ISSN 1392 – 124X INFORMACINĖS TECHNOLOGIJOS IR VALDYMAS, 1996, Nr. 2(3)

TRACE SEMANTICS FOR AGGREGATES

Valentinas Kriauciukas

Department of Mathematical Logic, Institute of Mathematics and Informatics
Akademijos 4, Vilnius 2600, Lithuania

Henrikas Pranevicius

Department of Control Systems, Kaunas University of Technology
Studentu 50, Kaunas 3031, Lithuania

Abstract. Aggregates are systems of automata containing continuous and discrete components. Aggregate systems are good for specification and simulation of reactive systems including communication protocols, also as for some kind of hybrid systems. The aggregate model is a good basis for designing tools for specification, simulation, verification, testing, etc. The tools may use specification languages of different sorts: procedural, logical or object-oriented, but for definition of semantics for any of them we need the formally defined aggregate model. We present here a formal definition of aggregates based on the notion of trace. The earlier definitions were sufficiently strict for design of simulation tools, but were not detailed enough for other purposes, particularly, for verification. The presented aggregate model is also slightly more general and flexible than the one used before.

1. Introduction

The name of *aggregates* comes from *piece–linear aggregates* (PLA) — the model of automata proposed by Buslenko [1] for simulation of complex systems. Later, Pranevicius [2, 3] developed a method for formal description of PLA based on the notion of *controlling sequences* which are very like to *time–sequences* defined below. In H.Pranevicius' doctoral thesis [4] has been shown how PLA with semantics of controlling sequences can be used for formal specification, validation and simulation of complex systems. The proposed mathematical model lies in base of several implemented simulation systems [5, 6, 7, 8] running on machines of various platforms. The model permits not only simulation but also correctness analysis on base of the same formal specification. Combination of these tasks is very important for design of real–time systems.

An application of the aggregate model for formal specification and simulation of computer network protocols was proposed in [10]. For this task, the language Estelle/Ag [8] (close to ISO standard Estelle) was created using PLA model. The main difference between the two languages is that the basic structural unit of Estelle — module — in Estelle/Ag is defined as a piece–linear aggregate. The possibility to use the same aggregate specification for both the validation of general protocol properties and the simulation is presented in [11]. The method for invariant–based correctness analysis of aggregate specifications is proposed in [12]. System PRANAS (PRotocol ANAlysis System) [13] was developed for protocol validation and simulation using formal description of protocols by aggregates. The aggregate approach was successfully approved by specification and simulation of adaptive commutation protocols [14], event–driven and interval–marker protocols for local networks [15, 16], and other reactive systems.

The method of controlling sequences [2, 3] defines, in fact, operational semantics for the PLA model. This semantics defines how to interpret (or "animate") specifications given in form of aggregates, and is sufficient for simulation and testing tasks. Some correctness analysis using reachability graph of states is also possible, but only in the case of finite number of states. The operational semantics is appropriate for verification tasks, like proving service ensurance in protocols.

In verification, at least two specifications are given and it is required to prove existence of some definite relation between them, e.g., "implements", "provides", "satisfies". More specifically, in the case of aggregate specifications, the relations should be established between behaviour of aggregates. The notion of trace formalizes the behaviour notion. The formalization is

presented here as a necessary step toward verification of aggregate specifications.

The aggregate model is compatible with the model of timed input/output automata [23, 24], but the comparative analysis will be given elsewhere.

Section 2 describes internal structure of aggregates and explains behaviour of aggregates informally. Traces are introduced in Section 3 using modified notion of the time–axis.

2. Structure of aggregates

One can think about aggregates as about active devices which wait particular *events* and react to them by generating other events. The reactions may be delayed in time. Each aggregate has changeable *state* which is divided into a finite number of *components*. Active part of an aggregate consists of a finite number of *agents* each reacting to one particular *sort* of events. Agents change state components when corresponding events occur. Each aggregate G is fully defined by three sets: $S(G)$ of possible states, $I(G)$ of initial states, and $A(G)$ of agents defining behaviour of G. We are going to describe the structure of all these elements in this section. Section 2.1 presents state structure and introduce our understanding of events. Agents and transitions are considered in Section 2.2.

2.1. State components

The most convenient way to describe state is to name uniquely each state component and think about these names as about *variables* accepting values from definite sets, called *range* sets.

Definition 2.1. Let $V(G)$ *be a set of variables denoting state components of an aggregate* G, $R(x)$ *be a set of possible values of a component* $x \in V(G)$, *then the set of states of* G, *denoted* $S(G)$, *consists of all functions* $s: V(G) \to \boldsymbol{D}$, *with* $\boldsymbol{D} = \mathop{\mathrm{Y}}_{\upsilon \in V(G)} R(\upsilon)$ *such that* $s(\upsilon) \in R(\upsilon)$ *for each* $\upsilon \in V(G)$.

Part of states of G are called *initial*, their set is denoted $I(G)$.

Discrete and continuous components

One division of state components into types occurs already at the most general level of aggregate consideration: they are divided into *discrete* and *continuous*. This division characterizes aggregates as *hybrid systems* [21, 22] and reflects the essential difference of the component behaviour: discrete components have changes just in discrete time moments, while continuous ones change their values continuously all time except discrete time moments. In the aggregate model we consider, every continuous component z ranges over the set of reals $\mathbf{R}$, and just one type of continuous change is possible — the one described by the equation

$$dz/dt = -1, \qquad (1)$$

which is valid all time but discrete time moments, t is a variable for time. Hence, continuous state components decrease all time except some discrete moments. One may think about such component as about timers, because their zero values are considered as internal events of the aggregate. Negative values of a continuous component may mean that the timer is off, but this is not the only interpretation of the negative part of scale. Thanks to the constant speed continuous components in the negative part are like inverse clocks.

Discrete components are constants almost all time, the equation corresponding to (1) is

$$dz/dt = 0. \qquad (2)$$

As follows from these equations, aggregates are a particular case of *linear hybrid automata* [20].

The range sets of discrete state components may be defined as (and, in fact always are) definite data types, including queues, stacks, arrays, etc. For verification task, in particular, it is necessary to have data types defined axiomatically, i.e., abstract data types are needed. For simulation or testing, implemented concrete data types are used. It is natural to include into a specification language some sublanguage for specification of (abstract) data types. In this paper, details of types of discrete components also as of their abstraction level are not essential.

Events

We consider events as properties of states, and there are a few reasons to do so:

- occurrence of an event in time looks like a change of an component—the event may occur in a time moment t, but not at any moment before or after t;
- often events, like arrivals of messages in protocols, have the associated message structure which should be described, and this may be done in the same way as data types for discrete state components are defined;
- in hybrid systems, events often means satisfiability of particular conditions on values of components;
- it is convenient to describe interaction of aggregates using shared components, and so on.

Definition 2.2. Events *are states satisfying a particular property.*

In the aggregate model, a state with zero value of a continuous component z is an *internal event* of the aggregate. The condition describing the corresponding property is obvious: $z=0$. The name *internal* means that these events happen inside of the aggregate.

For discrete components, there is a big freedom to set conditions describing events. However, in the

present paper we consider just one sort of events related with discrete components called *shared*.

Shared components

Aggregates are interactive devices by their nature and their behaviour depends on behaviour an environment, say, other aggregates. Interaction between aggregates occurs through shared state components called in short *channels*. This means that state of an aggregate may be changed by a connected aggregate (or the environment) through change of channel values.

We consider just one type of interaction which is like a discrete message passing. Each channel shared by an aggregate G may be an *input* or an *output* channel for G, in $V(G)$ the corresponding two subsets are denoted $In(G)$ and $Out(G)$ respectively. We allow $In(G) \mathrm{I}\ Out(G) \neq \varnothing$, this lets a composition of aggregates to be an aggregate again. We assume that

- range sets of channels include one special emptiness value denoted $\varnothing$, and
- each channel may be an output channel for at most one aggregate.

For an output channel k for G, the condition $k \neq \varnothing$ describes an event which is *external* to G since it (usually) occurs outside of G. We pose the following restrictions on use of channels:

- for an input channel $k \in In(G) \setminus Out(G)$, G can not change value of k in any way — just watch it (and react, if necessary);
- for an output channel o, G is responsible to keep $o=\varnothing$ all time except discrete time moments.

As follows, all events, internal and external, occur just in discrete time moments, hence the reaction to them must be instantaneous. This allows very high level of synchronization (if it is necessary) between aggregates. The aggregates can delay their external reaction to events by putting the information about them into some kind of stores, like queues, stacks, etc., and by using timers to calculate time of the postponed reactions.

2.2. Agents and transitions

Agents of an aggregate G relate events and discrete transitions called *steps*. Any step of the aggregate must be result of an action of some of its agents. Agents are activated (we say *enabled*) by occurrences of events. Each agent A reacts to one particular event related to one particular component (variable), i.e., to $c=0$ for some continuous component c or to $k \neq \varnothing$ for some input channel k. There may be (and often is convenient to have) a few agents reacting to the same events.

In general, steps from a former state to a new one performed by agents are *nondeterministic*, i.e., new values of state components need not be uniquely determined by the former values. Such nondeterministic steps correspond to binary relations defined in the set $S(G)$ of possible states of G, therefore we define agents as binary relations on states. Steps of the aggregate are results of simultaneous actions of all enabled agents.

Definition 2.3. *Let S be a set of states, then an agent A over S is a relation $A \subseteq S \times S$, and the set $En(A) = \{s: \exists s' \in S: \langle s, s' \rangle \in A\}$ consists of states where A is enabled.*

The set $En(A)$ contains all states to which A reacts. By our assumption, $En(A) = \{s: s(z) = 0\}$, if A reacts to internal events described by the condition $z=0$, or

$$En(A) = \{s: s(k) \neq \varnothing\},$$

if A reacts to external events related to a channel k. Of course, there is an easy way to generalize aggregates by releasing these assumptions.

Actions of agents

Several agents, say A_1, K, A_n, of G may be enabled at the same state s. Then all possible steps of G from this state are described by intersection of all agents

$$B = \mathop{\mathrm{Y}}_{i=1}^{n} A_i.$$

There is possible that $s \notin En(B)$, then agents block each other and a *deadlock* occurs. We would like to exclude from consideration aggregates allowing deadlocks. Let note, that deadlock resolution is the true place, where techniques from constraint programming may be applied, say, methods of hierarchical constraints.

In each state, say s, of an aggregate G, there is defined a set of agents

$$A(G,s) = \{A: A \in A(G) \wedge s \in En(A)\}$$

(which may be empty) that are enabled in s. Let

$$Tran(G,s) = \mathop{\mathrm{I}}_{A \in A(G,s)} A, \qquad (3)$$

denote the most general transition allowable by all agents from $A(G, s)$, and $E(G) = \mathop{\mathrm{Y}}_{A \in A(G)} En(A)$ denote the set of all events of G. If $Tran(G, s)$ is enabled in s, i.e., $s \in En(Tran(G, s))$, then the aggregate G changes the state s to s' where $\langle s, s' \rangle \in Tran(G, s)$. If $Tran(G, s)$ is not enabled in s, then no a step is performed by agents of G.

Remark. As follows from the definition (3) of $Tran(G, s)$, the aggregate G perform steps only to states on which "agree" all enabled agents. This is *conjunctive* treatment of agents. The alternative could be *disjunctive* treatment defined with 'Y' instead 'I' in (3). We prefer the conjunctive treatment because of

possibility to specify changes of distinct state components by distinct agents and to be unconcerned about the whole state in each agent, compare, for example, [19]. It is closer to programming practice to describe change of the whole state by combined action of all agents (like operators of a programming language). □

3. Traces of aggregates

Each aggregate starts at some initial state, all further continuous transitions are fully determined by equations (1)–(2), while discrete transitions are determined by agents of the aggregate, though the environment of the aggregate also can act to state through shared components (channels) as was described above.

It is natural to describe changes (of state) depending on time by functions defined on the set of time moments (the time–axis), let call them *traces*. In spite of all good work of traces in description of continuous transitions, their use in case of discrete transitions (steps) is problematic. The problem arises because of the assumption about timeless of steps (see [17] for more deep topological consideration). Any step is a pair of states, and we would like to have two time moments $t_1 \neq t_2$ when these states are accessed, but a time interval between t_1 and t_2 should be of zero length because of timeless. In the classical model of time — the set of real numbers **R** —, $|t_1 - t_2| = 0$ means $t_1 = t_2$, i.e., such time moments cannot be distinct. That means that the topology of the real line (based on the metrics function $|x - y|$) is not well–suited for description of discrete changes under assumption about their timeless. We need another model of time.

The problem with the real line **R** as a time model is that there is not enough points in it. It should be possible to have a few points with "the same time" for specification of systems that may consequently visit a few different states "without time passing". One possible way out is to use line of nonstandard real numbers as in the nonstandard analysis [18], where discrete changes could take infinitely small time amounts. We propose another way and introduce a slightly modified (comparing with **R**) time model.

What we loose with introduction of the new time model is the uniqueness of the time axis. This (uncountable) multiplicity of them, called broken time–lines, is a price for the assumption about timeless of discrete transitions.

3.1. Broken time lines

Let N denote the set of natural numbers, N_n be the set of natural numbers that are less or equal to n, and N_∞ be N.

Definition 3.1 *For any* $k \in N \cup \{\infty\}$, *call a time–sequence a nondecreasing sequence*

$$\tau = \langle \tau_i : i \in N_k \wedge \tau_i \in \mathbf{R} \wedge \tau_0 = 0 \rangle$$

and denote $\mathrm{dom}(\tau) = N_k$. *The sequence* τ *is* finite *if* $k \in N$, *and* infinite *otherwise; the infinite sequence* τ *is* Zeno *if* $\lim_{i\to\infty} \tau_i < \infty$, *and* non–Zeno *otherwise.*

A time–sequence τ consists of time moments when steps (discrete transitions) start plus the moment τ_0 added as the launch time. Closed intervals $[\tau_i, \tau_{i+1}]$ between adjacent elements of the sequence are called *phases*. Length of a phase may be zero, then it does not contain *internal points* and is called *trivial*. From the time–sequence τ we obtain a *phase–sequence*

$$\varphi(\tau) = \langle \varphi_i(\tau) : i \in N_k \rangle,$$

where $N_k = dom(\tau)$ and

$$\varphi_i(\tau) = \begin{cases} [\tau_i, \tau_{i+1}] & \text{if } i < k, \\ [\tau_i, \infty) & \text{otherwise.} \end{cases}$$

The phase–sequence $\varphi(\tau)$ is finite if τ is finite, but then the last phase of $\varphi(\tau)$ is infinite. The real half–line $\mathbf{R}^+$ is covered by all phases of $\varphi(\tau)$ only if τ is non–Zeno. In the case of Zeno time–sequence, time "collapses" at some point because of infinite number of steps occurred infinitely often (the situation known as Zeno's paradox). The phase–sequence obtained from a Zeno time–sequence is also called Zeno.

Definition 3.2. A broken time–line *is disjoint union of all phases from some non–Zeno phase–sequence.*

We call broken time–lines simply time–lines, because the half–line of reals $\mathbf{R}^+$ is isomorphic to a broken time–line obtained from the phase–sequence $\langle [0, \infty) \rangle$.

Pairwise representation of time–lines

Let τ be a time–sequence, $\varphi(\tau)$ be a phase sequence obtained from τ, then the broken time–line

$$T = \dot{+}_{i \in dom(\tau)} \varphi_i(\tau), \qquad (4)$$

where $\dot{+}$ denotes disjoint union, which may be presented by the following set of pairs:

$$\{\langle x, i \rangle : x \in \varphi_i(\tau)\}, \qquad (5)$$

Each element $t \in \tau$ except τ_0 appears in T at least twice — in $\langle t, i \rangle$ and in $\langle t, i+1 \rangle$ — since it is end of two adjacent phases from $\varphi(\tau)$. These two copies of t form pair of time–moments necessary for description of a step. Any particular t may be repeated in τ many times, so there may be many successive steps in "the same time moment".

Distance measure (metrics) in time–lines are introduced by the equation

$$\left|\langle x,n\rangle - \langle y,m\rangle\right| = |x-y|,$$

so, $\left|\langle t,i\rangle - \langle t,i+1\rangle\right| = 0$, as it is expected. The temporal order in time–lines is the lexicographical order of pairs:

$$\langle x,n\rangle < \langle y,m\rangle = x < y \vee (x = y \wedge n < m).$$

For representation free definition (i.e., without explicit presentation of time–points as pairs as in (5) of time–lines, the distance measure and the temporal order should be given for each of them. This information is necessary and sufficient for discovery of structure of phase–sequences which is used here to define time–lines. Therefore, for a time–line T given, when we write that a phase p is from T, that means that p is from the phase–sequence which is in 1–1 correspondence with T. The time–sequence τ from (4) is also uniquely defined by T, its elements (except τ_0) are called *break–points* of T, since they are right–hand end–points of phases from T. The initial point of any time–line is denoted **0**. In the form pairs, $\mathbf{0} = \langle 0, 0\rangle$. The time–point following a break–point t in T is called *next* to t and denoted t'. If, in the pairwise representation (5), $t = \langle x, n\rangle$, then $t' = \langle x, n+1\rangle$. The function $\_'$ is partial because is defined only on (countably many of) break–points of T.

Operations and relation between time–lines

Time–lines may be modified and compared, we present operations that we need. Let **T** denote the set of all time–lines and $T \in \mathbf{T}$. Any non–break point $c \in T$ can be made a break–point by splitting the phase $[a, b] \in T$ which contains c into two phases $[a, c]$ and $[c, d]$. We call this modification *breaking* of time–lines. We write $T \le T_1$ for any line T_1 that can be obtained from T by breaking. The relation '$\le$' is a partial order in the set **T** and defines a lattice in it. The bottom element of this lattice is (isomorphic to) the line $\mathbf{R}^+$ without breaks. The meet and join operations in this lattice we denote $\wedge$ and $\vee$ respectively.

Traces

We would like to consider only continuous *traces*. We do not go into topological considerations, but interested reader should note that continuity requires to have a topology defined on the sets of arguments and values. For continuous components the value set is **R** with the usual topology. For the rest of components we assume that the corresponding value sets possess the *discrete topology*. The topology of time–lines is a mixture of the topology of **R** and of some discrete topology. We give the following definition which does not require deep knowledge of topology:

Definition 3.3. *Let T be a time–line and D be a set with a topology. A trace is a function $\theta: T \to D$ such that, for any phase $p \in T$, the restriction $\psi = \theta \lceil p$ of θ to p called also phase of θ is a continuous function.*

We denote $\theta(p) = \{\theta(t): t \in p\}$ the set of all values appeared during the phase p. The definition allows any changes of a trace $\theta: T \to D$ in break–points of T, they are discrete and called *steps*. The step in a break–point t is *stuttering* (in the same way as in [19]) if $\theta(t) = \theta(t')$. It can be deleted by joining t and t' because continuity of the trace will be preserved. We call traces *equivalent*, if they can be made equal (as functions) by deletion or introduction of stuttering steps. Introduction of stuttering steps into the trace θ means breaking of the time–line T.

We can compare different traces, if they are *synchronized*, i.e., defined on the same time–line. This synchronization is not a problem, time–lines T_1 and T_2 from any two traces $\alpha: T_1 \to \boldsymbol{D}$ and $\beta: T_2 \to \boldsymbol{D}$ always may be combined into $T = T_1 \vee T_2$, and traces α and β may be easily transformed into synchronized traces $\alpha_1: T \to \boldsymbol{D}$ and $\beta_1: T \to \boldsymbol{D}$, respectively, by introduction of necessary stuttering steps. We suppose, that this kind of synchronizing transformation is always performed when it is necessary, and do not introduce any (boring) notation describing it. The synchronization of time–lines also occurs when two (or more) systems are combined into one, and their local states are united into one global state.

State traces

Let V be a finite set of variables denoting state components. By definition 2.1 state is a function $s: V \to \boldsymbol{D}$, then *state trace* is a trace $\theta: T \to (V \to \boldsymbol{D})$ with states as values. There is the well–known 1–1 correspondence between functions from sets like $T \to (V \to \boldsymbol{D})$ and $T \times V \to \boldsymbol{D}$: for each time–point t, the value of $\theta(t)$ is a state from $V \to \boldsymbol{D}$, for each component $\upsilon \in V$, $\theta(t)(\upsilon)$ is a value of υ in moment t; we obviously can present the same information with two–argument function from $T \times V \to \boldsymbol{D}$ or, after change of arguments, from $V \times T \to \boldsymbol{D}$. The last set is in the mentioned 1–1 correspondence with $V \to (T \to \boldsymbol{D})$, which elements map each state component to a trace describing behaviour of this component. This short passage between different presentations of state traces reflects the obvious observation that any trace of a system consisting of components is a collection of synchronized traces of components. We use this fact when we want to *hide* or abstract from some of state components. For $U \subseteq V$, let $\theta \lceil U$ denote the state trace where only components from U are taken, so, $(\theta \lceil U)$: $T \to (U \to \boldsymbol{D})$.

3.3. Aggregate traces

Aggregate traces are just special case of state traces. In any phase, every discrete state component of an aggregate G is a constant according to (2), while every continuous component is uniformly decreasing function according to (1). Equations (1), (2) do not define changes of components in break–points, this is done by agents or by the environment of the aggregate.

For an input channel k for G, any moment of a change of value of k from $\varnothing$ to any value $\neq \varnothing$ is called a *predpoint* (of the external event $k \neq \varnothing$). Changes of state (i.e., some of input channels from $In(G)$) in predpoints are the only changes of the aggregate G not described by agents of G.

Definition 3.4. *For an aggregate G given, a state trace* $\theta: T \to S(G)$ *is a trace of G, if*

- $\theta(0) \in I(G)$;
- *if* $\theta(t) \in E(G)$ *for* $t \in T$, *then t is a break point in T;*
- *in any nontrivial phase from T, all components behave according equations (1) and (2);*
- *for any break–point* $t \in T$, *which is not a predpoint,* $\langle \theta(t), \theta(t') \rangle \in Tran(G, \theta(t))$, *or* $\theta(t) = \theta(t')$.

As follows from this definition, a change of a channel value means at least two consecutive break points, the first one is when a a channels becomes $\neq \varnothing$, the second is when it is made $= \varnothing$.

4. Resume

A formal definition of piece–liniar aggregates based on the notion trace is presented. The earlier definitions of aggregates were sufficiently strict for design of simulation models, but were not detailed enough for other purposes, particularly, for verification.

5. References

[1] **N.Buslenko**. On a class of complex systems. *Problems of applied mathematics and mechanics, Nauka, Moscow*, 1971, 56–68 (*In Russian*).

[2] **H.Pranevicius**. Use of controlling sequences for formal description systems for developing simulation models. *Programming*, 6, 1979, 69–74 (*In Russian*).

[3] **H.Pranevicius**. Models and methods for computer system investigation. *Mokslas, Vilnius*, 1982, 228 (*In Russian*).

[4] **H.Pranevicius**. Automatization of developing numerical and simulation models for computer systems. *PhD dissertation*, 1983 (*In Russian*).

[5] **H.Pranevicius, A.Chmieliauskas**. ASPECT — language for specification of services and protocols. *In XII-th Conference of Computer Networks. Moscow–Odesa*, 1987, 76–81 (*in Russian*).

[6] **H.Pranevicius, A.Chmieliauskas, V.Pilkauskas**. Protocol simulation and verification in PRANAS. *Packet Switching Networks, Institute of Electronics and Computer Science, Riga*, 1985, 209–231 (*in Russian*).

[7] **H.Pranevicius, V.Pilkauskas, A.Chmieliauskas**. The automatized system PRANAS-2 for validation and simulation of computer network protocols. *Automatics and Computer technique*, 6, *Riga*, 1991, 9–18 (*in Russian*).

[8] **H.Pranevicius, V.Pilkauskas, A.Chmieliauskas**. Validation and simulation of computer protocols using single specification Estelle/Ag. *In Proc. of the Internationals Conference of Local-Area Networks, Riga*, 1990, 17–21.

[9] **H.Pranevicius**. Aggregate approach for specification, validation, simulation and implementation of computer network protocols. *Lectures notes in Computer Science*, 502, *Springer-Verlag*, 1991, 433–477.

[10] **H.Pranevicius, N.Listopadskis**. Aggregate approach application for formal specification and modelling of protocols. *Acad. Sci. USSR, Moscow*, 1982, 63 (*In Russian*).

[11] **H.Pranevicius, A.Chmieliauskas.** Correctness analysis and performance prediction of protocols by means of aggregate approach and control sequences method. *Acad. Sci. USSR, Moscow*, 1983, 32 (*In Russian*).

[12] **H.Pranevicius, A.Panevezys**. Proof of correctness technique for aggregate models of protocols. *In IFAC/IMAC Symp. on distributed intelligence systems, Varna*, 1988, 100–105.

[13] **H.Pranevicius, A.Chmieliauskas, V.Pilkauskas.** Protocol simulation and verification in PRANAS. *Packet Switching Networks, Electronic and Computer Technical Institute, Riga* 1985, 209–231 (*In Russian*)

[14] **H.Pranevicius, S.Samoilenko, P.Moscinskis, J.Granskas**. Models for investigating methods of commutation. *Problems of Cybernetic, Moscow*, 1986 (*In Russian*).

[15] **H.Pranevicius, S.Samoilenko, P.Moscinskis.** Aggregate model of interval-marker access method in local networks. *Proceedings of Computer Networks, Moscow–Riga*, 1996 (*In Russian*).

[16] **H.Pranevicius, S.Samoilenko, L.Sintonen, P.Moscinskis.** The employment of an aggregate approach for the simulation of access methods in local computer networks. *Automatic and Computer Technics*, 6, *Riga*, 1995, 45–52 (*In Russian*).

[17] **A.Nerode, W.Kohn**. Models for hybrid systems: automata, topologies, controllability, observability. *Technical report, Mathematical Science Institute, Cornell University, USA*, 1993, 93–28.

[18] **M.Davis.** Applied Nonstandard Analysis. *J.Wiley&Sons*, 1977.

[19] **T.Lamport.** The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, Vol.16, No.3, 1994, 872–923.

[20] **R.Alur, C.Courcoubetis, T.A.Henzinger, P.–H.Ho.** Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. *In* [21].

[21] **J.W. de Bakker, K.Huizing, W.-P. de Roever, G.Rozenberg,** editors. Real Time: Theory in Practice. *Lecture Notes in Computer Science*, 600, *Springer-Verlag*, 1992.

[22] **R.L.Grossman, A.Nerode, A.P.Ravn, H. Rischel,** editors. Hybrid Systems. *Lecture Notes in Computer Science*, 736, *Springer-Verlag*, 1993.

[23] **N.A.Lynch, F.W.Vaandrager.** Forward and backward simulations — part I: Untimed systems. *Information and Computation*, Vol.121, No.2, 1995, 214–233.

[24] **N.A.Lynch, F.W.Vaandrager.** Forward and backward simulations — part II: Timing-based systems. *Report CS-R9314, CWI, Amsterdam*, 1993. (*To appear in Information and Computation*).

Agregatų trajektorinė semantika

Panaudojant trajektorijų sąvoką yra pateikiamas formalus atkarpomis–tiesinių agregatų apibrėžimas, įvertinantis verifikavimo uždavinių sprendimo specifiką. Ankstesniuose darbuose pateikiami agregatų apibrėžimai yra orientuoti į imitacinių modelių sudarymą.