Runtime Model Checking of Log-Message Traces Based on Extensions of Linear Temporal Logic of Past

Vladislav Podymov

National Research University Higher School of Economics,
3 Kochnovsky Proezd, Moscow 125319, Russia
valdus@yandex.ru

Abstract. The following problem is investigated: given a distributed system description, and a collection of finite traces (logs), one for each component of a system, verify whether the collection satisfies a given temporal property. In the investigated variation of the problem, trace events (log messages) are assumed to be marked with real-valued timestamps, and to carry data from limited finite domains related to system component identifiers, which is not uncommon for logs generated by real applications. Surprisingly, though for each certain detail of the problem specifics (such as data parametrization, real time, past tense, and so on) there exist logical verification methods, nevertheless we could not find any temporal logic formalism well-suited for all these details at once.
The main results of the paper are: syntax and semantics of a logic suited for the considered problem (which we call a real-time predicative indexed linear temporal logic of past and bounded past), a formal statement of a verification problem for this logic, and a decision procedure for this formally stated verification problem.

Keywords: runtime verification, linear temporal logic, logic of past, real-time logic, indexed logic

1 Introduction

Imagine yourself a network administrator maintaining a reliable computer network which you cannot fully observe, e.g. a large local area network, or a campus area network. Once configured and launched, the network meets a large scope of errors and faults, even if all network devices are properly tested. These might be, for example, configurational errors (e.g. some rarely used connection was not reconfigured during major updates), hardware faults (e.g. someone physically disabled crucial devices or links), or quality of service issues (e.g. some network node cannot manage current amount of traffic). To handle such anomalities, you have a large scope of mathematical methods and tools (see, for example, a little part of these in [1, 2]). In any case, to apply these means, you should put some effort to configure network devices to send you data required for network behavior analysis.

Consider a network device configured in such a way that its applications (e.g. well-known implementations of standard communication protocols) send some minimal amount of time-stamped log messages to a certain server. One of the ways to check if the network is working properly is to manually look through all the messages from all the devices and try to find any anomalities. In general, such manual expertise is time-consuming and does not guarantee that all anomalities were found. We try to provide formal means for automatization of such an expertise.

For clarity and simplicity we may focus on simple toy logs which can be considered as generated by an implementation of a well-known routing protocol OSPF[1]. Note that there exist results on verification of some aspects of OSPF protocol (such as [3–5]), but not the aspect we focus on. A brief description of the considered version of the protocol with lots of unnecessary technical details omitted is as follows. The network consists of routers linked to each other in some way. The network is divided into areas — groups of routers. Some of the routers may belong to several areas at once. Routers which do not belong to some common area are not linked and cannot communicate at all. OSPF protocol runs on each router, and its goal is to deliver information about the network topology to all routers. Each router

- sends special messages (HELLO) over its OSPF-related network interfaces to discover its neighbours (linked routers) in each area;
- if HELLO is received, responds and establishes a two-way OSPF communication channel; if the channel is established, then the routers are considered linked (adjacent neighbours) in a given area;
- along with discovery of area neighbours, elects two special routers: designated router, and backup designated router;
- exchanges known information about the whole network topology with the designated router;
- under special conditions may rediscover neighbours, reelect designated routers, switch designated and backup designated routers for balancing purposes, and reexchange information about topology with the designated router.

Considered log messages (sent by every router to a selected server) do not contain any information on the network topology, as the size of information on the whole network topology is huge, so it is inexpedient to send it from network nodes for the sake of logging only. This means that log messages should contain only information about simple events correlated to a high-level protocol description, such as: HELLO is sent, a neighbour became adjacent, designated router elected, and so on. The problem we investigate is: in particular, how to formally verify whether given OSPF logs collected from several devices contains any “abnormalities”; in general, what can be the most suited formal means for such log-based verification of distributed communication systems.

The most straightforward way to capture causality between network events represented by timestamped messages is to use any of temporal logic formalisms

[1] See, for example, https://www.ietf.org/rfc/rfc2328.txt (checked 17.07.2017)

and "classical" model checking methods [6, 7]. The usual way to apply these formalisms is to check whether a finite system (a finite collection of states, and a transition relation which produces an infinite number of infinite state sequences — traces) satisfies a specification (a formula of any suited, usually temporal, logic). However, the considered log verification problem has several discrepancies with a straightforward classical model checking approach:

1. no state-space description is given, only one finite execution trace is to be verified;
2. model-checking logic is usually (though not always) well-suited to capture future time, or no time at all, while in case of one finite trace what we often need is to consult past log events with respect to a current analyzed time;
3. log messages are parametrized by data; in our example — such as area identifiers and/or IP addresses (e.g. an event "router i elected router j to be designated in subnet S" is naturally parameterized by i, j, and S);
4. log messages are marked with real-valued timestamps.

Trying to handle the first discrepancy, we immediately obtain a model-checking scope often called "runtime model checking" which is (briefly) an extraction of a system real execution trace in runtime and checking properties of this trace (as fast as possible, ideally — in runtime). Runtime model checking of networks is usually intended to check properties of each particular "snapshot" of a network model in which no explicit time dependencies are captured (such as mentioned in [2, 8, 9] for a specific kind of networks). However, if not to limit ourselves with network-related formalisms, there exists an advanced general-purpose formalism [10, 11] which includes as a special case a single-trace verification based on linear temporal logic of past and bounded past (modalities to look through a certain number of time units to the past from now). This handles the first two discrepancies, but leave the other ones open: data parametrization provided by [10, 11] is rather unclear, seems to be limited, does not have means for uniform specification independent of a certain network structure, and due to software-monitoring intention of a proposed formalism is strongly connected to Java built-in data types and does not capture real-valued timestamps in a natural way.

The third discrepancy (but not the other ones) can be handled almost perfectly by indexed linear temporal logic (ILTL) [12], which extends classical linear temporal logic with quantifiers over special index variables which denote system subcomponent identifiers. ILTL is used in a certain model-checking area — model checking of parameterized systems — which investigates the following problem: given a specification, and a description which contains a number of parameters and denotes an infinite family of systems (each system defined by certain parameter values), check whether the specification holds for every system in the family (i.e. for every parameterized system instance). Surprisingly, while for non-indexed future-time logics and related verification problems (to check a temporal formula against infinite traces of a system state space) there exist runtime analogues (to check a temporal formula against a single finite trace only), for ILTL model checking problem (to check an indexed temporal formula against

a family of system instances) we could not find any runtime analogue (to check an indexed formula against certain parameter values of a system instance, and a finite execution trace), even more so, a real-timed analogue. Though there exist real-time temporal logics of future, in some cases extended by limited versions of past, such as mentioned in [13–15] (which handle the last discrepancy, and partially the second one), we could not find any logic containing other discussed desired features at the same time.

What we propose in this paper is

- (Section 2) syntax and semantics of (what we call) a real-time predicative indexed linear temporal logic of past and bounded past which covers all discrepancies mentioned above at once and thus seems to be suitable for a proposed runtime log verification problem (and which we abbreviate not nicely as RPILTLP due to the lack of imagination during the invention),
- (Section 2) a problem statement for runtime verification of RPILTLP-formula agaist a certain system instance and its finite execution trace,
- (Section 3) a decision algorithm for the stated problem.

Section 4 provides several exemplary OSPF-related properties expressed as RPILTLP-formulae which show the usefulness of runtime verification for RPILTLP.

2 Logic

Syntax and semantics of RPILTLP is partially motivated by those of ILTL. For those familiar with indexed temporal logics, before strict formal definitions we describe distinctions between ILTL and proposed RPILTLP as follows. Start with syntax of ILTL formulae [12] without next-time modality. Replace temporal modalities with their past-time and bounded past-time analogues. Replace atomic propositional expressions p_i of ILTL (atomic proposition p holds for a system instance component identified by an index i) with atomic predicative expressions of the form $p_i(j_1, \dots, j_k)$: a predicate p holds for arguments $j_1, \dots, j_k$ and a component identified by an index i. Extend quantification over subsets of a finite set of indexes ($\forall i : e.\varphi$, $\exists i : e.\varphi$, where e is an expression interpreted as a set of indexes) with limited quantification over sets of indexes. Interpret RPILTLP-formulae over a tuple consisting of timestamped log messages of system components, a current timestamp, and an additional information about the system instance.

We start formal syntax of RPILTLP-formulae with

- a finite set AP of *atomic propositions*,
- an *arity function* $ar : AP \to \{0, 1, 2, \dots\}$,
- a finite set T of *types*, and
- disjoint sets V_i and V_t of *index variables* and *type variables* respectively.

Syntax of *RPILTLP-formulae* is defined by the following grammar:

$$\varphi ::= p_x(y_1, \dots, y_m) \mid \varphi \; bop \; \varphi \mid uop \; \varphi \mid \exists y : e.\varphi \mid \forall y : e.\varphi \;,$$

where

- φ is an RPILTLP-formula,
- $p \in AP$, and $ar(p) = m$,
- $x \in V_i$, and $y, y_1, \ldots, y_m \in V_i \cup V_t$,
- *bop* is either a binary boolean operator (a conjunction $\wedge$, a disjunction $\vee$, or an implication $\rightarrow$), or a binary temporal modality (the list of temporal modalities is given below),
- *uop* is either a negation $\neg$, or a unary temporal modality (the list of modalities is given below),
- e is a (possibly empty) list of expressions of the following form, with $t \in T \cup V_t$, $z \in V_i$:
  - $y \in t$, $y = z$, or $y \neq z$, if $y \in V_i$;
  - $z \in y$, $y = t$, or $y \neq t$, if $y \in V_t$.

Temporal modalities and semantics of RPILTLP-formulae are based on real time. To represent it, we use the term *timestamp* to denote a nonnegative real numer. The set of all nonnegative real numbers is denoted by $\mathbb{R}^{\geq 0}$, and positive ones — by $\mathbb{R}^{>0}$. We use the following modalities of past (formal semantics is given further; $k \in \mathbb{R}^{>0}$):

- a unary modality $\diamond$ for "eventually in past",
- unary modalities $\diamond^{<k}$, $\diamond^{\leq k}$, $\diamond^{=k}$ for "eventually in past in less than k time units from now", "... not more than k time units from now" and "... exactly k time units from now",
- a unary modality $\Box$ for "always in past",
- unary modalities $\Box^{<k}$, $\Box^{=k}$, $\Box^{\leq k}$ with the same additional meaning for $< k$, $= k$, $\leq k$ as for $\diamond^{<k}$, $\diamond^{\leq k}$, and $\diamond^{=k}$.
- a binary modality $\mathbf{S}$: $\varphi \mathbf{S} \psi$ means "φ since ψ": ψ holds eventually in past, and since that moment up to a current moment φ holds.
- binary modalities $\mathbf{S}^{<k}$, $\mathbf{S}^{\leq k}$, $\mathbf{S}^{=k}$ with the same replacement of "eventually" with "eventually in certain time" as for $\diamond^{<k}$, $\diamond^{\leq k}$, $\diamond^{k}$.

A (*system*) *interpretation* $(Ind, \mathcal{T}, Val, \mathcal{L})$ of an RPILTLP-formula consists of the following components:

- a finite set Ind of *indexes*, $Ind \cap T = \emptyset$;
- a *type mapping* $\mathcal{T} : T \rightarrow 2^{Ind}$ which maps each type into a set of indexes;
- a *variable mapping* $Val : (V_i \rightarrow Ind) \cup (V_t \rightarrow T)$ which maps every index variable into an index, and every type variable into a type;
- a *log description* $\mathcal{L} : Ind \rightarrow (\mathbb{R}^{\geq 0} \rightarrow 2^{Events})$ (the set of events $Events$ is defined below) such that for each index i the number of timestamps tm for which $\mathcal{L}(i)(tm) \neq \emptyset$ is finite.

We write $\mathcal{L}(i, tm)$ along with $\mathcal{L}(i)(tm)$ for simplicity. A relation $ev \in \mathcal{L}(i, tm')$ means that the event ev occured on i-th system component at a time tm. An event is an expression of the form $p(d_1, \ldots, d_{ar(p)})$, where $p \in AP$ and $d_1, \ldots, d_{ar(p)} \in Ind \cup T$. Note that by definition for each index i the number of timestamps for which any event occur in a log description is finite, which means

that the description is actually a finite collection of finite traces of all system components.

To shorten the definition of formal semantics given below, we use the following auxiliary definitions. An interpretation $(Ind, \mathcal{T}, Val_1, \mathcal{L})$ is a *y-variant* of an interpretation $(Ind, \mathcal{T}, Val_2, \mathcal{L})$ for $y \in V_i \cup V_t$, if for each variable z such that $z \in V_i \cup V_t$ and $z \neq y$, $Val_1(z) = Val_2(z)$. *Terminal values* of an index variable x, a type t, and a type variable y in an interpretation $\mathfrak{I} = (Ind, \mathcal{T}, Val, \mathcal{L})$ are the index $\mathfrak{I}(x) = Val(x)$, the set of indexes $\mathfrak{I}(t) = \mathcal{T}(t)$, and the set of indexes $\mathfrak{I}(y) = \mathcal{T}(Val(y))$, respectively. An interpretation $\mathfrak{I}$ satisfies expressions $x \in y$, $x = y$, $x \neq y$, if $\mathfrak{I}(x) \in \mathfrak{I}(y)$, $\mathfrak{I}(x) = \mathfrak{I}(y)$, and $\mathfrak{I}(x) \neq \mathfrak{I}(y)$ respectively, where the latter three relations are interpreted in natural way.

The *satisfiability* relation $\mathfrak{I}, tm \models \varphi$ for an interpretation $\mathfrak{I}$, a timestamp tm, and an RPILTLP-formula φ is defined inductively as follows:

- $\mathfrak{I}, tm \models p_i(x_1, \ldots, x_{ar(p)})$ iff $p(Val(x_1), \ldots, Val(x_{ar(p)})) \in \mathcal{L}(i, tm)$;
- $\mathfrak{I}, tm \models \neg\varphi$ iff $\mathfrak{I}, tm \not\models \varphi$;
- $\mathfrak{I}, tm \models \varphi \wedge \psi$ iff $\mathfrak{I}, tm \models \varphi$ and $\mathfrak{I}, tm \models \psi$;
- $\mathfrak{I}, tm \models \varphi \vee \psi$ iff $\mathfrak{I}, tm \models \varphi$ or $\mathfrak{I}, tm \models \psi$;
- $\mathfrak{I}, tm \models \varphi \rightarrow \psi$ iff $\mathfrak{I}, tm \models \neg\varphi \vee \psi$;
- $\mathfrak{I}, tm \models \diamond\varphi$ iff there exists a timestamp tm' such that $tm' \leq tm$ and $\mathfrak{I}, tm' \models \varphi$;
- $\mathfrak{I}, tm \models \diamond^{op\ k}\varphi$ iff there exists a timestamp tm' such that $tm' \leq tm$, $tm - tm'$ *op* k, and $\mathfrak{I}, tm' \models \varphi$;
- $\mathfrak{I}, tm \models \Box\varphi$ iff for all timestamps tm' such that $tm' \leq tm$, the relation $\mathfrak{I}, tm' \models \varphi$ holds;
- $\mathfrak{I}, tm \models \Box^{op\ k}\varphi$ iff for all timestamps tm' such that $tm' \leq tm$ and $tm - tm'$ *op* k, the relation $\mathfrak{I}, tm' \models \varphi$ holds;
- $\mathfrak{I}, tm \models \varphi \mathbf{S} \psi$ iff there exists a timestamp tm' such that $tm' \leq tm$ and both of the following hold:
  - $\mathfrak{I}, tm' \models \psi$;
  - for all timestamps tm'' such that $tm' < tm'' \leq tm$ holds $\mathfrak{I}, tm'' \models \varphi$;
- $\mathfrak{I}, tm \models \varphi \mathbf{S}^{op\ k} \psi$ iff there exists a timestamp tm' such that $tm' \leq tm$, $tm - tm'$ *op* k, and both of the following hold:
  - $\mathfrak{I}, tm' \models \psi$;
  - for all timestamps tm'' such that $tm' < tm'' \leq tm$ holds $\mathfrak{I}, tm'' \models \varphi$;
- $\mathfrak{I}, tm \models \exists y : e.\varphi$ iff there exists an y-variant $\mathfrak{J}$ of $\mathfrak{I}$ such that $\mathfrak{J}$ satisfies all expressions in e, and $\mathfrak{J}, tm \models \varphi$;
- $\mathfrak{I}, tm \models \forall y : e.\varphi$ iff for all y-variants $\mathfrak{J}$ of $\mathfrak{I}$ satisfying all expressions in e holds $\mathfrak{J}, tm \models \varphi$.

We state a *runtime verification problem for RPILTLP* as follows: given an interpretation $\mathfrak{I}$, a timestamp tm, and an RPILTLP-formula φ, check whether the relation $\mathfrak{I}, tm \models \varphi$ holds.

3 Decision Procedure

This section provides a decision procedure for the runtime verification problem in RPILTLP, i.e. an algorithm for checking a relation $\mathfrak{I}, tm \models \varphi$, where φ is an RPILTLP-formula. To claim that this procedure is an "algorithm" (as stated in Introduction), we assume any reasonable finite representation of all used real numbers, which complies with usual "real" log data in which timestamps are actualy not real-valued, but rationally-valued. Note that the procedure cannot be derived from results on similar logics mentioned in Introduction: mentioned real-time logics and ILTL are undecidable; mentioned results on runtime verification (and ILTL) lack real time.

For clarity we assume that $\mathfrak{I} = (Ind, \mathcal{T}, Val, \mathcal{L})$.

The following statements obviously hold for any interpretation $\mathfrak{J}$, any timestamp tm', and any formulae ψ, χ:

- $\mathfrak{J}, tm' \models \psi \wedge \chi$ iff $\mathfrak{J}, tm' \models \neg(\neg\psi \vee \neg\chi)$;
- $\mathfrak{J}, tm' \models \psi \to \chi$ iff $\mathfrak{J}, tm' \models \neg\psi \vee \chi$;
- $\mathfrak{J}, tm' \models \diamond\psi$ iff $\mathfrak{J}, tm' \models true\ \mathbf{S}\psi$, where $true$ is a tautology (e.g. a formula of the form $\chi \vee \neg\chi$);
- $\mathfrak{J}, tm' \models \diamond^{op\ k}\psi$ iff $\mathfrak{J}, tm' \models true\ \mathbf{S}^{op\ k}\psi$;
- $\mathfrak{J}, tm' \models \Box\psi$ iff $\mathfrak{J}, tm' \models \neg \diamond \neg\psi$;
- $\mathfrak{J}, tm' \models \Box^{op\ k}\psi$ iff $\mathfrak{J}, tm' \models \neg \diamond^{op\ k} \neg\psi$;
- $\mathfrak{J}, tm' \models \psi\mathbf{S}\chi$ iff $\mathfrak{J}, tm' \models \psi\mathbf{S}^{\leq tm'}\chi$;
- $\mathfrak{J}, tm' \models \psi\mathbf{S}^{\leq k}\chi$ iff $\mathfrak{J}, tm' \models \psi\mathbf{S}^{<k}\chi$ or $\mathfrak{J}, tm \models \psi\mathbf{S}^{=k}\chi$;
- $\mathfrak{J}, tm' \models \forall y : e.\psi$ iff $\mathfrak{J}, tm' \models \neg\exists y : e.\neg\psi$.

Based on these statements, operations $\wedge$, $\to$, modalities $\diamond$, $\diamond^{op\ k}$, $\Box$, $\Box^{op\ k}$, $\mathbf{S}$, $\mathbf{S}^{\leq k}$, and universal quantifiers can be easily eliminated from φ. Thus, without loss of generality, the following case is considered: φ contains atomic expressions $p_x(y_1, \ldots, y_m)$, operations $\vee$, modalities $\mathbf{S}^{<k}$, $\mathbf{S}^{=k}$, existential quantifiers, and nothing else.

To check the relation $\mathfrak{I}, tm \models \varphi$, we solve a more general problem: to construct a set $TS(\mathfrak{I}, tm, \varphi)$ of all timestamps tm' such that $tm' \leq tm$, and $\mathfrak{I}, tm' \models \varphi$. Then $\mathfrak{I}, tm \models \varphi$ iff $tm \in TS(\mathfrak{I}, tm, \varphi)$.

To handle sets of the form $TS(\mathfrak{J}, tm', \psi)$ algorithmically, we represent them as finite unions of disjoint intervals. An *interval* (of timestamps) is an expression of the form $\langle l, h\rangle$, where

- l and h are timestamps (a lower bound and an upper bound respectively), $l \leq h$;
- $\langle$ is either ((the lower bound is open), or [(the lower bound is closed);
- $\rangle$ is either) (the higher bound is open), or] (the higher bound is closed);
- if any of the bounds is open, then $l < h$.

The set $\overline{\langle l, h\rangle}$ represented by an interval $\langle l, h\rangle$ is defined as follows: $\overline{(x, y)} = \{z \mid x < z < y\}$, $\overline{[x, y)} = \{z \mid x \leq z < y\}$, $\overline{(x, y]} = \{z \mid x < z \leq y\}$, $\overline{[x, y]} = \{z \mid x \leq z \leq y\}$. An *interval description* (of a set) is a finite sequence

of intervals $\langle l_1, h_1\rangle, \dots, \langle l_k, h_k\rangle$, such that $h_i \le l_{i+1}$ for $i \in \{1, \dots, k-1\}$, and if any of the bounds is closed, then $h_i < l_{i+1}$. The set $\overline{I}$ represented by an interval description I is the union of the sets represented by its intervals.

Thus, the main goal of the algorithm is to construct an interval description $ID(\mathfrak{I}, tm, \varphi)$ such that $\overline{ID(\mathfrak{I}, tm, \varphi)} = TS(\mathfrak{I}, tm, \varphi)$. The algoritm constructs $ID(\mathfrak{I}, tm, \varphi)$ inductively: $ID(\mathfrak{I}, tm, p_i(x_1, \dots, x_k))$ is constructed explicitly (base case below); an interval description of a complex formula is constructed with an operation on interval descriptions of its subformulae (constructed on earlier steps, in some cases for different interpretations).

Hereafter we use set-theoretical operations applied to interval descriptions instead of sets: for any interval descriptions ID_1, ID_2, and an operation $op \in \{\cup, \cap, \setminus\}$, $ID_1\ op\ ID_2$ is an interval description ID such that $\overline{ID} = \overline{ID_1}\ op\ \overline{ID_2}$. Strict definitions of these operations are given in Appendix. We admit that there may be classical results in which all these operations on disjoint sets of intervals are already described, but we did not look thoroughly (as it is an auxiliary technical construct), found no results for the considered case (mixed open and closed bounds), and decided to describe all needed operations from scratch.

Hereafter TS_η, ID_η, where η is a formula, denote a set $TS(\mathfrak{I}, tm, \eta)$, and an interval description $TS(\mathfrak{I}, tm, \eta)$, respectively.

Base case: $\varphi = p_i(x_1, \dots, x_{ar(p)})$, $p \in AP$. By the definition of a log description, the set TS_φ is finite. Let $TS_\varphi = \{tm_1, \dots, tm_k\}$, where $tm_1 < \dots < tm_k$. Then $ID_\varphi = ([tm_1, tm_1], \dots, [tm_k, tm_k])$.

Case 1: $\varphi = \neg\psi$. Obviously, $ID_\varphi = ([0, tm]) \setminus ID_\psi$.

Case 2: $\varphi = \psi \vee \chi$. Obviously, $ID_\varphi = ID_\psi \cup ID_\chi$.

Case 3: $\varphi = \psi \mathbf{S}^{<k} \chi$. A bit informally, the semantics of $S^{<k}$ states that TS_φ is a union of the following intervals $ext_{\tilde{tm}}$, one for each timestamp $\tilde{tm}$ of TS_χ:

- if $\overline{(\tilde{tm}, \tilde{tm} + k)} \subseteq TS_\psi$, then $ext_{\tilde{tm}} = [\tilde{tm}, \tilde{tm} + k)$;
- otherwise, $ext_{\tilde{tm}}$ is the maximal interval of the form $[\tilde{tm}, tm'\rangle$, such that $\rangle \in \{),]\}$, $\tilde{tm} \le tm' < \tilde{tm} + k$, and $\overline{(\tilde{tm}, tm'\rangle} \subseteq TS_\psi$.

If there exists an interval $\langle l, h\rangle$ of ID_ψ such that $l \le \tilde{tm} < h$, then

- $ext_{\tilde{tm}} = [\tilde{tm}, \tilde{tm} + k)$, if $h \ge \tilde{tm} + k$;
- $ext_{\tilde{tm}} = [\tilde{tm}, h\rangle$, if $h < \dot{m} + k$.

Otherwise, $ext_{\tilde{tm}} = [\tilde{tm}, \tilde{tm}]$.

It is sufficient to show how to construct an interval $ext_I = \bigcup\limits_{\tilde{tm} \in \overline{I}} ext_{\tilde{tm}}$ for each interval $I = \langle u, v\rangle$ of ID_χ; then ID_φ is a (finite) union of all these intervals. Let $I = \langle u, v\rangle$, and $ext_v = [v, H\rangle$. If the upper bound of I is closed, then $ext_I = \langle v, H\rangle$ with the same openness of the upper bound as in ext_v. Otherwise, the upper bound of I is open, and two subcases are possible:

1. There exists an interval $I' = \langle p, q\rangle$ of ID_ψ, such that $p < v < q$. Then for each timestamp tm'', such that $v < tm'' < v + k$ and $tm'' < q$ (if the bound q is open), or $tm'' \le q$ (if the bound q is closed), there exists a timestamp $\tilde{tm}$ such that $p < \tilde{tm} < v$ and $\overline{(\tilde{tm}, tm'')} \subseteq \overline{I'}$. It means that $ext_I = [v, H\rangle$ with the same openness of the upper bound as in ext_v.

2. $v \notin \overline{(p,q)}$ for any interval $\langle p,q\rangle$ of ID_ψ. Then for any timestamp $\tilde{tm}$ of the set $\overline{I}$ it holds that $\overline{ext_{\tilde{tm}}} \subseteq \overline{I}$. It means that $ext_I = I$.

Case 4: $\varphi = \psi \mathbf{S}^{=k} \chi$. In this case, TS_φ is the set of timestamps tm', $tm' \le tm$, such that $(tm' - k) \in TS_\chi$, and for all timestamps tm'', $tm' - k < tm'' \le tm'$, it holds that $tm'' \in TS_\psi$. Let $ID_\psi = (\langle l_1, h_1\rangle, \ldots, \langle l_n, h_n\rangle)$, and $ID_\chi = (\langle u_1, v_1\rangle, \ldots, \langle u_m, v_m\rangle)$. To obtain ID_φ from ID_ψ, it is sufficient to:

- raise all lower bounds of intervals of ID_ψ by k (exclude intervals if they become empty),
- make all of the lower bounds closed, and
- intersect the resulting interval description with intervals of ID_χ raised by k: $(\langle u_1 + k, v_1 + k\rangle, \ldots, \langle u_m + k, v_m + k\rangle)$.

Case 5: $\varphi = \exists y : e.\psi$. In this case, TS_φ is a union of sets $TS(\mathfrak{J}, tm, \psi)$ for all y-variants $\mathfrak{J}$ of $\mathfrak{I}$ satisfying all expressions in e. As the set of indexes is finite, the number of the sets $TS(\mathfrak{J}, tm, \psi)$ is finite. It means that ID_φ is a finite union of interval descriptions $ID(\mathfrak{J}, tm, \psi)$ (previously constructed by induction).

Though the main result of this section cannot be stated as "the runtime verification problem for RPILTLP is decidable" due to nonalgorithmical nature of real numbers, the procedure description together with a correctness explanation (given along with the description) allows to state a bit weaker but still useful result, if a description of the relation $\mathfrak{I}, tm \models \varphi$ uses rational numbers (not arbitrary real numbers) in any reasonable representation.

Theorem 1. *The runtime verification problem for RPILTLP is decidable, if all numbers used in the problem statement $\mathfrak{I}, tm \models \varphi$ are rational.*

4 Examples of Properties

To provide an example for usage of RPILTLP, it is sufficient to describe certain atomic propositions with arities, fix some number of types, describe an exemplary interpretation, and explain the meaning of all described components. Recall a simplified version of OSPF protocol in Introduction. The following exemplary messages (events) indicating high-level stages of a simplified OSPF protocol (and not only they) may be sent by routers:

- $HelloSent(a)$ — indicates that the router has sent a multicast HELLO message inside the area a;
- $HelloReceived(a, i)$ — indicates that the router has received a HELLO message inside the area a from the router indexed by i;
- $DRElected(a, i)$ — indicates that the router has elected its neighbour indexed by i to be the designated router in the area a;
- $ExchangeStarted(i)$ — indicates that the router exchanges information about a network topology with a router indexed by i.

Other events may be considered, but these exemplary event types are suffient to formulate useful properties of a timestamped protocol log. Exemplary atomic propositions and their arity is deduced from the messages in an obvious way.

Consider the following netowrk structure: the network consists of three routers divided into two areas; the first router belongs to the first area; the second router belongs to the second area; the third router is a border router which belongs to both areas. Then we may define three types: N as the whole network, and $A1$, $A2$ as the two areas — and a suitable interpretation may consist of:

- the set of indexes is $\{1, 2, 3\}$ to denote the first, the second, and the third routers;
- the mapping of the first type to $\{1, 2, 3\}$, of the second type — to $\{1, 3\}$, and of the third type — to $\{2, 3\}$.

Meaningful RPILTLP-formulae to be checked for a considered interpretation are, for example,

- a designated router shoud be elected in less than three seconds since the first HELLO message:
  $\forall i : .\forall a : .\Box((\diamond^{=3}(HelloSent_i(a) \vee \exists j : .HelloReceived_i(a, j))) \rightarrow \diamond\exists j : j \in a.DRElected_i(a, j))$;
- a router should exchange topology information with a designated router only:
  $\forall i : .\forall j : .\Box(ExchangeStart_i(j) \rightarrow \exists a : .((\neg\exists k : k \neq j.DRElected_i(a, k))\mathbf{S}DRElected_i(a, j))$;
- if a HELLO message is delivered, it happens within 3 seconds:
  $\forall i : .\forall a : .\forall j : .\Box(HelloReceived_i(a, j) \rightarrow \diamond^{\leq 3}HelloSent_j(a))$.

5 Conclusion

We proposed a real-time predicative indexed linear temporal logic of past and bounded past (RPILTLP), which may seem to be rather heavy formalism with lots of syntactical and semantical details comparing to more "classical" well-known temporal logics, but in return is suitable for runtime verification of finite collections of parametrized messages equipped with real-valued timestamps, which seem to be a rather natual representation of logging information from real running applications. Due to novelty of the logic itself, all implementation and experimental research based on "more-or-less real logs" should be considered as a future work, though we note that the proposed formalism was inspired by real collections of complex logging data generated by communication protocol implementations.

Acknowledgements. Support from the Basic Research Program of the National Research University Higher School of Economics is gratefully acknowledged.

Appendix

Consider interval descriptions

$$ID_1 = (\langle l_1, h_1\rangle, \ldots, \langle l_n, h_n\rangle),\ ID_2 = (\langle u_1, v_1\rangle, \ldots, \langle u_m, v_m\rangle)\ ,$$

and an interval $I = \langle l, h\rangle$. This small section shows how to construct interval descriptions $ID_1 \cup I$, $ID_1 \cup ID_2$, $ID_1 \cap I$, $ID_1 \cap ID_2$, $ID_1 \setminus I$, and $ID_1 \setminus ID_2$, such that $\overline{ID_1 \cup I} = \overline{ID_1} \cup \overline{I}$, $\overline{ID_1 \cup ID_2} = \overline{ID_1} \cup \overline{ID_2}$, $\overline{ID_1 \cap I} = \overline{ID_1} \cap \overline{I}$, $\overline{ID_1 \cap ID_2} = \overline{ID_1} \cap \overline{ID_2}$, $\overline{ID_1 \setminus I} = \overline{ID_1} \setminus \overline{I}$, and $\overline{ID_1 \setminus ID_2} = \overline{ID_1} \setminus \overline{ID_2}$.

- $ID_1 \cup I$ (obtained from ID_1). Add I in ID_1 syntactically according to the order of lower bounds. Do the following while possible: take two neighbouring intervals $\langle l', h'\rangle, \langle l'', h''\rangle$ such that $\overline{\langle l', h'\rangle} \cap \overline{\langle l'', h''\rangle} \neq \emptyset$, and replace them with
  - $\langle l', h'\rangle$ with unchanged type of bound h', if $h' > h''$;
  - $\langle l', h''\rangle$ with unchanged type of bound h'', if $h'' > h'$;
  - $\langle l', h'\rangle$, if $h' = h''$, and make the bound h' weakest between initial bounds h', h'' (closed bound is weaker than open bound).

  Type of bound l' remains unchanged in any case.
- $ID_1 \cup ID_2 = (((ID_1 \cup \langle u_1, v_1\rangle) \cup \ldots) \cup \langle u_m, v_m\rangle))$.
- $ID_1 \cap I$ (obtained from ID_1). If $\overline{\langle l_i, h_i\rangle} \cap \overline{\langle l, h\rangle} = \emptyset$, then delete $\langle l_i, h_i\rangle$ from ID_1. Otherwise $\overline{\langle l_i, h_i\rangle} \cap \overline{\langle l, h\rangle}$ can be represented as an interval in an obvious way — replace $\langle l_i, h_i\rangle$ with this intersected interval.
- $ID_1 \cap ID_2 = (ID_1 \cap \langle u_1, v_1\rangle) \cup \cdots \cup (ID_1 \cap \langle u_m, v_m\rangle)$.
- $ID_1 \setminus I$ (obtained from ID_1). If $\overline{\langle l_i, h_i\rangle} \cap \overline{I} = \emptyset$, leave $\langle l_i, h_1\rangle$ as is. Otherwise $\overline{\langle l_i, h_i\rangle} \setminus \overline{I}$ can be represented as either one interval, or two intervals in an obvious way; replace $\langle l_i, h_i\rangle$ with these intervals.
- $ID_1 \setminus ID_2 = ((ID_1 \setminus \langle u_1, v_1\rangle) \setminus \ldots) \setminus \langle u_m, v_m\rangle$.

References

1. Steinder, M., Sethi, A.S.: A survey of fault localization techniques in computer networks. Science of Computer Programming. 54, N 2, 165–194 (2004)
2. Ali, S.T., Sivaraman, V., Radford, A., Jha, S.: A survey of securing networks using software defined networking. IEEE Transactions on Reliability. 64, N 3, 1086–1097 (2015)
3. Sosnovich, A., Grumberg, O., Nakiby, G.: Finding security vulnerabilities in a network protocol using parameterized systems. Computer Aided Verification. CAV 2013. Lecture Notes in Computer Science. 8044, 724–739 (2013)
4. Zengin, A., Ozturk, M.M.: Formal verification and validation with DEVS-suite: OSPF case study. Simulation Modelling Practive and Theory. 29, 193–206 (2012)
5. Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal verification of standards for distance vector routing protocols. Journal of the ACM. 49, N 4, 538–576 (2002)
6. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (2001)
7. Baier, C., Katoen, J.-P.: Principles of model checking. MIT Press (2008)
8. Altukhov, V.S., Chemeritsky, E.V., Podymov, V.V., Zakharov, V.V.: VERMONT - A toolset for checking SDN packet forwarding policies on-line. SDN&NFV: The Next Generation of Computational Infrastructure: 2014 International Scientific and Technology Conference "Modern Networking Technologies (MoNeTec)". 7–12 (2014)

9. Majumdar, R., Tetali, S.D., Wang, Z.: Kuai: a model checker for software-defined networks. Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design. 163–170 (2014)
10. Barringer, H., Goldberg, A., Havelund, K., Sen, K.: Rule-based runtime verification. Proceedings of 5th International Conference on Verification, Model Checking, and Abstract Interpretation. Lecture Notes in Computer Science. 2937, 44–57 (2004)
11. Barringer, H., Rydeheard, D.E., Havelund, K.: Rule Systems for Run-time Monitoring: from Eagle to RuleR. Journal of Logic and Computation. 20, N 3, 675–706 (2010)
12. Bloem, R., Jacobs, S., Khalimov, A., Konnov, I., Rubin, S., Veith, H., Widder, J.: Decidability of parameterized verification. Synthesis Lectures on Distributed Computing Theory. 6, N 1, 1–170 (2015)
13. Maler, O., Nickovic, D., Pnueli, A.: Real time temporal logic: past, present, future. Proceedings of International Conference on Formal Modeling and Analysis of Timed Systems. Lecture Notes in Computer Science. 3829, 2–16 (2005)
14. Ouaknine, J., Worrell, J.: Some recent results in metric temporal logic. Proceedings of International Conference on Formal Modeling and Analysis of Timed Systems. Lecture Notes in Computer Science. 5215, 1–13 (2008)
15. Bouyer, P., Chevalier, F., Markey, N.: On the expressiveness of TPTL and MTL. Information and Computation. 208, N 1, 97–116 (2010)