have the same behavior. By picking various formal definitions of the terms "program" and "behavior" we get numerous variants of this problem. The study of the equivalence problem for computational models is always of basic interest in computer science beginning with the original works on the development of the formal concept of computer program [24,45,34,9,11]. Tackling the equivalence problem we comprehend to what extent the very nature of computations can be conceived by means of formal methods and how much specific changes in the structure of a computer program affect its behavior. The understanding of relationships between the syntactic and semantic components of programs is very important for specification, verification and optimization of programs, partial computations, reusing of programs, etc. That is why this problem significantly influences both the theory and the practice of computer science and software engineering.

The decidability of the equivalence problem essentially depends on the expressive power of a computational model and the exact meaning of the term "the same behavior". Two fundamental results of 50th governed the advancement in studies of the equivalence problem for computer programs.

When programs under consideration are deterministic, it is quite reasonable to assume that two programs have the same behaviour if they compute the same function, i.e. for every valid input they output identical results (if any). A mapping which associates each program π with the recursive function f_π computed by π is called *enumeration* of recursive functions. If a programming system PS has an effective interpreter that can simulate every program by presenting its description as a part of the input, then enumeration is called *computable*. If, moreover, a programming system PS is such that any other programming system PS' corresponding to computable enumeration can be effectively translated

M. Margenstern and Y. Rogozhin (Eds.): MCU 2001, LNCS 2055, pp. 133–152, 2001.
© Springer-Verlag Berlin Heidelberg 2001

into PS then the enumeration specified by PS is called *acceptable* (see [47,54]). In 1953 H.G.Rice [46] proved that if a programming system PS corresponds to an acceptable enumeration of recursive functions, then any non-trivial property of computer programs which refers only to functions computed by programs is undecidable. This implies the undecidability of the functional equivalence of computer programs for every natural universal programming system.

On the other hand, in 1956 A.A.Lyapunov and Y.I.Yanov [24] introduced a propositional model of sequential programs (Yanov schemata) and set up the equivalence problem for this model. Yanov scheme may be thought of (see [11]) as a finite transition system whose nodes are labelled with statements $A_1, \ldots, A_n$ and basic propositions $p_1, \ldots, p_m$. Each statement A is associated with a set of basic proposition $Sh(A)$. Nodes marked with statements are called transformers; each transformer has a single outgoing arc. Nodes marked with propositions are called recognizor; each recognizor has two outgoing arcs labelled with $\top$ and $\bot$. Transformers are understood as abstractions of program statements, whereas basic propositions stand for logic conditions. Two specific nodes — **entry** and **exit** are distinguished. A run of a Yanov scheme π starts from the **entry** given some initial evaluation δ^0 of basic propositions. When a run passes via a transformer labelled with A, the statement A is executed by changing arbitrary the values of propositions from $Sh(A)$. When a run passes via a recognizor labelled with basic proposition p, it checks the current value of p and follows the arc marked with this value. A run terminates as soon as it reaches the **exit**. A determinant of a run r is a sequence of pairs $(\delta^0, A^1), (\delta^1, A^2), \ldots, (\delta^{n-1}, A^n), \ldots$, where $A^1, A^2, \ldots, A^n, \ldots$ is a sequence of statements executed during this run, and $\delta^i,\ i \geq 1$ is the evaluation of basic proposition after executing A^i. A determinant $det(\pi)$ of a Yanov scheme π is said to be a set of determinants of all possible terminated runs of π. Two schemata π_1 and π_2 are called equivalent if $det(\pi_1) = det(\pi_2)$. In 1957 Yanov [58] found a complete axiomatization (equational calculus) for the equivalence relation thus defined and proved that the equivalence problem for this computational model is decidable.

Both results, Rice's theorem and the decidability of equivalence problem for Yanov schemata, were the pioneering steps from the opposite sides towards the boarder between decidability and undecidability of the equivalence problem for computational models. Since that time the refinement of this boarder has become the topic for a large body of research. In succeeding years a wide variety of devices defining computations, languages and translations were introduced [45,38,9,32,35]. Every time when an all-new model of computation made its appearance, the equivalence problem for this model challenged researchers. Considerable effort devoted to this problem yielded an amazing amount of results and techniques that substantially extend the capability of formal methods in computer science.

Now it is hardly possible to cover in a single survey all main results and approaches to the equivalence problem for the most important models of computation. In this paper we consider in some details the current state of art in studying the equivalence problem for two computational models — propositional sequen-

tial programs and first-order sequential programs. An overview of the techniques for proving the decidability of language equivalence and relational equivalence for devices defining languages and translations (relations) can be found in [8]. Since the publishing of this paper, some new results has been obtained [13,51] that substantially revised our understanding of decidability/undecidability frontier for the equivalence problem. A broad spectrum of results on the bisimilation equivalence problem for automata and process algebras is discussed in much details in [14].

1 Propositional Sequential Programs

In this section we introduce the concept of a propositional sequential program (PSP), its syntax, and semantics and discuss the principal results on the equivalence problem for this model of computations.

1.1 Syntax of PSP

Fix two finite alphabets $\mathcal{A} = \{a^1, \ldots, a^N\}$ and $\mathcal{P} = \{p_1, \ldots, p_M\}$.

The elements of $\mathcal{A}$ are called *basic actions*. Intuitively, the basic actions stand for the elementary program statements such as assignment statements and procedure calls. A finite sequence of basic actions is called a *term*. The set of all terms is denoted by $\mathcal{A}^*$. We write λ for the empty term, $|h|$ for the length of a term h, and hg for the concatenation of terms h and g.

The symbols of $\mathcal{P}$ are called *basic propositions*. We assume that basic propositions denote the primitive relations on program data. Each basic proposition may be evaluated either by $\perp$ (falsehood) or by $\top$ (true). A binary M-tuple $\delta = \langle d_1, \ldots, d_M \rangle$ of truth-values of all basic propositions is called a *condition*. We write $\mathcal{C}$ for the set of all conditions.

A *propositional sequential program* (PsP) over alphabets $\mathcal{A}$, $\mathcal{P}$ is a tuple $\pi = \langle V, \mathbf{entry}, \mathbf{exit}, \mathbf{loop}, B, T \rangle$, where

- V is a finite set of *program nodes*;
- **entry** is an *initial node*, $\mathbf{entry} \in V$,
- **exit** is a *terminal node*, $\mathbf{exit} \in V$,
- **loop** is a *dead node*, $\mathbf{loop} \in V$,
- $B : V \to \mathcal{A}^*$ is a *binding function*, associating every node with some term so that $B(\mathbf{entry}) = B(\mathbf{exit}) = B(\mathbf{loop}) = \lambda$;
- $T : (V - \{\mathbf{exit}\}) \times \mathcal{C} \to V$ is a total *transition function* such that $T(\mathbf{loop}, \delta) = \mathbf{loop}$ for every δ from $\mathcal{C}$.

In essence, a PSP may be thought of as a finite-state labelled transition system representing the control structure of a sequential program. We extend a binding function B to the sequences of program nodes by assuming $B(v_1, v_2, \ldots, v_k) = B(v_1)B(v_2)\ldots B(v_k)$. By the size $|\pi|$ of a given PSP π we mean the number $|V|$ of its internal nodes.

1.2 Semantics of PSP

The semantics of PSP is defined by means of dynamic Kripke structures (frames and models) (see [12]).

A *dynamic deterministic frame* (or simply a *frame*) over alphabet $\mathcal{A}$ is a triple $\mathcal{F} = \langle S, s_0, R\rangle$, where

- S is a non-empty set of *data states*,
- s_0 is an *initial state*, $s_0 \in S$,
- $R: \ S \times \mathcal{A} \ \rightarrow \ S$ is an *updating function*.

$R(s, a)$ is interpreted as a result of the application of an action a to a data state s. It is assumed that each basic action $a \in \mathcal{A}$ transforms deterministicaly one state into another; therefore, the dynamic frames under considerations are both functional and serial relative to every action $a \in \mathcal{A}$.

An updating function R can be naturally extended to the set $\mathcal{A}^*$ as follows

$$R^*(s, \lambda) = s, \ R^*(s, ha) = R(R^*(s, h), a).$$

We say that a state s'' is *reachable* from a state s' ($s' \preceq_{\mathcal{F}} s''$ in symbols) if $s'' = R^*(s', h)$ for some $h \in \mathcal{A}^*$. Denote by $[h]_{\mathcal{F}}$ the state $s = R^*(s_0, h)$ which is reachable from the initial state by means of $\mathcal{A}$-sequence h. As usual, the subscript $\mathcal{F}$ will be omitted when the frame is understood. We will deal only with data states that are reachable from the initial state. Therefore, we may assume, without loss of generality, that every state $s \in S$ is reachable from the initial state s_0, i.e. $S = \{[h]: \ h \in \mathcal{A}^*\}$.

A frame $\mathcal{F}_s = \langle S', s, R'\rangle$ is called a *subframe* of a frame $\mathcal{F} = \langle S, s_0, R\rangle$ generated by a state $s \in S$ if $S' = \{R^*(s, h): \ h \in \mathcal{A}^*\}$ and R' is a restriction of R to S'. We say that a frame $\mathcal{F}$ is

- *semigroup* if $\mathcal{F}$ can be mapped homomorphically onto every subframe $\mathcal{F}_s$,
- *homogeneous* if $\mathcal{F}$ is isomorphic to every subframe $\mathcal{F}_s$,
- *ordered* if $\preceq$ is a partial order on the set of data states S,
- *universal* if $[h] = [g]$ implies $h = g$ for every pair h, g of $\mathcal{A}$-sequences.

Taking the initial state $s_0 = [\lambda]$ for the unit, one may regard a semigroup frame $\mathcal{F}$ as a finitely generated monoid $\mathcal{F} = \langle S, *\rangle$ such that $[h] * [g] = [hg]$. Clearly, the universal frame $\mathcal{U}$ corresponds to the free monoid generated by $\mathcal{A}$. Consequently, an ordered semigroup frame is associated with a monoid whose unit $[\lambda]$ is irresolvable, i.e. $[\lambda] = [gh]$ implies $g = h = \lambda$, whereas a semigroup corresponding to a homogeneous frame is a left-contracted monoid, i.e. $[gh'] = [gh'']$ implies $[h'] = [h'']$.

A *dynamic deterministic model* (or simply a *model*) over alphabets $\mathcal{A}$, $\mathcal{P}$ is a pair $M = \langle \mathcal{F}, \xi\rangle$ such that

- $\mathcal{F} = \langle S, s_0, R\rangle$ is a frame over $\mathcal{A}$,
- $\xi: \ S \ \rightarrow \ \mathcal{C}$ is a *valuation function*, indicating truth-values of basic propositions at every data state.

Let $\pi = \langle V, \mathbf{entry}, \mathbf{exit}, \mathbf{loop}, B, T\rangle$ be some PSP and $M = \langle \mathcal{F}, \xi\rangle$ be a model based on the frame $\mathcal{F} = \langle S, s_0, R\rangle$. A finite or infinite sequence of pairs

$$r = (v_0, \delta_0), (v_1, \delta_1), \ldots, (v_m, \delta_m), \ldots \tag{1}$$

where $v_i \in V,\ \delta_i \in \mathcal{C},\ i \geq 0$, is called a *run* of π on M if (1) meets the following requirements:

1. $v_0 = \mathbf{entry}$;
2. $v_{i+1} = T(v_i, \delta_i)$ for every $i,\ i \geq 0$;
3. $\delta_i = \xi([B(v_0, v_1, \ldots, v_i)])$ for every $i,\ i \geq 0$;
4. (1) is a finite sequence iff $T(v_m, \delta_m) = \mathbf{exit}$ for some $m \geq 0$.

When a run r ends with a pair (v_m, δ_m) as its last element, we say that r *terminates*, having the data state $s_m = [B(v_0, v_1, \ldots, v_m)]$ as the *result*. Otherwise, when r is an infinite sequence we say that it *loops* and has no result. Since all PSPs and frames under consideration are deterministic, every program π has a unique run $\pi(M)$ on a given model M. We write $[\pi(M)]$ for the result of $\pi(M)$, assuming $[\pi(M)]$ is undefined when the run loops.

1.3 The Equivalence Problem for PSPs

Let π' and π'' be some PSPs, M a model, $\mathcal{M}$ be a set of models, and $\mathcal{F}$ a frame. Then π' and π'' are called

- *equivalent on* M ($\pi' \sim_M \pi''$ in symbols) if $[\pi'(M)] = [\pi''(M)]$, i.e. either both runs $r(\pi', M)$ and $r(\pi'', M)$ loop or both of them terminate with the same data state as their results,
- *equivalent on* $\mathcal{M}$ ($\pi' \sim_{\mathcal{M}} \pi''$ in symbols) if $\pi' \sim_M \pi''$ holds for every $M \in \mathcal{M}$,
- *equivalent on* $\mathcal{F}$ ($\pi' \sim_{\mathcal{F}} \pi''$ in symbols) if π' and π'' are equivalent on every model $M = \langle \mathcal{F}, \xi\rangle$ based on $\mathcal{F}$.

For a given set of models $\mathcal{M}$ (a frame $\mathcal{F}$) the *equivalence problem* w.r.t. $\mathcal{M}$ ($\mathcal{F}$) is to check for an arbitrary pair π_1, π_2 of PSPs whether $\pi' \sim_{\mathcal{M}} \pi''$ ($\pi' \sim_{\mathcal{F}} \pi''$) holds.

1.4 Decidable and Undecidable Cases of the Equivalence Problem for PSPs

Since we are interested in the algorithmic aspects of the equivalence problem, it is assumed that frames and models we deal with are effectively characterized in logic or algebraic terms (say, by means of dynamic logic formulae or semigroup identities). It is also clear that the undecidability of the identity problem "$[h'] = [h'']$?" for a frame $\mathcal{F}$ implies the undecidability of the equivalence problem for PSP on $\mathcal{F}$. Therefore, we direct our attention only to those semantics (frames and sets of models) that have rather simple and natural arrangement.

Yanov schemata. The computational model of Yanov schemata [24,58,11] was the first attempt to provide a precise mathematical basis for the common activities involved in reasoning about computer program. It is also the first case of computer program formalization whose equivalence problem was proved to be decidable. In the framework of our concept Yanov schemata may be thought of as PSPs whose semantics is based on the set of dynamic models characterized by PDL axioms

$$[a_i]p_j \equiv p_j$$

for every basic action a_i and basic proposition p_j which is not in the shift $Sh(a_i)$. In fact, to check the equivalence of Yanov schemata it suffices to consider their behaviour only on the universal frame $\mathcal{U}$ which is based on the free monoid generated by $\mathcal{A}$. This demonstrates close relationships between Yanov schemata and finite automata [45]. It was revealed in [48] and opened up fresh opportunities for applications of formal methods in computer program analysis.

Ordered frames. When a set of states S of a frame $\mathcal{F}$ is partially ordered w.r.t. $\preceq_{\mathcal{F}}$, this means that no actions in $\mathcal{A}$ are invertible and, hence, PSP's computations always progress. First results on the equivalence problem for PSPs on ordered frames were obtained by Letichevsky.

Theorem 1 ([17]). *Suppose $\mathcal{F}$ is a homogeneous ordered frame such that the identity problem "$[g] = [h]$?" is decidable in time $\psi(n)$. Then the equivalence problem "$\pi_1 \sim_{\mathcal{F}} \pi_2$?" is decidable in time $O(2^n\psi(n))$.*

In many cases the complexity of decision procedures can be improved by taking into account some specific properties of the frames.

Let $\mathcal{F} = \langle S, s_0, R\rangle$ be a semigroup frame. Considering it as a monoid, we write $\mathcal{F} \times \mathcal{F}$ for the direct product of the monoids. Suppose W is a finitely generated monoid, U is a submonoid of W, and w^+, w^* are two distinguished elements in W. Denote by $\circ$ and e a binary operation on W and the unit of W respectively. We say that the quadruple $K = \langle W, U, w^+, w^*\rangle$ is a k_0-*criteria system* for the semigroup frame $\mathcal{F}$, where k_0 is some positive integer, if K and $\mathcal{F}$ meet the following conditions:

1. there exists a homomorphism φ of $\mathcal{F} \times \mathcal{F}$ in U such that

$$[h] = [g] \Leftrightarrow w^+ \circ \varphi(\langle [g], [h]\rangle) \circ w^* = e$$

   holds for every pair g, h in $\mathcal{A}^*$,
2. for every element w in the coset $U \circ w^*$ the equation $y \circ w = e$ has at most k_0 pairwise different solutions y in the coset $w^+ \circ U$.

Theorem 2 ([59]). *Suppose an ordered semigroup frame $\mathcal{F}$ satisfies the following requirements:*

1. *the reachability problem "$[g] \preceq_{\mathcal{F}} [h]$?" is decidable in time $t_1(m)$, where $m = \max(|g|, |h|)$;*

2. *$\mathcal{F}$ has a k_0-criteria system $K = \langle W, U, w^+, w^* \rangle$ such that the identity problem "$w_1 = w_2$?" in W is decidable in time $t_2(m)$, where $m = \max(|w_1|, |w_2|)$.*

Then the equivalence problem "$\pi_1 \sim_{\mathcal{M}} \pi_2$?" is decidable in time $O(n^4(t_1(n^2) + t_2(cn^2) + \log n))$, where $n = \max(|\pi_1|, |\pi_2|)$.

By finding appropriate criterial systems one may construct uniformly polynomial-time decision procedures for the variety of well-known frames. Consider, for example, a partially commutative monoid $\mathcal{F}_I$ induced by some independency relation I [33] on the set of basic actions,

Corollary 1 ([43]). *The equivalence problem for PSP on the partially commutative frame $\mathcal{F}_I$ is decidable in time $O(n^2 \log n)$*

Some results on the equivalence problem for PSPs on ordered frames which are not semigroup were obtained in [61].

Let $\mathcal{F}$ be an arbitrary frame. For every pair of states $\langle s', s'' \rangle$ denote by $I(s_1, s_2)$ the set of pairs of terms $\langle h_1, h_2 \rangle$, such that $R^*(s_1, h_1) = R^*(s_2, h_2)$. Two pairs $\langle s'_1, s'_2 \rangle$ and $\langle s''_1, s''_2 \rangle$ are said to be *similar* ($\langle s'_1, s'_2 \rangle \approx \langle s''_1, s''_2 \rangle$ in symbols) if $I(s'_1, s'_2) = I(s''_1, s''_2)$ holds. We denote by $\langle s'_1, s'_2 \rangle / \approx$ the similarity class containing a pair $\langle s'_1, s'_2 \rangle$.

Theorem 3 ([61]). *Suppose that an ordered frame $\mathcal{F}$ satisfies the following requirements:*

1. *the reachability problem "$[h_1] \preceq_{\mathcal{F}} [h_2]$?" on $\mathcal{F}$ is decidable in time $\Phi(\max(|h_1|, |h_2|))$;*
2. *the similarity problem "$\langle [h'_1], [h'_2] \rangle \approx_{\mathcal{F}} \langle [h''_1], [h''_2] \rangle$?" is decidable in time $\Psi(\max(|h'_1|, |h'_2|, |h''_1|, |h''_2|))$;*
3. *for any terms h_1, h_2 the set $\{\langle [H_1], [H_2] \rangle / \approx_{\mathcal{F}} : [H_1 h_1] = [H_2 h_2]\}$ contains at most $\Theta(\max(|h_1|, |h_2|))$ similarity classes,*

where Φ, Ψ and Θ are monotonic recursive functions. Then the equivalence problem $\pi' \sim_{\mathcal{F}} \pi''$? is decidable in time $O((\Theta^2(n)\Phi(n^4\Theta(n))\Psi(n^4\Theta(n))n^6 \log n)$, where $n = \max(|\pi'|, |\pi'|)$.

Group frames. If $\preceq_{\mathcal{F}}$ is not a partial order on the set of states S of a semigroup frame $\mathcal{F}$, then the monoid $\mathcal{F}$ contains invertible elements. These elements form a subgroup in $\mathcal{F}$, and the decidability of the equivalence problem for PSPs on $\mathcal{F}$ depends essentially on this group.

Theorem 4 ([26,52]). *Suppose a homogeneous frame $\mathcal{F}$ is associated with a right-contracted monoid whose identity problem "$[g] = [h]$?" is decidable. Then the equivalence problem w.r.t. $\mathcal{F}$ is decidable iff it is decidable w.r.t. $\mathcal{F}'$, where $\mathcal{F}'$ is the maximal subgroup of $\mathcal{F}$.*

A uniform criterion for selecting groups with decidable problem of program equivalence was presented in [26,30].

Let $\mathcal{F}'\langle S, s_0, R\rangle$ be a frame corresponding to some group, $a_1, a_2, \ldots, a_k$ be a finite sequence of actions, and g, h be some terms. Then the sequence of states $[g], [ga_1], [ga_1, a_2], \ldots, [ga_1a_2 \ldots a_n]$ is called a *trajectory* in $\mathcal{F}'$. If $[ga_1a_2 \ldots a_n] = h$ then we say that this trajectory joins g to h. By the *distance* $\rho(g, h)$ between states g and h we shall mean the length of the shortest trajectory joining g to h. A set of states H, $H \subset S$, is called *complete* if for any h, $h \in H$, there exists a trajectory joining s_0 to h, all of whose elements belong to H, i.e. H is a simply connected subset of S containing the initial state s_0.

Let H be some complete set of states. Consider the relation $g <_H h$ on S which is true iff $g \neq h$ and each trajectory joining g to h passes through some state $[gh']$ such that $[h'] \in H$.

A frame $\mathcal{F}' = \langle S, s_0, R\rangle$ is called *β-frame* [30] if it contains a finite complete set H satisfying the following requirement:

> there exists a positive integer N such that, for any states g, h in $\mathcal{F}$ as soon as $g <_H h$ and $\rho(g, h) > N$ holds, there exists a state f which satisfies $g <_H f <_H h$.

Theorem 5 ([30]). *If $\mathcal{F}'$ is a β-frame then the equivalence problem for PSP on $\mathcal{F}'$ is decidable.*

It was demonstrated (see [17,26,30,52]) that a lot of well-known groups (finite groups, free groups, free commutative groups, direct products of free groups, etc.) fall into the category of β-groups and have a decidable problem of PSP equivalence. But in all these cases the complexity of the decision procedures is at least exponential of the size of programs.

On the other hand, Yu.Gurevich noticed that any Turing machine can be simulated by a PSP running on a non-trivial Abelian group. This provides a grounding for the following theorem.

Theorem 6 ([26]). *Suppose that a frame $\mathcal{F}$ is Abelian group and $rank(\mathcal{F}) \geq 2$. Then the equivalence problem for PSPs on $\mathcal{F}$ is undecidable.*

Letichevsky [26] showed that the equivalence problem for PSPs is decidable when $\mathcal{F}$ is Abelian group of rank 1.

Multitape automata. Combining some PSP's semantics that are easy for solving the equivalence problem one may get computational models which are extremely complicated for analysis. Thus, for example, by joining together a frame $\mathcal{F}_c$ based on a free commutative monoid and a valuation function ξ specified by Yanov shifts we get a model which has exactly the same computational power as deterministic multitape finite automata. In the framework of PSPs the semantics of multitape automata is specified by the following PDL axioms

$$\begin{aligned} [a_ia_j]Q &\equiv [a_ja_i]Q, \qquad a_i, a_j \in \mathcal{A} \\ p_j &\equiv [a_i]p_j, \qquad a_i \in \mathcal{A},\ p_j \in Sh(a_i) \end{aligned} \tag{2}$$

where a shift $Sh : \mathcal{A} \to 2^{\mathcal{P}}$ meets a requirement

$$a_i \neq a_j \;\Rightarrow\; Sh(a_i) \cap Sh(a_j) = \emptyset.$$

The notion of multitape finite automata was introduced by Rabin and Scott [45] in their classical paper of 1959. The equivalence problem for deterministic multitape finite automata is one of the most famous problem in automata theory. It remained open for more than 30 years. The difficulty of this problem is manifested in the following facts. In [45] it was noticed that the inclusion problem for deterministic multitape finite automata is undecidable. In 1968 Griffiths [20] proved the undecidability of the equivalence problem for nondeterministic multitape automata. On the other hand, by replacing axioms (2) by their weak variants

$$p_j \to [a_i]p_j, \qquad a_i \in \mathcal{A},\ p_j \in Sh(a_i),$$

one get a semantics composed of monotonic commutative models. In [42] it was demonstrated that the equivalence problem for PSPs on monotonic commutative models is decidable.

In [4,56] positive solutions to the two-tape case of the equivalence problem were given. It was demonstrated also (see [31,23,7]) that the equivalence problem is decidable on some specific classes of deterministic multitape finite automata. In 1991 Harju and Karhumaki reduced the equivalence problem for deterministic multitape automata to the multiplicity equivalence problem for one-tape automata, attacked the latter by generalizing Eilenberg' Equality Theorem [10], and finally proved the following theorem.

Theorem 7 ([13]). *The equivalence problem for the n-tape deterministic finite automata is decidable.*

To the extent of our knowledge there are no results so far on the complexity of the equivalence problem for multitape deterministic finite automata.

Reset actions. Suppose that $\mathcal{A}$ contains some distinguished actions $b_1, \ldots, b_k$ which reset any PSP's run to some prescribed states. These actions corresponds to the right zeros in monoids; their semantics is specified by the axioms

$$[a_j b_i]Q \equiv [b_i]Q, \qquad a_j \in \mathcal{A}.$$

The equivalence problem for semigroup frames with right zeros was studied in [25,16]. In [16] it was proved that the equivalence problem for PSPs with reset actions is decidable in time which is double-exponential of the size of programs to be analyzed. In [62] it was proved that this problem is decidable in time which is polynomial of the size of programs and exponential of the maximal number of occurrences of each reset action in programs. It was demonstrated also that a variety of problems in polynomial-time hierarchy (3-SAT, the emptiness problem for the intersections of n deterministic n-state finite automata) are interreducible with various cases of the equivalence problem for PSPs supplied with reset statements.

On the other hand, by providing deterministic two-tape finite automata with a single reset action we get one of the most simple deterministic model of computations whose equivalence problem is undecidable [36].

2 First-Order Sequential Programs

In this section we consider the concept of first-order sequential program (FOSP), its sintax, and semantics, and discuss the principal results on the equivalence problem for this model of computations.

2.1 Syntax of FOSP

Terms and substitutions. Fix three alphabets X, F, R of variables, function symbols, and relation symbols, respectively. Without loss of generality we assume that the set of variables $X = \{x_1, \ldots, x_N\}$ is finite.

The set of terms $Term_X$ over X and F is defined in the usual way. When a variable x_i occurs in a term t, we denote this fact by writing $x_i \in t$. A *substitution* on X is a mapping θ from X to $Term_X$. We write θ as a set of *bindings* $\{x_1/t_1, \ldots, x_N/t_N\}$, assuming, in contrast to [1], that $t_i = x_i$ is possible. The term t_i which binds a variable x_i in θ is denoted by $\theta[i]$. A set of variables $\{x_i : \ x_i \neq \theta[i]\}$ is called a *domain* $Dom(\theta)$ of a substitution θ. A set of variables $\{x_j : \ \exists k \ x_j \in \theta[j]\}$ is called a *range* of θ. We use ϵ to denote *empty substitution* $\{x_1/x_1, \ldots, x_N/x_N\}$, and write $\eta\theta$ for the composition of substitutions η and θ. The application of a substitution θ to an expression E (term, atom, formula) is defined as in [1] and denoted by $E\theta$. The set of all substitutions on X is denoted by $Subst_X^X$.

As far as sequential programs are concerned, a binding x_i/t_i may be thought of as an assignment $x_i := t_i$, while a substitution θ stands for a parallel composition of assignments.

Atoms and conditions. Formulae of the form $p(x_{i_1}, \ldots, x_{i_k})$, where p is a k-ary relation symbol and $x_{i_1}, \ldots, x_{i_k}$ are variables in X, are called *atoms*. A *literal* based on an atom A is either A itself or its negation $\neg A$. Consider some finite set of atoms $\mathcal{A} = \{A_1, \ldots, A_M\}$. Then a M-tuple $C = \langle L_1, \ldots, L_M \rangle$ of literals based on the atoms $A_1, \ldots, A_M$ is called a *condition*. We write $\mathcal{C}_\mathcal{A}$ for the set of all possible conditions based on the set $\mathcal{A}$ of atoms. Clearly, $|\mathcal{C}_\mathcal{A}| = 2^M$. Conditions stand for the conjunctions of basic tests used in conditional statements **if-then-else** and loop statements **while-do**.

Term interpretations. A *term interpretation* I for X, F, R is defined as follows (see [1]).

- the domain of I is the term universe $Term_X$;
- each k-ary function symbol f in F is assigned the mapping from $(Term_X)^k$ to $Term_X$ which maps a sequence $t_1, \ldots, t_k$ of terms to the term $f(t_1, \ldots, t_k)$;
- each k-ary relation symbol p in R is assigned a set p^I of k-tuples of terms.

Given a term interpretation I, an atom $A = p(x_{i_1}, \ldots, x_{i_k})$, and a substitution θ in $Subst_X^X$, we say that I *satisfies the atom* A (*the literal* $\neg A$) *for* θ iff $(x_{i_1}\theta, \ldots, x_{i_k}\theta) \in p^I$ (resp. $(x_{i_1}\theta, \ldots, x_{i_k}\theta) \notin p^I$) holds.

First-order sequential programs. Consider alphabets X, F, R, and a finite set of atoms $\mathcal{A}$. Then a *sequential program* over $X, F, \mathcal{A}$ is a finite-state labelled transition system represented by a tuple $\pi = \langle V, \mathbf{entry}, \mathbf{exit}, \mathbf{loop}, S, T\rangle$, where

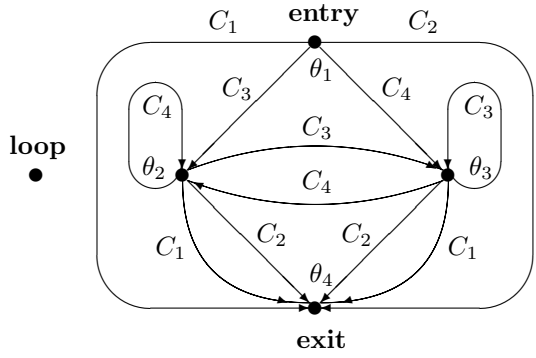
- V is a finite set of *program nodes*;
- $\mathbf{entry}, \mathbf{exit}, \mathbf{loop}$ are some distinguished program nodes;
- $S : V \to Subst_X^X$ is a total *assignment function* associating every program node with some substitution on X;
- $T : (V - \{\mathbf{exit}\}) \times \mathcal{C}_\mathcal{A} \to (V - \{\mathbf{entry}\})$ is a total *transition function* such that $T(\mathbf{loop}, C) = \mathbf{loop}$ holds for every condition C in $\mathcal{C}_\mathcal{A}$.

The nodes $\mathbf{entry}, \mathbf{exit}, \mathbf{loop}$ are called *initial, terminal*, and *dead* nodes, respectively. All other nodes in V are called *internal* nodes. By the *size* $|\pi|$ of a given program π we mean the total number of symbols involved in π. Given a finite sequence of program nodes $v_1, \ldots, v_{n-1}, v_n$ we write $S(v_1, \ldots, v_n)$ for the composition $S(v_n)S(v_{n-1}) \ldots S(v_1)$ of substitutions associated with the nodes.

Example 1. Consider a sequential program π written in traditional style.

```
i:=1; M:=0;
while i <= n
   do if a[i] > M then M:=a[i]; i:=i+1 od.
```

This program is intended for selecting the maximal positive element `M` in the integer array `a[1:n]`. According to the definition above it is represented by the following transition system over the set of variables $X = \{i, M\}$, the set of functions $F = \{a^{(1)}\}$, and the set of atoms $\mathcal{A} = \{A_1, A_2\}$, where $A_1 = (i \leq n)$ and $A_2 = (a(i) > M)$



$\theta_1 = \{i/1, M/0\}$,
$\theta_2 = \{i/i+1, M/M\}$,
$\theta_3 = \{i/i+1, M/a(i)\}$,
$\theta_4 = \epsilon$,
$C_1 = \langle \neg A_1, \neg A_2\rangle$,
$C_2 = \langle \neg A_1, A_2\rangle$,
$C_3 = \langle A_1, \neg A_2\rangle$,
$C_4 = \langle A_1, A_2\rangle$,

Fig. 1

2.2 Semantics of FOSP

Consider a program $\pi = \langle V, \mathbf{entry}, \mathbf{exit}, \mathbf{loop}, S, T\rangle$. A sequence of pairs

$$w = (v_0, C_0), (v_1, C_1), \ldots, (v_i, C_i), \ldots, \tag{3}$$

where $v_0, v_1, \ldots, v_i, \ldots$ are program nodes and $C_0, C_1, \ldots, C_i, \ldots$ are conditions, is said to be a *path* in π if $v_0 = \mathbf{entry}$ and $v_{i+1} = T(v_i, C_i)$ holds for every $i,\ i \geq 0$. A path w is called *total* if either w is infinite or it is finite and has a pair $(\mathbf{exit}, C)$ as its last element.

Given a term interpretation I, we say that a total path (3) is a *run of* π *on* I if for every pair (v_i, C_i) the interpretation I satisfies each literal L in C_i for the substitution $S(v_0, v_1, \ldots, v_i)$. The pairs $(v_i, C_i),\ i \geq 0$, are said to be the *states* of the run. Whenever a state (v, C) occurs in (3) then we say that the run w *reaches* this state. Since the programs under consideration are deterministic, every program π has a unique run on a given interpretation I. Denote this run by $\pi(I)$. If $\pi(I)$ reaches the *final state* (v_m, C_m) such that $v_m = \mathbf{exit}$, then we say that the run *terminates* having the substitution $S(v_0, v_1, \ldots, v_m)$ as the *result*. Otherwise, when $\pi(I)$ is an infinite sequence, we say that it *loops* and has no result. We write $r(\pi, I)$ for the result of $\pi(I)$, assuming $r(\pi, I)$ is undefined when the run loops.

Programs π' and π'' are called *equivalent* ($\pi' \sim \pi''$ in symbols) if $r(\pi', I) = r(\pi'', I)$ for every term interpretation I. The *equivalence problem* for sequential programs w.r.t. term interpretations is to check for an arbitrary pair π_1, π_2 of programs whether $\pi' \sim \pi''$ holds.

Along with the equivalence relation some other properties of FOSP are important for computer program analysis. A program π is called *empty* (*total*) if $r(\pi, I)$ is undefined (respectively, defined) for every interpretation I. A program π is called *free* if every total path (3) in π is a run $\pi(I)$ of π on some interpretation I.

2.3 Decidable and Undecidable Cases of the Equivalence Problem for FOSPs

Undecidability results. An extensive study of the equivalence problem for a first-order model of sequential programs began since 1968, when M.Paterson [39] introduced a concept of formalized computer program and set up decision problems for this model of computations. The definition of FOSP above is but a modification of this concept expressed in terms of transition systems and substitutions. In 1970 D.C.Luckham, D.M.Park and M.S.Paterson [32] and Letichevsky [27] showed that multihead one-tape finite automata can be simulated by means of FOSP. Since the emptiness problem for multihead finite automata is undecidable, we arrive then at the following

Theorem 8 ([32,27]). *The equivalence, emptiness, totality and freedom problems for FOSP are undecidable.*

A serious effort was mounted to reveal to what extent program structure, alphabets and interpretations affect the undecidability of the main decision problems for FOSPs.

In [18,36,37,41] it was shown that the boarder between decidability and undecidability for FOSPs is very subtle.

When $|X| = 1$ and all functions and predicates are unary we have FOSPs with inseparable memory. Programs of this kind corresponds to Yanov schemata. When $|X| = n$, all predicates are unary, and all substitutions involved in programs are ones of the form $\{x_i/f(x_i)\}$, we obtain deterministic n-tape finite automata. The main decision problems for both classes of FOSP are, clearly, solvable. But as soon as some minor modifications are made in these computational models, there comes a point of undecidability.

Theorem 9 ([36]). *Let $X = \{x_1, x_2\}$, $F = \{f^{(1)}\}$, $P = \{p^{(1)}\}$. Suppose also that only substitutions of the form $\{x_1/f(x_1)\}$, $\{x_2/f(x_2)\}$, and $\{x_2/x_1\}$ are available in programs. Then the emptiness problem for FOSPs over (X, F, P) is undecidable.*

The alphabet $A_1 = (X, F, P)$ used in the theorem above is the minimal alphabet which guarantees the undecidability of the emptiness problem for FOSPs, since any reduction of A_1 leads to the well-known decidable cases of FOSPs. It should be noticed also that FOSPs over A_1 may be thought of as deterministic two-tape finite automata supplied with an extra action for inserting a content of one tape into another.

Theorem 10 ([37]). *Let $X = \{x_1, x_2\}$, $F = \{c^{(0)}, f^{(1)}, g^{(1)}\}$, $P = \{p^{(1)}\}$. Suppose also that only substitutions of the form $\{x_1/f(x_1)\}$, $\{x_2/g(x_2)\}$ and $\{x_1/c, x_2/c\}$ are available in programs. Then the totality problem for FOSPs over (X, F, P) is decidable, whereas the emptiness and the equivalence problems are not.*

In this theorem we deal with a class of FOSPs located on the very boarder between decidable and undecidable. Programs from this class correspond to deterministic two-tape finite automata supplied with a reset action.

Paterson's and Letichevsky's results [40,27] were greatly strengthened by Itkin and Zwinogrodski [21]. They showed that the equivalence problem for FOSPs is undecidable for any "reasonable" equivalence definition based on the set of all interpretations and applicable to any FOSP. It is assumed that an equivalence of FOSPs is "reasonable" if it meets the following requirements specified below informally.

1. An equivalence relation is said to *consistent* if non-equivalence of two programs π_1 and π_2 implies the existence of interpretation I for which runs $\pi_1(I)$ and $\pi_2(I)$ differ.
2. An equivalence relation is said to be *nonsingular* if the existence of statement which, in some interpretation I, would be essential for the result of the run $\pi_1(I)$ and non-essential for the result of the run $\pi_2(I)$ implies non-equivalence of π_1 and π_2.

Theorem 11 ([21]). *If the equivalence relation $\simeq$ on FOSPs is consistent and non-singular then the equivalence problem "$\pi_1 \simeq \pi_2$?" is undecidable.*

In the theory of first-order sequential programs this theorem plays exactly the same role as Rice's theorem for universal computer models.

Decidability results. The theorem above implies that decidable cases of the equivalence problem for FOSPs could be obtained either by turning from functional equivalence to some crude non-interpretive equivalences which are more readily solved, or by imposing additional constraints on the structure of programs and selecting thus some wide classes of programs with solvable problem of functional equivalence.

The first line of investigations has led to a concept of logic-term equivalence of FOSPs [22] which is as follows. For every path (3) going in a program $\pi = \langle V, \mathbf{entry}, \mathbf{exit}, \mathbf{loop}, S, T\rangle$ from **entry** to **exit** a sequence

$$C_0S(v_0), C_1S(v_0, v_1), \ldots, C_iS(v_0, v_1, \ldots, v_i), \ldots$$

of conditions instantiated by compositions of substitutions $S(v_0, v_1, \ldots, v_i),\ i \geq 0$, assigned to program nodes $v_0, v_1, \ldots, v_i$ passed along this path, is referred to as a *determinant* of a path (3). Denote by $det(\pi)$ the set of determinants of all terminated paths in a program π. Two programs π_1 and π_2 are called *logic-term equivalent* if $det(\pi_1) = det(\pi_2)$.

It is apparent that $det(\pi)$ is a pure syntactical characteristic of a program π since it does not refer to any interpretations. It is easy to show also that $det(\pi_1) = det(\pi_2)$ implies $\pi_1 \sim \pi_2$. Therefore, the logic-term equivalence is a reasonable approximation to the interpretative equivalence. In [22] it was demonstrated that the logic-term equivalence can be checked effectively.

Theorem 12 ([22]). *The logic-term equivalence problem for FOSPs is decidable.*

A timed complexity of a decision procedure for the logic-term equivalence presented in [22] is double-exponential of the size of programs. Thereafter [5,29,49] the complexity of the logic-term equivalence checking procedures were improved substantially and reduced to $O(n^7)$. Due to the high efficiency of these algorithms, they are commonly used for data flow analysis and optimization of computer programs.

The alternative line of investigations provides the study of the equivalence problem for various classes of FOSPs. Results obtained in [27,32,36,37] show that the undecidability of the equivalence problem is induced by some specific features of substitutions involved in FOSP. By constraining the variety of substitutions used in programs we arrive at classes of FOSPs with decidable equivalence problem.

The first example of non-trivial class of programs with solvable equivalence problem was presented in [32,40]. A FOSP $\pi = \langle V, \mathbf{entry}, \mathbf{exit}, \mathbf{loop}, S, T\rangle$ is called *progressive* if all substitutions θ involved in π are of the form $\{x/t\}$,

and for every pair of adjacent nodes u, v such that $S(u, C) = v$, an inclusion $Dom(T(u)) \subseteq Rang(T(v))$ holds.

Theorem 13 ([32,40]). *The equivalence problem for progressive programs is decidable.*

A FOSP π is called *conservative* if $x_i \in \theta[i]$ holds for every variable x_i, $x_i \in X$, and for every substitution θ involved in π. Conservative programs were introduced in [32]. Further investigations have shown that the decidability of the equivalence problem for FOSPs is closely related to conservative property.

In [28] it was demonstrated that the equivalence problem is decidable for conservative FOSPs whose atoms are of the same arity N, where $N = |X|$. In fact, this class of FOSPs corresponds to the computational model of PSPs whose semantics is based on partially commutative frames $\mathcal{F}_I$ (see [43]).

In [19] it was proved that the equivalence problem is decidable for conservative FOSPs whose atoms are of the form $p(x_1)$ and all bindings $\{x_1/t\}$ for the variable x_1 are such that $x_1 \in t$.

Since all decidable cases above deal with free FOSP, this has led M.Paterson to a belief that the equivalence problem should be decidable for free programs. Paterson's conjecture is not ruled out so far and gained new testimonies in its favor.

In [50] Sabelfeld introduced a class of *weakly conservative* programs which extends the class of conservative programs. A FOSP π is called weakly conservative if $Dom(\theta) \subseteq Rang(\theta)$ for every substitution θ involved in π.

Theorem 14 ([50]). *The equivalence problem for free weakly conservative programs is decidable.*

This theorem generalizes the results obtained in [28,19]. The only difficulty in its application is that the freedom problem for conservative programs is generally undecidable.

In [60] a class of orthogonal programs was introduced. We say that terms t and s *match* if one of the terms occurs in the other. Otherwise, a pair of terms t, s is said to be *orthogonal*. A term with no variables is called *ground*. We also say that a substitution θ is *orthogonal* if it is not a renaming, and each pair of terms $\theta[i], \theta[j]$, $1 \leq i < j \leq N$, is orthogonal.

Example 2. A substitution $\theta = \{g(x_2, a)/x_1, x_3/x_2, g(f(x_1, a), x_2)/x_3\}$ is orthogonal and a term $f(x_1, a)$ matches θ.

Given alphabets (X, F, R), $X = \{x_1, x_2, \ldots, x_N\}$, and a finite set of atoms $\mathcal{A}$ we say that a program $\pi = \langle V, \mathbf{entry}, \mathbf{exit}, \mathbf{loop}, S, T\rangle$ over $X, F, \mathcal{A}$ is *orthogonal* if π meets the following requirements:

- every internal node v is associated with some orthogonal substitution $S(v)$ whose terms $S(v)[1], \ldots, S(v)[N]$ are non-ground;
- all atoms in $\mathcal{A}$ are ones of the form $p(x_1, \ldots, x_N)$, i.e. have all possible variables in X as their arguments.

An example of orthogonal program is depicted in Fig.1.

The class of orthogonal FOSP is in a sense dual to the class of progressive programs [32,40]. It is particularly remarkable that the orthogonal programs are free, though some of them are not weakly conservative.

Theorem 15 ([60]). *The equivalence problem for orthogonal programs is decidable.*

To the extent of our knowledge the classes of weakly conservative programs and orthogonal programs are the largest classes of FOSP with decidable equivalence problem.

3 Techniques for Proving Decidability/Undecidability of the Equivalence Problem

The undecidability results are usually obtained by applying common reduction technique to the well-known unsolvable problems (halting problem for Turing machines [26], emptiness problem for multihead finite automata [32], Post correspondence problem [21]).

A common way to prove the decidability of the equivalence problem for deterministic computational models is to find out a method for demonstrating the equivalence of two programs. This gives a semiprocedure for equivalence; the semiprocedure for nonequivalence trivially exists. The most-used techniques for proving the equivalence of sequential programs are "path" method, synchronization method, and reduction to a normal form.

Initially, the "path" method was introduced for proving lower bounds for computations [3]. This method takes an advantage of the specific phenomenon which is the very nature of some simple models of computation. It is as follows. Suppose that a program π (automaton) for some input data d has a run $r(\pi, d)$. If a length $|r(\pi, d)|$ exceeds some bound $L(\pi)$ which depends on program π only, then $r(\pi, d)$ can be broken down into fragments $r(\pi, d) = r_1 r_2 r_3$ so that joining the initial and the final fragments r_1 and r_3 we get a run of $r(\pi, d') = r_1 r_3$ of π for some other input data d'. This effect has much in common with Pumping Lemma which is widely used for proving non-regularity of formal languages [6]. It can be also applied for proving the equivalence of programs. Suppose that, given a pair of programs π_1, π_2, the results of some "long" runs of π_1 and π_2 reveal the difference between the functions realized by these programs. Then some "short" runs distinguish π_1 and π_2 as well. The equivalence problem is effectively solvable when the boundary between "long" and "short" computations depends recursively on some syntactic characteristics of π_1 and π_2. In this case one needs only to check the behaviour of programs on the finitely many runs to decide if they are equivalent or not. Usually the "path" method for equivalence checking is combined with the synchronization technique which reduces the equivalence problem to the emptiness problem for some devices that simulate synchronous runs of programs.

This approach was successfully employed in [16,5,22,25,28,30,40,45]. The main disadvantage of the "path" method is of its little use for practice since the number of "short" runs to be checked is very large (as a rule, it is exponential of the size of programs π_1 and π_2 under consideration). In [43,59,60,61] this deficiency has been overcome by encoding in algebraic terms the "difference" between intermediate states of computations $r(\pi_1, d)$ and $r(\pi_2, d)$. When a "difference" becomes too large, this indicates that results of $r(\pi_1, d)$ and $r(\pi_2, d)$ also differ. By choosing an appropriate encoding one could reduce the equivalence-checking problem "$\pi_1 \sim_{\mathcal{M}} \pi_2$?" to the well-known identity problem "$w_1 = w_2$?" in some specific monoids W associated with $\mathcal{M}$, and to the searching problem in a state space of polynomial size.

It is remarkable that whenever the "path" method gives a solution to the equivalence problem, then it always provides a solution to the inclusion problem, which is to find out whether π_1 and π_2 compute the same results whenever π_1 terminates. Since the inclusion problem for deterministic two-tape finite automata is undecidable, the "path" method is inapplicable to the equivalence problem for deterministic multitape finite automata.

The synchronization technique makes it possible to reduce the equivalence checking to the solution of the emptiness problem by constructing for given programs π_1 and π_2 an appropriate device $\pi_1 \times \pi_2$. It simulates synchronously computations of these programs so that a language accepted by $\pi_1 \times \pi_2$ is empty iff $\pi_1 \sim \pi_2$. The main difficulty in the using of this method is to provide the decidability of the emptiness problem for the crossproduct $\pi_1 \times \pi_2$. Nevertheless, in many cases (see [2,53,55,56,57,50,59,60]) this approach does work. A synchronization technique is discussed in much details in [8].

The key idea of a reduction to a normal form is in developing a complete equational calculus for an equivalence relation on programs. Usually, the completeness is proved by demonstrating that each pair of equivalent programs π_1 and π_2 can be transformed to the common normal form π_0. This approach has been used advantageously in [17,22,42,49,50,58].

References

1. K.Apt, From Logic Programming to Prolog, Prentice Hall, 1997.
2. E.Ashcroft, Z.Manna, A.Pnueli, A Decidable properties of monadic functional schemes, **J. ACM**, vol 20 (1973), N 3, p.489-499.
3. J.M.Barzdin, The complexity of recognition the symmetry by Turing machines. In *Problemy Kibernetiky*, **15**, 1965, p.245-248 (in Russian).
4. M.Bird, The equivalence problem for deterministic two-tape automata, *J. Comput. Syst. Sci.*, **7**, 1973, p.218-236.
5. A.O.Buda, Abstract computing machines. Tech. Report, Comp. Center of the USSR Academy of Science, Novosibirsk, N 108, 1978, 45 p (in Russian)..
6. D.Cozen, Automata and computability. Springer, 1997, 400 p.
7. K.Culik II, J. Karhumaki, HDT0L matching of computations of multitape automata, *Act. Inform.*, **27**, 1989, p.218-236.
8. K.Culik II, New techniques for proving the decidability of equivalence problems. *Theor. Comput. Sci.*, **71**, 1990, p.29-45.

9. J.W.De Bakker, D.A.Scott, A theory of programs. Unpublished notes, Vienna:IBM Seminar, 1969.
10. S.Eilenberg, Automata, Languages, and Machines. vol A, Academic Press, New-York, 1974.
11. A.P.Ershov, Theory of program schemata. In *Proc. of IFIP Congress 71*, Ljubljana, 1971, p.93-124.
12. D.Harel, Dynamic logics. In *Handbook of Philosophical Logics*, D.Gabbay and F.Guenthner (eds.), 1984, p.497-604.
13. T.Harju, J.Karhumaki, The equivalence of multi-tape finite automata. *Theoret. Comput. Sci.*, **78**, 1991, p.347-355.
14. Y.Hirshfeld, F.Moller, Decidable results in automata and process theory. *LNCS*, **1043**, 1996, p.102-148.
15. M.R.Garey, D.S.Johnson, A guide to the theory of NP-completeness, San-Francisco: W.H.Freeman & Company Publishers, 1979.
16. S.J.Garland, D.C.Luckham, Program schemes, recursion schemes and formal languages, *J. Comput. and Syst. Sci.*, **7**, 1973, p.119-160.
17. V.M.Glushkov, A.A.Letichevskii, Theory of algorithms and discrete processors, In *Advances in Information System Science*, vol 1 (1969), N 1.
18. A.B.Godlevsky, On some specific cases of halting problem and equivalence problem for automata. *Cybernetics*, 1973, N 4, p.90-97 (in Russian).
19. A.B.Godlevsky, On some special case of the functional equivalence problem for discrete transformers. *Cybernetics*, 1974, N 3, p.32-36 (in Russian).
20. T.V.Griffits, The unsolvability of the equivalence problem for λ-free nondeterministic generalized machines. *J. ACM*, **15**, 1968, p.409-413.
21. V.E.Itkin, Z.Zinogrodski, On program schemata equivalence. **J. Comput. and Syst. Sci.**, **6**, 1972, p.88-101.
22. V.E.Itkin, Logic-term equivalence of program schemata. *Cybernetics*, 1972, N 1, p.5-27 (in Russian)
23. E.Kinber, The inclusion problem for some classes of deterministic multitape automata. *Theoret. Comput. Sci.*, **26**, 1983, p.1-24.
24. A.A.Lapunov, Yu.I.Yanov, On logical program schemata, In *Proc. Conf. Perspectives of the Soviet Mathematical Machinery*, Moscow, March 12-17, 1956, Part III.
25. A.A.Letichevsky, On the equivalence of automata with final states on the free monoids having right zero. *Dokl. Akad. Nauk SSSR*, **182**, 1968, N 5 (in Russian).
26. A.A.Letichevsky, On the equivalence of automata over semigroup, *Theoretic Cybernetics*, **6**, 1970, p.3-71 (in Russian).
27. A.A.Letichevsky, Functional equivalence of finite transformers. II. *Cybernetics*, 1970, N 2, p.14-28 (in Russian)
28. A.A.Letichevsky, Functional equivalence of finite transformers. III. *Cybernetics*, 1972, N 1, p.1-4 (in Russian)
29. A.A.Letichevsky, Equivalence and optimization of programs. In *Programming theory*, Part 1, Novosibirsk, 1973, p. 166-180 (in Russian).
30. A.A.Letichevsky, L.B.Smikun, On a class of groups with solvable problem of automata equivalence, *Sov. Math. Dokl.*, **17**, 1976, N 2, p.341-344.
31. H.R.Lewis, A new decidable problem with applications. In *Proc 18th FOCS Conf.*, 1979, p.62-73.
32. 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.
33. A.Mazurkiewicz, Trace theory. *LNCS*, **255**, 1987, p.279-324.

34. J.McCarthy, A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*. Amsterdam:North-Holl. Publ. Co., 1963, p.33-70.
35. R.Milner, Processes: a mathematical model of computing agents. In *Proc. of Logic Colloquium'73*, 1973, p.157-174.
36. V.A.Nepomniaschy, On divergence problem for program schemes. *LNCS*, **45**, 1976, p.442-445.
37. V.A.Nepomniaschy, On the emptiness problem for program schemes. *Programming and Software Engineering*, 1977, N 4, p.3-13 (in Russian).
38. A.G.Oettinger, Automatic syntactic analysis and pushdown store. In Proc. Symposia on Applied Math., **12**, 1961.
39. M.S.Paterson, Programs schemata, *Machine Intelligence*, Edinburgh: Univ. Press, **3**, 1968, p.19-31.
40. M.S.Paterson, Decision problems in computational models, *SIGPLAN Notices*, **7**, 1972, p.74-82.
41. 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).
42. R.I.Podlovchenko, On the decidability of the equivalence problem on a class of program schemata having monotonic and partially commutative statements. *Programming and Software Engineering*, 1990, N 5, p.3-12 (in Russian).
43. R.I.Podlovchenko, V.A.Zakharov, On the polynomial-time algorithm deciding the commutative equivalence of program schemata, *Reports of the Russian Academy of Science*, **362**, 1998, N 6 (in Russian).
44. V.R.Pratt, Semantical considerations of Floyd-Hoare logic. In *Proc. 17th IEEE Symp. Found. Comput. Sci.*, 1976, p.109-121
45. M.O.Rabin, D.Scott, Finite automata and their decision problems. *IBM J. Res. Dev.*, **3**, 1959, N 2, p.114-125.
46. H.G.Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, bf 89, 1953, p. 25-59.
47. H.Rogers, Theory of recursive functions and effective computability. McGraw-Hill, 1967.
48. J.D.Rutledge, On Ianov's program schemata, *J. ACM*, **11**, 1964, p.1-9.
49. V.K.Sabelfeld, Logic-term equivalence is checkable in polynomial time. *Reports of the Soviet Academy of Science*, **249**, 1979, N 4, p.793-796 (in Russian).
50. V.K.Sabelfeld, An algorithm deciding functional equivalence in a new class of program schemata, *Theoret. Comput. Sci.*, **71**, 1990, p.265-279.
51. G.Senizergues, The equivalence problem for deterministic pushdown automata is decidable, *LNCS*, **1256**, 1997, p.271-281.
52. M.A.Taiclin,The equivalence of automata w.r.t. commutative semigroups, *Algebra and Logic*, **8**, 1969, p.553-600 (in Russian).
53. E.Tomita, K.Seino, The extended equivalence problem for a class of non-real-time deterministic pushdown automata. *Acta Informatica*, **32**, 1995, p.395-413.
54. V.A.Uspensky, A.L.Semenov, What are the gains of the theory of algorithms: basic developments connected with the concept of algorithm and with its application in mathematics. *LNCS*, bf 122, 1981, p.100-234.
55. L.G.Valiant, Decision procedures for families of deterministic pushdown automata, Report N 7, Univ. of Warwick Computer Center, 1973.
56. L.G.Valiant, The equivalence problem for deterministic finite-turn pushdown automata, *Information and Control*, **25**, 1974, p.123-133.
57. L.G.Valiant, M.S.Paterson, Deterministic one-counter automata, *J. of Comput. and Syst. Sci.*, **10**, 1975, p.340-350.

58. J.Yanov, To the equivalence and transformations of program schemata, *Reports of the Soviet Academy of Science*, **113**, 1957, N 1, p.39-42 (in Russian).
59. V.A.Zakharov, The efficient and unified approach to the decidability of the equivalence of propositional programs. In *LNCS*, **1443**, 1998, p. 246-258.
60. V.A.Zakharov, On the decidability of the equivalence problem for orthogonal sequential programs, *Grammars*, **2**, 1999, p.271-281.
61. V.A.Zakharov, On the uniform technique for designing decision procedures for the equivalence of propositional sequential programs. In Proc. of the 4th Int. Conf. on Discrete Models in Control System Theory, Krasnovidovo, 2000, p.25-29 (in Russian).
62. V.A.Zakharov, On the equivalence problem for sequential program schemata supplied with reset statements. In Proc. of the 4th Int. Conf. on Discrete Models in Control System Theory, Krasnovidovo, 2000, p.153-154 (in Russian).