UDK 004

Reasoning about Programmable Logic Controllers

Garanina N.O. (Institute of Informatics Systems SB RAS, Institute of Automation and Electrometry SB RAS)

Anureev I.S. (Institute of Informatics Systems SB RAS, Institute of Automation and Electrometry SB RAS)

Zyubin V.E. (Institute of Automation and Electrometry SB RAS)

Rozov A.S. (Institute of Automation and Electrometry SB RAS)

Liakh T.V. (Institute of Automation and Electrometry SB RAS)

Gorlatch S.(University of Muenster)

We address the formal verification of the control software of critical systems, i.e., ensuring the absence of design errors in a system with respect to requirements. Control systems are usually based on industrial controllers, also known as Programmable Logic Controllers (PLCs). A specific feature of a PLC is a scan cycle: 1) the inputs are read, 2) the PLC states change, and 3) the outputs are written. Therefore, in order to formally verify PLC, e.g., by model checking, it is necessary to reason both in terms of state transitions within a cycle and in terms of larger state transitions according to the scan-cyclic semantics.

We develop a formalization of PLC as a hyperprocess transition system and an LTL-based temporal logic cycle-LTL for reasoning about PLC.

Keywords: *formal verification, model checking, temporal logics, transition systems, programmable logic controllers*

1. Introduction

The long-term goal of our work is formal verification of control software specified in the process-oriented paradigm, in particular programs written in the Domain-Specific Language (DSL) Reflex [1, 7]. Formal verification of Programmable Logic Controllers (PLCs), which are important components of automatic control systems, is an active topic of practical and research work [2, 4]. The proposed approaches follow various PLC models [8]. In general, PLC functioning consists of infinite sequence of *scan cycles.* Each scan cycle includes a sequence of three phases: reading input, execution, and writing output.

We choose a process-oriented PLC modeling based on *hyperprocesses* [7]. This modeling method allows us to specify the main features of the PLC, such as scan cycle and timers; it describes a PLC as a synchronized system of interacting functional processes defined by a set of functional states and actions on these states. According to the classification [8], hyperprocesses model PLCs that abstract from scan cycle time (the environment is considered to be slow enough to assume zero time for the input/output and execution phases of a scan cycle) and use input and output controlling timers. This modeling provides a natural specification for the control multiprocess systems, which, due to abstracting from the scan cycle time, allows ordering process executions in a scan cycle. Such high synchronization of processes without interleaving makes possible the effective use of formal verification methods. A hyperprocess is a base for the process-oriented language Reflex that was used in a number of industrial projects, in particular, a plant for growing silicon single crystals using Czochralski method, and a vacuum system for the Big Solar Vacuum Telescope [6].

In this paper, we assume model checking as our formal verification method. Therefore, we present a hyperprocess as a special transition system, and the properties of a hyperprocess as formulas of a special temporal logic. A hyperprocess transition system is close to a concurrent multi-threaded system [3] enriched with synchronizing counter, process functional states, timers, and action primitives for their changing. The time for a PLC specified as a hyperprocess is clocked both inside of the scan cycle execution (taking into account changes of variable values) and outside of the execution at the reading of the inputs.

The logic for reasoning about PLC should allow the formulation of statements for these two kinds of clocks. In this paper, we develop the *cycle-LTL* logic – which is an LTL enriched with cycle temporal operators for reasoning about PLC states outside of a scan cycle. In this logic, the following properties are expressible for a simple example of a hand-dryer machine: "*If the sensor has detected hands, the dryer will turn on in the next scan cycle*" or "*If the temperature is higher than the critical value, the cooling process is always on.*" Note that if the switching on and off for the cooling process is performed after actions of other processes in a scan cycle, then the last property can be violated inside the cycle, but it can hold outside the cycle. This example illustrates the need to use cyclic temporal operators, in particular special cycle always-operator $\mathbf{G^c}$.

2. Hyperprocess Transition System

Let us give an informal description of the hyperprocess [7]. *Hyperprocess* is an ordered set of interacting processes that execute sequentially in a given order, forming a *scan cycle*. This cycle starts by reading the input from the environment into the hyperprocess system variables, and finishes by writing outputs to the environment. All hyperprocess variables are global. A feature of the processes that form a hyperprocess are *functional states* – labels that mark a sequence of process actions. The functional states of every process include the states of normal shutdown and abnormal shutdown: in these states, the process does not perform any actions. Process actions can change the values of hyperprocess variables (except input variables), affect the functional states of other processes, and set and reset timer values. Actions may have guard conditions depending on the hyperprocess variables and functional states of other processes. Our definition of the hyperprocess transition system is based on a description of the hyperprocesses and the operational semantics of the Reflex language [1, 7]. We hide the output phase inside the execution phase without loss of generality.

Definition 1. *(Hyperprocess transition systems, HTS)*
HTS is a tuple $H = (P, S, s_{ini}, A, R)$, where

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

Before defining HTS-components, we describe hyperprocess elements in general.

Definition 2. *(Hyperprocess elements)*
Hyperprocess elements are variables, functional states, process actions, and timers:

Variables. $V = \{v_1, \ldots, v_N\}$ is a set of hyperprocess variables which values are the result of the corresponding functions $v_i : S \mapsto D \cup \{\bot\}$. We distinguish *input variables* V_E and *process variables* V_P: $V = V_E \cup V_P$.

Functional states. For every $i \in [1..n]$ $F_i = \{f_i^1, ..., f_i^{m_i}, stop, err\}$ is *a set of functional states* of process p_i. The *stop* and *err* are *inactive* states, and other states are *active* states. The value of every functional state variable f_i is described by function $f_i : S \mapsto F_i$.

Actions. In functional state f_i^j, process p_i performs actions from set A. These actions form *the body of the functional state.* $L_i^j \in \mathbb{N}$ is the number of actions in this body. Variable a_i is *an action counter* and its value is *a position.* The value of the action counter is the result of function $a_i : S \mapsto [1..L_i^j] \cup \{\bot\}$. The next value of this counter is defined by functions $nxt^j : \mathbb{N} \times S \mapsto \mathbb{N} \cup \{\bot\}$. These functions implicitly include guards for actions because the results depend on a current HTS-state, in particular, on activity of other processes. If there is no the next action in the current functional state then $nxt^j(a_i) = \bot$. Functions $an^j : \mathbb{N} \cup \{\bot\} \mapsto A$ return the name of the action at position a_i.

Timers. *The timer* of process p_i is variable t_i with values as the result of functions $t_i : S \mapsto \mathbb{N} \cup \{\bot\}$. *The timer bound* at position a of functional state f_i^j is $N_i^{ja} \in \mathbb{N}$. For simplicity, we consider the processes with one timer per a functional state.

Let us define the components of HTS-system $H = (P, S, s_{ini}, A, R)$.

The set of states S

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

- the state of variables $v = (v_1(s), \dots, v_N(s))$ for variables' values in state s;
- the state of processes $sp = ((f_1(s), a_1(s), t_1(s)), \dots, (f_n(s), a_n(s), t_n(s)))$, where for every process p_i in state s, $f_i(s)$ is its current functional state, $a_i(s)$ is an action counter in state $f_i(s)$, and $t_i(s)$ is the value of its timer;
- process counter $pc(s)$ with values in $[0..n]$, where 0 is reserved for updating input.

The initial state s_{ini}

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

– $v_0 = (\bot_1, \dots, \bot_N)$,

– $sp_0 = ((f_1^1, 1, \bot)_1, (stop, \bot, \bot)_2, \dots, (stop, \bot, \bot)_n)$, and

– $pc_0 = 0$.

The action alphabet A

The alphabet includes a single environment action and process actions:

$A = \{upd0, skip, end, upd, tout, reset, startP, stopP, start, set, next, stop, err\}$.

0. The cycle action.

*upd*0 – change of values of input variables, i.e. reading environment inputs.

1. Service actions.

skip – a process does nothing in inactive states.

end – a process transfers control to the next process at the end of its active state body.

2. Actions for updating non-input variables.

upd – a process changes the values of some variables.

3. Timeout actions.

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

reset – a process resets its timer to zero.

4. Actions for functional states.

startP/*stopP* – a process transfers target process p_k to functional state $f_k^1/stop$;

start/*set* – a process transits to target functional state f_i^1/f;

stop/*err* – a process transits to functional state *stop*/*err*;

next – a process transits to the next functional state.

The labeled transition relation R

The transition relation R gives the semantics to actions of processes and the environment. Let $i \in [1..n], j \in [1..m_i]$, $a \in [1..L_i^j]$. We use the following notation.

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 $R(act) = (s, s')$, where

- $act = an^j(a)$, and $an^j(\bot) \in \{skip, end\}$;
- before-state s is such that $pc(s) = i$, $f_i(s) = f_i^j$, $a_i(s) = a$, the time constraint 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 constraint Tg_i can be $tgp_i = k$ or $tgs_i = k$, where tgp_i is variable in V_P with values in $[1..n]$ for specifying the target process, and tgs_i is variable in V_P with values in $[1..m_i]$ for specifying the target functional state of process p_i; non-mentioned left elements have arbitrary values;
- after-state s' specifies the changes of hyperprocess elements after action act: $y_k(s') = new_k$ ($k \in [1..m']$); non-mentioned right elements are not changed.

0. The external update action.

We use the notation like above to specify $R(upd0)$. At the beginning of the scan cycle, the values of input data are read to the input variables in V_E, the value of every ticking process timer is increased by 1, and the process counter points to the first process:

$$(t_{i_1} \neq \bot, \ldots, t_{i_k} \neq \bot, pc = 0) \xrightarrow{upd0}$$
$$(v'_1 = v_1, \ldots, v'_m = v_m, t'_{i_1} = t_{i_1} + 1, \ldots, t_{i_k} = t_{i_k} + 1, pc' = 1).$$

In the following definition of the transition relation R for process actions, we suppose that process p_i performs the action number a_i with name $an^j(a_i)$ in its functional state $f_i = f_i^j$.

1. Service actions.

Process p_i does nothing in inactive states *stop* and *err*, and passes control to the next process:

– $(f_i = stop, pc = i) \xrightarrow{skip} (pc' = |i+1|_{n+1})$.

– $(f_i = err, pc = i) \xrightarrow{skip} (pc' = |i+1|_{n+1})$.

When process p_i reaches the end of the body of its active functional state f_i^j, it goes to the first action of the body and transfers control to the next process:

– $(f_i = f_i^j, a_i = \perp, pc = i) \xrightarrow{end} (a_i' = 1, pc' = |i+1|_{n+1})$.

2. The action for updating variable values.

By this action, process p_i changes values of some variables from the set of non-input variables $\{v_1, \ldots, v_m\} \subseteq V_P$, and goes to the next action $nxt^j(a)$ of its current state:

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

3. Timeout actions.

In these cases, process p_i starts the timer t_i (if $t_i = \perp$), goes to the first action of its current state f_i^j, and transfers control to the next process:

– $(f_i = f_i^j, t_i = \perp, pc = i) \xrightarrow{tout} (a_i' = 1, t_i' = 0, pc' = |i+1|_{n+1})$;

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

In the case of timeout, process p_i stops the timer t_i and goes to the next action in its 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 $reset$-action results exactly as the first case of the $tout$-action:

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

4. Actions for functional states.

These two actions of process p_i force process p_k $(i \neq k)$ to go to start or stop state:

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

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

With these actions, process p_i goes to the first action of the corresponding functional states and stops the timer:

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

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

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

Process p_i perform these actions in case of it should do nothing from this moment because of normal or error shutdown: ?? и выполняет следующие действия текущего состояния??

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

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

According to the defined transition relation, the processes act sequentially in the current cycle, following the order specified by the process counter. Let an input state s_{inp} be a state with $pc(s_{inp}) = 0$. We define *a cyclic state* s_c as a state just after updating the input variables and before the processes start to act: $R(upd) = (s_{inp}, s_c)$. The set of cycle states is S_c.

We define two kinds of paths in HTS. *Standard path* $\pi = s_0, s_1, \ldots$ is a sequence of states $s_i \in S$ such that $\forall i \geq 0\ \exists a \in A : R(a) = (s_i, s_{i+1})$. Let $\pi(i)$ be i^{th} state on path π. *Cycle path* $\sigma = c_0, c_1, \ldots$ is an infinite sequence of cycle states $c_i \in S_c$ such that for every $i \geq 0$ there exists a finite standard path π_i of length n_i with $\pi_i(0) = c_i$ and $\pi_i(n_i) = c_{i+1}$. If ρ is the standard or cycle path, let $\rho(k)$ be k^{th} state on this path, ρ^k be the suffix of ρ starting from $\rho(k)$, and c^π be the number of the first cycle state on standard path π.

3. Temporal logic cycle-LTL

The syntax of our cycle-LTL logic includes propositions P, boolean connections, standard LTL temporal operators, inner-cycle temporal operators, and cycle temporal operators:

$$\varphi ::= P \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi\mathbf{U}\varphi \mid$$
$$\mathbf{X^i}\varphi \mid \mathbf{F^i}\varphi \mid \mathbf{G^i}\varphi \mid \varphi\mathbf{U^i}\varphi \mid \mathbf{X^c}\varphi \mid \mathbf{F^c}\varphi \mid \mathbf{G^c}\varphi \mid \varphi\mathbf{U^c}\varphi$$

The inner-cycle operators $\mathbf{X^i}$, $\mathbf{F^i}$, $\mathbf{G^i}$, and $\mathbf{U^i}$ are used for formulating properties of a control system which have to hold during a particular cycle execution phase, while *the cycle operators* $\mathbf{X^c}$, $\mathbf{F^c}$, $\mathbf{G^c}$, and $\mathbf{U^c}$ are used for formulating properties of the control system which have to hold at the beginning of cycles.

Let the set of standard LTL formulas be Φ^s, the set of formulas started with inner-cycle operators be Φ^i, the set of formulas started with cycle operators be Φ^c, and the set of all cycle-LTL formulas be Φ^{sic}.

The semantics of the cycle-LTL are defined for new inner-cycle and cycle temporal operators only. We define the semantics of inner-cycle operators on standard paths and semantics of cycle operators on standard and cycle path. The semantics of standard LTL formulas can be found in [3]. Let H be a hyperprocess transition system, π be an infinite standard path, σ be a cycle path, and $\varphi, \psi \in \Phi^{sic}$ be formula of cycle-LTL..

The semantics of formulas in Φ^i.

- $H, \pi \models \mathbf{X^i}\varphi$ iff $\pi^1 \notin S_c$ and $H, \pi^1 \models \varphi$;

- $H, \pi \models \mathbf{F^i}\varphi$ iff there exists $0 \leq k < c^\pi$ such that $H, \pi^k \models \varphi$;
- $H, \pi \models \mathbf{G^i}\varphi$ iff for all $0 \leq k < c^\pi$ $H, \pi^k \models \varphi$;
- $H, \pi \models \varphi\mathbf{U^i}\psi$ iff there exists $0 \leq k < c^\pi$ such that $H, \pi^k \models \psi$ and for all $0 \leq j < k$ $H, \pi^j \models \varphi$.

The semantics of formulas in Φ^c.

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

- $H, \pi \models \xi$ iff $H, \sigma \models \xi$ with $\sigma(0) = \pi(c^\pi)$;
- $H, \sigma \models \mathbf{X^c}\varphi$ iff $H, \sigma^1 \models \varphi$;
- $H, \sigma \models \mathbf{F^c}\varphi$ iff there exists $k \geq 0$ such that $H, \sigma^k \models \varphi$;
- $H, \sigma \models \mathbf{G^c}\varphi$ iff for all $k \geq 0$ $H, \sigma^k \models \varphi$;
- $H, \sigma \models \varphi\mathbf{U^c}\psi$ iff there exists $k \geq 0$ such that $H, \sigma^k \models \psi$ and for all $0 \leq j < k$ $H, \sigma^j \models \varphi$.

Let us give some informal comments for semantics of cycle-LTL formulas. We consider the cases when a formula of some type is a subformula of other type formula at the first nesting level of temporal operators: $\varphi \in nl^1(\psi)$. Let $\varphi^s, \psi^s \in \Phi^s$, $\varphi^i, \psi^i \in \Phi^i$, and $\varphi^c, \psi^c \in \Phi^c$. We have six cases.

1. $\varphi^i \in nl^1(\psi^c)$: this assertion states that φ^i holds during the execution part of scan cycles explicitly specified by ψ^c.
2. $\varphi^i \in nl^1(\psi^s)$: this assertion states that φ^i holds during the execution part of scan cycles implicitly specified by ψ^s.
3. $\varphi^c \in nl^1(\psi^i)$: if φ^c is in boolean connection with formulas in $\Phi^s \cup \Phi^i$, this assertion binds a property of the execution phase of a scan cycle explicitly specified by ψ^i to a cycle property of the next cycle.
4. $\varphi^c \in nl^1(\psi^s)$: if φ^c is in boolean connection with formulas in $\Phi^s \cup \Phi^i$, this assertion binds a property of the execution phase of a scan cycle implicitly specified by ψ^s to a cycle property of the next cycle.
5. $\varphi^s \in nl^1(\psi^c)$: this assertion states that φ^s holds at some cycle state explicitly specified by ψ^c. If ψ^c include operator $\mathbf{G^c}$ or $\mathbf{U^c}$, φ^s will hold periodically with respect to scan cycle.
6. $\varphi^s \in nl^1(\psi^i)$: this assertion states that φ^s holds at some state in execution part of a scan cycle specified by ψ^i.

As an illustration, let us define the cycle-LTL specifications for the properties of a hand-dryer and cooling machine as example control systems:

1. *If the sensor has detected hands, the dryer will turn on in the next scan cycle*:

   $\mathbf{G^c}(hands = on \rightarrow \mathbf{X^c} dryer = on)$.
2. *If the temperature is higher than the critical value, the cooling is always on*:

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

The above examples of formulas for control system properties use only cycle states and observable input-output system variables. This formulation can be used for high-level properties of implementation independent models of control system viewed as a black box. However, if some high-level property fails for some implementation of the control system, then checking low-level properties of the system may be required. These properties are formulated in terms of process states and actions. They are hypotheses which use inner-cycle operators to localize the error within the scan cycle. Verifying such hypotheses can be less time-consuming then analyzing a counterexample for the failed high-level property. The following properties for a system that combines a lighting control system and a burglar alarm system illustrates causality between a low-level hypothesis and a high-level property: failing the former implies failing the latter.

1. *When a break-in is detected, all lamps should flash*:

   $\mathbf{G^c}(alarm \rightarrow \mathbf{X^c} alarm\_light = on)$.
2. *When the alarm sensor is on then the security alarm subsystem should send the alarm message to all other subsystems ASAP*:

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

4. Conclusion

In this paper, we develop the hyperprocess transition systems (HTS) for modeling PLC and the novel cycle-LTL logic for specifying PLC properties. Our HTS-model naturally captures features of PLC such as scan cycles and timers. Our cycle-LTL temporal logic enables reasoning about PLC properties w.r.t. both small-step time inside scan cycles and big-step time over scan cycles. Expressing the big-step properties in a standard LTL would be much more cumbersome.

We plan to prove that the model checking for HTS and cycle-LTL is reduced to the standard LTL model checking. For this we will translate HTS into the Kripke structure and cycle-LTL formulas into LTL formulas. The method of this translation will provide the basis for the correct translation of the process-oriented language Reflex into the Promela language used by the SPIN verifier [5]. We also plan to develop and implement a special model checking algorithm for verifying cycle-LTL formulas in HTS, which will have lower time complexity than the standard model-checking algorithm for LTL due to the use of HTS features such as cyclicity and ordering action processes.

References

1. Anureev I.S. Operational Semantics of Reflex // System Informatics. 2019. V. 14. P. 1–10.
2. Brinksma E., Mader A. Verification and Optimization of a PLC Control Schedule // In: Havelund K., Penix J., Visser W. (eds) SPIN Model Checking and Software Verification. SPIN. Springer, LNCS. 2000. V. 1885. P. 73–92.
3. Clarke E.M., Henzinger Th.A., Veith H., Bloem R. // Handbook of Model Checking. Springer International Publishing. 2018. Ch. 18.
4. Gourcuff V., de Smet O., Faure J.-M. Improving large-sized PLC programs verification using abstractions. // IFAC Proceedings. 2008. V. 41. № 2. P. 5101–5106.
5. Holzmann G. The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional, Boston, 2003.
6. Kovadlo P.G., Lubkov A.A., Bevzov, A.N. et al. Automation system for the large solar vacuum telescope. // Optoelectronics, instrumentation and data processing. 2016. V. 52, P. 187–195.
7. Liakh T.V., Rozov A.S., Zyubin V.E. Reflex Language: a Practical Notation for Cyber-Physical Systems // System Informatics. 2018. V. 12. P. 85–104.
8. Mader A. A Classification of PLC Models and Applications. // In: Boel R., Stremersch G. (eds) Discrete Event Systems. SECS. Springer, Boston, MA, 2000. V. 569. P. 239–246.