[Kleshchev, Artemieva, 2006] Kleshchev A.S., Artemieva I.L. Domain ontologies and their mathematical models. In the Proceedings of the XII-th International Conference "Knowledge-Dialog-Solution" - KDS 2006, June 20-25, Varna, Bulgaria, Sofia: FOI-COMMERGE-2006: 107-115. – ISBN 954-16-0038-7.

[Noy, 2001] Noy N., McGuinness D. Ontology Development 101 : A Guide to Creating Your First Ontology.- Knowledge Systems Lab, Stanfo. URL: http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

[Ontos Miner] Ontos Miner: Data Mining System from Text Documents in Russian. URL: http://www.avicomp.ru/rus/ontos/academic_solutions_rus.php.

[Zagorulko et al., 2006] Zagorulko Yu. A., Borovikova O.I., Kononenko I.S., Sidorova E.A. Approach to developing domain ontology for knowledge portal of computational linguistics. In Computational linguistics and intellectual technologies: Papers of International Conference “Dialogue 2006” (Bekasovo, 31 May – 4 June, 2006). – Moscow: RSUH Publishing Centre, 2006: 148-151.

Authors’ Information

Irene L. Artemieva – artemeva@iacp.dvo.ru

Institute for Automation & Control Processes, Far Eastern Branch of the Russian Academy of Sciences; 5 Radio Street, Vladivostok, Russia

ONTOLOGY VIEW ON AUTOMATA THEORY

Sergey Krivoi, Lyudmila Matveyeva, Yelena Lukianova, Olga Sedleckaya

Abstract: *The summary of automata theory ontology is presented in the paper. It is based on the following dependences: a type of an automaton – the language accepted by the automaton – applications. The given ontology does not claim to be exhaustive as automata theory is very extensive and it is a complicated problem to survey all its aspects within one article. Only the main properties of automata and their applications are considered.*

1. Introduction

Correct design and verification of hardware and software systems are very important current problems. Design problem is difficult to formalize and consequently difficult to automatize; verification problem is hard as it accumulates a variety of different tasks engaged from adjacent scientific domains. One of such adjacent scientific domains is automata theory which is applied for partial solution of design and verification problems. Automata theory plays a crucial role particularly for reactive systems properties verification since such important problems as properties recognition, a definite state or a set of states reachability, and accepted language emptiness are decidable for most types of automata.

The paper presents brief automata theory ontology based on the following dependences: ***an automaton – the language accepted by this automaton – applications***. Only the main properties of automata and their applications are considered here due to limited size of the given paper.

2. Finite Automata on Finite Words

All the main problems in theory of finite automata working on finite words are already decided, so from this point of view the theory is complete. The main results received at present in this domain have an applied nature, i.e. methods of automata theory are used now for specified applied domains tasks solving. However, finite automata theory has exerted influence on further development of general automata theory. It is showed up via numerous variations of the finite automaton notion: finite automata over infinite words [3, 12], timed automata [2], hybrid automata [9], automata over trees [10], etc.

The notion of a finite automaton on finite words (finite sets of symbols) in finite alphabet is a basic notion on which ontology is built up. Three types of such automaton are in common use: *Mealy automata, Moore automata,* and *X–automata (or automata without outputs).* Let us define these types.

Mealy Automata. Let X = {x, x,…, x } and Y = {y, y,…, y } be finite alphabets, i.e. finite sets of pair-wise different elements, which are named as symbols or signals.

Definition 1. Mealy automaton is the 5-tuple *(A, X, Y, f, g)* such that *A* is a finite set of automaton states, *X* is a finite set called the input alphabet, *Y* is a finite set called the output alphabet, $f : A \times X \to A$ is a transition function, and $g: A \times X \to Y$ is an output function.

Usually, an automaton is denoted by a symbol of a set of its states, i.e. *A = (A, X, Y, f, g).* If *f(a,x) = a'*, then it is said, that automaton *A* passes on to the state *a'* under the action of input signal $x \in X$ or the signal *x* transfers the automaton A from the state *a* to the state *a'*. If *g(a,x) = y*, then it is said, that the automata A transforms input signal $x \in X$ into output signal $y \in Y$ being in the state a.

An automaton operation is described in the following way. A signal (symbol) $x \in X$ is given at the input of the automaton, then the automaton state is changed as a result in accordance with the current automaton state and its transition and output functions; and an output signal (symbol) $y \in Y$ appeared at its output. A word (a sequence of output symbols) appears at an automaton output if a word (a sequence of input symbols) is given at its input too. One may consider in that case the automaton A as an alphabetical information transformer which maps semigroup *F(X)* words into the semigroup *F(Y)* words. One may consider the sequence of input signals as a function of the natural argument – discrete automaton time. This circumstance allows us to consider an automaton as a discrete dynamic system which changes its states in time under the action of internal and external factors.

Automaton *A* is named *initial*, if a certain state a_0 is chosen (which is named *initial (start) state*) in the automaton set of states. It is considered that initial automaton is at inititial state a_0 at initial point of time (before giving some word $p \in F(X)$ at its input).

Automaton *A = (A, X, Y, f, g)* is named *finite* if all three sets *A, X,* and *Y* are finite and — *infinite* if even one of them is infinite.

An automaton is called *complete* or *completely specified* if its transition function and output function are completely specified and — *partial* if even one of these functions is partial.

An automaton, which component *f* is not a function but is a certain relation (i.e. the condition of a transition univocacy is not carried out in that sort of automata), is called *nondeterministic*. If the relation *f* is the function then the automaton is called *deterministic*. Therefore, the state *b* is unambiguously found for deterministic automaton *A* with start state a_0, when the automaton passes on in state *b* under the action of the word $p \in F(X)$. But there may be several states of this kind in a nondeterministic automaton. It is clear that the class of Mealy nondeterministic automata is the subclass of the nondeterministic automata class.

Moore Automata. Moore automaton presents the special case of Mealy automaton.

Definition 2. The automaton *(A, X, Y, f, g)* is named *Moore automaton* if its output function *g(a,x)* can be expressed by means of a transition function *f(a,x)* via the eguation *g(a,x) = h(f(a,x)),* where $h: A \to Y$. The function *h* is named as an *automaton marking function* and its value *h(a)* on the state *a* is named as the *mark* of this state.

Despite the fact that Moore automaton is the special case of Mealy automaton, Moore automata are studied separately in automata theory as far as in a number of cases their specific properties provide the opportunity to build more substantial theory then Moor automata one.

X-automata. *X-automaton* is 4-tuple *(A, X, f, F)*, but if *X*-automaton is initial, then it is 5-tuple *(A, X, f,* a_0*, F),* where $f: A \times X \to A$ is the transition finction, $F \subseteq A$ is a certain subset of the automaton states which elements are called final states and $a_0 \in A$ is a start state of the automaton.

3. Brief Finite Automata Ontology

Brief finite automata ontology is introduces in figure 1 which is presented in the form of a graph. Arcs of the graph link together different scientific domains which are sided with one or another automata theory or its applications.

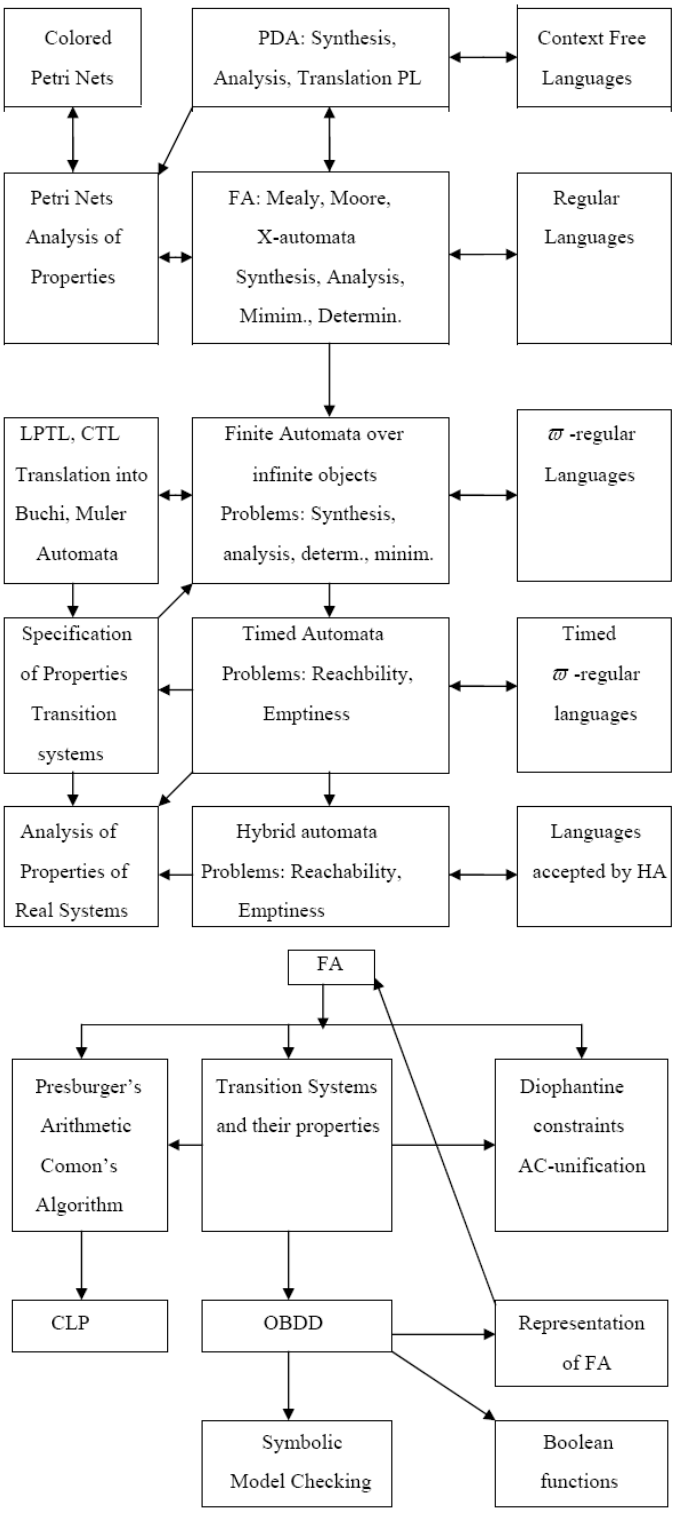

Figure 1

4. Finite Automata on Infinite Words

Let X be a finite alphabet. ϖ -language is an arbitrary subset X^{ϖ} of all infinite words in the alphabet X. A ϖ -automaton is finite object for the acceptance of ϖ -languages over a certain alphabet X. Several types of a ϖ -automata exist. Büchi automata and Muller automata are the main of them.

4.1. Büchi Automata and Muller Automata

Definition 3. A Büchi automata is 5-tuple (A, X, f, A_0, F), where X is a finite input alphabet, A is a set of the automaton states, $A_0 \subseteq A$ is a set of start states, $f \subseteq A \times X \times A$ is a set of arcs (a transition relation), and $F \subseteq A$ a set of final states. The tuple (A, X, f, A_0) is called a *transition table*. So, a Büchi automaton is the extension of a finite state automaton to infinite inputs. It accepts an infinite input sequence, iff there exists a run of the automaton (in case of a deterministic automaton, there is exactly one possible run) which has infinitely many states in the set of final states.

The automaton starts in one of the start states and if $(s, x, s') \in f$, then the automaton is able to change its state from s to s' by means of reading input symbol $x \in X$. One can say that $r = s_0 x_1 s_1 x_2 s_2 x_3 \ldots$ is the trace of the automathon A over the word $\sigma = x_1 x_2 x_3 \ldots$ in the alphabet X, which joins $s_0 \in A_0$ and $(s_{i-1}, x_i, s_i) \in f$ for all $i \geq 1$. The set $inf(r)$ for the trace r consists of the states $s \in A$ such that $s = s_i$ infinitely often, $i \geq 0$. The trace r of the automaton A on the word $\sigma \in X^{\varpi}$ is called the *accepting trace* iff $\inf(r) \cap F \neq \varnothing$. In other words, the trace r is accepting trace iff a certain state from the set F is repeated infinitely ofter in the trace r. The language $L(A)$ is called the language accepted by automaton A if it consists of the words σ such that the automaton A has the trace r which accepts σ.

ϖ -language is named *regular ϖ -language* if it is accepted by a certain Büchi automata.

Let $L(A) = \{p \in X^{\varpi} \mid A \text{ accepts } p\}$ be the ϖ -language which is accepted by a certain automaton A. If $L = L(A)$ for a certan Büchi automaton, then one can say that the language L is Büchi accepted. Muller atomata are the generalization of Büchi automata relatively the set of final states.

Definition 4. A Muller atomata over an alphabet X is 5-tuple $A = (A, X, f, a_0, F)$, where X is the finite alphabet, A is a finite set of states, $a_0 \in A$ is a start state, $f \subseteq A \times X \times A$ is transition relation, $F \subseteq B(A)$ is a set of subsets of the set of final states. It is said, that an automaton trace $\sigma = s_0 x_1 s_1 x_2 s_2 x_3 \ldots$is *effective*, if certain of the trace states appear infinite of times and serve as states from the set F. Automaton A accepts the word $p \in X^{\varpi}$, if the trace which corresponds to the word p is effective. ϖ -language $L \subseteq X^{\varpi}$ is named as accepted by Muller automaton, if it consists of all ϖ -words which this automaton accepts.

4.2. Properties of Regular ϖ –languages

All the results for ϖ –languages and ϖ -automata are presented in table 1. They mean that the operations are closed with respect to given classes of automata.

Table 1

Class of ϖ language	Operations
MA = BA = DMA $\cup$ DBA	union, intersection, complement union, intersection

5. Timed Automata

Let B be a set of clock variables which are nonnegative rational quantities $D+$. The set of timing constraints $C(B)$ is defined as the following:

a) all the elements, which belong to $C(B)$, are inequalities and look like $y \prec c$ and $c \prec y$, where $\prec$ means < or $\leq$, and c is nonnegative rational quantity (number);

b) if ϕ and ψ belong to $C(B)$, then $\phi \wedge \psi \in C(B)$.

Let us notice, that if *B* includes *k* clocks, then every timing constraint defines some convex set of *k*-dimensional Euclidean space. Therefore, if two points satisfy timing constraint, then all the points which belong to the segment connecting these points also satisfy this timing constraint.

Definition 5. Timed automaton (TA) *A* is 6-tuple *(X, A, A_0, B, I, T)*, where *X* is a finite alphabet TA, *A* is a finite set of states, $A_0 \subseteq A$ is a set of start states, *B* is a set of clocks, $I : A \rightarrow C(B)$ is the mapping of a set of states into timing constraints, which is named *states invariants*; $T \subseteq S \times X \times C(B) \times 2^B \times A$ gives the set of transitions.

Every 5-tuple (a, x, ϕ, λ, a') corresponds to the transition from the state *a* into the state *a'* marked with the symbol $x \in X$. The constraint ϕ defines the moment of time when this transition becomes possible and clock values from the set $\lambda \in B$ are nulled when this transition is occurred. That is, a transition may be taken only if the current values of the clocks satisfy the associated constraints.

5.1. The Timed Automaton Model

The transition graph *T(B) = (X, V, V_0, R)* with infinite nodes number serves as the model of TA *A.*

Every node from *V* corresponds to the pair *(a, v)* which consists of the state $a \in A$ ans clock value $v: B \rightarrow D+$ (*D+* — nonnegative rational quantities). The set of start nodes is the following: $V_0 = \{(a,v) : a \in A_0 \wedge \forall y \in B[v(y) = 0]\}$.

It is necessary to introduce some notation for the definition of transitions in *T(B).* Let us define $v[\lambda = 0]$ as the value of clocks for $\lambda \subseteq B$, at that the values coincide with *v* for clocks from $B \setminus \lambda$ and all the clocks from λ have the value 0.

Let us define *v+d* for $d \in D+$ as the clock values *v(y)+d* which are taken for all clocks from $y \in B$, and the clock values *v-d* are defined similarly.

It follows from the definitions above that there are two main types of TA transitions:

- *transition by delay d* corresponds to defined time in case when the automaton is in some state *a*. At that it is written: *(a, v){d} (a, v + d)*, where $d \in D+$, if for all $0 \leq e \leq d$ the invariant *I(a)* is true for *v + e*;
- *transition by action* corresponds to running transition from the set *T*. In that case it is written: *(a, v){x}(a', v')*, where $x \in X$, in case if such transition *(a, x, ϕ, λ, a')* exists that *v* satisfies ϕ and $v' = v[\lambda = 0]$.

One can receive transition relation *R* for *T(B)* via joining a set of transitions by delay and a set of transitions by action. The notation *(a,v)R(a', v')* or *(a, v){x}(a', v')* means: such elements *a"* and *v"* exist, that *(a, v){d} (a", v"){x} (a', v')*, for some $d \in D+$ and $x \in X$.

5.2. Timed Languages

Let $X = \{x_1,\ldots,x_n\}$ be an alphabet and *R* is the set of rational quantities. A *time sequence* $\tau = \tau_1 \tau_2 \ldots$ is a infinite sequence of time values $\tau \in R$ with $\tau_i > 0$, satisfying the following constraints:

(1) Monotonicity: τ increases strictly monotonically; that is, $\tau_i > \tau_{i+1}$ for all $i \geq 1$;

(2) Progress: For every $t \in R$, there is some $i \geq 1$ such that $\tau_i > t$.

A *timed word* over an alphabet *X* is a pair (σ, τ), where $\sigma = \sigma_1 \sigma_2 \ldots$ is a infinite word over *X* and τ is a time sequence. A *timed language* over *X* is a set of timed words over *X*.

A timed word (σ, τ) is named *input word* of an automaton; if this timed word is viewed as an input to an automaton, it presents the symbol σ_i at time τ_i.

Timed languages language-theoretic operations such as union, intersection, and complementation are defined the same way as for regular languages. In addition we define the *Untime* operation (for timed language *L* over *X*) which discards the time values associated with the symbols, that is, it considers the projection of a time trace (σ, τ) on the first component. That is, *Untime(L)* is ϖ-language which include all the words σ such that $(\sigma, \tau) \in L$ for some time sequence τ. Main results for languages and language operations are presented in table 2.

Table 2

Timed languages class	Operations
TMA = TBA	union, intersection
$\cup$	
DTMA	union, intersection, complement
$\cup$	
DTBA	union, intersection

6. Hybrid Automata

Hybrid automata generalize timed automata. A hybrid automaton (HA) is a dynamical system wih both discrete and continuous components. For example, an automobile engine whose fuel injection (continuous) is regulated by a microprocessor (discrete) is a hybrid system. HA provide a mathematical model for different real systems like digital computer systems that interact with analog environment in real time. Particularly, distributed processes with drifting clocks, real-time schedulers, and protocols for the control of manufacturing plants, vehicles, and robots would be modeled by means of HA.

Two problems, that are central to the analysis of HA theory are:

- the reachability problem and
- ϖ -language emptiness problem (more general problem).

Let us present here basic notions and main results of the HA theory. Let $D_{\geq 0}$ means the set of nonnegative real numbers: $D_{\geq 0} = \{x \in D \mid x \geq 0\}$.

Rectangular regions. Given a positive integer $n > 0$, $n \in N$, a subset of D^n is called a *region*. A bounded and closed region is *compact*. The region $R \subseteq D^n$ is called *rectangular* if it is Cartesian product of n intervals (possibly unbounded), all of whose finite endpoints are rational. R_i means the projection of R on i-th coordinate, so that $R = R_1 \times R_2 \times \ldots \times R_n$. The set of all rectangular regions in D^n is denoted R^n.

Definition 6. *n-dimensional rectangular automaton* (RA) A is 9-tuple $A = (G, X, init, inv, flow, pre, post, jump, obs)$, where $G = (V, E)$ – finite directed graph, X is finite *observation alphabet*, three vertex labeling functions $init: V \to R^n$, $inv: V \to R^n$, and $flow: V \to R^n$, and four edge labeling functions $pre: E \to R^n$, $post: V \to R^n$, $jump: E \to 2^{\{1,2,\ldots,n\}}$, and $obs: E \to X$.

RA may have so-called ε *-moves* (empty transitions). *n-dimensional RA with* ε *moves* differs from RA presented above in that the function $obs : E \to X^{\varepsilon}$, where $X^{\varepsilon} = X \cup \{\varepsilon\}$ augments the observation alphabet with the null observation $\varepsilon \notin X$.

The function *init* defines the set of initial states of RA. When the discrete state begins at vertex $v \in V$, the continuous state must begin in the initial region $init(v)$.

The functions *pre, post,* and *jump* constraint the behavior of the automaton state during edge steps. The edge $e = (v, w)$ may be traversed only if the discrete state resides at vertex v and the continuous state lies in the region $pre(v)$. For every i in the jump set $jump(e)$, the i-th coordinate of the continuous state is nondeterministically assigned a new value in the interval $post(e)_i$. For each $i \notin jump(e)$, the i-th coordinate of the continuous state is not changed and must lie in $post(e)_i$.

The *observation* function *obs* identifies every edge traversal with an observation from X or X^{ε}.

The *invariant* function *inv* and *flow* function *flow* constrain the behavior of the automaton state during time steps. While the discrete state resides at vertex v, the continuous state nondeterministically follows a smooth (C^{∞}) trajectory within the invariant region $inv(v)$, whose first time derivative remains within the flow region $flow(v)$. RA with ε -moves may traverse ε -edges during time steps.

If we replace rectangular regions with arbitrary linear regions in the definition ofRA, we obtain the *linear hybrid automata* (LHA). Thus RA are the subclass of LHA in which all defining regions are rectangular.

Initialization and bounded nondeterminism. RA *A* is *initialized* if for every edge *e=(v, w)* of *A*, and every coordinate $i \in [1,2,\ldots,n]$ with $flow(v)_i \neq flow(w)_i$, we have $i \in jump(e)$.

It follows that whenever the i-th continuous coordinate of an initialized automaton changes its dymanics, as given by the *flow* function, then its value is nondeterministically reinitialized according to the *post* function.

RA *A* has *bounded nondeterminism* if

(1) for every vertex $v \in V$, the regions *init(v)* and *flow(v)* are bounded, and

(2) for every edge $e \in E$, and every coordinate $i \in [1,\ldots,n]$ with $i \in jump(e)$ the interval $post(e)_i$ is bounded.

Note that bounded nondeterminism does not imply finite branching. It ensures that the edge and time successors of a bounded region are bounded.

7. Automata Applications for Systems Verification

Finite-state automata are well used for modeling of concurrent and interacting reactive systems. This modeling imposes that either the set of an automaton states, or the symbols of its input alphabet represent the states of modeled system. Main advantage at that automata use for systems verification is the following: both the system model and its specification are presented equally. In that case, Kripke model is directly correlated with ϖ-automaton (Büchi or Muller automaton), which all states are final, and the set of possible system behaviors is set by ϖ-language *L(A)*, which is accepted by corresponding automaton *A*. At that, the algorithm exists which translates a temporal logic formula into ϖ-automaton.

Let AP be a set of atomic propositions, then Kripke model *(S, R,* S_0*, f)*, where $f : S \to 2^{AP}$, one may translate into the automaton $A = (A \cup a_0, X, Q, a_0, A \cup a_0)$, which input alphabet is the powerset of the set of atomic propositions $X = 2^{AP}$. At that for every pare of states *a, a'* $\in A$ relation *Q* includes the triple *(a,x,a')* iff *(a,a')* $\in R$ and *x = f(a')*, in which connection $(a_0,x,a) \in Q$ iff $a_0 \in S_0$ and *x = f(a)*, where *R* – consequence relation in Kripke model.

8. Examples of Some Properties and Their Specification

Let us consider some hypothetical reactive system. The properties of such systems are divided into two classes:

- ***safety properties***, usually state that something bad never happens**;**
- ***liveness properties*****,** which state that something good eventually happens.

Safety properties are usually expressed by means of the following temporal logic formula $\Box \neg p$, where formula *p* expresses an unwanted event (a state) in a system.

The example of safety property is *mutual exclusion* property: $\Box (\neg p \vee \neg q)$, where formulas *p* and *q* defines the states at which a system can never be at the same time. The example of liveness property is *fairness* property; it states that a system must some time answer received inquiry, so fairness is concerned with guaranteeing that processes get a chance to proceed.

Let us extend this list of properties by some additional properties which belong to the class of liveness properties and are the following:

- guarantee states, that some event occurs at least once, but its repetition is not guaranteed. This property is expressed by LPTL-formula as $\Diamond p$;

- obligation states, that formula *p* must always be executed or formula *q* is executed in the same state as formula *p*. This property is expressed by LPTL-formula as $\Box p \vee \Diamond q$. **obligation** property can be obtained by disjunciton of **safety** and **guarantee** properties;

- response states, that the event defined by formula *p* occurs infinitely often. This property is expressed by LPTL-formula as $\Box \Diamond p$;

- persistence states, that the event defined by formula p occurs continuously from a certain point on. This property is expressed by LPTL-formula as $\diamond \Box p$;

- reactivity can be obtained by disjunciton of **persistence** and **response** properties**.** This property is expressed by LPTL-formula as $\diamond \Box p \vee \Box \diamond p$;

- unconditional fairness states, that eligible process eventually run or the event, which is defined by formula q, occurs infinitely often irrespective of property p. This property is expressed by LPTL-formula as $\Box \diamond p$;

- weak fairness states, if an activity is continually enabled (no temporary disabling) than it has to be executed infinitely often; or in other words, when formula p is true all the time, then formula q must be true infinitely often. This property is expressed by LPTL-formula as $\Box p \rightarrow \Box \diamond q$;

- strong fairness states, if an activity is infinitely often enabled (not necessarily always) then it has to be executed infinitely often; or in other words, when formula p is true infinitely often, then formula q must be true infinitely often also. This property is expressed by LPTL-formula as $\Box \diamond p \rightarrow \Box \diamond q$.

Bibliography

1. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. -М. :Мир.-1979. - 535 с.
2. Alur R., Dill D.L. A theory of timed automata. - Theoretical Computer Science. -1994. -126. - PP. 183-235.
3. Thomas W. Automata on infinite objects. Handbook on theoretical computer science. - 1990. - PP. 135-191.
4. Годлевский А.Б., Кривой С. Л. Трансформационный синтез эффективных алгоритмов с учетом дополнительных спецификаций. Кибернетика, - 1986. - N 6. - С.34 - 43.
5. Глушков В.М. Абстрактная теория автоматов. - Успехи математических наук. -1961. - 16. - вып. 5. - С. 3-62.
6. Глушков В.М. Синтез цифровых автоматов. - М: Физматгиз. - 1962. - 476 с.
7. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра, языки, программирование. -Киев: Наукова думка. -1985. 327с.
8. Perrin D. Finite automata. In Handbook of Theoretical Computer Science. vol. 2, -Elsevier. -1990. -PP. 1-58.
9. Henzinger T.A., Kopke P. W, Puri A., Varaiya P. What's Decidable About Hybrid Automata? In the Proceed. of the 27-th Annual ACM Symposium on Theory of Computing (STOC 1995). - 1995. - PP. 373-382.
10. Comon H. Constraint solving on terms: Automata techniques (Preliminary lecture notes). - Intern. Summer School on Constraints in Computational Logics: Gif-sur-Yvette, France, September 5-8. -1999. - 22 p.
11. Капитонова Ю.В., Кривой С. Л., Летичевский А. А., Луцкий Г.М. Лекции по дискретной математике. БХВ: Санкт-Петербург, 2004, 624 с.
12. Трахтенброт Б. А., Барздинь Я. М. Конечные автоматы (Поведение и синтез). -М.: Наука. -1970. - 400 с.
13. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. -М.: Мир. -1973. - 256 с.
14. Arnold A. Finite Transition Systems: Semantics of Communicating Systems. -Paris: Prentiuce Hall. -1994. - 177 p.
15. Ben-Ari M. Mathematical Logic for Computer Science. Springer Verlag London Limited. - 2001.-305 p.
16. Emerson E.A. Temporal and modal logics. Handbook of Theoretical Computer Science: Elsevier. - vol. B. -1990. - PP.995-1072.
17. Peterson G.L. Myths about the mutual exclusion problem. - Information Processing Letters, - 1981. - v.12.-N 3. - P.115-116
18. Wolper P. Temporal logic can be more expressive. - Information and Control. -v. 99. -1983. -P. 56 - 72.
19. Clarke E.M., Grumberg Jr. O., Peled D. Model Checking. - The MIT Press: Cambridge, Massachusetts, London, England. -2001. -356 p

Authors Information

S.L. Krivoy, L.Ye. Matveyeva *– Institute of Cybernetics, NAS of Ukraine, Kiev, Ukraine, e-mail: krivoi@i.com.ua; luda@iss.org.ua*

E. A. Lukianova *– Crimean Vernadski's University, Crimea, Simferopol, Ukraine*

O. Sedleckaja *– Institute of Theoretical and Applied Informatics, Technical University of Czestochova, Czestochova, Poland*