ISSN 0146-4116, Automatic Control and Computer Sciences, 2021, Vol. 55, No. 7, pp. 763–775. © Allerton Press, Inc., 2021.
Russian Text © The Author(s), 2020, published in Modelirovanie i Analiz Informatsionnykh Sistem, 2020, No. 4, pp. 412–427.

A Temporal Logic for Programmable Logic Controllers

N. O. Garanina[a, b, *], I. S. Anureev[a, b], V. E. Zyubin[b], S. M. Staroletov[b, c], T. V. Liakh[b], A. S. Rozov[b], and S. P. Gorlatch[d]

[a] *A. P. Ershov Institute of Informatics Systems (IIS), SB RAS, 6, Acad. Lavrentjev ave., Novosibirsk, 630090 Russia*
[b] *Institute of Automation and Electrometry SB RAS, 1 Academician Koptyug ave., Novosibirsk, 630090 Russia*
[c] *Polzunov Altai State Technical University, 46 Lenina ave., Barnaul, 656038 Russia*
[d] *University of Muenster, 62 Einsteinstr., Muenster, 48149 Germany*
**e-mail: garanina@iis.nsk.su*

Received November 12, 2020; revised December 2, 2020; accepted December 16, 2020

Abstract—We investigate the formal verification of the control software of critical systems, i.e., the verification of the compliance of the designed system with the requirements. The most important class of control software consists of programs for programmable logic controllers (PLCs). A special feature of PLC programs is the scan cycle: 1) the inputs are read, 2) the PLC states are changed, and 3) the outputs are written. Therefore, for formal verification of PLC programs, for example by model checking, it is necessary to be able to describe transition systems that take into account this specificity. In addition, it is required to determine properties of systems that model PLC programs, both with respect to transitions within the cycle as well as larger transitions in accordance with the semantics of the scan cycle. In this paper, we introduce a formal model of a PLC program as a system of hyperprocess transitions and the temporal *cycle-LTL* logic based on the LTL logic for formalizing the properties of the PLC. A special feature of the cycle-LTL logic is the ability to consider the properties of control systems in two ways: as an impact of the environment on the control system and as an impact of the control system on the environment. We define modifications of the standard temporal operators of the LTL logic for each of these cases, as well as for properties inside the scan cycle. Examples of requirements defined in our logic are considered. The translation of cycle-LTL formulas into LTL formulas is described and its correctness is proved. Thereby we demonstrate the possibility of reducing the problem of verification by model checking for the requirements defined in the cycle-LTL logic to the model checking problem for the requirements defined in the standard LTL logic.

Keywords: formal verification, temporal logics, transition systems, programmable logic controllers (PLC)

DOI: 10.3103/S0146411621070038

INTRODUCTION

In the last decade, in connection with solving problems that impose increased security requirements on control systems based on programmable logic controllers (PLCs), researchers have already developed solutions for formal modeling of programs for such controllers and proving their properties, in particular, expressed in the form of formulas of temporal logics.

The long-term goal of our work is the formal verification of programs for automatic control systems written within the *process-oriented* paradigm, and, in particular, programs written in the process-oriented Reflex language [1, 2]. Formal verification of programs for PLCs, which are important components of automatic control systems, is an urgent topic of theoretical and practical work [3, 4]. The proposed approaches follow various formal PLC models [3–7]. In general, the operation of a PLC consists of an infinite sequence of *control cycles*. Each control cycle includes a sequence of three phases: (1) reading the inputs, (2) execution, and (3) writing the outputs.

We use process-oriented modeling of PLC programs based on *hyperprocesses* [2]. This modeling method allows us to naturally determine the main features of PLC programs, such as the control cycle and timers; it describes a PLC program as a synchronized system of interacting processes determined by a set of functional states and actions in these states. According to the classification in [5], a hyperprocess simulates a program for a PLC that abstracts from the execution time of the control cycle[1] and uses input and output control timers. Such modeling provides a natural specification for multiprocessor control systems, which, by abstracting from the time of the control cycle, allows one to strictly order the sequence of execution of processes in the control cycle. Such a strong synchronization of processes without alternation makes it possible to effectively use formal verification methods. Hyperprocess is the basis for the subject-oriented Reflex language, which has shown its productivity in several industrial projects, in particular, in the equipment for growing single crystals of silicon by the Czochralskimethod [8] and the vacuum system for a large solar vacuum telescope [9].

In this paper, we use model checking as our formal verification method. Therefore, a hyperprocess is presented as a transition system of a special kind, and the properties of the hyperprocess are expressed as formulas of a temporal logic. The hyperprocess transition system is close to a parallel multithreaded system [10], enriched with a synchronizing counter, functional states of processes, timers, and action primitives for changing them. The time for a PLC program defined as a hyperprocess, is clocked not only within the execution phase of the control cycle (with each change in the values characterizing the states of the processes), but also on a larger scale – for a sequence of control cycles, i.e., when reading inputs/writing outputs.

A logic for specifying requirements for the PLC program should enable formulating statements regarding the above two types of clocking. In this paper, we introduce the *cycle-LTL* logic, which enriches the LTL temporal logic [11] with cyclic temporal operators for analyzing the states of a PLC program at the beginning and at the end of the control cycle, as well as with intracycle temporal operators for defining the properties of the PLC program. In this logic, for some abstract control algorithm, it is possible to express the following properties: *"If the sensor is triggered, the device will turn on in the next control cycle"* or *"If the temperature is above the critical value, the cooling process is always on."* Note that if the cooling process is switched on and off after the actions of other processes in the execution phase of the control cycle, the latter property may be violated inside this phase, but may be preserved outside it. This example illustrates the need to use cyclic temporal operators, in particular, the operator "always" $\mathbf{G}^{\mathbf{i}}$.

Let us provide an overview of the formalisms currently used in this area of research. Initially, an extension of the time automaton, PLC-automaton [12], was developed to describe the behavior of programs for PLCs in the automata form, while further research mainly focused on the use of logical systems, rather than automata.

In [13], methods for verifying PLC programs are evaluated in both text and graphical form using various model verification tools, and specific types of real programs are provided. The current problems in this area are also described—in particular, the problem of combinatorial explosion, the modeling of timers, the definition of properties for checking, and the problem of modeling the execution cycle of such programs, which is the subject of this work.

There are known methods for representing PLC programs in the form of LTL formulas. According to the approach in [14], the value of each variable should be changed only once in one place of the program during one complete execution of the PLC working cycle. Therefore, a change in the value of each program variable is represented by two explicit and one implicit LTL formulas. Thus, it is proposed to reduce PLC programming to the construction of an LTL specification for each program variable. In this case, the Turing completeness of languages of the IEC 61131-3 standard is proven using Minsky machines.

In [15], methods of automatic mining of invariants are presented using a causal graph of event relationships for a running PLC program in order to obtain TPTL [16] formulas for safety properties. It is noted that, at run time, it is difficult to enforce a rule based on the LTL formula that requires one action to be followed by another, since the absence of a required event during the limited testing time does not imply its absence at a later time.

In [17], a method is proposed for automatically obtaining specifications for given patterns of LTL formulas by running execution cycles of IEC 61131-3 programs. In [18], the PLC code was translated into the input language of the SMV verifier to check models according to the identified invariants.

The development of domain-specific extensions of temporal logics to specify requirements for industrial controllers has begun recently. An overview on this topic is given in [19], where, in particular, the ST-LTL logic is proposed [20]. The main improvements over LTL are the use of the previous variable values instead of the next state operator, as well as the introduction of the *WhileNScanCycles* operator to work with the values of control variables at the Nth step, which significantly differs from our approach and leads to more complex specifications and proofs. In [21], the real-time temporal logic MITL[a,b] is introduced, where all temporal modalities are limited to the time interval [a,b], and then the STL logic (signal temporal logic) is introduced based on filtering signals and transition from real numbers to natural numbers.

[1] The environment is considered slow enough to count the time of the I/O phases and the execution of the control cycle as zero.

Thus, our approach is in line with the current trends in the development of approaches to formal specification and verification of programs, which focus on expanding existing logics in the context of control cycle modeling. The first version of our approach is presented in [22].

The rest of the article is organized as follows. In Section 1, we define a hyperprocess transition system. Section 2 describes the syntax and semantics of the cycle-LTL temporal logic. In Section 3, we show that every formula of cycle-LTL can be expressed in the temporal logic LTL, and that the model checking problem for cycle-LTL and the hyperprocess transition system is decidable. Section 4 contains a conclusion and a plan for future work.

HYPERPROCESS TRANSITION SYSTEM

Let us provide an informal description of a hyperprocess [2]. *A hyperprocess* is an ordered set of interacting processes that execute sequentially in a given order, forming *a control cycle*. This cycle begins by reading input data from the environment into the system input variables of the hyperprocess and ends with writing the output data (control impacts) to the corresponding output variables. The values of the output variables are obtained during the execution phase. We assume that changes in the data from the environment occur slowly enough for all processes to have enough time to work in the execution phase. Thus, one control cycle can be considered as a logical unit of time for a hyperprocess. All hyperprocess variables are global. A special feature of the processes that form a hyperprocess are their *functional states* – labels denoting a certain sequence of process actions. Among the functional states, the states of normal stop and erroneous stop, in which the process does not perform any actions, are distinguished. The actions of the processes can change the values of the hyperprocess variables (except for the input variables), have a limited effect on the functional states of other processes, and set and reset the timer values. Actions may have guard conditions that depend on the variables of the hyperprocess and the functional states of other processes. Our definition of a hyperprocess transition system is based on the description of hyperprocesses and the operational semantics of the Reflex language [1, 2].

Definition 1. (Hyperprocess Transition System, HTS)

The hyperprocess transition system is the five-tuple $H = (P, S, s_{ini}, A, R)$, where

- $P = \{p_1, \dots, p_n\}$ is the ordered set of processes;
- S is the nonempty set of states;
- s_{ini} is the initial state;
- A is the action alphabet;
- R is the labeled transition relation $R : A \to 2^{S \times S}$.

Before defining the HTS components, let us describe the elements of a hyperprocess in general.

Definition 2. (Elements of a hyperprocess)

The elements of a hyperprocess are variables, functional states, process actions, and timers:

• **Variables.** $V = \{v_1, \dots, v_N\}$ is the set of *hyperprocess variables*. The values of the variables in a particular state of a hyperprocess are defined by the corresponding functions $v_i : S \to D \cup \{\perp\}$. We distinguish between *input variables* V_I, *output variables* V_O, *and internal variables* V_L: $V = V_I \cup V_O \cup V_L$.

• **Functional states.** For each $i \in [1..n]$, $F_i = \{f_i^1, \dots, f_i^{m_i}, stop, err\}$ is the set of *functional states of the process* p_i. *Inactive* states *stop* and *err* are distinguished. The remaining states are *active*. The value of the functional state variable f_i for each process p_i in the state of the hyperprocess is described by the function $f_i : S \to F_i$.

• **Actions.** In an active functional state f_i^j, the process p_i performs actions from the set A. These actions form *the body of the functional state*. $L_i^j \in \mathbb{N}$ is the number of actions in the body f_i^j. For each process, a_i is *the action counter*, and its value is *the position*. The value of the action counter in the hyperprocess state is the result of the function $a_i : S \to [1..L_i^j] \cup \{\perp\}$. The next action of the process in the functional state is determined by the functions $nxt^j : \mathbb{N} \times S \to \mathbb{N} \cup \{\perp\}$. These functions implicitly set guard conditions for actions, since they depend on the current state of the hyperprocess, in particular, on the state of activity of other processes. If there is no next action in the current functional state, i.e., the end of its body has been reached, then $nxt^j(a_i) = \perp$. The functions $an^j : \mathbb{N} \cup \{\perp\} \to A$ return the name of the action at the position a_i.

• **Timers.** *The timer* of the process p_i is a variable t_i whose values in the hyperprocess state are the result of the function $t_i : S \to \mathbb{F} \cup \{\perp\}$. *The timeout value* at the position a of the functional state f_i^j is $N_i^{j_a} \in \mathbb{N}$. For the sake of simplicity, here we only consider processes that have no more than one timer start per functional state.

Now we proceed to define the components of the hyperprocess transition system $H = (P, S, s_{ini}, A, R)$.

Set of states S

The state $s = (v, sp, pc) \in S$ includes the following elements:

• evaluation of variables $v = (v_1(s), \ldots, v_N(s))$: vector of values of variables in the state s;

• state of processes $sp = ((f_1(s), a_1(s), t_1(s)), \ldots, (f_n(s), a_n(s), t_n(s)))$, where for each process p_i in the state s, $f_i(s)$ is its current functional state, $a_i(s)$ is the counter of actions in the state $f_i(s)$, and $t_i(s)$ is the value of its timer;

• counter of processes $pc(s)$ with values in $[0..n+1]$, where 0 is reserved for reading input data, and $n+1$ for writing output data.

Initial state s_{ini}

$s_{ini} = (v_0, sp_0, pc_0)$, where

• $v_0 = (\perp_1, \ldots, \perp_N)$,

• $sp_0 = ((f_1^1, 1, \perp)_1, (stop, \perp, \perp)_2, \ldots, (stop, \perp, \perp)_n)$, and

• $pc_0 = 0$.

Action alphabet A

The alphabet includes input and output data update actions, as well as process actions:

$A = \{inp, out, skip, end, upd, tout, reset, startP, stopP, start, set, next, stop, err\}$.

0. Cycle actions.

inp – changing the values of input variables (reading inputs from the environment).

out – changing the values of output variables (writing outputs to the environment).

1. Service actions.

skip – the process does nothing in inactive states.

end – upon termination of a process in an active state, the progress is transferred to the next process.

2. The action of updating the values of internal variables.

upd – the process changes the values of some internal variables.

3. Actions with the timer.

tout – the process starts the timer and performs actions in the current functional state until the timeout occurs;

reset – the process resets its timer to zero.

4. Actions with functional states.

$startP/stopP$ – the process translates the target process p_k into the functional states $f_k^1/stop$;

$start/set$ – the process goes to the target functional state f_i^1/f;

$stop/err$ – the process goes to the functional state $stop/err$;

next – the process goes to the next functional state.

Relation of labeled transitions R

The transition relation R defines the semantics of process actions and input updates. Let

$i \in [1..n]$, $j \in [1..m_i]$, $a \in [1..L_i^j]$. We will use the following notation for process actions.

The expression

$$(f_i = f_i^j, a_i = a, T_i, Tg_i, pc = i) \xrightarrow{act} (y_1' = new_1, \ldots, y_{m'}' = new_{m'})$$

means that for all states s that satisfy the precondition (the expression to the left of the arrow) and all states s' that satisfy the postcondition (the expression to the right of the arrow) $R(act) = (s, s')$ is true, and along with this

- $act = an^j(a)$, and $an^j(\bot) \in \{skip, end\}$;
- the starting state s is such that $pc(s) = i$, $f_i(s) = f_i^j$, $a_i(s) = a$, the time bound T_i can be $t_i(s) \neq \bot$,

$t_i(s) = \bot$, $t_i(s) < N_i^j$, or $t_i(s) = N_i^j$, and the target limitation Tg_i can be $tgp_i = k$ or $tgs_i = k$, where tgp_i is a variable from V_L with values in $[1..n]$ to set the target process, and tgs_i is a variable in V_L with values in $[1..m_i]$ to set the target functional state of the process p_i; elements not mentioned on the left have arbitrary values;

the resulting state s' specifies changes in the elements of the hyperprocess after the action act: $y_k(s') = new_k$ $(k \in [1..m'])$; elements not mentioned on the right do not change.

0. Actions for updating input and output variables.

We use a similar notation to define $R(inp)$ and $R(out)$.

At the beginning of the control cycle, the input data values are read into the input variables from V_I, the value of each activated process timer is increased by 1, and the process counter is shifted to the first process:

— $(t_{i_1} \neq \bot, \ldots, t_{i_k} \neq \bot, pc = 0) \xrightarrow{inp} (u_1' = u_1, \ldots, u_K' = u_K, t_{i_1}' = t_{i_1} + 1, \ldots, t_{i_k}' = t_{i_k} + 1, pc' = 1)$.

The action of reading input data is the only action in the system that is, generally speaking, nondeterministic.

At the end of the control cycle, the values of the output data are written to the output variables from V_O, and the process counter is shifted to the beginning of the control cycle:

— $(pc = n + 1) \xrightarrow{out} (w_1' = w_1, \ldots, w_M' = w_M, pc' = 0)$

This action is deterministic.

In defining the transition relation R for process actions, we consider the process p_i that performs an action number a_i named $an^j(a_i)$ in an active functional state $f_i = f_i^j$, or in an inactive state.

1. Service actions.

The process p_i does nothing in the inactive states $stop$ and err, and the move is passed to the next process:

— $(f_i = stop, pc = i) \xrightarrow{skip} (pc' = i + 1)$.

— $(f_i = err, pc = i) \xrightarrow{skip} (pc' = i + 1)$.

When the process p_i reaches the end of the body of the active functional state f_i^j, it goes to the first action of this state, which it will perform in the next control cycle (unless another process transfers it to a different state), and the move is passed to the next process:

— $(f_i = f_i^j, a_i = \bot, pc = i) \xrightarrow{end} (a_i' = 1, pc' = i + 1)$.

2. The action of updating the values of process variables.

With this action, the process p_i changes the values of some internal variables $\{v_1, \ldots, v_m\} \subseteq V_L$, and passes to the next action $nxt^j(a)$ of the current functional state:

— $(f_i = f_i^j, pc = i) \xrightarrow{upd} (v_1' = d_1, \ldots, v_m' = d_m, a_i' = nxt^j(a))$;

This action is deterministic.

3. Actions with the timer.

In the following cases, the process p_i starts the timer t_i (if $t_i = \bot$), proceeds to the first action of the current functional state f_i^j, and the move is passed to the next process:

— $(f_i = f_i^j, t_i = \bot, pc = i) \xrightarrow{tout} (a_i' = 1, t_i' = 0, pc' = i + 1)$;

— $(f_i = f_i^j, t_i < N_i^j, pc = i) \xrightarrow{tout} (a_i' = 1, pc' = i + 1)$.

If the timeout event occurs, the process p_i stops the timer t_i and passes to the next actions of the current functional state:

— $(f_i = f_i^j, t_i = N_i^j, pc = i) \xrightarrow{tout} (a_i' = nxt^j(a), t_i' = \perp)$.

The result of the action *reset* is similar to the result of the first case of the action *tout*:

— $(f_i = f_i^j, t_i \neq \perp, pc = i) \xrightarrow{reset} (a_i' = 1, t_i' = 0, pc' = i + 1))$.

4. Actions with functional states.

The following two actions of the process p_i transfer the process p_k $(i \neq k))$ to the state *stop*, *err* or its first functional state. Note that the hyperprocess model satisfies the principle of limited encapsulation: a process cannot transfer another process into an arbitrary functional state.

— $(f_i = f_i^j, tgp_i = k, pc = i) \xrightarrow{startP} (f_k' = f_k^1, a_k' = 1, t_k' = \perp, a_i' = nxt^j(a))$;

— $(f_i = f_i^j, tgp_i = k, pc = i) \xrightarrow{stopP} (f_k' = stop, a_k' = \perp, t_k' = \perp, a_i' = nxt^j(a))$;

— $(f_i = f_i^j, tgp_i = k, pc = i) \xrightarrow{errP} (f_k' = err, a_k' = \perp, t_k' = \perp, a_k' = nxt^j(a))$;

With the following actions, the process p_i sets that in the next control cycle it will start from the first action of the corresponding functional state, and stops the timer:

— $(f_i = f_i^j, tgs_i = k, pc = i) \xrightarrow{set} (f_i' = f_i^k, a_i' = 1, t_i' = \perp)$;

— $(f_i = f_i^j, pc = i) \xrightarrow{start} (f_i' = f^1, a_i' = 1, t_i' = \perp)$;

— $(f_i = f_i^j, pc = i) \xrightarrow{next} (f_i' = f_i^{j+1}, a_i' = 1, t_i' = \perp)$.

The process p_i goes into a state of normal or erroneous shutdown:

— $(f_i = f_i^j, pc = i) \xrightarrow{stop} (f_i' = stop, a_i' = \perp, t_i' = \perp)$;

— $(f_i = f_i^j, pc = i) \xrightarrow{err} (f_i' = err, a_i' = \perp, t_i' = \perp)$.

According to the definition of the transition relation, processes act sequentially in the current control cycle in the order specified by the process counter. Note that the transition relation is nondeterministic only for the action of reading the input data *inp*, which is not an action of any process. Let *the output states* s_{out} be the states in which $pc(s_{out}) = 0$. In these states, the output variables received new values. Thus, the output states reflect the dependence of the output data (reactions of the control system) on the input data (on the control object). Let's define *the input states* s_{inp} as states immediately after reading the inputs to the input variables and before the processes started to act: $R(inp) = (s_{out}, s_{inp})$. Thus, the input states reflect the possible dependences of the input data (reactions of the control object) on the output data of the control system at the previous control cycle. Due to the determinism of the actions of processes, the nearest output state is uniquely determined for a given input state; however, different inputs can still lead to the same output: for example, a certain range of input values can be processed in the same way. Let S_i be a set of input states, and S_o be a set of output states.

Let us define three kinds of paths in HTS. *The standard path* $\pi = s_0, s_1, \ldots$ is a sequence of states $s_j \in S$ such that $\forall j \geq 0 \exists a \in A : R(a) = (s_j, s_{j+1})$. Let $\pi(j)$ be the jth state on the path π. *An input path* $\sigma = i_0, i_1, \ldots$ is an infinite sequence of input states $i_j \in S_i$ such that for each $j \geq 0$ there is a finite standard path π_j of length n_j with $\pi_j(0) = i_j$ and $\pi_j(n_j) = i_{j+1}$. *An -output path* $\rho = o_0, o_1, \ldots$ is an infinite sequence of - output states $o_j \in S_o$ such that for each $j \geq 0$ there is a finite standard path π_j of length n_j with $\pi_j(0) = o_j$ and $\pi_j(n_j) = o_{j+1}$. An arbitrary path $\chi = x_0, x_1, \ldots$ is a *subpath* of the path $\chi' = x_0', x_1', \ldots$ if $x_0, x_1 \ldots$ is a subsequence of $x_0', x_1', \ldots$. Accordingly, χ' is *a superpath* of χ. If the path χ goes inside H, then we write $\chi \in H$.

Further in the paper, χ is a standard, input or output path, π is a standard path, ρ is an input path, σ is an output path. Let us give a definition of additional paths that are used when defining the semantics of the temporal operators of cycle-LTL.

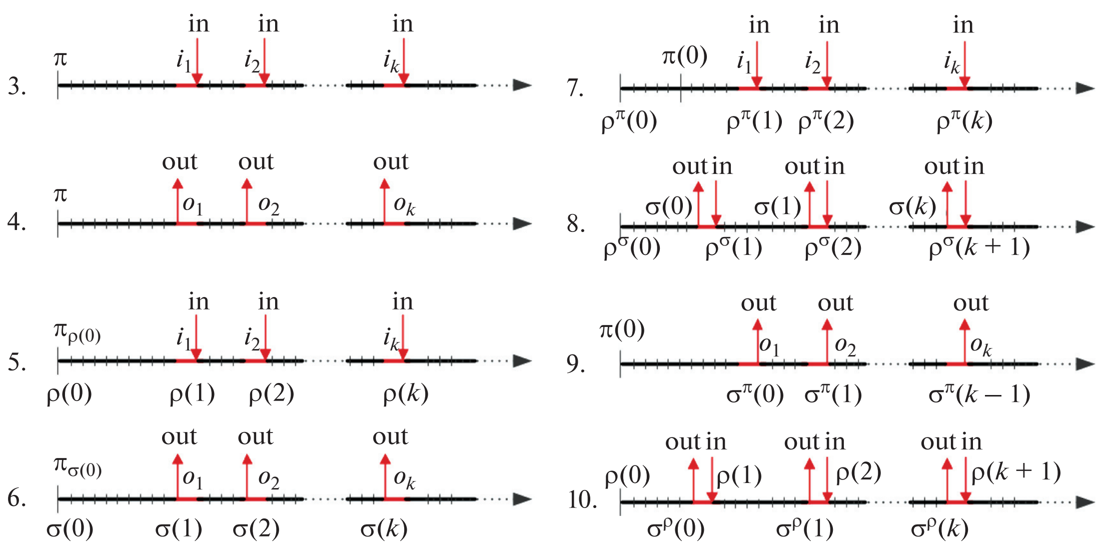

Fig. 1. Paths and their elements.

Definition 3. (Paths and their elements)

1. $\chi(k)$ is the k-th state on the path χ;

2. χ^k is the suffix path of χ such that $\chi^k(0) = \chi(k)$;

3. i_k^π is the number of the k-th input state on the path π;

4. o_k^π is the number of the k-th output state on the path π;

5. π_ρ is the path containing a given path ρ so that their beginnings coincide; i.e., ρ is a subpath of π and $\pi_\rho(0) = \rho(0)$ (this path is the only one, since the processes process input data deterministically);

6. π_σ are the paths containing a given path σ so that their origins coincide; i.e., σ is a subpath of π and $\pi_\sigma(0) = \sigma(0)$;

7. ρ^π are the input paths that start earlier than the given path π, but are contained in it, starting from their second state; i.e., $\pi = \pi^k_{\rho^\pi}$ where k is such that $\rho^\pi(1) = \pi(i_1^\pi)$;

8. ρ^σ are the input paths, the superpaths of which contains the given path σ from the nearest output state; i.e., σ is a subpath of π_{ρ^σ} and $\sigma(0) = \pi_{\rho^\sigma}(o_1^{\pi_{\rho^\sigma}})$;

9. σ^π is the output path that starts later than the specified path π, but is contained in it, and its beginning is the first output state of π; i.e., $\pi_{\sigma^\pi} = \pi^k$, where k is such that $\sigma^\pi(0) = \pi(o_1^\pi)$ (this path is the only one, since all its states lie on the given path π);

10. σ^ρ is the output path, the superpath of which contains the given path ρ from the nearest preceding input state; i.e., σ is a subpath of π_ρ and $\sigma(0) = \pi_\rho(o_1^{\pi_\rho})$ (this path is the only one, since processes generate deterministic output for given input data).

Figure 1 illustrates the concepts of definition 3.

CYCLE-LTL: A TEMPORAL LOGIC FOR PLC

The syntax of the cycle-LTL logic contains atomic statements P, Boolean connectives, standard temporal operators of LTL, input, internal and output cycle temporal operators: 1em

$$\varphi ::= P \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi\mathbf{U}\varphi \mid$$
$$\mathbf{X}^\mathrm{i}\varphi \mid \mathbf{F}^\mathrm{i}\varphi \mid \mathbf{G}^\mathrm{i}\varphi \mid \varphi\mathbf{U}^\mathrm{i}\varphi \mid \mathbf{X}^\mathrm{e}\varphi \mid \mathbf{F}^\mathrm{e}\varphi \mid \mathbf{G}^\mathrm{e}\varphi \mid \varphi\mathbf{U}^\mathrm{e}\varphi \mid \mathbf{X}^\mathrm{o}\varphi \mid \mathbf{F}^\mathrm{o}\varphi \mid \mathbf{G}^\mathrm{o}\varphi \mid \varphi\mathbf{U}^\mathrm{o}\varphi.$$

Internal cycle temporal operators $\mathbf{X^e}$, $\mathbf{F^e}$, $\mathbf{G^e}$, and $\mathbf{U^e}$ are used to formulate properties that must hold during the execution phase of control cycles, while using *the input and output cycle operators* $\mathbf{X^i}$, $\mathbf{F^i}$, $\mathbf{G^i}$, $\mathbf{U^i}$, $\mathbf{X^o}$, $\mathbf{F^o}$, $\mathbf{G^o}$, and $\mathbf{U^o}$, you can describe the properties of control systems that hold at the beginning and at the end of control cycles.

Let Φ^s be the set of formulas starting with standard LTL operators, Φ^e be the set of formulas starting with the internal operators, Φ^i be the set of formulas starting with the input operators, Φ^o be the set of formulas starting with the output operators, and Φ^c be the set of all cycle-LTL formulas.

We traditionally define **the semantics** of the cycle-LTL logic in terms of the relation of satisfiability $\vDash$ between a formula and a path in HTS: $H,\chi \vDash \xi$, where H is the hyperprocess transition system, χ is an infinite standard, input or output path, and $\xi \in \Phi^c$. We consider all kinds of paths for formulas with cyclic temporal operators, and input/output paths for standard LTL formulas. The semantics of standard LTL formulas on standard paths can be found in [11]. Let H be a hyperprocess transition system, π be an infinite standard path, ρ be an input path, σ be an output path, and $\varphi, \psi \in \Phi^c$ be the cycle-LTL formulas. 1em

Semantics of formulas Φ^s.

Let $\xi \in \Phi^s$.

- $H,\rho \vDash \xi \Leftrightarrow H,\pi_\rho \vDash \xi$;
- $H,\sigma \vDash \xi \Leftrightarrow$ for all π_σ, $H,\pi_\sigma \vDash \xi$ is true;

Semantics of formulas Φ^e.

Let $\xi \in \Phi^e$.

- $H,\rho \vDash \xi \Leftrightarrow H,\pi_\rho \vDash \xi$;
- $H,\sigma \vDash \xi \Leftrightarrow$ for all π_σ, $H,\pi_\sigma \vDash \xi$ is true;
- $H,\pi \vDash \mathbf{X^e}\varphi \Leftrightarrow \pi(1) \notin S_i \cup S_o$ and $H,\pi^1 \vDash \varphi$;
- $H,\pi \vDash \mathbf{F^e}\varphi \Leftrightarrow$ there is $0 \le k < o_1^\pi$ such that $\pi(k) \notin S_i \cup S_o$ and $H,\pi^k \vDash \varphi$;
- $H,\pi \vDash \mathbf{G^e}\varphi \Leftrightarrow$ for all $0 \le k < o_1^\pi$ $\pi(k) \notin S_i \cup S_o$ and $H,\pi^k \vDash \varphi$;
- $H,\pi \vDash \varphi\mathbf{U^e}\psi \Leftrightarrow$ there is $0 \le k < o_1^\pi$ such that $\pi(k) \notin S_i \cup S_o$ and $H,\pi^k \vDash \psi$ and for all $0 \le j < k$ $\pi(j) \notin S_i \cup S_o$ and $H,\pi^j \vDash \varphi$.

Semantics of formulas Φ^i.

Let $\xi \in \Phi^i$.

- $H,\pi \vDash \xi \Leftrightarrow$ for all ρ^π, $H,\rho^\pi \vDash \xi$ is true;
- $H,\sigma \vDash \xi \Leftrightarrow$ for all ρ^σ, $H,\rho^\sigma \vDash \xi$ is true;
- $H,\rho \vDash \mathbf{X^i}\varphi \Leftrightarrow H,\rho^1 \vDash \varphi$;
- $H,\rho \vDash \mathbf{F^i}\varphi \Leftrightarrow$ there is $k \ge 0$ such that $H,\rho^k \vDash \varphi$;
- $H,\rho \vDash \mathbf{G^i}\varphi \Leftrightarrow$ for all $k \ge 0$ $H,\rho^k \vDash \varphi$;
- $H,\rho \vDash \varphi\mathbf{U^i}\psi \Leftrightarrow$ there is $k \ge 0$ such that $H,\rho^k \vDash \psi$ and for all $0 \le j < k$ $H,\rho^j \vDash \varphi$.

Semantics of formulas Φ^o.

Let $\xi \in \Phi^o$.

- $H,\pi \vDash \xi \Leftrightarrow H,\sigma^\pi \vDash \xi$;
- $H,\rho \vDash \xi \Leftrightarrow H,\sigma^\rho \vDash \xi$;
- $H,\sigma \vDash \mathbf{X^o}\varphi \Leftrightarrow H,\sigma^1 \vDash \varphi$;

- $H, \sigma \models \mathbf{F}^{\mathbf{o}}\varphi \Leftrightarrow$ there is $k \geq 0$ such that $H, \sigma^k \models \varphi$;
- $H, \sigma \models \mathbf{G}^{\mathbf{o}}\varphi \Leftrightarrow$ for all $k \geq 0$ $H, \sigma^k \models \varphi$;
- $H, \sigma \models \varphi\mathbf{U}^{\mathbf{o}}\psi \Leftrightarrow$ there is $k \geq 0$ such that $H, \sigma^k \models \psi$ and for all $0 \leq j < k$ $H, \sigma^j \models \varphi$.

Let us illustrate the specifications in the cycle-LTL language for control systems using the example of a fan heater and a temperature maintenance device:

- *If hands appear under the dryer, it will be turned on in the next control cycle*:

$\mathbf{G}^{\mathbf{i}}(hands = on \rightarrow \mathbf{X}^{\mathbf{i}}dryer = on)$.

- *If the temperature is higher than the maximum value, then the cooling process is always on*:

$\mathbf{G}^{\mathbf{i}}(temp > 95° \rightarrow cooler = on)$.

- *The heating element is ~~switched~~ on if the temperature has dropped below the minimum value*:

$\mathbf{G}^{\mathbf{o}}(temp < 5° \rightarrow heater = on)$.

- *If the heating element is turned on, then over time the temperature will rise above the minimum value*:

$\mathbf{G}^{\mathbf{o}}(heater = on \rightarrow \mathbf{F}^{\mathbf{i}}temp \geq 5°)$.

Note the variability of the cycle-LTL syntax when describing the properties of a hyperprocess. In the formulas of examples 1 and 3, you can use both the operator $\mathbf{G}^{\mathbf{i}}$ (with $\mathbf{X}^{\mathbf{i}}$), and $\mathbf{G}^{\mathbf{o}}$: in both cases, the properties of the dependence of the values of the nearest output on the input data are set. The choice of one or another representation of a system property depends on the user's preferences.

The above examples of system properties use only cyclic states and observable input—output variables of the system. Formulas of this kind can be used to define high-level properties of control systems and algorithms, independent of the implementation, when the control system is viewed as a black box. However, if a high-level property has not been satisfied for some implementation of the control system, then for further analysis it may be necessary to check the low-level properties of the system, formulated in terms of the states and actions of the processes that implement it. These properties of the system are defined in the form of hypotheses using intracycle logic operators, which allows to localize the error within the execution phase of the control cycle. In some cases, checking such hypotheses is less labor-intensive than analyzing a counterexample for a high-level property. Consider a system that combines a lighting control system and a burglar alarm. The following two properties of this system illustrate the causal relationship between low-level hypotheses and high-level properties: non-satisfiability of a low-level formula 2 entails non-satisfiability of a high-level formula 1.

- *When a break-in is detected, all the lights flash in emergency mode*:

$\mathbf{G}^{\mathbf{i}}(alarm \rightarrow \mathbf{X}^{\mathbf{i}}alarm\_light = on)$.

- *If the break-in sensor is triggered, the process of external interaction of the security alarm system sends a signal message to all connected systems, including the lighting control system, during the next control cycle*:

$\mathbf{G}^{\mathbf{i}}(alarm \rightarrow \mathbf{F}^{\mathbf{e}}alarm\_message\_sent)$.

The choice of hypothesis 2 is due to the assumption that in the case of sending a signal message, it will be received in time and, after receiving it, the emergency operation of the lamps will immediately work.

CONVERTING THE LOGIC FORMULAS OF CYCLE-LTL TO LTL

We consider formulas $\xi, \xi' \in \Phi^c$ of the cycle-LTL logic to be equivalent ($\xi \equiv \xi'$) if and only if $H, \chi \models \xi \Leftrightarrow H, \chi \models \xi'$ for an arbitrary HTS H and an arbitrary path χ. In this section, we show that for formulas with cyclic operators $\Phi^e \cup \Phi^i \cup \Phi^o$, there are standard LTL formulas equivalent to them with additional atomic statements.

Let us add two auxiliary Boolean variables *Input* and *Output* to the description H of the HTS states. Let the variable *Input* be true only in the input states from S_i, and the variable *Output* be true only in the output states from S_o. Then for the corresponding atomic statements it is true that $H, \pi \models Input$ for all paths π such that $\pi(0) \in S_i$, and $H, \pi \not\models Input$ for all paths π such that $\pi(0) \notin S_i$, as well as $H, \pi \models Output$ for all paths π such that $\pi(0) \in S_o$, and $H, \pi \not\models Output$ for all paths π such that $\pi(0) \notin S_o$. Let H be a hyperprocess transition system, π be an infinite standard path, ρ be an input path, σ be an output path, and $Exe = \neg(Input \vee Output)$. Since the proofs of the propositions below use induction on the structure of the formula, we can assume that below the subformulas of formulas with cycle operators φ and ψ are LTL formulas.

Proposition 1. Cycle-LTL formulas with initial inner cycle operators are equivalent to the following LTL formulas:

- $\mathbf{X}^{\mathbf{e}}\varphi \equiv \mathbf{X}(\varphi \wedge Exe)$;
- $\mathbf{F}^{\mathbf{e}}\varphi \equiv (Exe)\mathbf{U}(\varphi \wedge Exe)$;
- $\mathbf{G}^{\mathbf{e}}\varphi \equiv (\varphi \wedge Exe)\mathbf{U}(\neg Exe)$;
- $\varphi\mathbf{U}^{\mathbf{e}}\psi \equiv (\varphi \wedge Exe)\mathbf{U}(\psi \wedge Exe)$.

Proof.

The proof uses induction on the structure of the formula. Let us first consider the standard paths. Note that $\pi(k) \notin S_i \cup S_o \Leftrightarrow H, \pi^k \vDash \neg(Input \vee Output) \Leftrightarrow H, \pi^k \vDash Exe$.

- $H, \pi \vDash \mathbf{X}^{\mathbf{e}}\varphi \Leftrightarrow (\pi(1) \notin S_i \cup S_o) \wedge (H, \pi^1 \vDash \varphi) \Leftrightarrow$
$$(H, \pi^1 \vDash Exe) \wedge (H, \pi^1 \vDash \varphi) \Leftrightarrow H, \pi^1 \vDash \varphi \wedge Exe \Leftrightarrow H, \pi \vDash \mathbf{X}(\varphi \wedge Exe);$$
- $H, \pi \vDash \mathbf{F}^{\mathbf{e}}\varphi \Leftrightarrow \exists 0 \le k < o_1^\pi : (\pi(k) \notin S_i \cup S_o) \wedge (H, \pi^k \vDash \varphi) \Leftrightarrow$
$$\exists 0 \le k < o_1^\pi : (H, \pi^k \vDash Exe) \wedge (H, \pi^k \vDash \varphi) \overset{def.o_1^\pi}{\Longleftrightarrow}$$
$$\exists 0 \le k < o_1^\pi : (H, \pi^k \vDash \varphi \wedge Exe) \wedge \forall 0 \le j \le k\ (H, \pi^j \vDash Exe) \Leftrightarrow H, \pi \vDash (Exe)\mathbf{U}(\varphi \wedge Exe);$$
- $H, \pi \vDash \mathbf{G}^{\mathbf{e}}\varphi \Leftrightarrow \forall 0 \le k < o_1^\pi : (\pi(k) \notin S_i \cup S_o) \wedge (H, \pi^k \vDash \varphi) \Leftrightarrow$
$$\forall 0 \le k < o_1^\pi : (H, \pi^k \vDash Exe) \wedge (H, \pi^k \vDash \varphi) \wedge (H, \pi^{o_1^\pi} \vDash Output) \Leftrightarrow H, \pi \vDash (\varphi \wedge Exe)\mathbf{U}(\neg Exe);$$
- $H, \pi \vDash \varphi\mathbf{U}^{\mathbf{e}}\psi \Longleftrightarrow \exists 0 \le k < o_1^\pi : (\pi(k) \notin S_i \cup S_o) \wedge (H, \pi^k \vDash \psi) \wedge$
$$\forall 0 \le j < k : (\pi(j) \notin S_i \cup S_o) \wedge (H, \pi^j \vDash \varphi) \Leftrightarrow$$
$$\exists 0 \le k < o_1^\pi : (H, \pi^k \vDash \psi \wedge Exe) \wedge \forall 0 \le j < k : (H, \pi^j \vDash \varphi \wedge Exe) \Leftrightarrow H, \pi \vDash (\varphi \wedge Exe)\mathbf{U}(\psi \wedge Exe).$$

The equivalence of the formulas $\mathbf{\Phi}^e$ and the above formulas $\mathbf{\Phi}^s$ on standard paths is proven. Let $\xi \in \mathbf{\Phi}^e$ and $\xi' \in \mathbf{\Phi}^s$ be the corresponding LTL-formula. Let us show the equivalence of these formulas on the input and output paths.

- $H, \rho \vDash \xi \Leftrightarrow H, \pi_\rho \vDash \xi \Leftrightarrow H, \pi_\rho \vDash \xi' \Leftrightarrow H, \rho \vDash \xi'$;
- $H, \sigma \vDash \xi \Leftrightarrow \forall \pi_\sigma : H, \pi_\sigma \vDash \xi \Leftrightarrow \forall \pi_\sigma : H, \pi_\sigma \vDash \xi' \Leftrightarrow H, \sigma \vDash \xi'$.

Proposition 2. Cycle-LTL formulas with initial input cycle operators are equivalent to the following LTL formulas:

- $\mathbf{X}^{\mathbf{i}}\varphi \equiv \mathbf{X}(\neg Input\mathbf{U}(Input \wedge \varphi))$;
- $\mathbf{F}^{\mathbf{i}}\varphi \equiv \mathbf{F}(Input \wedge \varphi)$;
- $\mathbf{G}^{\mathbf{i}}\varphi \equiv \mathbf{G}(Input \rightarrow \varphi)$;
- $\varphi\mathbf{U}^{\mathbf{i}}\psi \equiv (Input \rightarrow \varphi)\mathbf{U}(Input \wedge \psi)$.

Proof.

The proof uses induction by the structure of the formula. Let us first consider the input paths.

- $H, \rho \vDash \mathbf{X}^{\mathbf{i}}\varphi \Longleftrightarrow H, \rho^1 \vDash \varphi \overset{def.\pi_\rho, Input}{\Longleftrightarrow}$
$$\forall 1 \le k < i_2^\pi : (H, \pi_\rho^k \vDash \neg Input) \wedge (H, \pi_\rho^{i_2^\pi} \vDash Input \wedge \varphi) \Leftrightarrow$$
$$H, \pi_\rho \vDash \mathbf{X}(\neg Input\mathbf{U}(Input \wedge \varphi)) \Leftrightarrow H, \rho \vDash \mathbf{X}(\neg Input\mathbf{U}(Input \wedge \varphi));$$
- $H, \rho \vDash \mathbf{F}^{\mathbf{i}}\varphi \Leftrightarrow \exists k \ge 0 : H, \rho^k \vDash \varphi \overset{def.\rho}{\Longleftrightarrow}$

$$\exists k \geq 0 : (H, \rho^k \vDash \varphi) \wedge (H, \rho^k \vDash \mathit{Input}) \overset{def.\pi_\rho}{\Longleftrightarrow} \exists j \geq 0 : H, \pi_\rho^j \vDash \mathit{Input} \wedge \varphi \Leftrightarrow$$
$$H, \pi_\rho \vDash \mathbf{F}(\mathit{Input} \wedge \varphi) \Leftrightarrow H, \rho \vDash \mathbf{F}(\mathit{Input} \wedge \varphi);$$

- $H, \rho \vDash \mathbf{G}^{\mathbf{i}}\varphi \Leftrightarrow \forall k \geq 0\ H, \rho^k \vDash \varphi \overset{def.\rho}{\Longleftrightarrow}$

$$\forall k \geq 0\ (H, \rho^k \vDash \varphi) \wedge (H, \rho^k \vDash \mathit{Input}) \overset{def.\pi_\rho, Input}{\Longleftrightarrow} \forall j \geq 0\ H, \pi_\rho^j \vDash (\mathit{Input} \rightarrow \varphi)$$
$$\Leftrightarrow H, \pi_\rho \vDash \mathbf{G}(\mathit{Input} \rightarrow \varphi); \Leftrightarrow H, \rho \vDash \mathbf{G}(\mathit{Input} \rightarrow \varphi);$$

- $H, \rho \vDash \varphi \mathbf{U}^{\mathbf{i}} \psi \Leftrightarrow \exists k \geq 0 : (H, \rho^k \vDash \psi) \wedge \forall 0 \leq j < k : (H, \rho^j \vDash \varphi) \overset{def.\rho}{\Longleftrightarrow}$

$$\exists k \geq 0 : (H, \rho^k \vDash \mathit{Input} \wedge \psi) \wedge \forall 0 \leq j < k : (H, \rho^j \vDash \mathit{Input} \wedge \varphi) \overset{def.\pi_\rho, Input}{\Longleftrightarrow}$$
$$\exists l \geq 0 : (H, \pi_\rho^l \vDash \mathit{Input} \wedge \psi) \wedge \forall 0 \leq i < l : (H, \pi_\rho^i \vDash \mathit{Input} \rightarrow \varphi) \Leftrightarrow$$
$$H, \pi_\rho \vDash (\mathit{Input} \rightarrow \varphi)\mathbf{U}(\mathit{Input} \wedge \psi) \Leftrightarrow H, \rho \vDash (\mathit{Input} \rightarrow \varphi)\mathbf{U}(\mathit{Input} \wedge \psi).$$

The equivalence of the formulas Φ^i and the above formulas from Φ^s on the input paths is proved. Let $\xi \in \Phi^i$ and $\xi' \in \Phi^s$ be the corresponding LTL formula. We show the equivalence of these formulas on the standard and output paths. The proof is based on the definition of the paths π_ρ, ρ^π, σ^π, and ρ^σ.

- $H, \pi \vDash \xi \Leftrightarrow \forall \rho^\pi : H, \rho^\pi \vDash \xi \Leftrightarrow \forall \pi'_{\rho^\pi} : H, \pi'_{\rho^\pi} \vDash \xi' \overset{ind.by.struc.\xi'}{\Longleftrightarrow} H, \pi \vDash \xi';$
- $H, \sigma \vDash \xi \Leftrightarrow \forall \rho^\sigma : H, \rho^\sigma \vDash \xi \Leftrightarrow \forall \pi'_{\rho^\sigma} : H, \pi'_{\rho^\sigma} \vDash \xi' \Leftrightarrow \forall \sigma'^{\pi'_{\rho^\sigma}} : H, \sigma'^{\pi'_{\rho^\sigma}} \vDash \xi' \overset{ind.by.struc.\xi'}{\Longleftrightarrow} H, \sigma \vDash \xi'.$

Proposition 3. Cycle-LTL formulas with initial output cycle operators are equivalent to the following LTL formulas:

- $\mathbf{X}^{\mathbf{o}}\varphi \equiv \mathbf{X}(\neg \mathit{Output}\mathbf{U}(\mathit{Output} \wedge \varphi));$
- $\mathbf{F}^{\mathbf{o}}\varphi \equiv \mathbf{F}(\mathit{Output} \wedge \varphi);$
- $\mathbf{G}^{\mathbf{o}}\varphi \equiv \mathbf{G}(\mathit{Output} \rightarrow \varphi);$
- $\varphi \mathbf{U}^{\mathbf{o}} \psi \equiv (\mathit{Output} \rightarrow \varphi)\mathbf{U}(\mathit{Output} \wedge \psi).$

Proof.

The proof uses induction on the structure of the formula. Let us first consider the output paths.

- $H, \sigma \vDash \mathbf{X}^{\mathbf{i}}\varphi \Leftrightarrow H, \sigma^1 \vDash \varphi \overset{def.\pi_\sigma, Output}{\Longleftrightarrow}$

$$\forall 1 \leq k < o_2^\pi\ (H, \pi_\sigma^k \vDash \neg \mathit{Output}) \wedge (H, \pi_\sigma^{o_2^\pi} \vDash \mathit{Output} \wedge \varphi) \Leftrightarrow$$
$$H, \pi_\sigma \vDash \mathbf{X}(\neg \mathit{Output}\mathbf{U}(\mathit{Output} \wedge \varphi)) \Leftrightarrow H, \sigma \vDash \mathbf{X}(\neg \mathit{Output}\mathbf{U}(\mathit{Output} \wedge \varphi));$$

- $H, \sigma \vDash \mathbf{F}^{\mathbf{o}}\varphi \Leftrightarrow \exists\, k \geq 0 : H, \sigma^k \vDash \varphi \overset{def.\sigma}{\Leftrightarrow}$

$$\exists\, k \geq 0 : (H, \sigma^k \vDash \varphi) \wedge (H, \sigma^k \vDash \mathit{Output}) \overset{def.\pi_\sigma}{\Leftrightarrow}$$
$$\exists\, j \geq 0 : H, \pi_\sigma^j \vDash \mathit{Output} \wedge \varphi \Leftrightarrow H, \pi_\sigma \vDash \mathbf{F}(\mathit{Output} \wedge \varphi) \Leftrightarrow$$
$$H, \sigma \vDash \mathbf{F}(\mathit{Output} \wedge \varphi);$$

- $H, \sigma \vDash \mathbf{G}^{\mathbf{o}}\varphi \Leftrightarrow \forall\, k \geq 0\ H, \sigma^k \vDash \varphi \overset{def.\sigma}{\Leftrightarrow}$

$$\forall\, k \geq 0\ (H, \sigma^k \vDash \varphi) \wedge (H, \sigma^k \vDash \mathit{Output}) \overset{def.\pi_\sigma, Output}{\Longleftrightarrow}$$
$$\forall\, j \geq 0\ H, \pi_\sigma^j \vDash (\mathit{Output} \rightarrow \varphi) \Leftrightarrow H, \pi_\sigma \vDash \mathbf{G}(\mathit{Output} \rightarrow \varphi) \Leftrightarrow$$
$$H, \sigma \vDash \mathbf{G}(\mathit{Output} \rightarrow \varphi);$$

- $H,\sigma \vDash \varphi\mathbf{U}^{\mathbf{o}}\psi \Leftrightarrow \exists\, k \geq 0\ (H,\sigma^k \vDash \psi) \wedge \forall\, 0 \leq j < k\ (H,\sigma^j \vDash \varphi) \overset{def.\sigma}{\Leftrightarrow}$

$$\exists\, k \geq 0 : (H,\sigma^k \vDash Output \wedge \psi) \wedge \forall\, 0 \leq j < k\ (H,\sigma^j \vDash Output \wedge \varphi) \overset{def.\pi_\sigma, Output}{\Longleftrightarrow}$$

$$\exists\, l \geq 0 : (H,\pi_\sigma^l \vDash Output \wedge \psi) \wedge \forall\, 0 \leq i < l\ (H,\pi_\sigma^i \vDash Output \rightarrow \varphi) \Leftrightarrow$$

$$H,\pi_\sigma \vDash (Output \rightarrow \varphi)\mathbf{U}(Output \wedge \psi) \Leftrightarrow H,\sigma \vDash (Output \rightarrow \varphi)\mathbf{U}(Output \wedge \psi).$$

The equivalence of the formulas Φ^o and the above formulas from Φ^s on the input paths is proved. Let $\xi \in \Phi^o$ and $\xi' \in \Phi^s$ be the corresponding LTL formula. We show the equivalence of these formulas on standard and input paths. The proof is based on the definition of the paths π_ρ, σ^π, ρ^π and σ^ρ.

- $H,\pi \vDash \xi \Leftrightarrow H,\sigma^\pi \vDash \xi \Leftrightarrow H,\pi'_{\sigma^\pi} \vDash \xi' \overset{ind.by.struc.\xi'}{\Longleftrightarrow} H,\pi \vDash \xi'$;
- $H,\rho \vDash \xi \Leftrightarrow H,\sigma^\rho \vDash \xi; \Leftrightarrow H,\pi'_{\sigma^\rho} \vDash \xi' \Leftrightarrow H,\rho'^{\pi'_{\sigma^\rho}} \vDash \xi' \overset{ind.by.struc.\xi'}{\Longleftrightarrow} H,\rho \vDash \xi'$.

This **proposition** is a direct consequence of the previous ones.

Proposition 4. For each formula from $\Phi^e \cup \Phi^i \cup \Phi^o$, there is an equivalent LTL formula of linearly longer length.

The task of verifying models for cycle-LTL logic formulas and hyperprocess transition systems is to determine the semantics of cycle-LTL formulas in a given HTS, i.e., in calculating the set where-> $\{\chi \mid H,\chi \vDash \xi'$, where H – HTS, $i \in H, \xi \in \Phi^c\}$. Note that for any HTS, it is possible to construct a Kripke structure containing the same set of standard paths in polynomial time. Due to this remark, Proposition 4, and the estimation of the complexity of the model checking problem for LTL [11], the following theorem is satisfied:

Theorem 1. There is an algorithm that solves the model checking problem for cycle-LTL formulas and hyperprocess transition systems. Its complexity polynomially depends on the size of the hyperprocess transition system, and exponentially depends on the length of the cycle-LTL formula.

CONCLUSION

In this paper, we have developed a hyperprocess transition system (HTS) for modeling software for programmable logic controllers (PLCs) and a new cycle-LTL logic for formulating PLC properties. This HTS model naturally captures the features of PLC programs, such as control cycles and timers. The proposed temporal logic cycle-LTL allows us to describe the properties of PLC programs both of small time steps of the inner phase of the execution of control cycles, and of large time steps of the control cycles themselves. We described the translation of cycle-LTL logic formulas into LTL logic formulas and proved its correctness. This translation demonstrates that the formulation of the properties of control systems directly in terms of LTL logic turns out to be much more complicated and confusing. We have shown that due to the correctness of this translation, the problem of verifying models for HTS and cycle-LTL is decidable.

We plan to use the results presented in this article to ensure the correct translation of the process-oriented Reflex language into the Promela language used by the SPIN verifier [23]. In addition, we are going to develop and implement a special model checking algorithm for cycle-LTL and HTS. We expect that this algorithm in many cases will have a lower time complexity than the standard model checking algorithm for LTL, due to the use of such features of HTS as cyclicity, strict order of process actions, and determinism of these actions. We also plan to explore the possibility of using the cycle-LTL logic to formulate the properties of more general transition systems.

FUNDING

This work has been funded by the state budget of the Russian Federation (project no. AAAA-A19-119120290056-0 of the Institute of Automation and Electrometry).

REFERENCES

1. Anureev, I., Operational semantics of annotated Reflex programs, *Model. Anal. Inf. Sist.,* 2019, vol. 26, no. 6, pp. 181–192.
2. Anureev, I., Garanina, N., Liakh, T., Rozov, A., and Zyubin, V., Towards safe cyber-physical systems: The Reflex language and its transformational semantics, *IEEE International Siberian Conference on Control and Communications,* IEEE, 2019, pp. 18–20.
3. Brinksma, E. and Mader, A., *Verification and optimization of a PLC control schedule, SPIN 2000 – SPIN Model Checking and Software Verification,* Springer, 2000, pp. 73–92.
4. Gourcuff, V., de Smet, O., and Faure, J.-M., Improving large-sized PLC programs verification using abstractions, *IFAC Proc. Vol.,* 2008, vol. 41, no. 2, pp. 5101–5106.
5. Mader, A., A classification of PLC models and applications, in *Discrete Event Systems,* Springer, 2000, pp. 239–246.
6. Wan, H., Chen, G., Song, X., and Gu, M., Formalization and verification of PLC timers in Coq, *Proc. of 33rd Annual IEEE International Computer Software and Applications Conference,* IEEE, 2009, pp. 315–323.
7. Yoo, J., Cha, S., and Jee, E., A verification framework for FBD based software in nuclear power plants, *Proc. of 15th Asia-Pacific Software Engineering Conference,* IEEE, 2008, pp. 385–392.
8. Bulavskij, D., Zyubin, V., Karlson, N., Krivoruchko, V., and Mironov, V., An automated control system for a silicon single-crystal growth furnace, *Optoelectron. Instrum. Data Process.,* 1996, vol. 32, no. 2, pp. 25–30.
9. Kovadlo, P.G., Lubkov, A., Bevzov, A., et al., Automation system for the large solar vacuum telescope, *Optoelectron. Instrum. Data Process.,* 2016, vol. 52, pp. 187–195.
10. Gupta, A., Kahlon, V., Qadeer, S., and Touili, T., *Handbook of Model Checking,* Springer Int. Publ., 2018, ch. 18, pp. 573–577.
11. Clarke, E.M., Henzinger, T.A., and Veith, H., *Handbook of Model Checking,* Springer Int. Publ., 2018, ch. 1, pp. 1–13.
12. Dierks, H., PLC-automata: A new class of implementable real-time automata, in *International AMAST Workshop on Aspects of Real-Time Systems and Concurrent and Distributed Software,* Springer, 1997, vol. 1231, pp. 111–125.
13. Ovatman, T., An overview of model checking practices on verification of PLC software, *Software Syst. Model.,* 2016, vol. 4, no. 15, pp. 937–960.
14. Kuzmin, E., Ryabukhin, D., and Sokolov, V.A., On the expressiveness of the approach to constructing PLC-programs by LTL-specification, *Autom. Control Comput. Sci.,* 2016, vol. 7, no. 50, pp. 510–519.
15. Zhang, M., Towards automated safety vetting of PLC code in real-world plants, *IEEE Symposium on Security and Privacy,* IEEE, 2019, pp. 522–538.
16. Rajeev, A. and Henzinger, T., A really temporal logic, *J. ACM,* 1994, vol. 41, no. 1, pp. 181–203.
17. Xiong, J., A user-friendly verification approach for IEC 61131-3 PLC programs, *Electronics,* 2020, vol. 4, no. 9.
18. Beckert, B., Regression verification for programmable logic controller software, *International Conference on Formal Engineering Methods,* Springer, 2015, vol. 9407.
19. Ljungkrantz, O., An empirical study of control logic specifications for programmable logic controllers, *Empirical Software Eng.,* 2014, vol. 3, no. 19, pp. 655–677.
20. Ljungkrantz, O., Åkesson, K., Fabian, M., and Yuan, C., A formal specification language for PLC-based control logic, *8th IEEE International Conference on Industrial Informatics,* IEEE, 2010, pp. 1067–1072.
21. Maler, O. and Nickovic, D., Monitoring temporal properties of continuous signals, in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems,* Springer, 2004, pp. 152–166.
22. Garanina, N., Anureev, I., Zyubin, V., Rozov, A., Liakh, T., and Gorlatch, S., Reasoning about programmable logic controllers, *Syst. Inf.,* 2020, vol. 17, pp. 33–42.
23. Holzmann, G., *The SPIN Model Checker: Primer and Reference Manual,* Addison-Wesley Professional, 2003.

Translated by T. N. Sokolova

SPELL: 1. OK