On the model checking of finite state transducers over semigroups

A.R. Gnatenko<gnatenko.cmc@gmail.com>
V.A. Zakharov<zakh@cs.msu.su>

Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia
National Research University High School of Economics, 20, Myasnitskaya str., Moscow, 101000, Russia

Abstract. Sequential reactive systems represent programs that interact with the environment by receiving signals or requests and react to these requests by performing operations with data. Such systems simulate various software like computer drivers, real-time systems, control procedures, online protocols etc. In this paper, we study the verification problem for the programs of this kind. We use finite state transducers over semigroups as formal models of reactive systems. We introduce a new specification language LP-CTL* to describe the behavior of transducers. This language is based on the well-known temporal logic CTL* and has two distinguished features: 1) each temporal operator is parameterized with a regular expression over input alphabet of the transducer, and 2) each atomic proposition is specified by a regular expression over the output alphabet of the transducer. We develop a tabular algorithm for model checking of finite state transducers over semigroups against LP-CTL* formulae, prove its correctness, and estimate its complexity. We also consider a special fragment of LP-CTL* language, where temporal operators are parameterized with regular expressions over one-letter alphabet, and show that this fragment may be used to specify usual Kripke structures, while it is more expressive than usual CTL*.

Keywords: reactive program; transducer; verification; model checking; temporal logic; finite state automaton; regular language

DOI: 10.15514/ISPRAS-2018-30(3)-21


1. Introduction

Finite state machines are widely used in the field of computer science and formal methods for various purposes. While finite automata represent regular sets, transducers stand for regular (or, rational) relations and, therefore, can serve as models of programs and algorithms that operate with input and output data. For example, transducers are used as formal models in software engineering to represent numerous algorithms, protocols and drivers that manipulate with strings, dataflows, etc. [1, 15, 25].

By extending the concept of ordinary transducers, we build a new formal model for sequential reactive systems. These systems are software programs or hardware devices that receive requests (control signals, commands) from the environment and perform in response some manipulations (actions, transformations) with data, interactions with the environment, mechanical movements, etc. While the flow of requests can be represented by finite or infinite words in some fixed alphabet, the sequence of actions of the system needs a more sophisticated interpretation. The key point here is that different sequences of actions may bring a computing system to the same result. To capture this effect the collection of actions performed by a reactive system can be viewed as a generating set of some algebraic structure (e.g. semigroup, group, ring, etc.) and particular algebraic properties of basic actions should be taken into account when choosing adequate formal models for this class of information processing systems. Let us illustrate this consideration by several examples.

- A network switch with several input and output ports. A switch is a device, which receives data packets on its input port, modifies their heads and commutes them to one of the output ports. Once received a special control signal, this switch changes its packet-forwarding table and, thus, its behaviour. Since packets from different flows may be processed in any order, the switch can be modeled by a transducer, which operates over a free partially commutative semigroup, or a trace monoid. Trace monoids are commonly used as an algebraic foundation of concurrent computations and process calculi (see, e.g., [9]).

- A real-time device that control the operation of some industrial equipment (say, a boiling system). Such device receives data from temperature and pressure sensors and switches some processes on and off according to its instructions and the current state of the system. It seems reasonable that for some actions the order of their implementation is not important (routine actions), while others must follow in a strictly specified order (e.g. an execution of some complex operation). Moreover, there are also actions which bring system to certain predefined operation mode (set-up actions). These actions are implemented in the emergency situations. A partially commutative semigroup with right-zero elements 0 which satisfy the equalities $x0 = 0$ for every element x provides an adequate interpretation for such operations.

- A system supervisor that maintains a log-file. For each entry its date and time is recorded in the file and there is no way to delete entries from the log — only to append it. Thus, for any two basic actions (record operations to the log-file) it is crucial in which order they are performed and such a supervisor can be modeled by a transducer over a free semigroup. Verification techniques for such reactive systems are considered in [17]; this is the main topic of this paper as well.

- A radio-controlled robot, that moves on a surface. It can make one step moves in...
any of direction. When it receives a control signal in a state $q'$ it must choose and carry out a sequence of steps and enter to the next state $q''$. At some distinguished state $q_f$ the robot reports its current location. Movements of the robot may be regarded as basic actions, and the simplest model of computation which is suitable for analyzing a behaviour of this robot is a nondeterministic finite state transducer operating on a free Abelian group of rank 2.

To construct a reliable system or network it is crucial for its components to have a correct behaviour. For example, a network switch must process received data packets within a specified number of execution steps and the boiling system should never be overheated, that is, will never remain for a long time in a particular condition without appropriate responses from the control device. By using finite state transducers as formal models of reactive systems, one can develop verification algorithms for these models to solve such problems as equivalence checking, deductive verification or model checking.

The study of the equivalence checking problem for classical transducers began in the early 60s. It was established that the equivalence checking problem for non-deterministic transducers is undecidable [13] even over 1-letter input alphabet [16]. However, the undecidability displays itself only in the case of unbounded transductions when an input word may have arbitrary many images. The equivalence checking problem was shown to be decidable for deterministic [4], functional (single-valued) [5, 19], and k-valued transducers [6, 26]. In a series of papers [20-22] techniques for checking bounded valuedness, k-valuedness and equivalence of finite state transducers over words were developed. Recently in [29] 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 [23] 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 [25] extended 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 program model.

Meanwhile, very few papers on the model checking problem for transducers are known. Transducers can be conveniently used as auxiliary means in regular model checking of parameterized distributed systems where configurations are represented as words over a finite alphabet. In such models, a transition relation on these configurations may be regarded as a rational relation and, thus, it may be specified by finite state transducers (see [7, 28]). In these papers, finite state transducers just play the role of verification instrument, but not an object of verification. However, as far as we know, a deeper investigation of the model checking problem for the reactive systems represented by transducers has not yet been carried out. We think that this is due the following main reason. A transducer is a model of computation which, given an input word, computes an output word. The letters of input and output alphabets can be regarded as valuations (tuples of truth values) of some set of basic predicates. Therefore, a transducer can be viewed as some special representation of a labeled transition system (Kripke structure) (see [2]). From this viewpoint model checking problem for finite state transducers conform well to standard model checking scheme for finite structures, and, hence, it is not worthy of any particular treatment.

However, our viewpoint is quite different. Transducer is a more complex model of computation than a finite state automaton (transition systems). Its behaviorism characterized by the correspondence between input and output words. A typical property of such behaviour to be checked is whether for every (some) input word from a given pattern a transducer outputs a word from another given pattern. Therefore, when formally expressing the requirements of this kind one needs not only temporal operators to specify an order in which events occur but also some means to refer to such patterns. Conventional Temporal Logics like LTL or CTL are not sufficient in this case; they should be modified in such a way as to acquire an ability to express correspondences between the sets (languages) of input words and the sets (languages) of output words. This could be achieved by supplying temporal operators with parameters. Every such pattern is a formal description of a language $L$ over an input alphabet $\gamma$; automata, formal grammars, regular expressions, language equations are suitable for this purpose. The basic properties of output words can be also represented by languages over an output alphabet. Then, for instance, an expression $G:\gamma^*P$ can be understood as the requirement that for every input word $w$ from the language $L$ the output word $h$ of a transducer belongs to the language $P$.

The advantages of this approach are twofold. On the one hand, such extensions of Temporal Logics make it possible to express explicitly relationships between input and output words and specify thus desirable behaviours of transducers. On the other hand, it can be hoped that such extensions could rather easily assimilate some well-known model checking techniques (see [3, 8]) developed for conventional Temporal Logics. The first attempt to implement this approach was made in [17]. The authors of this paper introduced an $L^P-LTL$ specification language based on $LTL$ temporal logic and developed a checking procedure for transducers over free monoids against specifications from $L^P-LTL$. It was shown that this procedure has double exponential time complexity.

In this paper we continue this line of research and "raise" the specification language introduced in [17] to the level of $L^P-CTL$. We will focus only on one task related to the use of new logic for the verification of reactive systems, namely, the development of a general model checking algorithm for finite state transducers against specifications in $L^P-CTL$. Such issues as expressive power of $L^P-CTL$, complexity
of model checking and satisfiability checking problems, the influence of types of languages used as parameters and basic predicates in \( \text{LP-}\text{CTL}^* \) on decidability and complexity of model checking problem remain topic of our further research and will be covered in our subsequent works. We also leave beyond this work a number of applied questions, which are worthy of consideration in a separate paper. For example, it is important to understand to what extent the already developed model checking tools can be adapted to the new temporal logic. And, of course, in the future we will have a well-chosen series of examples that illustrate the new possibilities of using \( \text{LP-}\text{CTL}^* \) to describe the behavior of reactive systems.

The paper is organized as follows. In Section 2, we define the concept of finite state transducer over semigroup as a formal model of sequential reactive systems (see [29]) and in Section 3, we describe the syntax and the semantics of \( \text{LP-}\text{CTL}^* \) as a formal language for specifying behaviour of sequential reactive systems. In Section 3 we also set up formally model checking problem for finite state transducers against \( \text{LP-}\text{CTL}^* \) formulae. In Section 4, we present an \( \text{LP-}\text{CTL}^* \) model checking algorithm for the case when parameters of temporal operators and basic predicates are regular languages represented by finite state automata. The model checking algorithm we designed has time complexity which is linear of the size of a transducer but exponential of the size of \( \text{LP-}\text{CTL}^* \) formula. This complexity estimate is in contrast to the case of conventional \( \text{CTL} \) model checking: its time complexity is linear of both the size of a model and the size of a \( \text{CTL} \) formula. To explain this effect in Section 5 we show how \( \text{LP-}\text{CTL}^* \) formulae can be also checked on the conventional Kripke structures. Finally, we compare \( \text{LP-}\text{CTL}^* \) with some other known extensions Temporal Logics and discuss some topics for further research.

2. Finite state transducers as models of reactive systems

In this section, we introduce a Finite State Transducer as a formal model of a reactive computing system which receives control signals from the environment and reacts to these signals by performing operations with data. Let \( \mathcal{C} \) be a finite set of signals. Finite words over \( \mathcal{C} \) are called signal flows; the set of all signal flows is denoted by \( \mathcal{C}^* \). Given a pair of signal flows \( u \) and \( v \) we write \( uv \) for their concatenation, and denote by \( \varepsilon \) the empty flow.

Let \( \mathcal{A} = \{a_1, ..., a_n\} \) be a finite set of elements called basic actions; these actions stand for the elementary operations performed by a reactive system. Finite words over \( \mathcal{A} \) are called compound actions; they denote sequential compositions of basic actions. Since different sequences of basic actions could produce the same result, one may interpret compound actions over a semigroup \( (S, \ast, \varepsilon) \) generated by a set of basic actions \( \mathcal{A} \). The elements of \( S \) are called data states. Every compound action \( h = a_{i_1}a_{i_2}...a_{i_k} \) is evaluated by the data state \( [h] = [a_{i_1}] \ast [a_{i_2}] \ast ... \ast [a_{i_k}] \). For example, if a reactive system just keeps a track of input requests by adding certain records to a log-file then a free semigroup will be suitable for interpretation of these operations. But when a robot moves on a 2-dimensional surface then the actions (movements) performed by this robot may be regarded as generating elements of Abelian group \( G \) of rank 2, and the positions on the surface occupied by this robot can be specified by the elements from \( G \). In this paper we restrict ourselves to the consideration of free semigroups when \( |h| = h \) holds for every compound action \( h \), and \( \varepsilon \) is the word concatenation operation.

Let \( \mathcal{C} \) be a set of signals and \( \mathcal{A} \) be a set of basic actions that are interpreted over a semigroup \((S, \ast, \varepsilon)\). Then a Finite State Transducer (in what follows, FST) is a quintuple \( \Pi = (Q, \mathcal{C}, \mathcal{A}, q_{\text{init}}, T) \), where

- \( Q \) is a finite set of control states;
- \( q_{\text{init}} \in Q \) is an initial control state;
- \( T \subseteq Q \times \mathcal{C} \times Q \times \mathcal{A} \) is a finite transition relation.

Each tuple \( (q', c, q'', h) \) in \( T \) is called a transition: when a transducer is in a control state \( q' \) and receives a signal \( c \), it changes its state to \( q'' \) and performs a compound action \( h \). We denote such transition by \( q' \xrightarrow{c, h} q'' \). A run of a FST \( \Pi \) is any finite sequence of transitions

\[
q_1 \xrightarrow{c_1, h_1} q_2 \xrightarrow{c_2, h_2} q_3 \xrightarrow{c_3, h_3} ... \xrightarrow{c_n, h_n} q_{n+1},
\]

this run transduces a signal flow \( w = c_1c_2...c_n \) into a data state \( [h_1h_2...h_n] \).

The behaviour of a FST \( \Pi = (Q, \mathcal{C}, \mathcal{A}, q_{\text{init}}, T) \) over a semigroup (data space) \((S, \ast, \varepsilon)\) is presented formally by a transition system \( TS(\Pi, S) = (D, \mathcal{C}, d_{\text{init}}, T') \), where

- \( D = Q \times S \) is (in general case, infinite) set of states of computation,
- \( d_{\text{init}} = (q_{\text{init}}, \varepsilon) \) is the initial state, and
- \( T' \subseteq D \times \mathcal{C} \times D \) is a transition relation such that for every states of computation \( d' = (q', s'), d'' = (q'', s'') \) and every signal \( c \in \mathcal{C} \) the relationship

\[
(d', c, d'') \in T' \quad \Rightarrow \quad \exists h \in \mathcal{A}^* (q', c, q'', h) \in T \text{ and } s'' = s' \ast [h]
\]

holds.

As usual, a transition \((d', c, d'') \in T' \) is denoted by \( d' \xrightarrow{c} d'' \). A trajectory in a transition system \( TS(\Pi, S) \) is a pair \( tr = (d_0, \alpha) \), where \( d_0 \in D \) and \( \alpha = (c_1, d_1), (c_2, d_2), ..., (c_i, d_i) \) is a sequence of pairs \((c_i, d_i)\) such that \( d_{i-1} \xrightarrow{c_i} d_i \) holds for every \( i \geq 1 \). A trajectory represents a possible scenario of a behaviour of a sequential reactive system: when receiving a signal flow \( c_1c_2...c_p \), the reactive system performs a sequence of basic actions \( h \) and follows sequentially via the states of computation \( d_1, d_2, ..., d_p \). By \( tr \rceil \alpha \) we mean the trajectory\((d_1, \alpha \rceil \alpha)\), where \( \alpha \rceil = (c_{i+p}, d_{i+p}), (c_{i+p+2}, d_{i+p+2}), ... \) is a suffix of \( \alpha \), respectively.
3. **LP-CTL** specification language

When designing sequential reactive systems one should be provided with a suitable formalism to specify the requirements for their desirable behaviour. For example, one may expect that

- a mobile robot, receiving an equal number of control signals "go_left" and "go_right", will always return to its original position,
- a network switch will never commute data packets from different packet flows into the same output buffer,
- it is not possible for the interrupt service routine to complete the processing of one interrupt before it receives a request to handle another.

These and many other requirements which refer to the correspondences between control flows and compound actions in the course of FST runs can be specified by means of Temporal Logics. When choosing a suitable temporal logic as a formal specification language of FST behaviours one should take into account two principal features of our model of sequential reactive systems:

- since a FST operates over a data space which is semigroup, the basic predicates must be interpreted over semigroups as well, and
- since a behaviour of a FST depends not on the time flow itself but on a signal flow which it receives as an input, temporal operators must be parameterized by certain descriptions of admissible signal flows.

To adapt traditional temporal logic formalism to these specific features of FST behaviours the authors of [17] introduced a new variant of Linear Temporal Logic (**LTL**). We assume that in general case one may be interested in checking the correctness of FST's responses to arbitrary set of signal flows. Every set of control flows may be regarded as a language over the alphabet of control signals. Therefore, it is reasonable to supply temporal operators ("globally" \(G\), "eventually" \(F\), etc.) with certain descriptions of such languages as parameters. In more specific cases we may confine ourselves with considering only a certain family of languages (finite, regular, context-free, etc.) \(L\) used as parameters of temporal operators. These languages will be called environment behaviour patterns.

A reactive system performs finite sequences of basic actions in response to control signals from the environment and thus follows in the course of its run via a sequence of data states, which are elements of a semigroup \((S, \circ)\). Therefore, basic predicates used in **LTL** formulae may be viewed as some sets of data states \(S', S'' \subseteq S\). These sets can be also specified in language-theoretic fashion. Any language \(P\) over the alphabet of basic actions \(A\) corresponds to a predicate (set of data states) \(S_p = \{[h] \mid h \in P\}\). As in the case of environment behaviour patterns we may distinguish a certain class \(P\) of languages and use them as specifications of basic predicates. When these languages are used as parameters in temporal formulae then it will be assumed that they are defined constructively by means of automata, grammars, Turing machines, etc.

Thus, we arrive at the concept of **LP**-variants of Temporal Logics. In [17] the syntax and semantics of **LP-LTL** was studied in some details in the case when both environment behaviour patterns and basic predicates are regular languages presented by finite automata. In this paper we make one step further and extend the concept of **LP**-variants of Temporal Logics to **CTL**. Select an arbitrary family of environment behaviour patterns \(L\) and a family of basic predicates \(P\). The set of **LP-CTL** formulae consists of state formulae and trajectory formulae, which are defined as follows:

- each basic predicate \(P \in P\) is a state formula;
- if \(\varphi_1, \varphi_2\) are state formulae then \(\neg \varphi_1, \varphi_1 \land \varphi_2\) and \(\varphi_1 \lor \varphi_2\) are state formulae;
- if \(\psi\) is a trajectory formula then \(A\psi\) and \(E\psi\) are state formulae;
- if \(\varphi\) is a state formula then \(\varphi\) is a trajectory formula;
- if \(\psi_1, \psi_2\) are trajectory formulae then \(\neg \psi_1, \psi_1 \land \psi_2\) and \(\psi_1 \lor \psi_2\) are trajectory formulae;
- if \(\varphi_1, \varphi_2\) are state formulae, \(c \in C\), and \(L \in L\) then \(X_c \varphi, Y_c \varphi, F_L \varphi, G_L \varphi\) and \(\varphi_1 U \varphi_2\) are trajectory formulae.

The specification language **LP-CTL** is the set of all state formulae constructed as defined above.

Now we introduce the semantics of **LP-CTL** formulae. These formulae are interpreted over transition systems. Let \(M = (TS(\Pi, S), d)\) be a transition system, \(d\) is a state of computation in this system, and \(tr\) be a trajectory in \(M\). Then for every state formula \(\varphi\) we write \(M, d \models \varphi\) to denote the fact that the assertion \(\varphi\) is true in the state \(d\) of \(M\), and for every trajectory formula \(\psi\) we write \(M, tr \models \psi\) to denote the fact that the assertion \(\psi\) holds for the trajectory \(tr\) in \(M\).

In the definition below it is assumed that \(M\) is a transition system, \(d = (q, s)\) is a state of computation in \(M\), and \(tr = (d_0, \alpha)\) is a trajectory in \(M\) such that \(\alpha = (c_1, d_1), (c_2, d_2), ..., (c_i, d_i), \ldots\). We define the satisfiability relation \(\models\) by induction on the height of formulae:

- \(M, d \models P \iff s \in P\);
- \(M, d \models \neg \varphi \iff \text{it is not true that } M, d \models \varphi\);
- \(M, d \models \varphi_1 \land \varphi_2 \iff M, d \models \varphi_1\) and \(M, d \models \varphi_2\);
- \(M, d \models E\varphi \iff \text{there exists a trajectory } tr' = (d, \alpha') \text{ in } M\) such that \(M, tr' \models \varphi\);
- \(M, d \models A\varphi \iff \text{for any trajectory } tr' = (d, \alpha') \text{ in } M\) \(it\text{'s true that } S M, tr' \models \varphi\);
- \(M, tr \models \psi_1 \land \psi_2 \iff M, tr \models \psi_1\) and \(M, tr \not\models \psi_2\);
For every formula $\varphi$ the following equalities hold

1. $\equiv AX_c \varphi \equiv \neg E_c \neg \varphi$
2. $\equiv AY_c \varphi \equiv \neg E_c \neg \varphi$
3. $\equiv AF_c \varphi \equiv \neg E_G \neg \varphi$
4. $\equiv EF_c \varphi \equiv E[true U_c \varphi]$
5. $\equiv AG_c \varphi \equiv \neg EF_c \neg \varphi$
6. $\equiv A[\varphi U_c \varphi] \equiv \neg E[\neg \psi U_c (\neg \varphi \land \neg \psi)] \land \neg E_G \neg \psi$

Certainly, some other relationships like fixed-point identities are also valid in $\mathcal{LPT}-\mathcal{CTL}^*$ (see [17]) but they will not be involved in this paper.

We can now bound our consideration with those $\mathcal{LPT}-\mathcal{CTL}^*$ formulae which are constructed using only $\neg, \land, \lor, \Rightarrow, \Leftrightarrow$ of $\mathcal{LPT}-\mathcal{CTL}^*$. It should be noticed that $\mathcal{M}$ is, in general, infinite. Therefore, to obtain an effective model checking procedure we need a construction that will model the behaviour of $\mathcal{M}$ w.r.t. target formula $\varphi$.

For every basic predicate $P \in \mathcal{P}$ let $A_P = (Q_P, A_{init}, \delta_P, F_P)$ be a minimal DFA recognizing this language. Here $Q_P$ is a finite set of states, $init_P$ is an initial state, $F_P$ is a set of accepting states and $\delta_P : Q_P \times A \rightarrow Q_P$ is a transition function.

The latter can be extended to the set $A^*$ in the usual fashion:

$$\delta_P(q_P, \varepsilon) = q_P \quad \text{and} \quad \delta_P(q_P, \gamma a) = \delta_P(\delta_P(q_P, \gamma), a).$$

Let $P_1, P_2, \ldots, P_k$ be all basic predicates occurred in the formula $\varphi$. Given a transducer $\Pi = (Q, \mathcal{C}, A_{init}, T)$ and a formula $\varphi$, we build a checking machine --- a transducer $\mathcal{M} = (Q, \mathcal{C}, A, q_{init}, T)$, where

$\hat{Q} = Q \times Q_P \times \cdots \times Q_P$ is a set of states (to avoid misunderstanding we will call them metastates);

$\hat{q}_{init} = (q_{init}, init_{P_1}, \ldots, init_{P_k})$ is an initial metastate;

$\hat{T} \subseteq \hat{Q} \times \mathcal{C} \times \hat{Q} \times A^*$ is a transition relation, such that:

$$(\hat{q'}, c, \hat{q'', h}) \in \hat{T} \quad \text{and} \quad \delta_P(q'' p_j, h) = q''_{P_j} \quad \text{for all} \ j \leq 1 \leq k.$$
automata $A_{P_1}, ..., A_{P_k}$ in response to every signal $c$. Recall that the elements of the free monoid are words $s$ from $A^*$. The checking machine $MC$ induces a relation $\sim$ on the set $D$: for an arbitrary pair $d' = (q', s')$ and $d'' = (q'', s'')$ of states of computation of $\Pi$ over $A^*$

$$d' \sim d'' \iff \left( \delta_{P_j}(\text{init}_{P_j}, s') = \delta_{P_i}(\text{init}_{P_i}, s'') \text{ for all } j \right).$$

The relation $\sim$ is clearly an equivalence relation of finite index, and every equivalence class of states of computation in $MC$ corresponds to a metastate of the checking machine $MC$. As it can be seen from the definition of $\sim$, if two states of computation $d'$ and $d''$ are equivalent and there is a trajectory $tr' = (d', \alpha')$ in $M$, where $\alpha' = (c_1, d_1'), (c_2, d_2'), ..., $ from one of these states, then there is also a corresponding trajectory $tr'' = (d'', \alpha'')$, where $\alpha'' = (c_1, d_1''), (c_2, d_2''), ...$ from the other state, such that $d'_i \sim d''_i$ holds for every $i, i \geq 1$. Actually, this means that $\sim$ is a bisimulation relation on the state space of the transition system $M$. It is well known (see [3, 8]) that bisimulation preserves the satisfiability of CTL formulae. The Proposition below shows that the same is true for $LP-CTL$. This means that the checking machine provides a finite contraction of the infinite transition system $M = TS(\Pi, A^*)$ w.r.t. satisfiability of $LP-CTL$ formulae.

**Proposition 3.** Suppose that $d'$ and $d''$ are two states of computation in $MC$ such that $d' \sim d''$. Then $M, d' \models \varphi \iff M, d'' \models \varphi$.

**Proof.** It is carried out by induction on the nesting depth of $\varphi$. If $\varphi$ is a basic predicate the assertion follows from the definition of $\sim$. The cases when $\varphi = \neg \psi$ and $\varphi = \psi_1 \wedge \psi_2$ are obvious. We focus only on the case of $\varphi = E[\psi_1, \chi]$: the other cases when $\varphi$ is of the form $EX, \psi, YE, \psi$, or $EG, \psi$ can be treated similarly.

Suppose that $M, d' \models E[\psi_1, \chi]$. Then, by the definition of $LP-CTL$ semantics, there exists a trajectory $tr' = (d', \alpha')$ such that $M, tr' = \psi_1 U \chi$ and $\alpha' = (c_1, d_1'), (c_2, d_2'), ...$. As it was noticed above, there is also a corresponding trajectory $tr'' = (d'', \alpha'')$ in $M$, where $\alpha'' = (c_1, d_1''), (c_2, d_2''), ...$, such that $d'_i \sim d''_i$ holds for every $i, i \geq 1$. Then, by induction hypotheses, $M, d'_1 \models \psi \iff M, d''_1 \models \psi$ and $M, d'_i \models \chi \iff M, d''_i \models \chi$ hold for every $i, i \geq 1$.

Since $M, tr' = \psi_1 U \chi$, there exists $i$ such that

1. $c_1 c_2 ... c_i L \in L$ and $M, tr'^i \models \chi$;
2. for all $j < i$ if $c_1 c_2 ... c_j L$ then $M, tr'^j \models \psi$.

However, taking into account the fact that $\psi$ and $\chi$ are state formulas, we must recognize that $M, tr''^j \models \chi$ and $M, tr''^i \models \psi$ every time when $M, tr'^i \models \psi$. Thus, we arrive at the conclusion that $M, tr''^j \models \psi_1 U \chi$ and, hence, $M, d'' \models E[\psi_1, \chi]$.

Each metastate $\hat{q} = (q_0, q_1, ..., q_k)$ of the checking machine $MC$ represents an equivalence class $D_{\hat{q}}$ which includes all states $d = (q, h) \in D$ such that $q = q_0$ and $\delta_{P_j}(\text{init}_{P_j}, h) = q_j$ for all $j, 1 \leq j \leq k$. By using Proposition 3, we can correctly introduce a new satisfiability relation $\models_0$ on the metastates of the checking machine:

$$\hat{q} \models_0 \varphi \iff \text{for some } d \in D_{\hat{q}}: M, d \models \varphi.$$
recognizes the language $L$. Then the nodes of $\Gamma_0(M, L)$ are all pairs $(\bar{q}, q_L) \in \bar{Q} \times Q_L$. This DLG has an arc of the form $(\bar{q}', q_L') \xrightarrow{c, h} (\bar{q}'', q_L'')$ iff $\bar{q}' \xrightarrow{c, h} \bar{q}''$ is a transition of $M$ and $\delta_t(q_L', c) = q_L''$.

We then delete all those nodes $(\bar{q}, q_L)$ of $\Gamma_0(M, L)$ for which the relations $\psi \notin \text{label}(\bar{q})$ and $q_L \in F_L$ hold simultaneously and discard all arcs incoming to or outgoing from such nodes. A DLG thus reduced is denoted by $\Gamma_0'(M, L)$.

A subformula $E[\psi U_t \chi]$ is added to the set $\text{label}(\bar{q})$ iff $\Gamma_0'(M, L)$ includes the node $(\bar{q}, \text{init}_t)$ and there exists a directed path in this graph from this node to some node $(\bar{q}', q_L')$ such that $\chi \in \text{label}(\bar{q}')$ and $q_L' \in F_L$.

For a subformula $E G \psi$ we construct a DLG $\Gamma_G(M, L)$ in the same fashion and delete all the nodes $(\bar{q}, q_L)$ for which the relations $\psi \notin \text{label}(\bar{q})$ and $q_L \in F_L$ hold simultaneously. As the result we obtain the reduced DLG $\Gamma_0'(M, L)$.

The subformula $E G \psi$ is added to the set $\text{label}(\bar{q})$ iff $\Gamma_0'(M, L)$ includes the node $(\bar{q}, \text{init}_t)$ and there exists a directed path in this graph from this node to some nontrivial strongly connected component (SCC), that is, to a subgraph, every node of which is reachable from any other node by some non-empty path.

As soon as all the subformulae from $\text{Sub}(\varphi)$ (including the formula $\varphi$) are processed we obtain the result of the model checking as

$$\Pi \models \varphi \iff \varphi \in \text{label}(\bar{q}_{\text{init}}).$$

The correctness of this assertion is based on the following relationship: $\bar{q} \models_0 \varphi \iff \varphi \in \text{label}(\bar{q})$. It can be proved by applying induction on the nesting depth of formulae with the help of Proposition 3. We also need Propositions 4 and 5 to justify the induction step for formula $E[\psi U_t \chi]$ and $E G \psi$.

Suppose, for every metastate $\bar{q} \in \bar{Q}$ it is true that $\bar{q} \models_0 \psi \iff \psi \in \text{label}(\bar{q})$ and $\bar{q} \models_0 \chi \iff \chi \in \text{label}(\bar{q})$. This statement is used as an inductive hypothesis.

**Proposition 4.** Let $\bar{q}_0 \in \bar{Q}$ be an arbitrary metastate in $M$. Then $\bar{q}_0 \models_0 E[\psi U_t \chi]$ iff some node $(\bar{q}', q_L')$ in $\Gamma_0'(M, L)$, such that $\bar{q}' \models_0 \chi$ and $q_L' \in F_L$, is reachable from the node $(\bar{q}_0, \text{init}_t)$ by a directed path.

**Proposition 5.** Let $\bar{q}_0 \in \bar{Q}$ be an arbitrary metastate in $M$. Then $\bar{q} \models_0 E G \psi$ iff some nontrivial strongly connected component is reachable from the node $(\bar{q}, \text{init}_t)$ in $\Gamma_0'(M, L)$ by a directed path.

The proofs of these Propositions are straightforward adaptations of the correctness proof of the tabular model checking algorithm for $\text{CTL}$ which is discussed in much greater details in [8]. However, for completeness of the exposition we give here a proof of Proposition 5. The proof of Proposition 4 follows the similar line of reasoning.

**Proof of Proposition 5 (Sketch).**
that basic predicates \( \{ p_i \}_{i=1}^k \) occurred in \( \varphi \) are recognized by minimal DFAs \( \{ A_{L_i} = (Q_{L_i}, \delta_{L_i}, \epsilon_{L_i}, F_{L_i}) \}_{i=1}^k \). Suppose also that environment patterns \( \{ L_i \}_{i=1}^k \) used in \( \varphi \) are recognized by minimal DFAs \( \{ A_{L_i} = (Q_{L_i}, \delta_{L_i}, \epsilon_{L_i}, F_{L_i}) \}_{i=1}^k \). Then the size of \( \varphi \) is the sum \( \| \varphi \| = |\text{Sub}(\varphi)| + \sum_{i=1}^k |Q_{L_i}| + \sum_{i=1}^k |Q_{L_i}| \).

As it can be seen from the description of our model checking algorithm, the size of auxiliary graphs \( G_{L_i}(M, L) \) and \( G_{L_i}(M, L) \) used in this algorithm does not exceed the value \( \| \varphi \| = \max |Q_{L_i}| : 1 \leq i \leq s \). These graphs are processed in no more than \( |\text{Sub}(\varphi)| \) steps. So, the total time complexity of our model checking algorithm does not exceed the value \( \| \varphi \| \cdot \max |Q_{L_i}| : 1 \leq i \leq s \) which is \( O(\| \varphi \| \cdot 2^{|\varphi|}) \).

Because of these considerations, we get the following

**Theorem 1.** Model checking of a finite state transducer \( \Pi \) operating over a free monoid against a formula \( \varphi \in \mathcal{CP}\text{-CTL}^* \) can be performed in time \( O(\| \varphi \| \cdot 2^{|\varphi|}) \).

When a more general case of model checking problem of FSTs against \( \mathcal{CP}\text{-CTL}^* \) formulae is concerned we can rely on the well-known combining approach which is based on the interleaving application of model checking algorithms for CTL and LTL. The details can be found in [8]. The similar procedure for \( \mathcal{CP}\text{-CTL}^* \) can be obtained in the same fashion by means of \( \mathcal{CP}\text{-CTL}^* \) model checking algorithm described above and \( \mathcal{CP}\text{-LTL} \) model checking algorithm developed in. Since this approach does not take into account any specific features of \( \mathcal{CP}\text{-CTL}^* \) formulae, we will not give a complete description of it.

### 5. \( \mathcal{LP}\text{-LTL}^* \) and ordinary Kripke structures

In this section, we consider the model checking problem for two subfamilies of \( \mathcal{LP}\text{-CTL}^* \) whose semantics can be defined on ordinary Kripke structures.

Recall, that a *Kripke structure* over a finite set \( AP \) of atomic propositions is a quadruple \( M = (Q, q_{init}, R, \rho) \), where \( Q \) is a finite set of states which includes an initial state \( q_{init} \), \( R \subseteq Q \times Q \) is a transition relation and \( \rho : Q \rightarrow 2^{AP} \) is a labeling function which for each state \( q \) gives a matching set \( \rho(q) \subseteq AP \) of all atomic propositions that are evaluated to *true* in this state. As usual, the size of \( M \) is the sum \( |M| = |Q| + |R| \). Below we present two modifications of \( \mathcal{LP}\text{-CTL}^* \) that are well suited for model checking of Kripke structures.

Given a Kripke structure \( M = (Q, q_{init}, R, \rho) \), consider a set of \( \mathcal{LP}\text{-CTL}^* \) formulae where \( L \) is a family of regular languages over one-letter alphabet \( \{ c \} \) and \( P = AP \) (we denote this formulae by \( \mathcal{LP}\text{-1-CTL}^* \)) and a transition system \( M_c = (Q, c, q_{init}, R_c, L) \) where \( (q', c, q'') \in R_c \) iff \( (q', q'') \in R \). Then for \( q \in Q \) the relation \( q \models P \) holds iff \( P \in \rho(q) \). The semantics of more complex formulae is defined exactly as in Section 3.

Some \( \mathcal{LP}\text{-1-CTL}^* \) formulae have an ability to keep track of the number of steps of the run. For example, an \( \mathcal{LP}\text{-1-LTL} \) formula \( A_G q \), where \( L = \{ c^n \} \) is a regular language which contains all 1-letter words of even length, expresses the assertion that \( \varphi \) holds at every even step of a run. By using the techniques of Ehrenfeucht-Fraisse games for Temporal Logics developed and studied in [11] one can prove that this property cannot be specified by means of usual LTL. This certifies that \( \mathcal{LP}\text{-1-CTL}^* \) is more expressive than \( CTL^* \) and justifies its use as a new specification language for finite state transducers and Kripke structures.

Observe, that given a set \( AP \) of all atomic propositions used in formulae we can use the \( M \) directly as a checking machine for the algorithm described in Section 4. Suppose that formula \( \varphi \) refers to 1-letter regular languages \( L_1, L_2, \ldots, L_s \) as the parameters of temporal operators, and every language \( L_i, 1 \leq i \leq s \), is recognized by a DFA with a set of states \( Q_{L_i} \). Then the size of the graphs used in this algorithm does not exceed the value \( |M| \cdot \max |Q_{L_i}| : 1 \leq i \leq s \) which is \( O(|M| \cdot \| \varphi \|) \), where \( \| \varphi \| = |\text{Sub}(\varphi)| + \sum_{i=0}^s |Q_{L_i}| \).

Another modification of the Kripke structure \( M \) allows one to encode more detailed information of the computation flow. Let \( \Sigma = 2^{AP} \). For each state \( q \in M \) there exists a letter \( \sigma(q) \in \Sigma \) corresponding to the label \( \rho(q) \) assigned to this state.

Let \( M_{AP} = (Q \cup \{ err \}, q_{init}, R_{AP}, \rho_{AP}) \) be a transition system for \( M \), where for every \( q \in Q \) the following equalities hold: \( \rho_{AP}(q) = \rho(q) \), \( \rho_{AP}(err) = \{ err \} \) and \( R_{AP} \subseteq Q \times 2^{AP} \times Q \) is a minimal transition relation such that:

- for each transition \((q', q'')\) of the Kripke structure \( M \) there exists a *fair transition* \((q', \sigma(q''), q'')\) and erroneous transitions \((q', \sigma, err)\) for each \( \sigma \neq \sigma(q'') \);
- \((err, \sigma, err) \in R_{AP}\) holds for each \( \sigma \in \Sigma \) and \((err, \sigma, q) \in R_{AP}\) holds for each \( q \neq err \).

Then consider a specification language \( \mathcal{LP}\text{-n-CTL}^* \) which is a set of all such formulae where \( L \) is a family of regular languages over \( \Sigma \) and \( P = AP \). To model check a transition system \( M_{AP} \) against these formulae one needs to process only the states in \( Q \) and only the fair transitions. To do so, we replace all state formulae of type \( A\varphi \) with \( A(G \to err \to \varphi) \) and all state formulae of type \( E\varphi \) with \( E(G \to err \land \varphi) \). The transition system \( M_{AP} \) thus obtained may as well be used as a checking machine for the model checking algorithm described in Section 4.

Thereby, the following theorem holds.

**Theorem 2.**

1. There exists an algorithm for model checking of a Kripke structure \( M \) against a formula \( \varphi \in \mathcal{LP}\text{-1-CTL}^* \) with time complexity \( O(|M| \cdot \| \varphi \|) \).
2. There exists an algorithm for model checking of a Kripke structure $M$ against a formula $\varphi \in LP$-CTL with time complexity $O(|M| \cdot |\varphi| \cdot 2^{\Delta_{API}})$.

As it can be seen from this theorem, the exponential complexity of model checking procedure described in Section 4 is due to the language-theoretic nature of basic predicates used in LP-CTL$^*$.  

6. Related papers and conclusion

Actually, the idea of providing parameterization of temporal operators is not new. In [27] right-linear grammar patterns were offered to define new temporal operators. The same kind of temporal patterns but specified by means of finite state automata were introduced in [18, 24]. For these extensions it was proved that they have the same expressiveness as SIS and that satisfiability checking problem in these logics is PSPACE-complete. We did not pursue a goal of merely expanding the expressive possibilities of CTL$^*$; our aim was to make CTL$^*$ more adequate for describing the behaviour of reactive systems. Almost the same kind of parametrization is used in Dynamic LTL. However, our extension of CTL$^*$ differs from that which was developed in [14], since in our logic basic predicates are also parameterized. The LP-CTL$^*$ formulae allows one to specify and verify the behaviour of finite state transducers that operate over semigroups as well as classical Kripke structures. Moreover, when Kripke structures are concerned LP-CTL$^*$ has more expressive power than conventional temporal logics. But the place of LP-CTL$^*$ in the expressive hierarchy of specification languages, such as SIS, PDL or $\mu$-calculus, has not yet been established and remains a matter for our further research.

The results of this paper combined with the results of [17] provide positive solution to model checking for transducers over free semigroups. Free semigroups is the most simple algebraic structure which can be used for interpretation of basic actions performed by transducers when they are regarded as formal models of sequential reactive systems. Next, we are going to find out whether model checking algorithms could be built for transducers operating over more specific semigroups. Some preliminary results showed that this is not an easy problem. In [12] we proved that it is undecidable for the case of Abelian groups and free commutative semigroups. It is also interesting how much the complexity of model checking algorithms for LP-CTL$^*$ depends on languages that are used as parameters of temporal operators. We assume that model checking problem becomes undecidable when context-free languages are allowed for this purpose. The complexity issues of model checking for regular variant of LP-CTL$^*$ also need further research. We assume that even for regular LP-CTL this problem is PSPACE-complete.

As for practical application of the results obtained, the most important issue is that of adapting the existing means of working with finite automata to widely known model checking tools (like SPIN, v-SMV, etc.) in order to be able to effectively implement the proposed model checking algorithms for LP-CTL$^*$.  

Acknowledgments

The authors of the article thank the anonymous reviewers for their valuable comments and advice on improving the article. This work was supported by the Russian Foundation for Basic Research, Grant N 18-01-00854.

References


О верификации конечных автоматов-преобразователей над полугруппами

1 А.Г. Гнатенко <gnatenko.cmc@gmail.com>
2 В.А. Захаров <zakh@cs.msu.su>

1 Московский государственный университет им. М. В. Ломоносова, 119991, Российская Федерация, Москва, Ленинские горы, д. 1
2 Национальный исследовательский университет Высшая школа экономики, 101000, Российская Федерация, Москва, ул. Мясницкая, д. 20

Аннотация. Последовательные реагирующие системы – это программы, которые взаимодействуют с окружением, получая от него сигналы или запросы, и реагируют на эти запросы, проводя операции с данными. Подобные системы могут служить моделью для многих программ: драйверов, систем реального времени, сетевых протоколов и др. В статье исследуются задача верификации программ такого вида. В качестве формальных моделей для реагирующих систем мы используем конечные автоматы-преобразователи, работающие над полугруппами. Для описания поведения автоматов-преобразователей введён новый язык спецификаций LP-CTL*. В его основе положена темпоральная логика CTL*. Этот язык спецификаций имеет две характерные особенности: 1) каждый темпоральный оператор снабжён регулярным выражением над входным алфавитом автомата, и 2) каждое атомарное высказывание задается регулярным выражением над выходным алфавитом автомата-преобразователя. В данной работе представлен табличный алгоритм проверки выполнимости формул LP-CTL* на моделях конечных автоматов-преобразователей, работающих над свободными полугруппами. Показано, что этот фрагмент алгоритма применим для спецификаций обычных моделей Кршке, а при этом его выразительные возможности превосходят обычную логику CTL*.

Ключевые слова: реагирующая система, автомат-преобразователь, верификация, проверка на модели, темпоральная логика, конечный автомат, регулярный язык.

DOI: 10.15514/ISPRAS-2018-30(3)-21

Список литературы

Гнatenко А.Р., Захаров В.А. О верификации конечных автоматов-преобразователей над полугруппами. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 303-324


