On the model checking of sequential reactive systems

D.G. Kozlova[1] V.A. Zakharov[2]

[1] Faculty of Computational Mathematics and Cybernetics,
Lomonosov Moscow State University, Moscow, RU-119899, Russia,
[2] Faculty of Computer Science,
National Research University Higher School of Economics,
Moscow, Russia
(Corresponding author: zakh@cs.msu.su)

Abstract. By sequential reactive system we mean a program which operates in the interaction with the environment permanently receiving data (requests) from it. At receiving a piece of data a program performs a sequence of actions (response) and displays the current result. Such programs usually arise at implementation of computer drivers, online algorithms, control procedures. Basic actions performed by these programs may be regarded as generating elements of a certain semigroup. This consideration opens the way to model sequential reactive systems by finite state transducers that operate over semigroups. This model of computation is suitable for synthesis, optimization, verification and testing of sequential reactive systems. In this paper we originate a framework for developing verification techniques for sequential reactive systems by utilizing finite state transducers as a formal model. To this end we introduce a LTL-based formal language which may be suitable for specification of the behaviour of sequential reactive systems and adopt a well known LTL-based model checking techniques for verification of finite state transducers against these specifications.

1 Introduction

Finite state transducers extend the finite state automata to model functions and relations on strings or lists. They are used in many fields as diverse as computational linguistics [14] and model-based testing [1, 22]. In software engineering transducers provide a suitable formal model for various on-line algorithms and device drivers for manipulating with strings, transforming images, filtering dataflows, inserting fingerprints, sorting data, etc.

An ordinary model of finite state transducers over words can be further extended to encompass a more wide class of sequential reactive programs. These programs operate in the interaction with the environment permanently receiving data (requests) from it. At receiving a piece of data such program performs a sequence of actions. When certain control points are achieved a program outputs the current results of computation as a response. What matters is that different sequences of actions may yield the same result. Therefore, the basic actions of a program may be viewed as generating elements of some appropriate semigroup, and the result of computation may be regarded as the composition of actions performed by the program.

Let us consider some examples. Imagine that a radio-controlled robot moves on the earth surface. It can make one step moves in any of 4 directions N, E, S, W. When such robot receives a control signal syg in a state q it must choose and carry out a sequence of steps (say, N, N, W, S), and enter to the next state q'. At some distinguished state q_{fin} robot reports its current location. Movements of the robot may be regarded as basic actions, and the most simple model of computation which is suitable for analyzing a behaviour of this robot is non-deterministic finite state transducer operating on free Abelian group of rank 2. Next, consider a network switch which receives as input packet flows alternating with control instructions. Following to its flow table a switch sends modified copies of every packet into one or another output port. A flow table is updated at receiving a control instruction. Modifications and forwardings of a data packet may be regarded as basic actions. When a switch forwards two packets from different packet flows to different ports, the corresponding actions can be performed in an arbitrary order. Therefore, such a switch can be modeled by a finite state transducer operating on a partially commutative semigroup. Semigroups of this kind are also known as traces; they are thoroughly studied in [9].

When designing sequential reactive systems software engineers want to be confident of their correct behaviour. For example, in the case of radio-controlled robot it may be required that it never appears in the north-west sector of the surface, obligatory passes via certain locations, and can be always returned to the starting point at receiving a particular sequences of control messages. When a network switch is concerned, its computations should comply with the requirements of forwarding policies (see, e.g. [6, 7]) such as the absence of forwarding loops, non-interference of certain packet flows, etc. To analyze the behaviour of sequential reactive systems one may use the concept of finite state transducer over finitely generated semigroups as a formal model of such systems and develop various verification techniques (equivalence checking, model checking, deductive verification, etc.) for these class of transducers.

Equivalence checking problem for finite state transducers has been studied in much details in many papers. Its study for classical transducers that operate on words began in the early 60s. First, it was shown that the equivalence checking problem is undecidable for non-deterministic transducers [11] even over 1-letter input alphabet [12]. But the undecidability displays itself only in the case of unbounded transduction when an input word may have arbitrary many images. At the next stage bound-valued transducers were studied. The equivalence checking problem was shown also to be decidable for deterministic [4], functional (single-valued) [3, 18], and k-valued transducers [8, 23]. In a series of papers [16, 17, 19] techniques for checking bounded valuedness, k-valuedness and equivalence of finite state transducers over words were developed. Recently in [25] equivalence checking problem was shown to be decidable for finite state transducers that operate over finitely generated semigroups embeddable in decidable groups.

There are also papers where equivalence checking problem for transducers is studied in the framework of program verification. The authors of [20] proposed models of communication protocols as finite state transducers operating on bit strings. They set up the verification problem as equivalence checking between the protocol transducer and the specification transducer. The authors of [22] extend finite state transducers with symbolic alphabets which are represented as parametric theories. They showed that a number of classical problems for extended transducers, including equivalence checking problem, are decidable modulo underlying theories. In [1] a model of streaming transducers was proposed for programs that access and modify sequences of data items in a single pass. It was shown that a number of verification problems such as equivalence checking, assertion checking, and checking correctness with respect to pre/post conditions, are decidable for this transducer model.

Unlike equivalence checking, model checking of (or related with) transducers is less well studied. Transducers found a usage in regular model checking of parameterized distributed systems. In some formal models of these systems configurations are modeled as words over finite alphabet. In such a situation a transition relation on these configurations is a binary relation on finite words which can be adequately specified by finite state transducers (see [5, 24]). In this line of research transducers play the role of verification instrument, but not an object of verification. As for verification of transducers, to the extend of our knowledge no special purpose study of model checking problem for finite state transducers has been conducted so far. In our opinion, this is due the following reason. Usually, both the influence of the environment upon a reactive system and its response is defined in terms of a set of basic predicates. The letters of input and output alphabets of a transducer are regarded as valuations (tuples of truth values) of these predicates, and transducers are viewed as special presentation of finite labeled transition system (Kripke structure) (see [2]). From this viewpoint model checking problem for finite state transducers conforms well to standard model checking scheme for finite structures, and, therefore, are not worthy of any particular treatment.

However, these arguments become invalid when a response of a reactive system at every step of its computation is regarded as a composition of actions produced by the system so far. In this case the predicates which specify the basic properties of reactive systems behaviour are defined on finite sequences of actions, i.e. every such predicate is a language over an alphabet of output actions. More complex dynamic properties can be expressed by LTL formulae. It should be remarked that these formulae must express not only the properties of output sequences of actions but relationships between input sequences of requests from the environment (signal flows) and output sequences of responding actions (compound actions). This can be achieved through the introduction of behaviour patterns of the environment as the sets of signal flows and the using of these patterns as parameters of temporal operators.

In this paper we make an attempt to introduce a LTL-based formal language for specification of the behaviour of sequential reactive systems and to adapt a well known LTL-based model checking techniques [13, 21] for verification of finite state transducers. The paper is organized as follows. In the next section a concept of finite state transducer over semigroup (see [25]) as a formal model of sequential reactive systems is defined. In Section 3 we introduce $\mathcal{LP}$-LTL — a parameterized version of Linear Temporal Logics — as a formal language for specifying behaviour of sequential reactive systems. In this section we also set up model checking problem for finite state transducers. In Section 4 we present a $\mathcal{LP}$-LTL model checking algorithm for the case when both basic properties of reactive systems and behaviour patterns of the environment are defined by finite state automata. Finally, we outline some possible directions for further research.

2 Transducers as models of reactive systems

Let $\mathcal{C}$ and $\mathcal{A}$ be two finite sets. The elements of $\mathcal{C}$ are called *signals*; they may be viewed as abstractions of messages (control instructions, instrument or sensor readings, pieces of data, etc.) received by a reactive system from its environment. Finite sequences of signals (words over alphabet $\mathcal{C}$) are called *signal flows*. As usual, the set of all signal flows is denoted by $\mathcal{C}^*$. We write uv for concatenation of signal flows u and v, and ε for the empty signal flow.

The elements of $\mathcal{A}$ are called *basic actions*; they are the abstractions of operations (data processings, movements, etc.) performed by a reactive system in response to received signals. Finite sequences of basic actions (words over alphabet $\mathcal{A}$) are called *compound actions*.

Actions are interpreted over semigroups. Consider a semigroup $(S, e, \circ)$ generated by the set $\mathcal{A}$, where S is a set of semigroup elements, e is the neutral element, and $\circ$ is a composition operation. The elements of S may be regarded as *data states*. Every basic action $a, a \in \mathcal{A}$, when been applied to a data state $s, s \in S$, yields the result $s \circ a$. Every compound action $h = a_1 a_2 \dots a_k$ is interpreted as the composition $[h] = a_1 \circ a_2 \circ \dots \circ a_k$.

A *trajectory* on a semigroup $(S, e, \circ)$ is a pair $tr = (s_0, \alpha)$ such that $s_0 \in S$ and α is an infinite sequence

$$\alpha = (c_1, s_1), (c_2, s_2), \dots, (c_i, s_i), \dots,$$

where $c_i \in \mathcal{C}, s_i \in S$ for every $i, i \geq 0$. This sequence represents a possible behaviour of a reactive system as it becomes visible to an outside observer: every time at receiving a next signal c_i the system performs some compound action h_i and displays its effect $s_i = s_{i-1} \circ h_i$. Given a trajectory $tr = (s_0, \alpha)$ and an integer $i, i \geq 0$, we denote by $tr|^i$ the trajectory $(s_i, \alpha|^i)$, where $\alpha|^i = (c_{i+1}, s_{i+1}), (c_{i+2}, s_{i+2}), \dots$

A *finite state transducer* over a set of signals $\mathcal{C}$ and a set of basic actions $\mathcal{A}$ is a system $\pi = (\mathcal{C}, \mathcal{A}, Q, Q_0, T)$, where Q is a finite set of *control states*, $Q_0, Q_0 \subseteq Q$, is a set of *initial states*, and $T, T \subseteq Q \times \mathcal{C} \times Q \times \mathcal{A}^*$, is a *transition relation*. Every quadruple (q, c, q', h) in T is called a *transition*: when a transducer is in a control state q and receives a signal c it passes its control to a state q' and performs a compound action h. Such transitions are usually depicted as $q \xrightarrow{c,h} q'$. It is assumed that T is a total relation: for every control state q and a signal c the set T includes at least one transition of the kind $q \xrightarrow{c,h} q'$. A *run* of π is any sequence of transitions

$$run = q_0 \xrightarrow{c_1,h_1} q_1 \xrightarrow{c_2,h_2} q_2 \xrightarrow{c_3,h_3} \cdots \tag{1}$$

which begins from some initial state q_0. We write $run|^i$ for the suffix of the sequence run which begins from the state $q_i, i \geq 0$. The size $|\pi|$ of a transducer π is the number $|Q|$ of its state.

A finite state transducer can serve as a formal model of a sequential reactive system. At each step of its computation it receives a signal c from the environment and performs a transition $q \xrightarrow{c,h} q'$ by passing its control to a state q' and executing an action h. Usually behaviour of transducers is defined as transduction relation between input and output words. But it can be rather well defined in terms of trajectories as follows. Suppose that basic actions of a transducer $\pi = (\mathcal{C}, \mathcal{A}, Q, Q_0, T)$ are interpreted over a semigroup $(S, e, \circ)$. Then every run (1) of π generates a trajectory $tr(run) = (e, \alpha)$, where the sequence $\alpha = (c_1, s_1), (c_2, s_2), \ldots, (c_i, s_i), \ldots$, is such that $s_1 = e \circ h_1$, and $s_i = s_{i-1} \circ h_i$ holds for every $i, i \geq 2$. The set of all trajectories generated by the runs of π is denoted by $Tr(\pi, S)$. This set completely characterizes a behaviour of sequential reactive system modeled by a transducer π over a semigroup of actions $(S, e, \circ)$.

3 Specification language

Specification languages are intended to describe formally desirable (or erroneous) behaviours of computing systems. Since the behaviour of a sequential reactive system is presented as a set of trajectories, the expressions of an appropriate specification language should be interpreted over trajectories. Every trajectory displays how the data states from the set S changes as a reactive system receives signals and performs responding actions with the passage of time. Therefore, it is advantageous to take some variant of temporal logics as a framework of such a specification language.

The formulae of temporal logics are built of basic predicates by means of Boolean connectives and temporal operators. Basic predicates are defined on data states. In our model of sequential reactive systems data states are interpreted as elements of a semigroup $(S, e, \circ)$. Thus, basic predicates can be regarded as certain subsets of S. They can be formally introduced alternatively in different ways.

1. By means of parameterized algebraic equations in a semigroup: a data state s satisfies a basic predicate $Eq(p, X)$ iff s is such a value of a parameter p that an equation $Eq(p, X)$ has a solution in a semigroup $(S, e, \circ)$. For example, an equation $p \circ X = e$ specifies a set of data states s from which a computation of a reactive system can be restarted.

2. By any means — formal grammars, language equations, automata of various types, etc. — for defining formal languages over a set of basic actions $\mathcal{A}$. A data state s satisfies a predicate L, where L is a language over $\mathcal{A}$, iff $s = [h]$ for some compound action h such that $h \in L$. For example, a finite state automaton A distinguishes a set of data states s such that $s = [h]$ for some compound action h accepted by A.

A sequential reactive system modifies data states in response to incoming signals. These signals come to an input of a system in conformity with a certain scenario (pattern) of environment's behaviour. An environment behaviour pattern characterizes a set of possible signal flows that may affect a reactive system. Therefore, a specification of its behaviour must include some references to signal flows. This can be achieved by using formal descriptions of environment behaviour patterns as parameters of temporal operators. Since a signal flow is but a word over a set of signals $\mathcal{C}$, such descriptions can be provided by any means used for defining formal languages — grammars, equations, automata.

These contemplations bring us to the following concept of formal specification language $\mathcal{LP}$-LTL for sequential reactive systems. Given a set of signals $\mathcal{C}$, a set of basic actions $\mathcal{A}$, and a semigroup $(S, e, \circ)$ generated by basic actions, we say that any set of finite words (language) over the alphabet $\mathcal{C}$ is an *environment behaviour pattern* (or, simply, a *pattern*), and any subset $S', S' \subseteq S$, of data states is a *basic predicate.*

Select a family of patterns $\mathcal{L}$ and a family $\mathcal{P}$ of basic predicates. Then a set of $\mathcal{LP}$-LTL formulae is the minimal set $Form$ of expressions which satisfy the following rules:

1) every basic predicate $P, P \in \mathcal{P}$, is a $\mathcal{LP}$-LTL formula;
2) if φ, ψ are $\mathcal{LP}$-LTL formulae then $\neg\varphi$, $\varphi \wedge \psi$ and $\varphi \vee \psi$ belong to $Form$;
3) if $\varphi \in Form$ and $c \in \mathcal{C}$ then $X_c\varphi$, $Y_c\varphi$ belong to $Form$;
4) if $\varphi \in Form$ and $L \in \mathcal{L}$ then $F_L\varphi$, $G_L\varphi$ belong to $Form$ as well.

This definition is constructive, since $\mathcal{L}$ and $\mathcal{P}$ may be thought of as the set of names interpreted over patterns and basic predicates. The size $|\varphi|$ of a formula φ is the number of Boolean connectives and temporal operators occurred in φ.

The semantics of the specification language is defined in terms of satisfiability relation $\models$ of $\mathcal{LP}$-LTL formulae on trajectories. Let $tr = (s_0, \alpha)$ be a trajectory, where $\alpha = (c_1, s_1), (c_2, s_2), \dots, (c_i, s_i), \dots$, and φ be a $\mathcal{LP}$-LTL formula. Then

1) if $P \in \mathcal{P}$ then $tr \models P \iff s_0 \in P$;
2) $tr \models \neg\varphi \iff$ it is not true that $tr \models \varphi$;
3) $tr \models \varphi \wedge \psi \iff tr \models \varphi$ and $tr \models \psi$;
4) $tr \models \varphi \vee \psi \iff tr \models \varphi$ or $tr \models \psi$;
5) $tr \models X_c\varphi \iff c = c_1$ and $tr|^1 \models \varphi$;
6) $tr \models Y_c\varphi \iff c \neq c_1$ or $tr|^1 \models \varphi$;
7) $tr \models F_L\varphi \iff \exists\, i \geq 0 \ : \ c_1c_2\dots c_i \in L$ and $tr|^i \models \varphi$;
8) $tr \models G_L\varphi \iff \forall\, i \geq 0 \ : \ c_1c_2\dots c_i \in L$ implies $tr|^i \models \varphi$.

Clearly, some other parameterized temporal operators that are used in LTL like U (until), W (weak until), R (release) can be introduced in the same way. Moreover, some new temporal operators that are specific for $\mathcal{LP}$-LTL may be introduced. For example, to express some properties of trajectories one may need a weak eventuality operator $\widehat{F}_L$ which has the following semantics:

$tr \models \widehat{F}_L\varphi \iff$ either $\forall\ i \geq 0\ :\ c_1c_2\dots c_i \notin L,$ or $tr \models F_L\varphi.$

It is easy to make sure that parameterized temporal operators introduced above satisfy duality and fixed-point (expansion) properties.

Proposition 1. *Let φ be an arbitrary $\mathcal{LP}$-LTL formula, $c \in \mathcal{C}$, $L \subseteq \mathcal{C}^*$, and tr be an arbitrary trajectory. Then*

1) $tr \models \neg X_c\varphi \iff tr \models Y_c\neg\varphi,$
2) $tr \models \neg Y_c\varphi \iff tr \models X_c\neg\varphi,$
3) $tr \models \neg F_L\varphi \iff tr \models G_L\neg\varphi,$
4) $tr \models \neg G_L\varphi \iff tr \models F_L\neg\varphi.$

For every pattern L and a signal c denote by $Pref_1(L)$ the set $\{c : \exists w \in \mathcal{C}^* : cw \in L\}$ of 1-letter prefixes of signal flows in L, and by $Suff_c(L)$ the pattern $\{w : cw \in L\}$ which consists of maximal proper suffixes of those signal flows in L that begin with the signal c. We say that a family of patterns $\mathcal{L}$ is suffix-closed iff for every signal c and every pattern $L, L \in \mathcal{C}$, the pattern $Suff_c(L)$ also belongs to $\mathcal{L}$.

Proposition 2. *Suppose that a family of patterns $\mathcal{L}$ is suffix-closed, and let φ be a $\mathcal{LP}$-LTL formula, and tr be a trajectory. Then*

1) *if* $\varepsilon \in L$ *then* $tr \models F_L\varphi \iff tr \models \varphi \vee \bigvee_{c \in Pref_1(L)} X_cF_{Suff_c(L)}\varphi,$
2) *if* $\varepsilon \notin L$ *then* $tr \models F_L\varphi \iff tr \models \bigvee_{c \in Pref_1(L)} X_cF_{Suff_c(L)}\varphi,$
3) *if* $\varepsilon \in L$ *then* $tr \models G_L\varphi \iff tr \models \varphi \wedge \bigwedge_{c \in Pref_1(L)} Y_cF_{Suff_c(L)}\varphi,$
4) *if* $\varepsilon \notin L$ *then* $tr \models G_L\varphi \iff tr \models \bigwedge_{c \in Pref_1(L)} Y_cF_{Suff_c(L)}\varphi.$

As in the case of ordinary LTL these properties are important for building model checking and satisfiability checking procedures for $\mathcal{LP}$-LTL formulae.

4 Model checking sequential reactive systems against $\mathcal{LP}$-LTL specifications

Assume that sequential reactive systems are modeled by finite state transducers that operate over a set of signals $\mathcal{C}$ and the set of basic actions $\mathcal{A}$ interpreted in a semigroup $(S, e, \circ)$. Let $\mathcal{L}$ and $\mathcal{P}$ be families of admissible patterns and basic predicates. Then model checking (MC) problem for sequential reactive systems against $\mathcal{LP}$-LTL specifications is that of checking, given a finite state transducer

π and a $\mathcal{LP}$-LTL formula φ, whether $tr \models \varphi$ holds for every trajectory tr in $Tr(\pi, S)$ (or, in symbols, $Tr(\pi, S) \models \varphi$).

It is evident that decidability and complexity of MC problem for sequential reactive systems against $\mathcal{LP}$-LTL specifications essentially depend on 1) a semigroup $(S, e, \circ)$ used for interpretation of basic actions, 2) a family of basic predicates $\mathcal{P}$ on the set of data states S, and 3) a family of behaviour patterns of the environment $\mathcal{L}$ used for parametrization of temporal operators. In some cases this problem has an effective solution.

Here we consider the most simple case of MC problem when 1) basic actions are interpreted over free monoid $(S, e, \circ)$, where S is the set of compound actions $\mathcal{A}^*$, $e = \varepsilon$, and $\circ$ is concatenation operation on compound actions, 2) a family $\mathcal{P}$ of basic predicates is the collection of all regular sets of compound actions, 3) a family $\mathcal{L}$ of behaviour patterns of the environment is the collection of all regular sets of signal flows. $\mathcal{LP}$-LTL formulae of this type will be called *Reg-LTL* formulae. The main advantage of *Reg-LTL* is that the most simple model of computation — deterministic finite state automata — can be involved to define basic predicates and patterns occurred in these formulae.

By (non-initialized) *deterministic finite state automaton* we mean a quadruple $K = (\Sigma, Z, Z_{acc}, \Phi)$, where Σ is a finite input alphabet, Z is a finite set of states, $Z_{acc}, Z_{acc} \subseteq Z$, is a subset of accepting states, and $\Phi : Z \times \Sigma \to Z$ is a total transition function. A transition function can be extended to the set Σ^* in the usual fashion: $\Phi(z, \varepsilon) = z$, and $\Phi(z, bw) = \Phi(\Phi(z, b), w)$ for every state z, a letter b in Σ and a word $w, w \in \Sigma^*$. By initialized automaton we mean a pair (K, z_0), where z_0 is a state of an automaton K. An initialized automaton (K, z_0) accepts a word w if $\Phi(z_0, w) \in Z_{acc}$; thus, it specifies a regular language $L(K, z_0) = \{w \ : \ \Phi(z_0, w) \in Z_{acc}\}$ of all accepted words.

When finite state automata are used for specification of regular basic predicates they have the set of basic actions $\mathcal{A}$ as an input language; automata of this kind will be called $\mathcal{A}$-automata. When finite state automata are employed for specification of regular patterns of the environment they have the set of signals $\mathcal{C}$ as an input alphabet; automata of this sort will be called $\mathcal{C}$-automata. Thus, every atomic formula of *Reg-LTL* is an initialized $\mathcal{A}$-automaton (A, z_0), and temporal operators used in *Reg-LTL* are those of the form $X_c, Y_c, F_{(B,z_0)}, G_{(B,z_0)}$, where c is a signal, and (B, z_0) is an initialized $\mathcal{C}$-automaton. In what follows we will use letters Z, Z_{acc} and Φ as generic names of a set states, a subset of accepting states and a transition functions in automata that specify basic predicates and patterns of the environment.

The rules of *Reg-LTL* semantics can be redefined in terms of finite state automata. Suppose, for example, that a run of a transducer begins with a transition $q \xrightarrow{c,h} q'$. Then $tr(run) \models X_c(A, z_0) \iff h \in (A, z_0) \iff \Phi(z_0, h) \in Z_{acc}$. This effect also manifests itself for other formulae. Given a $\mathcal{A}$-automaton (A, z_0) and a compound action h, we say that the $\mathcal{A}$-automaton $(A, \Phi(z_0, h))$ is *h-shift* of basic predicate (A, z_0). In more general case, a h-shift of a *Reg-LTL* formula φ is a formula $shift(\varphi, h)$ which is obtained from φ by replacing every basic predicate (A, z_0) occurred in φ with its h-shift $(A, \Phi(z_0, h))$. Consider a run (1) of a transducer π. Then

$$\begin{aligned} tr(run) \models F_{(B,z_0)}\varphi &\iff \exists\ i \geq 0\ :\ \Phi(z_0, c_1c_2\dots c_i) \in Z_{acc} \text{ and} \\ & \qquad tr(run|^i) \models shift(\varphi, h_1h_2\dots h_i); \\ tr(run) \models G_{(B,z_0)}\varphi &\iff \forall\ i \geq 0\ :\ \Phi(z_0, c_1c_2\dots c_i) \in Z_{acc} \text{ implies} \\ & \qquad tr(run|^i) \models shift(\varphi, h_1h_2\dots h_i); \end{aligned}$$

These relationships are crucial in the designing of *Reg-LTL* model checking algorithm in Theorem 1.

For the sake of brevity we will skip references to a semigroup $(S, e, \circ)$ in our notation till the end of the section. It is assumed that this semigroup is a free monoid of finite words over $\mathcal{A}$ and MC problem $Tr(\pi) \models \varphi$ is studied for finite state transducers against *Reg-LTL* specifications.

The main result of this section is

Theorem 1. *Let $\pi = (\mathcal{C}, \mathcal{A}, Q, Q_0, T)$ be a finite state transducer operating on a free monoid of words, and φ be a Reg-LTL formula. Suppose that every regular component (a basic predicate or a pattern of the environment) of φ is specified by a deterministic finite state automata which has N states at the most. Then there exists a generalized Büchi automaton $M[\pi, \varphi]$ such that*

- *$M[\pi, \varphi]$ has $|\pi| 2^{O(|\varphi| N^{|\varphi|})}$ states at the most;*
- *$M[\pi, \varphi]$ can be constructed effectively by π and φ in time polynomial of its size;*
- *$M[\pi, \varphi]$ accepts empty ω-language iff $Tr(\pi) \models \varphi$.*

Proof. (Sketch) Our algorithm for the translation of a pair (π, φ) to a Büchi automaton $M[\pi, \varphi]$ follows the well-known scheme for translation of LTL formulae to Büchi automata which was introduced in [21]. We only emphasize those aspects of this translation which are specific for *Reg-LTL*.

1. Consider the formula $\psi = \neg\varphi$ and present it in negation normal form via duality laws (see Proposition 1). It should be noted that if a basic predicate is specified by an automaton (A, z_0) then $\neg(A, z_0) \equiv (\bar{A}, z_0)$, where $\bar{A}$ is a complementation of A. Thus, we eliminate all negations in ψ.

2. Define the closure $cl(\psi)$ of ψ as the minimal set of *Reg-LTL* formulae which complies with the following rules:

- $\psi \in cl(\psi)$,
- $(A, z_0) \in cl(\psi) \ \Rightarrow\ \forall\ z \in Z:\ (A, z) \in cl(\psi)$
- $f \vee g \in cl(\psi) \ \Rightarrow\ f, g \in cl(\psi)$,
- $f \wedge g \in cl(\psi) \ \Rightarrow\ f, g \in cl(\psi)$,
- $X_c f \in cl(\psi) \ \Rightarrow\ shift(f, h) \in cl(\psi)$ for every $h \in \mathcal{A}^*$,
- $Y_c f \in cl(\psi) \ \Rightarrow\ shift(f, h) \in cl(\psi)$ for every $h \in \mathcal{A}^*$,
- $F_{(B,z_0)} f \in cl(\psi) \ \Rightarrow\ f \in cl(\psi)$ and $\forall\ c \in \mathcal{C}:\ X_c F_{(B,\Phi(z_0,c))} f \in cl(\psi)$,
- $G_{(B,z_0)} f \in cl(\psi) \ \Rightarrow\ f \in cl(\psi)$ and $\forall\ c \in \mathcal{C}:\ Y_c G_{(B,\Phi(z_0,c))} f \in cl(\psi)$.

As it can be seen from the definition of $cl(\psi)$ this set may contain $O(|\varphi|N^{|\varphi|})$ at the most.

3. Build the collection $CS(\psi)$ of all subsets of $cl(\psi)$ which are both locally consistent and saturated. A subset K of $cl(\psi)$ is called locally consistent if it satisfies the following requirements:

- if $(A, z_0) \in K$ then $z_0 \in Z_{acc}$;
- if $X_{c_1} f \in K$ and $X_{c_2} f \in K$ then $c_1 = c_2$,

and it is called saturated if it fulfills the rules listed below:

- if $f \vee g \in K$ then $f \in K$ or $g \in K$;
- if $f \wedge g \in K$ then $f \in K$ and $g \in K$;
- if $F_{(B,z_0)} f \in K$ then either $X_c F_{(B,\Phi(z_0,c))} \in K$ for some signal c, or $f \in K$ in the case of $z_0 \in F_{acc}$;
- if $G_{(B,z_0)} f \in K$ then $Y_c G_{(B,\Phi(z_0,c))} \in K$ for every signal c, and, moreover, f is also in K in the case of $z_0 \in F_{acc}$.

4. Build a generalized Büchi automaton $M[\pi, \varphi] = (Q \times CS(\psi), Init, \Delta, \mathbf{F})$ over the input alphabet $\mathcal{C} \times \mathcal{A}^*$, where

- $Q \times CS(\psi)$ is the set of states of the automaton,
- $Init = \{(q_0, K) : \ q_0 \in Q_0, \psi \in K\}$ is the set of initial states,
- $\Delta = \Delta_1 \cup \Delta_2 \cup \Delta_3$ is a transition relation which is defined as follows:
  - $(q, K) \xrightarrow{c,h} (q', K') \in \Delta_1$ iff 1) $q \xrightarrow{c,h} q' \in T$, 2) a set K contains at least one formulae $X_c\varphi$, and 3) $\{shift(\varphi, h) : X_c\varphi \in K \text{ or } Y_c\varphi \in K\} \subseteq K'$;
  - $(q, K) \xrightarrow{c,h} (q', K') \in \Delta_2$ iff 1) $q \xrightarrow{c,h} q' \in T$, 2) a set K does not contain any X-formulae, and 3) $\{shift(\varphi, h) : Y_c\varphi \in K\} \subseteq K'$;
  - $(q, K) \xrightarrow{c,h} (q', K) \in \Delta_3$ iff 1) $q \xrightarrow{c,h} q' \in T$, and 2) a set K does not contain neither X-formulae, nor Y-formulae.
- $\mathbf{F} = \{\mathbf{F}_\varphi : \varphi \text{ is a } F\text{-formula in } cl(\psi)\}$ is a family of acceptance conditions, where for every $\varphi = F_{(B,z)} f$ the acceptance condition $\mathbf{F}_\varphi$ is a set of all such pairs (q, K) that satisfy a requirement:
  $F_{(B,z')} shift(f, h) \in K \ \Rightarrow \ shift(f, h) \in K$.

5. Following the same line of reasoning as in [21] one could show that $M[\pi, \varphi]$ has an accepting computation iff the set $Tr(\pi)$ includes a trace tr such that $tr \models \psi$. Thus, $M[\pi, \varphi]$ is empty iff $Tr(\pi) \models \varphi$. $\square$

Since emptiness of generalized Büchi automata can be checked in polynomial time we arrived at

Corollary 1. *Regular models checking of sequential reactive systems can be performed effectively in time polynomial of the size of a model (finite state transducer) and double exponential of the size of a specification (Reg-LTL formula).*

5 Conclusion

The main contribution of this paper is twofold:

1. we introduce a new framework for formal verification of sequential reactive systems; it includes a concept of finite state transducer over semigroups as a formal model of sequential reactive systems, and a formal language for specifying behaviour of transducers.
2. we set up a model checking problem for finite state transducers operating over semigroups and show that conventional model checking techniques is applicable to this problem (at least in the case of transducers over free monoids).

There are questions and problems that still remain open for further research. What is an expressive power of $\mathcal{LP}\text{-}LTL$? We surmise that some $\mathcal{LP}\text{-}LTL$-specific operators could be introduced to make this language more convenient in practice. We believe also that other temporal logics (say, CTL) could be also adapted appropriately for specification of sequential reactive systems behaviour. Model checking algorithm presented in Theorem 1 needs further improvement. To this end complexity issues of $\mathcal{LP}\text{-}LTL$ need to be studied. We are sure that a more advanced on-the-fly approach used in LTL model checking [10] could be applied to efficient verification of transducers against $\mathcal{LP}\text{-}LTL$. In this paper we presented in some details a solution to verification problem for finite state transducers over free semigroups. But we believe that this result can be extended further to comprise the cases of partially commutative semigroups (traces [9]), free groups and free Abelian groups.

References

1. Alur R., Cerny P.: Streaming transducers for algorithmic verification of single-pass list-processing programs. Proc. of 38-th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (2011), p. 599-610.
2. Alur R., Moarref S., and Topcu U.: Pattern-based refinement of assume-guarantee specifications in reactive synthesis. Proc. of 21-st International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2015.
3. Blattner M, Head T.: Single-valued a-transducers. Journal of Computer and System Sciences. **15** (1977), p. 310-327.
4. Blattner M, Head T.: The decidability of equivalence for deterministic finite transducers. Journal of Computer and System Sciences. **19** (1979), p. 45-49.
5. Bouajjani A., Jonsson B., Nilsson M., Touili T.: Regular Model Checking. Proc. of 12-th International Conference on Computer Aided Verification, LNCS **1855** (2000), p. 403-418.
6. M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford.: A NICE way to Test OpenFlow Applications. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, April 2012, p. 1-10.
7. Chemeritsky E. V., Smeliansky R. L., Zakharov V. A.: A formal model and verification problems for software defined networks. Automatic Control and Computer Sciences. **48** (2014), p. 398-406.

8. Culik K., Karhumaki J.: The equivalence of finite-valued transducers (on HDTOL languages) is decidable. Theoretical Computer Science. **47** (1986), p. 71-84.
9. Diekert V., Metivier Y.: Partial commutation and traces. Handbook of Formal Languages. **3** (1997), p. 457-533.
10. Gerth R., Peled D., Vardi M. Y., Wolper P.: Simple on-the-fly automatic verification of linear temporal logic. Proc. of 15-th IFIP International Symposium on Protocol Specification, Testing and Verification, (1995), p 3-18.
11. Griffiths T.: The unsolvability of the equivalence problem for ε-free nondeterministic generalized machines. Journal of the ACM **15** (1968), p. 409-413.
12. Ibarra O.: The unsolvability of the equivalence problem for Efree NGSM's with unary input (output) alphabet and applications. SIAM Journal on Computing, 1978, v. 4.
13. Kesten Y., Manna Z., McGuire H., Pnueli A.: A decision algorithm for full propositional temporal logic. Proc. of 5-th International Conference on Computer Aided Verification, LNCS **697** (1993), p. 97-109.
14. Mohri M.: Finite-state transducers in language and speech processing. Computational Linguistics. **23** (1997), p. 269-311.
15. Reutenauer C., Schuzenberger M.P.: Minimization of rational word functions. SIAM Journal of Computing. **30** (1991), p. 669-685.
16. Sakarovitch J., de Souza R.: On the decomposition of k-valued rational relations. Proc. of 25-th International Symposium on Theoretical Aspects of Computer Science. (2008), p.621-632.
17. Sakarovitch J., de Souza R.: On the decidability of bounded valuedness for transducers. Proc. of the 33-rd International Symposium on MFCS. (2008), p. 588-600.
18. Schutzenberger M. P.: Sur les relations rationnelles. Proc. of Conference on Automata Theory and Formal Languages. (1975), p. 209-213.
19. de Souza R.: On the decidability of the equivalence for k-valued transducers. Proc. of 12-th International Conference on Developments in Language Theory. (2008), p. 252-263.
20. Thakkar J., Kanade A., Alur R.: A transducer-based algorithmic verification of retransmission protocols over noisy channels. Proc. of IFIP Joint International Conference on Formal Techniques for Distributed Systems, LNCS, **7892** (2013), p. 209-224.
21. Vardi M.Y., Wolper P.: Reasoning about infinite computations. Information and Computation. **115** (1994), p. 137.
22. Veanes M., Hooimeijer P., Livshits B., et al.: Symbolic finite state transducers: algorithms and applications. Proc. of the 39-th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. ACM SIGPLAN Notices. **147** (2012), p. 137-150.
23. Weber A.: Decomposing finite-valued transducers and deciding their equivalence. SIAM Journal on Computing. **22** (1993), p. 175-202.
24. Wolper P., Boigelot B.: Verifying systems with innite but regular state spaces. Proc. 10-th Int. Conf. on Computer Aided Verication (CAV-1998). LNCS. **1427** (1998), p. 8897.
25. Zakharov V.A.: Equivalence checking problem for finite state transducers over semigroups. Proc. of the 6-th International Conference on Algebraic Informatics (CAI-2015). LNCS. **9270** (2015), p. 208-221.