Conformance Checking for Programmable Logic Controller Programs and Specifications

Dániel Darvas*†, István Majzik* and Enrique Blanco Viñuela†
*Budapest University of Technology and Economics, Department of Measurement and Information Systems
Budapest, Hungary, Email: {darvas,majzik}@mit.bme.hu
†European Organization for Nuclear Research (CERN), Beams Department
Geneva, Switzerland, Email: {ddarvas,eblanco}@cern.ch

Abstract—**Verification of industrial control systems' software is an important task, as the cost of failure in these systems is typically high. Formal verification methods can complement the currently used testing techniques, especially if requirements are formally specified. Behavioural specifications can be used to perform conformance checking against the implementation. However, the typical conformance relations are often more sensitive to differences than the controlled processes in case of many control systems, resulting in counterexamples during verification that are considered as false positives in practice. To overcome this issue, we introduce conformance relations adapted to control systems based on programmable logic controllers (PLCs) with different levels of permissibility. The relations can be selected by the control engineers, depending on the required compliance levels. Defining the new relations and a model checking-based method to check them makes conformance checking a powerful tool for the verification of industrial control systems.**

I. Introduction

Industrial control systems (ICS) often rely on *programmable logic controllers* (PLCs). PLCs are robust industrial computers performing various control tasks. The typical execution schema of a PLC is cyclic: the user-defined program is executed in an infinite loop (so-called *PLC* or *scan cycle*). At the beginning of the loop the physical inputs are read, then the program is executed using these stable input values. At the end of each loop the physical outputs are assigned which are then stable until the end of the next cycle. The duration of the scan cycle may vary in most of the PLCs.

While PLC programs were rather simple in the past, they tend to be more and more complex. As the complexity and the criticality grow, the precise verification of PLC programs comes more and more into focus. We can observe this phenomenon at CERN (European Organization for Nuclear Research), where complex control systems are being used to operate the facilities of the institute. Current best practices apply testing to check the correctness of PLC programs, however testing does not provide an exhaustive analysis.

Formal verification is a good candidate to complement testing in order to improve the verification of PLC-based control software. Verification needs detailed, precise specifications: without defined requirements it is not possible to argue about correctness. Different works [1], [2] aim to provide formal specification methods for PLC-based systems. The latest specification methods provide solutions for formalizing the requirements and allow new use cases for verification: besides temporal property specifications (verified by model checking) these offer more detailed behaviour specifications that allow conformance checking of the implementation.

Various *behavioural equivalence relations* were introduced in the past for these purposes, such as the widely-known trace equivalence, (bi)simulation, and ioco relations [3], [4], [5]. For reactive systems [6] these relations are useful in practice, but this is not always the case for ICS. A PLC with a typical cycle time of 1–100 ms may control a slow process (e.g. cryogenic system) where certain responses of the plant are expected in minutes or even later. For many signals, a delay by 1–100 ms has no significant impact. Furthermore, these slight changes of the outputs, often caused by code reorganization, cannot be easily avoided in a complex system. These acceptable differences appear as false positives in the equivalence checking. As typically only a single counterexample is provided, it is difficult to identify and exclude all these cases.

Our key observation is that the widely-known conformance relations and even the PLC-specific conformance relations are typically too strict. The ICS domain needs conformance relations with configurable sensitivity to be useful in the development process. In summary, *we need to develop new conformance relations specifically targeting the PLC-based ICS to make conformance checking useful in practice.*

In this paper we introduce conformance relations for PLC software development, inspired by the needs observed in ICS at CERN. We define a set of configurable, permissive relations which responds to the identified special needs of the targeted domain. These relations are formalised in this paper, and their verification using model checkers is defined. The *structure of the paper* is the following. Section II presents the analysis of the special needs of the targeted domain. Section III defines the conformance relations designed on this basis. The checking methods for these relations are discussed in Section IV. Section V is dedicated to the validation of the presented ideas. Section VI overviews the related work on conformance methods, focusing especially on PLC-related usage. Section VII summarizes the paper and the future work.

II. Motivation – Domain Requirements

This section overviews the special needs[1] of the industrial control systems domain to provide motivation for our work.

[1]"Needs" will always refer to the requirements for conformance relations.

Authors' manuscript. Published in *11th IEEE International Symposium on Industrial Embedded Systems (SIES)*, IEEE, 2016. The final publication is available at IEEE Xplore. DOI: 10.1109/SIES.2016.7509409.

Use Cases: We consider three main cases of applying conformance checking:

- **Specification–implementation.** An implementation and the corresponding specification can be compared to check the conformance of a generated or manually created implementation and the specification. Comparing a specification and an implementation can also be useful in re-engineering: the properties of a specification extracted from a legacy implementation can be systematically checked.
- **Implementation–implementation.** Two implementations can be compared e.g. to check different implementations of the same specification, or to check that an extended version of an implementation still provides the previous behaviour (in addition to some new behaviours).
- **Specification–specification.** Two specifications can be compared e.g. to be sure that the new version of the specification preserves the behaviour given by the previous version, or an optimised/reorganised specification still behaves in the same way as the previous version.

Permissibility of the Relations: We have extracted some motivational examples from CERN's PLC-based control systems (Fig. 1). The main observation is the fact that small differences of the control outputs may be observable, but practically equivalent from the point of view of the properties of the control functions (Fig. 1a). Accordingly, those differences are often treated as false positives. Focusing on the relevant differences implies the need for *more permissive conformance relations*. However, this need co-exists with the need for strict equivalence, depending on the role of the outputs. Often some outputs are insensitive to small delays (e.g. an output used only for information by the supervision system or controlling slow processes), while others might not allow any differences. Therefore the conformance relations should not be selected for two compared artefacts (specification or implementation), but for pairs of outputs (one output per compared artefact).

The level of permissibility is given separately for each output pairs, and not included in the specification. This way the specification can be kept "clean", describing the ideal required behaviour. Also, this makes the comparison of two implementations possible. Being able to select different relations for different output pairs also gives flexibility in checking artefacts (specification or implementation) with different input/output signatures. For instance, the behaviour of an artefact M can be compared to M', that is M extended with new outputs. By selecting conformance relations only for the variable pairs contained in both M and M' it is possible to analyse if the extension had any side-effect on the base behaviour. However, this does not mean that only one variable pair's conformance can be checked in a single model checking run, the checks for several variable pairs can be conjuncted.

Even if an output may allow some delay, in some cases the amount of allowed delay should be fixed during the whole execution (e.g. *out*$'$ compared to *out* in Fig. 1b), in other cases the delay may vary during an execution (e.g. *out*$''$ compared to *out* in Fig. 1b). The allowed differences depend on the usage of the signal. Moreover, for some signals, the sum of pulse durations should be equal to call them conformant (Fig. 1c), for some other signals the pulse duration is secondary, but the number of rising or falling edges should be the same (Fig. 1d).

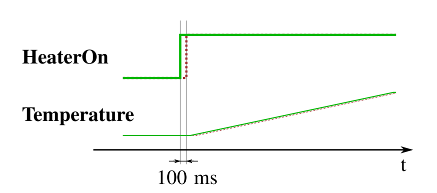
Based on the discussion above, the following three main conformance relation categories are defined, depending on the permissibility of the conformance.

- **Strict equivalence relation.** This relation requires the same output sequences for the same input sequences under the same timing conditions in the two artefacts (specification or implementation).
- **Permissive conformance relations with fixed delay.** These relations permit delays by certain number of PLC cycles between the outputs of the two artefacts for the same inputs and timing, however the delay should be constant, therefore the shape of the two output signals will be basically identical.
- **Permissive conformance relations with variable delay.** These relations also allow delays in the outputs, and the amount of delay may vary during the execution (between defined limits). In certain cases such relation provides enough restriction for the conformance checking, e.g. in case of status outputs used for informative purposes in the supervision systems.

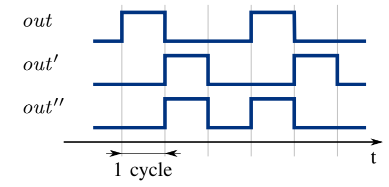
Cycle Time: Although PLCs may have interrupts, interrupt-driven reactive behaviour seems to be uncommon in practice. In this paper we do not consider user-defined interrupts and we treat PLCs as transformational systems [6], transforming an input sequence to an output sequence under some timing conditions. However, e.g. communication- or operating system-related interrupts may occur during execution. A side effect of them is that the length of a scan cycle may vary in most of the PLCs. This causes non-determinism in the otherwise deterministic execution. As the PLC timers' delay parameters are defined in physical time units, not in number of cycles, this can have observable collateral effects (Fig. 1e). The length of a scan cycle also depends on the number and type of executed instructions, or on the used hardware. Accordingly, we consider variable cycle lengths (cycle durations) during the execution. However, we assume that the two compared artefacts process the same sequence of inputs (otherwise they will be trivially inequivalent). To assure this, during equivalence checking we consider the same input and cycle length sequences for both artefacts. Note that programs running on non-fail-safe PLCs typically do not have fixed cycle length, therefore the implementation should not rely on the exact value of the cycle length (this is typically treated as non-deterministic by the developers), therefore these assumptions should not pose any restriction.

Extracted Requirements: By analysing the systems in use and the motivational examples, the conformance relations should satisfy the following main needs.

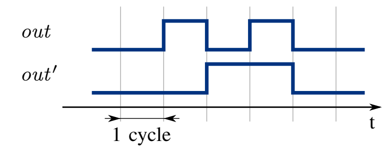
N1. *Permissive conformance relations are needed*, permitting delays between signals, adapted to the PLCs and



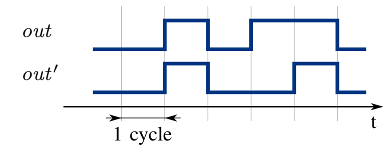
(a) A delay of one PLC cycle often causes an unnoticeable difference.



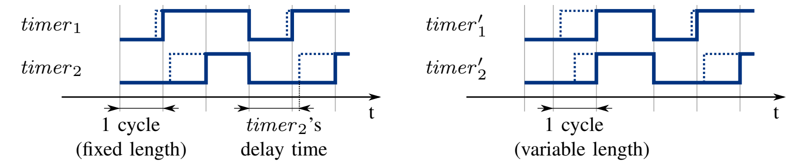
(b) The allowed delays can be constant (same for the whole execution, out') or varying (out'').



(c) The varying delay may modify the amount of rising/falling edges.



(d) For some signals the edge number preservation is more important than the pulse duration preservation.



(e) The fixed PLC cycle length can hide some behaviours. The timeouts (rising edges) of $timer_1$ and $timer_2$ never happen in the same cycle using fixed lengths, but it is possible using variable cycle length ($timer'_1$ and $timer'_2$).

Fig. 1: Motivational examples

the process control domain.

N2. *Multiple relations are needed* that can be adjusted to the desired level of conformance for each corresponding output pair.

N3. The conformance relations (and their checking method) should be able to handle *time-dependent behaviours*, reflecting the PLC's typical behaviour (cyclic behaviour with generally non-fixed cycle time).

N4. Different *use cases* (not only code-code conformance, but also specification-specification and specification-code conformance) need to be supported.

N5. A general checking method is needed that can *scale up to complex cases*.

III. PLC Conformance Relations

This section introduces the conformance relations defined upon the identified needs. Section III-A describes the modelling and formalization of the artefacts and conformance checking problem. Section III-B defines the conformance relations.

A. Formalization and Assumptions

For the formal definition of conformance relations, we provide the formalization of behaviour semantics of PLC programs and we introduce the necessary symbols. The formalization is harmonised with the definitions of [7].

As mentioned previously, a good compromise had to be found between precision and performance of verification for time modelling (N3). The assumption that the PLC cycle length is fixed is too vague (cf. Fig. 1e), and it may hide possible behaviours. On the other hand, if the time is modelled precisely, checking the conformance relations becomes troublesome. We follow an approach similar to the one proposed in [8]: the length of each PLC cycle is non-deterministic with an accuracy of 1 ms (same as the accuracy of the PLC timers in most implementations). We do not model a finer granularity of time, i.e. the global clock does not advance during a cycle in our models. In other words, our PLC cycle model has two phases: a non-deterministic delay without any computation and the real execution modelled with execution time 0.

Generic Modelling for Different Artefacts: Section II presented different use cases, involving checking of specifications and implementations in arbitrary combinations. As all combinations of artefacts might be compared, it is useful to have a common representation for both specifications and implementations. PLCspecif is a recent attempt to describe PLC components formally [1], [9]. The formal semantics of PLCspecif is defined based on automata, making it simple to be included in verification. We use PLCspecif as specification language throughout the paper, but the presented conformance relations can be used with other formal specification methods adapted to the PLC domain as well, if they can be represented as automata.

To be able to compare PLC implementations and PLCspecif specifications, we need a similar, automata-based representation for the implementations. This is provided by a tool named PLCverif [10]. The usage of PLCspecif and PLCverif together provides a simple way to represent different artefacts in compatible, comparable ways. More details about the generation of automata can be read later, in Section IV-B. From this point, we often do not distinguish between implementation and specification, as typically they are represented in the same way and can be used interchangeably. In this case, we refer to both as *artefact* and we denote them by M or M'.

Formalization of the Semantics: To be able to formally define the conformance relations, first the model and behaviour of PLC artefacts should be defined. This we do it in a generic way, covering both specifications and implementations.

Def. 1 (Model of a PLC artefact): The model of a PLC artefact is a structure $M = \langle V_I, V_O, V_L, val, \pi, s_0 \rangle$, where V_I is the set of input variables, V_O is the set of output variables,

V_L is the set of internal variables, $val\colon (V_I \cup V_O \cup V_L) \to 2^{Val}$ is the set of possible values for each variable (where *Val* is the set of all representable values, typically Boolean and bounded integer values in PLCs), π is the behaviour description (source code or specification) of the artefact with an initial state s_0.

Let us denote the input value space[2] by I, the output value space by O, the internal state space by S (similarly to [7]) and the set of potential cycle lengths in ms by $T = \{1, \dots \tau\}$ (where τ is a configurable upper limit on the cycle length, enforced by the PLC's watchdog). Then I^ω is the set of possible infinite input sequences. $\underline{i} = (i_1, i_2, \dots) \in I^\omega$ denotes an infinite input sequence, where each i_j is a vector assigning values for each input variable $v \in V_I$ of the artefact, thus can also be considered as a function $V_I \to Val$ $(\forall v, i_j : i_j(v) \in Val(v))$. The definitions of O^ω, $\underline{o}$, T^ω and $\underline{t}$ are similar. The item i_j contains the input values observed (sampled) at the beginning of cycle j, which cycle has a length of t_j and after this t_j time will provide outputs as defined in o_j. Based on these symbols, the trace semantics of a PLC artefact can be drawn up.

Def. 2 (Semantics of a PLC artefact (based on [7])): The *observable* behaviour b_M of a PLC artefact M is the function $b_M\colon I^\omega \times T^\omega \to O^\omega$, defined by π, s_0.

Notice that behaviour description π, s_0 contain more information than the observable behaviour b_M. Furthermore, b_M is a simplified view of the real behaviour for conformance checking purposes, due to the way of time modelling.

B. Definition of Conformance Relations

In this section we define in total six conformance relations ($\mathsf{pconf}_1, \dots, \mathsf{pconf}_6$) in three categories, with different levels of permissibility (reflecting N1). We recall that different output variables might need different levels of conformance: for some outputs we may require strict equivalence, while some other outputs may be allowed to be delayed. Therefore each of the conformance relations target the conformance of two corresponding output variables, not whole modules. Also, they can be separately parametrised (responding to N2).

To keep the formal definitions short, the following symbols are defined: $\underline{w} \triangleq \Pi_v(b_M(\underline{i}, \underline{t}))$ and $\underline{w}' \triangleq \Pi_{v'}(b_{M'}(\underline{i}, \underline{t}))$, where Π_v denotes projection of the behaviour of the PLC artefact to variable v, i.e. $\underline{w}$ is the sequence of values of output variable v given by M for the sequences $\underline{i}, \underline{t}$[3]. We assume that variables v and v' are corresponding to each other in M and M', respectively, thus the set of their possible values are the same $(val(v) = val(v'))$.

1) Strict Equivalence: Strict equivalence ($M \ \mathsf{pconf}_1^{v,v'}\ M'$) is the simplest and strictest conformance relation between two variables of two artefacts defined here. It is satisfied if the two artefacts, M and M', assign always the same value to the output variables v and v', if the same input sequence is provided under the same timing conditions. This relation is similar to the perfect equivalence relation of [7], with the previously discussed time extensions.

Def. 3 (pconf_1): $M \ \mathsf{pconf}_1^{v,v'}\ M' \iff \forall \underline{i} \in I^\omega, \underline{t} \in T^\omega : \underline{w} = \underline{w}'$.

2) Permissive Conformance Relations With Fixed Delay: As mentioned previously, the fixed permissive conformance relations allow delays between the corresponding outputs of the two compared artefacts. More precisely, $M \ \mathsf{pconf}_2^{v,v',n} M'$ is satisfied, if the delay of output variable v' compared to v is exactly $n \in \mathbb{Z}$ cycles. If n is positive, the v' output of M' is delayed compared to the output v of M.

For the formalization the following notation is introduced. If $\underline{w} = (w_1, w_2, w_3, \dots)$ is the output sequence of variable v, then $\underline{w}_{-1} = (*, w_1, w_2, \dots)$, i.e. $\underline{w}_{-1}$ will correspond to the output sequence of the variable v delayed by one cycle. The symbol $*$ denotes a "do not care" value. Any equality should be evaluated to true that contains a $*$ (e.g. $2 = *$ is true). More generally: $\underline{w}_{-n} = (\underbrace{*, \dots, *}_{n}, w_1, w_2, \dots)$ and $\underline{w}_{+n} = (w_{1+n}, w_{2+n}, \dots)$. Notice that $\underline{w} = \underline{w}'_{+n} \iff \underline{w}_{-n} = \underline{w}'$.

Def. 4 (pconf_2): $M \ \mathsf{pconf}_2^{v,v',n}\ M' \iff \forall \underline{i} \in I^\omega, \underline{t} \in T^\omega : \underline{w} = \underline{w}'_{-n}$.

For example, in Fig. 1b, $M \ \mathsf{pconf}_2^{out,out',1}\ M'$ is satisfied, but $M \ \mathsf{pconf}_2^{out,out'',1}\ M'$ is not.

The relation $\mathsf{pconf}_3^{v,v',K}$ generalizes $\mathsf{pconf}_2^{v,v',n}$. The parameter K of $\mathsf{pconf}_3^{v,v',K}$ is a set $K \in 2^{\mathbb{Z}}$ instead of a single number. $M \ \mathsf{pconf}_3^{v,v',K} M'$ iff $M \ \mathsf{pconf}_2^{v,v',n} M'$ holds for an $n \in K$. Using these notations, the formal definition of this conformance relation is the following.

Def. 5 (pconf_3): $M \ \mathsf{pconf}_3^{v,v',K}\ M' \iff \exists n \in K : M \ \mathsf{pconf}_2^{v,v',n}\ M'$.

3) Permissive Conformance Relations With Variable Delay: Certain modifications in the specification or the implementation cause shifts only in certain cycles of the execution, not consistently (see Fig. 1b for example). If these are permitted differences, the previously defined relations are still too strict. Therefore we introduce a relation allowing variable shifts in the output sequences.

The key idea of $\mathsf{pconf}_4^{v,v',K}$ is the following. Given the two output sequences $\underline{w} = (w_1, w_2, \dots)$ and $\underline{w}' = (w'_1, w'_2, \dots)$, and a set of allowed delays $K = \{k_1, k_2, \dots\}$, the two sequences are considered as conformant if for each w_i there is a $w'_j = w_i$ in i's K-surrounding, i.e. $w_i = w'_{i+k_1} \vee w_i = w'_{i+k_2} \vee \dots$; and for each w'_i there is a $w_j = w'_i$ in i's $-K$-surrounding (where $-K$ is the elementwise negation of K).

In the formal definition we use $\varphi(\underline{w}, \underline{w}', K)$ which stands for "for each w_i there is a $w'_j = w_i$ in i's K-surrounding". Formally (for infinite vectors):

$$\varphi(\underline{w}, \underline{w}', K) \iff \forall i = 1, \dots : \exists s \in K : 1 \le i+s \wedge w_i = w'_{i+s}$$

An illustration of φ can be seen in Fig. 2. It shows that for the given example vectors $\underline{u}, \underline{u}'$ the $\varphi(\underline{u}, \underline{u}', \{-1, 0, 1, 2\})$ is not satisfied because there is no match for $u_6 = 1$ in $\underline{u}'$. However, $\varphi(\underline{u}', \underline{u}, \{-2, -1, 0, 1\})$ is satisfied for the same vectors, as there is a match for each u'_i in $\underline{u}$ (see the arrows

[2]Each element of I assigns a value for each input variable of the artefact.

[3]Notice that $\underline{w}$ is a function of $\underline{i}, \underline{t}, v, M$ and $\underline{w}'$ is a function of $\underline{i}, \underline{t}, v', M'$. However, we omit to denote this in the following to keep the definitions compact and readable.

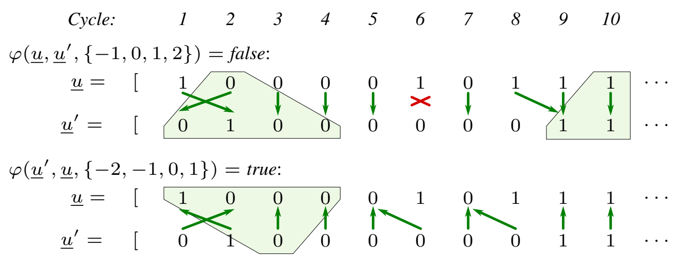

Cycle: 1 2 3 4 5 6 7 8 9 10
$\varphi(\underline{u}, \underline{u}', \{-1, 0, 1, 2\}) = false$:
$\underline{u} =$ [1 0 0 0 0 1 0 1 1 1 ...
$\underline{u}' =$ [0 1 0 0 0 0 0 0 1 1 ...
$\varphi(\underline{u}', \underline{u}, \{-2, -1, 0, 1\}) = true$:
$\underline{u} =$ [1 0 0 0 0 1 0 1 1 1 ...
$\underline{u}' =$ [0 1 0 0 0 0 0 0 1 1 ...

Fig. 2: Example for relation φ

in Fig. 2). The shaded area illustrates the range in which the corresponding value is searched, e.g. in the first example, u_2 should equal to any of $u'_1, \ldots, u'_4$ to satisfy the relation φ.

Def. 6 (pconf_4*):* $M \ \mathsf{pconf}_4^{v,v',K} \ M' \iff \forall \underline{i} \in I^\omega, \underline{t} \in T^\omega : \varphi(\underline{w}, \underline{w}', K) \wedge \varphi(\underline{w}', \underline{w}, -K)$.

For example, in Fig. 1b, $M \ \mathsf{pconf}_4^{out,out',\{-1,0,1\}} \ M'$ and $M \ \mathsf{pconf}_4^{out,out'',\{-1,0,1\}} \ M'$ are both satisfied.

Following the ideas described in Section II, we define two additional restrictions to $\mathsf{pconf}_4^{v,v',K}$. The relation $\mathsf{pconf}_5^{v,v',K}$ requires *total pulse duration preservation* in addition to $\mathsf{pconf}_4^{v,v',K}$, thus for each allowed value of v their number of occurrences should be the same for the outputs of the two artefacts for every possible execution. The number of occurrences of value e in vector $\underline{w}$ is denoted by $\underline{w}\#e$ in the formal definition.

Def. 7 (pconf_5*):* $M \ \mathsf{pconf}_5^{v,v',K} \ M' \Leftrightarrow M \ \mathsf{pconf}_4^{v,v',K} \ M' \wedge \forall \underline{i} \in I^\omega, \underline{t} \in T^\omega : \forall e \in val(v) : \underline{w}\#e = \underline{w}'\#e$.

For example, $M \ \mathsf{pconf}_5^{out,out',\{-1,0,1\}} \ M'$ is satisfied in Fig. 1c, but $M \ \mathsf{pconf}_5^{out,out',\{-1,0,1\}} \ M'$ is not satisfied in Fig. 1d, as the total pulse duration for out' is shorter than for out.

The relation $\mathsf{pconf}_6^{v,v',K}$ reflects the common usage of edge-driven signals, thus it is only defined for Boolean variables. The additional restriction of $\mathsf{pconf}_6^{v,v',K}$ compared to $\mathsf{pconf}_4^{v,v',K}$ is that the *number of rising edges* should be the same in the two outputs for every possible execution. To formalise this, we introduce the rising edge vector $\uparrow(\underline{w}) = (e_1, e_2, \ldots)$, where e_i is true iff $i > 1$, $w_{i-1} = false$ and $w_i = true$.

Def. 8 (pconf_6*):* $M \ \mathsf{pconf}_6^{v,v',K} \ M' \Leftrightarrow M \ \mathsf{pconf}_4^{v,v',K} \ M' \wedge \forall \underline{i} \in I^\omega, \underline{t} \in T^\omega : \ \uparrow(\underline{w})\#true = \ \uparrow(\underline{w}')\#true$.

For example, in Fig. 1c, $M \ \mathsf{pconf}_5^{out,out',\{-1,0,1\}} \ M'$ is not satisfied, as out' has fewer rising edges than out. However, in Fig. 1d, $M \ \mathsf{pconf}_5^{out,out',\{-1,0,1\}} \ M'$ is satisfied.

Notice that we focus on checking whether the two compared artefacts are conformant given the same inputs and cycle lengths. Consequently, we cannot show differences between two codes implementing the same specification with different complexity, thus with different execution times. A stricter relation could show these differences, but it could even mark any code non-conformant with itself, as on different hardware the execution time is different. We use a higher abstraction level, not taking the internal interrupts and hardware differences into account. In practice, the developers do that too: if some deterministic, physical time conditions have to be satisfied, PLC timers are used, or a fixed cycle time can be assumed in a fail-safe PLC. On the other hand, the relations introduced here can show the differences when the two artefacts give different outputs for the same inputs (wrong functionality) or when the delay between the two corresponding outputs (in number of cycles) is out of the given acceptable range.

IV. Checking the PLC Conformance Relations

Obviously, to make the conformance checking useful in practice it is not enough to define the conformance relations. A method has to be established to check if two artefacts are conformant or not. Section IV-A overviews the proposed conformance checking approach. It consists of generating models (detailed in Section IV-B) and temporal logic (TL) expressions (detailed in Section IV-C), then evaluating the conformance using a model checker.

A. *Overview of the Approach*

We recall the primary needs affecting the implementation of conformance checking: the method should be generic to support different use cases, thus different artefacts (N4) and it should be able to scale up to large input artefacts (N5).

PLCverif [10] already provides a solution for efficient model checking – using requirement-specific reductions – of PLC programs (mainly for Structured Text programs) through its intermediate model format – similar to the automata models generated from PLCspecif specifications. We have decided to reuse and to build on these solutions, thus to use model checking for conformance checking. An advantage of reusing the PLCverif intermediate models is that we can benefit from its built-in model reductions [10]. Also, the model (and the TL expression) can be translated to the concrete syntax of various model checkers by PLCverif, thus we can use different model checkers for conformance checking.

To reuse the workflow of PLCverif, the following steps have to be executed:

- Generating (compatible) verification models representing the two artefacts to be compared,
- Building composite verification models for the conformance checking case,
- Generating temporal logic expressions representing the conformance criteria (i.e. the conformance relation is valid between the two models if the temporal logic expressions are satisfied on the composite verification model), and
- Performing model checking using the composite verification models and the generated temporal logic expressions.

The first and last steps are already supported by PLCspecif and PLCverif as described in [8], [9]. The following two sections discuss the remaining two steps of the list above.

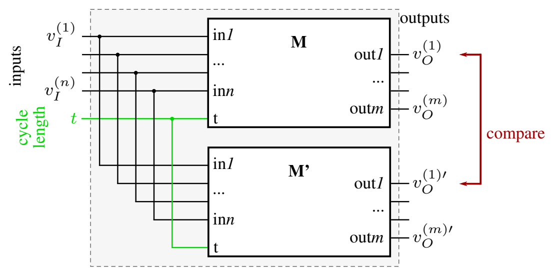
B. *Model Generation*

As mentioned before, the applied toolchain can produce models separately for the compared artefacts M and M'. The additional task is to generate a *composite verification model*

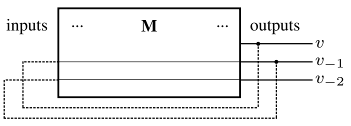
$\Gamma_{M,M'}$ on which the evaluation of a TL expression can decide the satisfaction of the selected conformance relations.

First trials showed that in case of certain model checkers (e.g. nuXmv [11]) the performance may significantly drop when complex temporal logic expressions are checked. Therefore the model $\Gamma_{M,M'}$ is constructed in a way that it contains some parts of the conformance criteria directly, as described in the following.

- The corresponding input variables (identified manually or using heuristics) are merged: one of them is deleted and the other is used in both M and M'[4] (see Fig. 3a).
- The cycle time representation (sequence of cycle times) is merged in a similar way (M and M' will have the same, non-deterministic PLC cycle length, given by $\underline{t}$).
- If the TL expression refers to delayed variables, i.e. it uses not only the current, but a previous cycle's value of a certain variable (cf. Section IV-C), the delayed variables are also included as new output variables in $\Gamma_{M,M'}$, as illustrated in Fig. 3b. v_{-1} represents a variable whose value equals to the value of v in the previous cycle.
- For $\mathsf{pconf}_5^{v,v',K}$ and $\mathsf{pconf}_6^{v,v',K}$ some other helper variables (P^v, Q^v) are defined (see later, in Section IV-C).



(a) Structure of the composite verification model $\Gamma_{M,M'}$



(b) Delayed variables

Fig. 3: Modelling methods

C. Temporal Logic Expression Generation

The next task is to represent the required conformance relations in TL according to the definitions of the relations. As $\Gamma_{M,M'}$ is constructed in a way that the two compared artefacts have the same inputs and timing conditions (that is included in every conformance relation), the TL expressions need to express only the comparison of outputs.

Here we overview briefly the ideas behind each of the six relations' TL representation. The formal definitions can be found in Table I.

- The representation of the strict equivalence relation ($\mathsf{pconf}_1^{v,v'}$) is trivial, only the values of the two corresponding output variables (v, v') have to be compared.

[4]This implies the assumption that variables representing physical inputs are not modified by the user program.

- To check $\mathsf{pconf}_2^{v,v',n}$, the shifted values should be taken into account too, thus additional output variables should be generated during the construction of $\Gamma_{M,M'}$, as discussed in Section IV-B. It has to be noted that the outputs can only be delayed, the future values (e.g. v_{+1}) should not be referred in the TL expressions.
- To check $\mathsf{pconf}_3^{v,v',K}$ it is enough to check if $\mathsf{pconf}_2^{v,v',n}$ holds for at least one $n \in K$.
- The main idea of checking $\mathsf{pconf}_4^{v,v',K}$ is already introduced by defining the function φ in Section III-B. However, this could cause looking ahead in time (see Fig. 2). To avoid this and the complex TL expressions imposed by looking ahead, we determine the largest look-aheads (denoted by $\mu(K)$ and $\mu(-K)$) and we shift all comparisons towards the past by these amounts of cycles.
- Checking relation $\mathsf{pconf}_5^{v,v',K}$ requires additionally to check the different values of the output variables. For Boolean variables we automatically introduce in the model generation phase a variable P^v in the verification model $\Gamma_{M,M'}$ that is incremented by one if v is true at the end of a cycle and decremented by one if v' is true at the end of the cycle. Therefore $P^v = 0$ means that v and v' were true for the same amount of cycles. The additional requirement is expressed as P^v has to go back to zero always in the future. We do not define checking of this relation for non-Boolean variables in this paper, but it can be achieved easily with a simple extension of the verification model. Let us add a new variable S^v to $\Gamma_{M,M'}$ with the same type as v. The value of S^v should be non-deterministic and this value should be kept during an execution. In this way the previously introduced P^v can count the differences in how many times v and v' had the value S^v. Practically, this is a universal quantification of the expression to be checked without the need of more expressive language than CTL.
- Checking relation $\mathsf{pconf}_6^{v,v',K}$ requires additionally to check the number of rising edges on the Boolean outputs v and v'. We introduce a variable Q^v in the model $\Gamma_{M,M'}$ that is incremented by one if v has a rising edge at the end of a cycle and decremented by one if v' has a rising edge at the end of the cycle. Therefore $Q^v = 0$ means that v and v' had the same number of rising edges in the preceding cycles. The additional requirement is expressed as "Q^v has to go back to zero always in the future".

Verification Model Semantics: The CTL expressions in Table I were defined assuming that one transition in the verification model represents one PLC cycle, e.g. $\mathsf{AG}(v = v')$ means that v and v' equal to each other *at the end of all PLC cycles*. The semantics of the intermediate model of PLCverif is defined differently: one transition in $\Gamma_{M,M'}$ represents only one computation. However, the states representing ends of PLC cycles are labelled by the label EoC (end of cycle). Based on that the CTL expressions can be systematically transformed to have the same meaning on the verification model as before. These transformations can be seen in Table II. (The idea can

TABLE I: CTL Representations of the pconf Relations

M pconf $M' \iff$	Satisfaction of the CTL representation
1. $\mathsf{pconf}_1^{v,v'}$	$\Gamma_{M,M'} \vDash \mathsf{AG}(v = v')$
2. $\mathsf{pconf}_2^{v,v',n}$	$\begin{cases} \Gamma_{M,M'} \vDash \mathsf{AG}(v = v'_{-n}) & \text{if } n \geq 0 \\ \Gamma_{M,M'} \vDash \mathsf{AG}(v_n = v') & \text{if } n < 0 \end{cases}$
3. $\mathsf{pconf}_3^{v,v',\{k_1,k_2,\dots\}}$	$\bigvee_{i \in \{k_1,k_2,\dots\}} M \ \mathsf{pconf}_2^{v,i} \ M'$
4. $\mathsf{pconf}_4^{v,v',K}$	$\Gamma_{M,M'} \vDash \mathsf{AG}($ $(\bigvee_{i \in K} v_{-\mu(K)} = v'_{i-\mu(K)}) \wedge$ $(\bigvee_{i \in K} v_{-i-\mu(-K)} = v'_{-\mu(-K)}))$ where $\mu(K) \triangleq \max(K \cup \{0\})$
5. $\mathsf{pconf}_5^{v,v',K}$	$M \ \mathsf{pconf}_4^{v,v',K} \ M' \wedge \mathsf{AG}\ \mathsf{AF}(P^v = 0)$
6. $\mathsf{pconf}_6^{v,v',K}$	$M \ \mathsf{pconf}_4^{v,v',K} \ M' \wedge \mathsf{AG}\ \mathsf{AF}(Q^v = 0)$

TABLE II: Translation of CTL Temporal Operators

Expression on PLC model	**Expression on verification model**
(1 transition = 1 cycle)	*(1 transition = 1 computation)*
$\mathsf{AF}(\alpha)$	$\mathsf{AF}(EoC \wedge \alpha)$
$\mathsf{AG}(\alpha)$	$\mathsf{AG}(EoC \rightarrow \alpha)$

be extended to all CTL and LTL temporal operators.)

V. Validation of the Approach

In this section, we briefly overview two applications of the relations for evaluation purposes. The first case is included for the initial validation of scalability, the second is to show the practical benefits of the permissive relations.

Scalability Validation: To justify that the proposed method can scale up to large artefacts, we describe the verification of the safety logic of a superconducting magnet test controller of CERN. This controller is responsible for allowing or forbidding certain magnet tests based on various signals coming from the environment or other subsystems. Even without the non-safety parts (e.g. communication, supervision), the controller is significantly complex: the potential state space of the formal model generated from the implementation of the safety logic has about 3×10^{979} states (the size of the state vector is approx. 3250 bits, the size of the reachable state space is not known) before reductions. During the development of this safety logic we have created the formal specification of the system based on the client's requirements. The model generated from the PLCspecif formal specification is significantly smaller, still its *reachable* state space contains 10^{140} states (before requirement-specific reductions).

PLCverif was able to efficiently reduce the *composite* verification model generated for the current conformance requirement: the size of reachable state space of the composite model was 10^{139} (after requirement-specific reductions). The model was then transformed to the concrete syntax of nuXmv [11]. The CTL expression generated from the conformance checking problem (following Table I) was evaluated[5] in 538 s. As the checked system was a safety logic without stateful behaviour, mostly strict conformance relations were used. This analysis revealed a problem in the safety system, although it has been previously verified using model checking. The detected problem was not covered by the pre-defined correctness criteria of model checking, showing that conformance checking with formal specification can provide an easy-to-use, efficient verification method. After fixing several errors, the conformance between the final implementation and the specification was proven in 482 s. More details can be found in [12] about this case study.

Validation of the Permissive Relations: In this second case we have used an object from the UNICOS framework's object library, widely used at CERN for PLC software. We have introduced a small modification in the code causing a one-cycle-long delay in two of the outputs used for informational purposes, i.e. a delay that does not cause any problems. By using strict equivalence checking, the two code versions differed, a difference was found in less than a second[6].

Without permissive conformance relations the conformance checking would get stuck: having and analysing only a single counterexample, it is not known whether any other important difference exists, i.e. if the modification caused any relevant side-effect. We have repeated the conformance checking, but taking into account that various conformance relations can be used for the different outputs. First, we have identified the level of required conformance between each corresponding output pairs. We have selected a permissive conformance relation ($\mathsf{pconf}_4^{v,v',\{-1,0,1\}}$) for the two outputs where the strict equivalence is not required. After this, the conformance checking demonstrated in less than a second that the two code versions are conformant, there is no inequivalence besides the permissible delay of the above-mentioned variables. This example shows the advantage of the permissive conformance relations: using them the developer can be sure that his modifications did not affect any other behaviours, and also it did not cause intolerable changes in the "informative" outputs.

The reachable state space of each PLC program model was around 10^8 states (before requirement-specific reductions), while the composite model contained only 7×10^4 reachable states (after requirement-specific reductions). The runtime of the model checker was 0.6 s for the conformance checking using both strict and permissive relations.

VI. Related Work

Equivalence and conformance relations have a long history. Already in 1981, Back summarised and described various refinement and equivalence relations in [3]. Tretmans described the ioco relation [5] a decade later to check the conformance between a specification and an implementation. [13], [14] are recent works introducing newer relations responding to various needs. However, they all mainly target reactive systems, where the level of required conformance is typically high. In cyclic transformational systems, such as PLC-based control systems this leads to numerous discrepancies considered to be "false positive", not relevant from the practical point of view.

[5]For all measurements we have used PLCverif 2.0.2 and nuXmv 1.0.1 on Windows 7 x64, executed on a PC with Intel® Core™ i7-3770 3.4 GHz CPU and 8 GB RAM.

[6]This validation example is published, refer to DOI 10.5281/zenodo.45415.

Sülflow and Drechsler [15] applied strict, non-timed equivalence checking based on a SAT solver to verify PLC programs against their specification. Provost et al. [16] check the strict conformance between a specification (given as a Mealy machine [17]) and an implementation by generating test sequences. Besides these work, applying equivalence and conformance relations specifically to PLC-based systems was not targeted until the very recent work of Weigl, Beckert, Ulewicz et al. [7], [18], [19]. Their goal is to perform regression verification on PLC programs, i.e. to check the conformance of two different PLC codes. For this purpose two relations are defined: the *perfect equivalence*, when the two PLC programs should give the same output sequence if the same input sequence was given; and the *conditional equivalence*, which relaxes the requirements of perfect equivalence: the two codes should produce the same output only if a certain condition is met (practically the two implementation can give different output for impossible input scenarios).

One of the goals of their work is to complement testing by defining and checking sensitive conformance relations. For example, a one-cycle-long delay in the output might mean a delay of 1–100 ms on the output which needs high precision measurements to be checked using testing. However, in many cases such a small delay does not affect the controlled plant, therefore in some cases the two compared artefacts can be considered as conformant to each other.

The main differences between our current work and the work in [7], [18], [19] are the following:

1) The conformance relations defined in the current work are more permissive (responding to N1),
2) The time assumptions are more relaxed here (the fix cycle time assumption is not necessary, responding to N3),
3) More use-cases are considered in the current work (not only code-code conformance responding to N4), and
4) The model generation method of our work is more generic, based on an intermediate model, supporting model reductions and the usage of various model checkers without exponential complexity in model generation (responding to N5).

VII. Conclusion and Future Work

In this work we have identified the needs for conformance relations in case of PLC-based industrial control systems in order to make conformance checking useful in the development process. Different use cases and motivating scenarios were identified and, based on them, six conformance relations were drawn up with different permissibility. The conformance relations are generic in the sense that they can be used both on specification and implementation, not only between two implementations.

These conformance relations can be checked as model checking problems. This is made possible by defining a model and generating temporal logic criteria for checking the conformance relations. The implementation builds on PLCverif, providing an efficient and generic approach.

Future work includes extensive validation, mainly on the control systems of CERN. These validation works may lead to new conformance relations or alternations of the currently defined ones if the current ones do not cover all the needs. Various extensions are possible, e.g. the conformance relations can take the invariant properties defined in the PLCspecif formal specification into account. Another possible extension is to make the relations more permissive in the value dimension, e.g. two values can be considered as equal if the difference between them is less than a certain constant or percentage; or to add relations that are stricter about the physical timing of the output signals.

References

[1] D. Darvas, E. Blanco Viñuela, and I. Majzik, "A formal specification method for PLC-based applications," in *Proc. of the 15th Int. Conf. on Accelerator & Large Experimental Physics Control Systems*. JACoW, 2015, pp. 907–910.

[2] G. Godena, T. Lukman, and G. Kandare, "A new approach to control systems software development," in *Case studies in control – Putting theory to work*. Springer, 2013, pp. 363–406.

[3] R.-J. R. Back, "On correct refinement of programs," *Journal of Computer and System Sciences*, vol. 23, no. 1, pp. 49–68, 1981.

[4] D. Park, "Concurrency and automata on infinite sequences," in *Theoretical Computer Science*, ser. LNCS. Springer, 1981, vol. 104, pp. 167–183.

[5] J. Tretmans, "Test generation with inputs, outputs and repetitive quiescence," *Software-Concepts and Tools*, vol. 17, no. 3, pp. 103–120, 1996.

[6] D. Harel and A. Pnueli, "On the development of reactive systems," in *Logics and Models of Concurrent Syst.* Springer, 1985, pp. 477–498.

[7] B. Beckert, M. Ulbrich, B. Vogel-Heuser, and A. Weigl, "Regression verification for programmable logic controller software," in *Formal Methods and Software Engineering*, ser. LNCS. Springer, 2015, vol. 9407, pp. 234–251.

[8] B. Fernández Adiego *et al.*, "Modelling and formal verification of timing aspects in large PLC programs," in *Proc. of the 19th IFAC World Congress*. IFAC, 2014, pp. 3333–3339.

[9] D. Darvas, E. Blanco Viñuela, and I. Majzik, "Syntax and semantics of PLCspecif," CERN, Report EDMS 1523877, 2015. [Online]. Available: http://edms.cern.ch/document/1523877

[10] D. Darvas *et al.*, "Formal verification of complex properties on PLC programs," in *Formal Techniques for Distributed Objects, Components, and Systems*, ser. LNCS. Springer, 2014, vol. 8461, pp. 284–299.

[11] R. Cavada *et al.*, "The nuXmv symbolic model checker," in *Computer Aided Verification*, ser. LNCS. Springer, 2014, vol. 8559, pp. 334–342.

[12] D. Darvas, I. Majzik, and E. Blanco Viñuela, "Formal verification of safety PLC based control software," in *Integrated Formal Methods*, ser. LNCS. Springer, 2016, vol. 9681.

[13] T. Lambolais, A.-L. Courbis, H.-V. Luong, and T.-L. Phan, "Interoperability analysis of systems," in *Proc. of the 18th IFAC World Congress*. Elsevier, 2011, pp. 7879–7884.

[14] T.-L. Phan, "Modeling and verification techniques for incremental development of UML architectures," in *Doctoral Symposium of the European Conf. on Object-Oriented Programming*, 2013.

[15] A. Sülflow and R. Drechsler, "Verification of PLC programs using formal proof techniques," in *Formal Methods for Automation and Safety in Railway and Automotive Systems*. L'Harmattan, 2008, pp. 43–50.

[16] J. Provost, J.-M. Roussel, and J.-M. Faure, "Generation of single input change test sequences for conformance test of programmable logic controllers," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 3, pp. 1696–1704, 2014.

[17] ——, "Translating Grafcet specifications into Mealy machines for conformance test purposes," *Control Engineering Practice*, vol. 19, no. 9, pp. 947–957, 2011.

[18] S. Ulewicz, B. Vogel-Heuser, M. Ulbrich, A. Weigl, and B. Beckert, "Proving equivalence between control software variants for programmable logic controllers," in *Proc. of the 20th IEEE Int. Conf. on Emerging Technologies and Factory Automation*. IEEE, 2015.

[19] A. Weigl, "Regression verification for programmable logic controller software," Master's thesis, Karlsruhe Institute of Technology, 2015.