INFORMATICA, 2012, Vol. 23, No. 3, 427–441

Applications of Finite Linear Temporal Logic to Piecewise Linear Aggregates

Henrikas PRANEVICIUS[1], Stanislovas NORGĖLA[2]
[1]*The Faculty of Informatics, Kaunas University of Technology*
Studentų 50, LT-51368 Kaunas, Lithuania
[2]*The Faculty of Mathematics and Informatics, Vilnius University*
Naugarduko 24, LT-03225 Vilnius, Lithuania
e-mail: henrikas.pranevicius@ktu.lt, stasys.norgela@mif.vu.lt

Received: July 2010; accepted: June 2012

Abstract. In this paper, we consider piecewise linear aggregates (PLA) and a possibility to use a linear temporal logic for analysis of their performance over finite structures (finite linear temporal logic (LTL)). We describe a calculus where the search is performed with respect to a context of the formula. An important aspect of finite LTL is the simplicity of its model of time and actions. PLA is used for description of numerous complex systems. The answers about the behavior of the aggregate are got by finding an interpretation in which all the formulas describing the work of the aggregate are true. This is illustrated by formalizing alternative bit protocol (ABP) task. We describe the ABP by putting it in the form of a planning problem. From the obtained model, we can find a finite sequence of actions to be executed in order to achieve the goal. In addition, an alternative bit protocol problem is described using the planning domain description language (PDDL). We report the results of experiments conducted using the LPG-TD planner.

Keywords: artificial knowledge representation, artificial intelligence planning, piecewise linear aggregates, finite linear temporal logic.

1. Introduction

At first, temporal logic was being created in order to formalize time tense in natural language. It occurred later that temporal logic can be successfully used as a specification language to solve the problems arising in the area of informatics. One of the first uses of temporal logic was a formalization and verification of concurrent and distributed systems. The circle of application expanded later. Now temporal logic is used in such fields as program specification, temporal databases, knowledge representation, and natural language. Temporal logics are classified according to whether time is assumed to have a linear or branching structure. CTL (computation tree logic; Clarke *et al.*, 2000) is the most used of branching time logics. CTL formulas are composed of path quantifiers and temporal operators. Both CTL and LTL (linear temporal logic) can be thought of as fragments of powerful computation tree logic CTL*. The complexity of the satisfiability problem for the logics CTL, CTL* and LTL is PSPACE or EXP. The complexity of model checking problem for CTL* and LTL formulas is PSPACE. Only the model checking problem for

formulas of CTL can be solved in polynomial time. That is why the most widely used algorithms for the search of goal is model checking for the CTL formulas (Holzman, 2003). However problems specified using LTL formulas cannot be expressed by CTL and vice versa.

In this paper, we consider the application of finite LTL to the problems related to PLA. The model of PLA is widely used for behavior and performance analysis of complex systems (Pranevicius, 1991, 2008; Pranevicius and Miseviciene, 2006). Some methods for correctness analysis of the PLA models have been developed: reachable states, invariants, application of predicate logic. The PLA verification methods permit to analyze the following properties of systems: statistical and dynamical deadlocks, termination, boundedness, completeness, absence of redundancy, invariants veracity. Applications of satisfiability checking mechanisms for LTL formulas over finite structures (Cerrito and Cialdea Mayer, 1998; Cialdea Mayer *et al.*, 2007) to the problems of PLA have not been examined. Complex structures (for example, Büchi automaton) are used to describe a linear temporal logic model. Finite sequence of nodes, to which propositional variables are assigned, describes a finite model. Finite model can be obtained using known model search algorithms for LTL formulas but it can be obtained faster using procedures which are focused on finite model search. Theoretically, using a model checking method it is possible to verify if a given formula is satisfiable or not. To this end, it is necessary to run through all possible interpretations of the formula. However that would be a primitive, highly inefficient algorithm (EXP complexity for CTL formulas and PSPACE complexity for LTL formulas). We describe a calculus where the search is performed with respect to a context of the formula. This gives a possibility to create executable interactive systems for the most tasks described by PLA. The use of finite linear temporal logic as a specification language for the problems cost in the PLA framework has several advantages. An important aspect of finite linear temporal logic is the simplicity of its model of time and actions. We have a natural representation of a world that changes over time. This logic is more expressive than classical propositional logic. Note that finite quantified LTL is undecidable (Cerrito *et al.*, 1999b). The validity problem for first-order linear temporal logic over *finite* time structures is not recursively axiomatizable. The search of answer is reduced to model search. The behavior of PLA is described by formulas in which the degree of modal operators does not exceed two. This allows the described formalizations to apply practically.

The structure of the paper is the following. The Section 2 presents a general overview of the PLA model. A finite LTL is presented in the Section 3 of the paper. The Section 4 presents a description of an alternative bit protocol by LTL. For the sake of simplicity, only a formalization method (idea) is given. In order to increase the effectiveness of the algorithm more nodes are introduced by applying the specific case. More formulas are got however all of them are described in the Section 4. In Section 5, we report the results of experiments conducted using the LPG-TD logical inference system.

We describe an alternative bit protocol task by putting it in the form of a *planning problem* (Kautz and Selman, 1992, 1999; Kautz *et al.*, 1996). Since there is a large number of planners (such as IPP, BLACKBOX, STAN, LPG-TD, SGPLAN6, MIPS-XXL),

we were not willing to implement our own solution to the problem (we should find a derivation in the calculus, which is described in the Section 3) but instead we have used one of the existing planners, namely, LPG-TD. Experiments were also carried out with two well known planners, SGPLAN6 and MIPS-XXL, which are accessible on the internet. But the best results were obtained with LPG-TD planner, so only experiment using this planner is described in Section 5. We apply a satisfiability approach to the planning problem: a plan corresponds to a model of the problem specification.

2. Piecewise Linear Aggregate Formalization Approach

PLA is a special case of automaton models. In the application of the PLA approach for system specification, the system is represented as a set of interacting piecewise linear aggregates (Pranevicius, 2008; Pranevicius and Ceponyte, 1992; Pranevicius and Miseviciene, 2003). The PLA rendered as an object defined by a set of states *Z*, a set of input signals X, and a set of output signals Y. The behavior of an aggregate is considered at discrete time moments $t \in T$. The states $z \in Z$, input signals $x \in X$, and output signals $y \in Y$ are considered to be time functions. Transition and output operators, H and G, must be known as well.

The state $z \in Z$ of the piecewise linear aggregate is the same as a state of a piece-linear Markov process, i.e., $z(t) = (\upsilon(t),\ z_\upsilon(t))$ where $\upsilon(t)$ is a discrete state component taking values on a countable set, and $z_\upsilon(t)$ is a continuous component with coordinates $z_{\upsilon 1}(t), z_{\upsilon 2}(t), \ldots, z_{\upsilon k}(t)$. When there are no inputs, an aggregate the state changes as follows:

$$\upsilon(t) = \text{const}, \quad \frac{dz_\upsilon(t)}{dt} = -\alpha_\upsilon,$$

where $\alpha_\upsilon = (\alpha_{\upsilon 1}, \alpha_{\upsilon 2}, \ldots, \alpha_{\upsilon k})$ is a constant vector.

The state of the aggregate can change in two cases only: either when an input signal arrives at the aggregate or when a continuous component takes a definite value. The theoretical basis of piecewise linear aggregates is their representation as a piece-linear Markov processes.

Continuous coordinates, which are used in PLA, define time moments when internal events occur. The aggregate state $z(t_m)$ can be changed only at discrete time moments t_m, $m = 1, 2, 3, \ldots$. The state stays to be fixed in every interval $[t_m, t_{m+1}]$, $m = 0, 1, 2, \ldots$, where t_0 – the initial moment of system behavior. When the state $z(t_m)$, $m = 0, 1, 2, \ldots$ of the system is known, the moment t_{m+1} of the next event is determined by a moment of input signal arrival at the aggregate or by the equation:

$$t_{m+1} = \min\left\{w\left(e_i'', t_m\right)\right\}, \quad 1 \leqslant i \leqslant \tilde{f},$$

where $\tilde{f}$ is the number of internal events. A class of the next event e_{m+1} is specified by the input signal. It depends on whether this signal arrives at the time moment t_{m+1} or it

is determined by control coordinate having the minimum value at the moment t_m, i.e., when the coordinate $w(e_i'', t_m)$ reaches its minimum, $e_{m+1} \in E''$.

The new state of the aggregate is defined using H operator:

$$z(t_{m+1}) = H\big[z(t_m), e_i\big], \quad e_i \in E' \cup E'',$$

where E' and E'' stand for the subsets of external and internal events, respectively.

The output signals y_i can be selected by an aggregate from the set of output signals $Y = \{y_1, y_2, \ldots, y_m\}$. This may happen only at moments of fixing the events from the subsets E' and E''. The operator G determines the content of the output signals:

$$y = G\big[z(t_m), e_i\big], \quad e_i \in E' \cup E'', \; y \in \tilde{Y}.$$

PLAs are widely applied for specification, modeling, simulation and analysis of complex systems (Pranevicius, 1991; Pranevicius and Miseviciene, 2006).

3. Linear Temporal Logic Over Finite Time Structures

The language of finite linear temporal logic considered in this paper extends classical propositional logic by means of adding the unary modal operators $\Box$ (always), $\Diamond$ (eventually), $\circ$ (next). The formula $\Box A$ means that A is true now and will always be true, $\Diamond A$ that A is either true now or sometime in the future and $\circ A$ that A holds in the next state.

A temporal structure is a finite sequence of elements called states or time points. Interpretation M consists of states s_0, s_1, …, s_n representations from N to set of subsets of literals. To every s_i, a subset $\nu(i)$ of propositional variables is assigned. The satisfiability relation $M_i \models F$ is inductively defined as follows:

- $M_i \models p$ if $p \in \nu(i)$, p is a propositional letter;
- $M_i \models \neg A$ if A is not satisfiable;
- $M_i \models A \wedge B$ if $M_i \models A$ and $M_i \models B$;
- $M_i \models A \vee B$ if either $M_i \models A$ or $M_i \models B$;
- $M_i \models A \rightarrow B$ if either A is not satisfiable or $M_i \models B$;
- $M_i \models \Box A$ if for all $j \geqslant i$, $M_i \models A$;
- $M_i \models \Diamond A$ if there exists $j \geqslant i$ such that $M_i \models A$;
- $M_i \models \circ A$ if $M_{i+1} \models A$ and $i \neq n$; if $i = n$, then failure.

The *degree* of modal operators in a propositional formula of classical logic is 0. If the degree of modal operators in a formula A is n, then the *degree* of time operators in the formulas $\Box A$, $\Diamond A$, and $\circ A$ is $n + 1$.

Next, we will describe a tableaux calculus, which makes it possible to find at least one finite model (if the formula is satisfiable) or to show that the formula hasn't got finite models. The calculus is built using the one described in work Cerrito and Cialdea Mayer (1997). The main features making our calculus different are the following:

- endpoint s_m of model interval is created;
- notion failure (defined as $\perp$) is used in rules;
- there are rules for logical operations $\neg$, $\wedge$, $\vee$ and time operators $\Box$, $\Diamond$, $\circ$.

We denote by $s, s_1, s_2, \ldots;\ s', s'', \ldots$ natural numbers. The meaning of prefixed formula $F : s$ is " F is true in the state s". Prefixed formula $F : [s_i, s_j]$ (here $s_i \leqslant s_j$) encodes the situation where F is true in each state s of finite time interval, here $s_i \leqslant s \leqslant s_j$. We consider the formulas in negation normal form. Recall that a formula is in negation normal form if all negations immediately precede atoms.

Tableaux calculus:

$$(\vee) \qquad \frac{F \vee G\colon s_i}{F\colon s_i \ \mid \ G\colon s_i}, \qquad \frac{F \vee G\colon [s_i, s_m]}{\begin{array}{c} F\colon s_i \ \mid \ G\colon s_i \\ F \vee G\colon [s', s_m] \ \mid \ F \vee G : [s', s_m] \\ s' = s_i + 1 \ \mid \ s' = s_i + 1 \end{array}},$$

$$(\wedge) \qquad \frac{F \wedge G\colon s_i}{\begin{array}{c} F\colon s_i \\ G\colon s_i \end{array}}, \qquad \frac{F \wedge G\colon [s_i, s_m]}{\begin{array}{c} F\colon [s_i, s_m] \\ G\colon [s_i, s_m] \end{array}},$$

$$(\Box) \qquad \frac{\Box F \ : \ s_i}{F \ : \ [s_i, s_m]}, \qquad \frac{\Box F \ : \ [s_i, s_m]}{F \ : \ [s_i, s_m]},$$

$$(\Diamond) \qquad \frac{\Diamond F\colon s_i}{\begin{array}{c} F\colon s' \\ s_i \leqslant s' \end{array}}, \qquad \frac{\Diamond F \ : \ [s_i, s_m]}{F \ : \ s_m},$$

where s' is new in the whole tableau,

$$(\circ) \qquad \frac{\circ F \ : \ s_i}{F \ : \ s_i + 1}, \qquad \frac{\circ F \ : \ [s_i, s_m]}{\perp}, \qquad \frac{\circ F \ : \ s_m}{\perp},$$

$$(synch) \qquad \frac{\Delta}{\begin{array}{c} s = s' \ \mid \ s < s' \ \mid \ s' < s \\ s, s' \in \Delta \end{array}}.$$

In the above formula, we denote by Δ the set of all time frames and inequalities in a branch of the search tree. It is called *limitation set*.

On every branch of the model search tree we get limitation sets. By applying synchronization rule $synch$, we do not get any new formulas. The goal of this rule is to put the time frames in an absolute order.

A branch of the model search tree is closed if one of the following conditions holds true:

- limitation set is *contradiction*. This means that it contains inequalities of the form: $s < s'$, $s' < s$;
- there are $\perp$ in this branch;

- there are two prefixed formulas of the following form in this branch of the model search tree: $A : s$, $\neg A : s'$ and from the limitation set of the analyzed branch we get that $s = s'$.

A branch is open if all its time frames are in the absolute order and the following conditions are satisfied:

- if in branch exists prefixed formula in form: $A \wedge B : s$, then there will be such $s' = s$, that in branch will such prefixed formulas $A : s'$, $B : s'$ too;
- if there exists in the branch a prefixed formula of the type: $A \vee B : s$, then there will be such a number $s' = s$ that in branch will be also include either the prefixed formula $A : s'$ or the prefixed formula $B : s'$;
- if there exists in the branch a prefixed formula of the type $\Box A : s$, then for each state $s' \geqslant s$ the branch will also include the prefixed formula $A : s'$;
- if there exists in the branch a prefixed formula of the type $\Diamond A : s$, then there will be such a number $s' \geqslant s$ that the formula $A : s'$ will belong to the branch, too;
- if there exists in the branch a prefixed formula of the type $\circ A : s$, then there will be such a number $s' = s + 1$ that the formula $A : s'$ will belong to the branch, too.

We denote by Σ the set of formulas consisting of the initial formula, formulas describing actions, successor state axioms, loop avoidance axioms and the goal formula. On the time frame j, the search of the model tree is performed using the input data $\Gamma = \{A : j \mid A \in \Sigma\}$. Finding a model amounts to finding an open branch of the search tree in Γ.

Usually the search for a model (model can be finite also) is faster performed using the described tableaux calculus than using known methods from the literature which are designated for the search of models for linear temporal logic formulas (models can be infinite also).

As an example, the search of the model for the formula $\Box\Diamond p$, which is described in the book (Ben-Ari, 2001, page 246), can be compared with the search in our calculus:

$$\Box\Diamond p : s_0,$$
$$\Diamond p : [s_0, s_m],$$
$$p : s' s_0 \leqslant s' \leqslant s_m.$$

We have found a model which has one node s' in which *p* is true.

Below we present another example which shows how quickly the calculus settles the fact that the formula $\Box\Diamond\neg p \wedge \Box \circ p$ does not have a finite model:

$$\Box\Diamond\neg p \wedge \Box \circ p : s_0,$$
$$\Box\Diamond\neg p : s_0,$$
$$\Box \circ p : s_0,$$
$$\circ p : [s_0, s_m],$$
$$\neg p : s_m,$$
$$\bot.$$

The simplicity with which finite temporal logic model is searched opens new opportunities in finding automated solution for problems. It will be illustrated by alternative bit with manager task. We did not implement our calculus. There are several available planners on the internet. We have chosen one of them which can be adopted to search for a model in our calculus.

The protocols are modeled as state-transition systems. Such structures are called Kripke structures.

4. A Finite LTL-Based Description of Alternative Bit Protocol

4.1. *Conceptual Model of Alternative Bit Protocol*

A protocol manages the transmission of information packets between two protocol objects: Sender and Receiver. The packets are transmitted through a half-duplex channel. After sending out the packet, the sender starts timer and waits for an acknowledgement. When the time expires, the packet is resent. After receiving the acknowledgement, it is assumed that the packet transmission cycle is over. Every transmitted packet contains packet number that can be either 0 or 1. This causes protocol name. After receiving the packet, the receiver sends an acknowledgement packet. It is admitted that an information packet and an acknowledgement may be either disorded or lost during transmission.

4.2. *A Description of Protocol by Finite LTL*

The protocol model consists of the following three aggregates: Sender, unreliable Channel and Receiver. The structure of the aggregate model is depicted in Fig. 1.

Next, we present the description of both the Sender aggregate and the Channel aggregate. To describe the model of the protocol aggregate using LTL two changes are done. Firstly, continuous coordinates used in PLA model are changed in the following way:

$$w(e_i'', t_m) = \begin{cases} 0, & \text{if operation initiating event } e_i'' \text{ is not active,} \\ 1, & \text{otherwise.} \end{cases}$$

After such changes have been done, all PLA coordinates describing model state are discrete. This permits using a propositional logic for description of both discrete and continuous coordinates of the aggregate model.

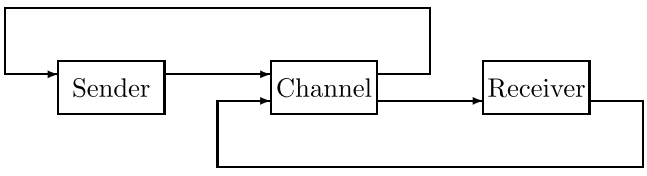

Fig. 1. Structure of the aggregate model.

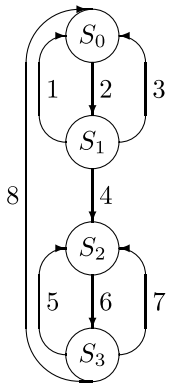

Fig. 2. State graph of the sender.

Aggregate *Sender*
Propositional variables are as follows:

$get\_conf(B)$ – confirmation has been received with alternating bit value B;
$out\_packet(B)$ – packet has been sent with alternating bit value B;
$conf\_u$ – confirmation packet is not damaged;
$packet\_formation$ – operation of packet formation is active;
$timer\_on$ – timer is switched on.

States are as follows:

S_0: $packet\_formation,\ Bit1 = 1,\ \neg timer\_on$;
S_1: $packet\_formation,\ Bit1 = 1,\ timer\_on$;
S_2: $packet\_formation,\ Bit1 = 0,\ \neg timer\_on$;
S_3: $packet\_formation,\ Bit1 = 0, timer\_on$;

The state graph of the sender is shown in Fig. 2:
The meaning of arcs in this graph is as follows:

1 – $get\_conf(B) \wedge conf\_d$;
2 – $packet\_formation(B = 1)$;
3 – $\neg timer\_on$;
4 – $get\_conf(B)$;
5 – $get\_conf(B) \wedge conf\_d$;
6 – $packet\_formation(B = 0)$;
7 – $\neg timer\_on$;
8 – $get\_conf(B)$.

Actions are as follows: $change\_Bit, form, end\_up, timer\_end\_up$.
Confirmation packet has arrived:

$$\Box((get\_conf(B) \wedge (B = Bit1) \wedge conf\_u) \rightarrow \circ do\_change\_Bit),$$
$$\Box((get\_conf(B) \wedge (B = Bit1) \wedge conf\_u \wedge do\_change\_Bit) \rightarrow \circ(\neg Bit1 \wedge packet\_formation)),$$

$\Box((get\_conf(B) \wedge (B = Bit1) \wedge conf\_u) \rightarrow do\_form)$,
$\Box((get\_conf(B) \wedge \neg(B = Bit1)) \rightarrow do\_form)$,
$\Box(do\_form \rightarrow \circ packet\_formation)$.

Formation of packet has ended:

$\Box(packet\_formation \rightarrow do\_end\_up)$,
$\Box(do\_end\_up \rightarrow \circ(\neg packet\_formation \wedge timer\_on \wedge out\_packet(Bit1)))$.

Timer has ended:

$\Box(timer\_on \rightarrow do\_timer\_end\_up)$,
$\Box(do\_timer\_end\_up \rightarrow \circ(packet\_formation \wedge \neg timer\_on))$.

Propositional variables of the *Channel* aggregate are as follows:

$in\_packet(B)$ – the packet is transmitted to channel with alternating bit value B;
$in\_acknowledgement(B)$ – the packet is transmitted to channel with alternating bit value B;
$out\_acknowledgement(B)$ – the acknowledgement has been transmitted to receiver;
$out\_packet(B)$ – the packet has been transmitted to receiver;
$lost$ – transmitted packet/ acknowledgement is lost.

Actions are as follows:

$trans\_acknowledgement$ – acknowledgement is transmitted;
$trans\_packet$ – packet is transmitted;
$trans(B = Bit2)$.

States are as follows:

S_0: $do\_trans\_packet, do\_trans(B = Bit2), \neg do\_trans\_acknowledgement$.

The packet with alternating bit value *Bit2* is transmitted through the channel.

S_1: $\neg do\_trans\_packet, do\_trans(B = Bit2), do\_trans\_acknowledgement$.

The acknowledgement with alternating bit value *Bit2* is transmitted through the channel.

S_2: $do\_trans\_packet, do\_trans(B = Bit2), \neg do\_trans\_acknowledgement$.

The channel is empty.

The state graph of the channel is shown in Fig. 3:

The meaning of arcs of this graph is as follows:

1 – $in\_acknowledgment(B)$;
2 – $in\_packet(B)$;
3 – $trans\_packet \wedge out\_packet(Bit2)$;
4 – $in\_packet(B)$;
5 – $in\_packet(B) \wedge lost$;

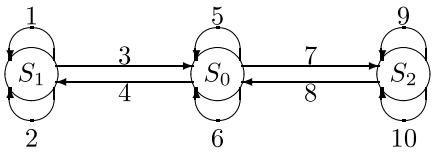

Fig. 3. State graph of the channel.

6 – $in\_acknowledgement(B) \wedge lost$;
7 – $in\_acknowledgement(B)$;
8 – $trans\_acknowledgement \wedge out\_acknowledgement(Bit2)$;
9 – $in\_acknowledgement(B)$;
10 – $in\_packet(B)$.

Actions are as follows:

Packet has sent to channel:

$\Box((in\_packet(B) \wedge \neg do\_trans\_acknowledgement \wedge \neg do\_trans_packetlost)$
$\rightarrow \circ\neg do\_trans\_packet)$,
$\Box((in\_packet(B) \wedge do\_trans\_acknowledgement) \rightarrow \circ\neg do\_trans\_packet)$
$\Box((in\_packet(B) \wedge do\_trans\_packet) \rightarrow \circ\neg do\_trans\_packet)$
$\Box(in\_packet(B) \wedge \neg do\_trans\_acknowledgement \wedge \neg do\_trans\_packet \wedge \neg lost)$
$\rightarrow \circ do\_trans\_packet)$
$\Box((in\_packet(B) \wedge \neg do\_trans\_acknowledgement \wedge do\_trans\_packet \wedge \neg lost)$
$\rightarrow \circ do\_trans(B = Bit2))$.

The acknowledgement has sent to channel:

$\Box((in\_acknowledgement(B) \wedge \neg do\_trans\_acknowledgement$
$\wedge \neg do\_trans\_packet \wedge lost) \rightarrow \circ\neg do\_trans\_acknowledgement)$
$\Box((in\_acknowledgement(B) \wedge do\_trans\_acknowledgement)$
$\rightarrow \circ do\_trans\_acknowledgement)$
$\Box((in\_acknowledgement(B) \wedge do\_trans\_packet)$
$\rightarrow \circ do\_trans\_acknowledgement)$
$\Box((in\_acknowledgement(B) \wedge \neg do\_trans\_acknowledgement$
$\wedge \neg do\_trans\_packet \wedge \neg lost) \rightarrow \circ do\_trans\_acknowledgement)$
$\Box((in\_acknowledgement(B) \wedge \neg do\_trans\_acknowledgement$
$\wedge \neg do\_trans\_packet \wedge \neg lost) \rightarrow \circ do\_trans(B = Bit2))$.

The acknowledgement has transmitted:

$\Box((do\_trans\_acknowledgement \wedge do\_trans(B = Bit2)$
$\rightarrow \circ\neg do\_trans\_acknowledgement)$
$\Box((do\_trans\_acknowledgement \wedge do\_trans(B = Bit2)$
$\rightarrow \circ out\_acknowledgement(B))$

The packet has transmitted:

$\Box((do\_trans\_packet \wedge do\_trans(B = Bit2)) \rightarrow \circ\neg do\_trans\_packet)$
$\Box((do\_trans\_packet \wedge do\_trans(B = Bit2)) \rightarrow out\_packet(B = Bit2))$.

Below we present example properties of the analysed protocol:

- *A state must be always achieved after losing either a packet or an acknowledgement that is transmitted through the channel (when a timer is off)*;
- $\Box(((trans\_packet \vee trans\_acknowledgement) \wedge lost) \rightarrow \neg timer\_on)$;
- *A sender must always get an acknowledgement when a packet for sending has been formed after a definite time*;
- $\Box((packet\_formation \wedge out\_packet(B)) \rightarrow get\_conf(B))$.

The activation of any operation depends on values assigned to the state. It will be described by the formula $\Box(C \rightarrow do(e))$. This formula means that operation *e* is activated always when *C* is true. After activation, the coordinates change. This fact is recorded using the formula $\Box(do(e) \rightarrow \circ H)$. This formula means that always, when operation *e* is activated, at the next time moment the formula *H* becomes to be true. The formula *H* describes the coordinates that have changed. Depending on the task, which is being solved using PLA, sometimes the formula $\Box(C \wedge do(e)) \rightarrow \circ H)$ must be prefered. It should be noted that more than one operation can be activated. An initial situation is described using a propositional logic formula. When any state is accessed, the output operator is activated, and the result, which includes the coordinates of particular state, is obtained. In this way, temporal logic formulas, where can be no more than another time operator occurring in operation scope of any time operator, are used for the description of the PLA behaviors. An answer for created goals is obtained by ending interpretation (Kripke structure) in which all the formulas describing aggregate behavior are true.

5. Experiment

In order to accomplish an experiment, the alternative bit protocol problem was described using PDDL (Fox and Long, 2003; Gerevini *et al.*, 2004). Many actions such as

$$\Box(do\_send \rightarrow \circ(sent \wedge \neg ready\_to\_send))$$

were transformed into actions of PDDL easily without putting on extra effort.

```
(:actiondo_send
:precondition(and(not(sent))(ready_to_send))
:effect(and(sent)(not(ready_to_send)))
.)
```

However, for example, actions (the complete list of variables and actions can be obtained via internet (home page of S. Norgėla: `http://www.mif.vu.lt/katedros/cs/`) having"OR" operator in their effect field were transformed into more than one action. This is done in order to eliminate "OR" operator since PDDL syntax does not allow disjunctive operators to appear in the effect field. By abolishing the operator, we have actions with the same preconditions but different effects, so the planner decides which

action to use. Moreover, action "do_receive" is transformed into more than one action too. This is needed to avoid not only "OR" operator but also conditional effects. So if an action either has "OR" operator in its effect field or is using conditional effects, it is transformed into more than one action of PDDL. In addition, to describe such situations as packet deformation or packet loss and to count packets, fluents are needed. For example, functions "i" and "k" are implemented to identify packet loss. When packet is sent via the channel, "i" value is incremented by one. When "i" value is equal to "k" value which is defined in problem file, init field, the packet is lost and "i" starts to count from zero. So the packet loss (as well as packet deformation) frequencies can be specified in problem file to obtain different plans.

```
(: actiondo_receive_1
:precondition(and(timer)(sent)(sendin_bit)(< (i)(k))(< (j)(1)))
:effect(and(received)(receiving_bit)
  (increase(i)1)(increase(j)1)(not(sent)))
)

(: actiondo_receive_2
: precondition(and(timer)(sent)(not(sending_bit))(< (i)(k))(<(j)(1)))
: effect(and(received)(not(receiving_bit))(increase(i)1)
  (increase(j)1)(not(sent)))
)
```

Basically, in order to arrange an experiment a planner which support fluents and negative preconditions is needed. There are not so many planners which fulfill this requirement. We have chosen LPG-TD planner (Gerevini *et al.*, 2004) because it runs on the Microsoft Windows platform. The experiment is performed by trying five times to send a particular number of packets via the channel using different packet loss and packet deformation rates. For example, three packets can be sent via the channel five times under the conditions that packet transformation rate is equal to $\frac{1}{4}$ (that means every fourth packet is deformed) and packet loss frequency $\frac{1}{2}$ (that means every second packet is lost) The obtained results include the number of actions and the time averages. It is also taken into account that the sender informs the aggregate manager about failure after three attempts to send the same packet. The results of experiments are summarized in table. For example, if a packet is sent five times via the channel with packet deformation and packet loss frequencies $\frac{1}{2}$ (that means every second packet is either deformed or lost), then the average length plan needed to send a packet is 38 actions and the average time taken to find the plan is 1.42 seconds. If two packets are sent five times via the channel with packet deformation frequency $\frac{1}{4}$ (that means every forth packet is deformed) and packet loss frequency $\frac{1}{2}$ (that means every second packet is lost), then the average length plan needed to send two packets is 29 actions and the average time needed to find the plan is 2.2 seconds. The Table 1 shows that if packet deformation and loss rates are higher, then

Table 1
Results of experiments

P	1/2, 1/2	1/2, 1/4	1/4, 1/2	1/4, 1/4	1/8, 1/8
1	38a, 1.42s	28a, 2.10s	29a, 2.2s	8a, 0.05s	8a, 0.04s
2	53a, 18.18s	52a, 23s	52a, 20s	27a, 0.88s	15a, 0.08s
3	75a, 52.2s	75a, 82.3s	77a, 80.2s	60a, 0.5s	21a, 0.1s

bigger plans will be needed. Of course, the length of the plan depends on the number of packets needed to be sent. Results were as we had expected. The first row of the table is used to specify the packet and loss rates. The first column indicates the number of packets to be sent.

The knowledge, which represents a relation between sender and receiver is described using finite linear temporal logic formulas. The core of computer-assisted system for answering to questions about the alternative bit protocol aggregate operation at various time moments has been created. Questions are formulated in the form of $\diamond A$. Is there any possible situation that A is true? For example, can we access the situation where packet acknowledgement waiting time has ended and the acknowledgement packet was not received? This question is described by the formula $\diamond(\neg timer \wedge \neg ack\_received)$. A range of software (Fox and Long, 2003; Gerevini *et al.*, 2004) has been developed for this type of information processing. This software is available on the internet. The next step is to describe restrictions and to create proof-search tactic. To make the search more effective, the knowledge about impossible events and situations, which are not needed to be considered, must be added to the system.

6. Concluding Remarks

PLA are automata very well suited for describing many complex systems. Those include protocols of computer networks, processing of distributed informations logistics and business systems among others. One of the problems tackled is to develop efficient methods for verification and validation of the systems that are amenable to PLA based descriptions. In the literature, there exist various methods for simulating the behavior of a PLA. The majority of them are theoretical by nature. In this paper, we have proposed an innovative and not yet considered approach to formalizing the notion of the automata, which is centered around the use of linear temporal logic (LTL) formulas. For verification of the systems described using PLA, we apply finite LTL. We reduce the verification problem to a planning problem. The latter was investigated in many studies including those which involved time intervals for planning. However, there are only a few specific applications where finite timer temporal logic was used. In most of the papers, only possibility to apply the LTL mechanism is discussed. In this work, we extend the application spectrum of temporal logic. Taking as an example the alternative bit protocol, we show that our method can be efficient. Furthermore, the majority of LTL applications employ the

model checking method. We instead use a different approach for verification, namely a model search method. This opens new possibilities for verification and allows considering queries of different type. We not only check whether the provided query satisfies the specifications of the problem (as the model checking method is doing), but also search for at least one model. When a query is specified using a CTL formula, solving problems of this type is not possible in practice because algorithms for searching an answer have exponential worst-case complexity. However, the problem becomes tractable when the query is described using LTL formulas and a finite model is searched. One of the merits of the method presented in this paper hinges on the fact that there exist many planners, for example, IPP,BLACBOX, ... Thus, it is possible to use already existing tools (by adopting them to searching a finite model) for solving the problems related to verification and validation of various systems when specifications of the problems as well as queries are described using linear temporal logic formulas. We have shown that whenever a complex system is formalized as a PLA with no random elements, then it can also be described by simple LTL formula (in which the degree of model operators does not exceed two) and solved by applying already existing tools (planners).

References

Ben-Ari, M. (2001). *Mathematical Logic for Computer Science*, Springer, Berlin.

Cerrito, S., Cialdea Mayer, M. (1997). A prefixed tableau calculus for plan generation in linera temporal logic. *Technical report*, RT-DIA-24-97.

Cerrito, S., Cialdea Mayer, M. (1998). Using linear temporal logic to model and solve planning problems. In: *Proceedings of the 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications*, pp. 141–152.

Cerrito, S., Cialdea Mayer, M., Praud, S. (1999). A tableau calculus for first order linear temporal logic over bound time structures. *Technical report LRI*, 1207.

Cerrito, S., Cialdea Mayer, M., Praud, S. (1999). First order linear temporal logic over finite time structures. In: *LNAI*, Vol. 1705, pp. 62–76.

Cialdea Mayer, M., Limongelli, C., Orlandini, A., Poggioni, V. (2007). Linear temporal logic as an executable semantics for planning languages. *Journal Logic, Language and Information*, 16, 63–89.

Clarke, E.M., Grumberg, O., Peled, D. (2000). *Model Checking*. MIT Press, Cambridge.

Fox, M., Long, D. (2003). PDDL 2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20, 61–124.

Gerevini, A., Saetti, A., Serina, I. (2004). LPG-TD: a fully automated planner for PDDL2.2 domains. In: *14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, Whistler, Canada.

Holzman, G.J. (2003). *The SPIN Model Checker*. Addison-Wesley, Reading.

Kautz, H., Selman, B. (1992). Planning as satisfiability, In: *Proceedings of the 10th European Conference in Artificial Intelligence*, Vienna, Austria, pp. 360–363.

Kautz, H., Selman, B. (1999). Unifying SAT-based and graph based planning. In: *Proceedings of the 16th International Joint Conference of Artificial Intelligence*, Stockholm, pp. 318–325.

Kautz, H., McAllester, D., Selman, B. (1996). Encoding plans in propositional logic. In: *Proceedings of the 4th International Conference on Knowledge Representation and Reasoning*, pp. 374–385.

Pranevicius, H. (1991). Aggregate approach for specification, validation and implementation of computer network protocols, *Lecture Notes in Computer Science*, 502, 433–477.

Pranevičius, H. (2008). *Analysis and Formalizations of Complex Systems*. Kauno Technologijos Universitetas, Kaunas (in Lithuanian).

Pranevicius, H., Ceponyte, R. (1992). Application of logic programming based for validation of computers network protocols aggregate specifications. *Automatic and Computing Technique*, 2, 22–27.

Pranevicius, H., Miseviciene, R. (2003). Transformation of aggregate specifications to the predicate logic models. In: *The International Workshop on Harbour, Maritime and Multimodal Logistics Modelling & Simulation*, Riga, pp. 378–384.

Pranevicius, H., Miseviciene, R. (2006). Knowlegde based verification of aggregate specifications. In: *Proceedings of Fifth Mexican International Conference on Artificial Intelligence*, pp. 3–11.

H. Pranevicius is author of a few monographs in area of formal specification and analysis of distributed systems using piece-linear aggregate approach. He is a professor, hab. dr, head of Business Informatics Department. His research interests include the use formal methods for verification and simulation of complex software systems.

S. Norgėla was awarted the candidate of physical-mathematical sciences at Leningrad Steklov Institute in 1978. He is an associate professor of Vilnius University at the Computer Sciences Chair. His research interests include the problems of artificial intelligence, nonclassical logics etc.

Baigtinės tiesinio laiko logikos takymai atkarpomis tiesiniams agregatams

Henrikas PRANEVICIUS, Stanislovas NORGĖLA

Atkarpomis tiesinių agregatų darbas aprašomas baigtinės laiko logikos formulėmis. Eksperimentuota su alternatyvaus bito agregatu. Formulės užrašytos PDDL kalba. Eksperimentui panaudota programa LPG-TD.