Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Modelling and Formal Verification of Timing Aspects in Large PLC Programs

Borja Fernández Adiego [*] **Dániel Darvas** [*]
Enrique Blanco Viñuela [*] **Jean-Charles Tournier** [*]
Víctor M. González Suárez [**] **Jan Olaf Blech** [***]

[*] *CERN, European Organization for Nuclear Research, CH-1211 Geneva 23, Switzerland (e-mail: {borja.fernandez.adiego, daniel.darvas, enrique.blanco, jean-charles.tournier}@cern.ch).*
[**] *ISA, University of Oviedo, Campus de Viesques 33204 - Gijón, Spain (e-mail: victor@isa.uniovi.es)*
[***] *RMIT University, Melbourne, Australia (e-mail: janolaf.blech@rmit.edu.au)*

Abstract: One of the main obstacle that prevents model checking from being widely used in industrial control systems is the complexity of building formal models out of PLC programs, especially when timing aspects need to be integrated. This paper brings an answer to this obstacle by proposing a methodology to model and verify timing aspects of PLC programs. Two approaches are proposed to allow the users to balance the trade-off between the complexity of the model, i.e. its number of states, and the set of specifications possible to be verified. A tool supporting the methodology which allows to produce models for different model checkers directly from PLC programs has been developed. Verification of timing aspects for real-life PLC programs are presented in this paper using NuSMV.

Keywords: PLC, timers, formal verification, model checking, automata, abstraction

1. INTRODUCTION

CERN, the European Organization for Nuclear Research, relies on a large number of PLC (Programmable Logic Controller) applications to operate its different particle accelerators. These applications are critical to CERN operation, thus guaranteeing that their behaviour conform to their specifications is of highest importance. Formal verification, and especially model checking, appears to be a promising technique to ensure that PLC programs meet their initial specifications. However, this technique is not widely used in industry due to the complexity of building the formal model of a PLC program: building such formal model requires an in-depth knowledge of the system to model (hardware and software) as well as the underlying model checker. Moreover, when timing aspect needs to be taken into account, i.e. PLC time and timers, the modelization task becomes even more complex as the resulting models, without a refined representation, are usually too large in terms of state space to be handled by model checkers.

In this paper, we are proposing a methodology to model PLC time and timers. The methodology is integrated into the generic framework described in Darvas et al. (2013) allowing to generate formal models automatically out of PLC programs. Two approaches are proposed in order to take timing aspects into consideration: *realistic* and *abstract* modelization. The realistic approach represents the behaviour of timers and the internal representation of time in PLCs with high fidelity. Such modelling allows to verify time-related properties to ensure that a given action will (or will not) be performed after or before a given delay (e.g. *PLC output set to true 500 ms after a given input has been set to true*). While this modelization is powerful in terms of expressivity, it may produce models that are too big to be handled by model checkers and thus, leads to the second modelling approach. The abstract approach omits the modelization of time itself and gives a non-deterministic model of timers. Compared to the first approach, this one drastically reduces the state space of the generated model and therefore allows to verify large PLC programs while still providing the ability to verify some time-related specifications. The properties that can be verified by applying this second modelization are for example liveness properties (e.g. *PLC output will be set to true after its input is set to false*). The requirements verified using the abstract time modelization remain valid on the realistic model, as the realistic approach is a refinement of the abstract one. Finally, a tool implementing the two types of time modelization and generating formal models for NuSMV (Cimatti et al. (2002)), BIP (Basu et al. (2011)) and UPPAAL (Amnell et al. (2001)) has been developed and applied to CERN's control systems.

1.1 Related Work

Although modelling timing behaviour of PLC programs has previously been studied in the literature, none of them provides a general methodology which allows automatically generating formal models including timing aspects, and performing verification on these models at the same time. Moreover, all approaches found in the literature are bounded to a specific model checker and thus prevent to take advantages of the different types of model checkers.

Indeed, Mader and Wupper (1999) or Perin and Faure (2013) proposes an approach for modelling PLC timers using timed automaton models, but does not present verification results. As time is considered as a linear and monotonic function, the generated models would have a huge state space, making verification impossible if this approach would be applied to large systems, as the systems developed at CERN. Similarly, Mokadem et al. (2010) presents a case study where a global model for a timed multitask PLC program is created for verification purposes. This approach is similar to the one proposed by Mader and Wupper (1999) but verification is performed with UPPAAL using clocks and therefore with monotonic time representation. In Wang et al. (2013), several aspects of PLC control systems including timers are modelled, using the component-based BIP framework. In this case, they assume fixed PLC cycle length which is a big constrain, and timer models are not precise enough compare to real PLC timers. In addition, verification results are not presented.

The rest of the paper is structured as follows: Section 2 introduces the notion of time and timers in PLCs. Section 3 gives an overview of the proposed methodology which allows to generate formal models for various model checkers out of PLC programs. Section 4 presents in detail the two proposed ways to model timing aspects of PLC programs, along with a case study applying the modelization. In addition, this section demonstrates formally that a realistic time modelization is a refinement of its abstraction. Finally, Section 5 analyses the two approaches by highlighting their advantages and disadvantages, and concludes the paper.

2. TIMED PLC CONTROL SYSTEMS

This section gives an overview of PLC control systems, with a focus on timing aspects. In addition, a case study is presented which will be used throughout the remaining sections to illustrate the modelization approach proposed in this paper.

2.1 Timing behaviour of PLCs

A PLC is an industrial computer which performs a synchronous and cyclic process called *scan cycle*, consisting of the following main steps: *(1)* reading the input values to the memory, *(2)* interpreting and executing the program logic using the read data, and finally *(3)* writing the computed output values to the real outputs.

In standard PLCs, i.e. non-safety PLCs, the cycle time is not fixed, but there is an upper limit enforced by a watchdog module. If the PLC cycle time is bigger than this upper limit, e.g. due to an infinite loop in the PLC program, the PLC executes a special part of the program responsible for handling timing errors. By contrast, safety PLCs have a fixed cycle time.

Timing operations, such as timers, are defined by IEC 61131 and can be considered as a function block that delays a signal or produces a pulse. Different types of timers can be found in PLCs, one of the most common timers is TON (Timer On-delay) (see Fig. 1). This timer has 2 inputs variables: *IN* and *PT*. *IN* is a Boolean input signal and *PT* is the delay time. The timer has 2 outputs: *Q* and *ET*. *Q* is the Boolean output variable, its value will be true after the predefined delay (*PT*) when *IN* performs a rising edge, and it will be false if *IN* is false. *ET* is the elapsed time, its value is increased until *PT*, starting when a rising edge occurred on *IN*.

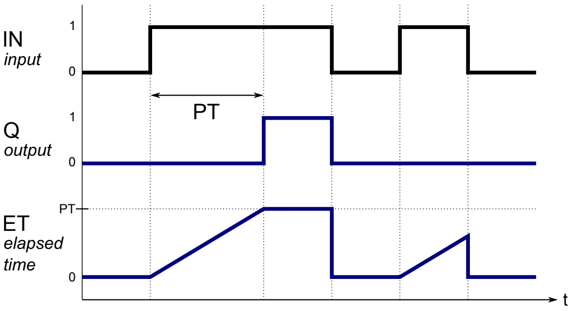


Fig. 1. TON time diagram

PLC timers use a specific data type for timing operations called *TIME*. This data type is defined by IEC 61131 as a finite variable which states that "*The range of values and precision of representation in these data types is implementation-dependent*". Representing time by a finite variable leads to a non-monotonic time representation as the variable can overflow (c.f. the upper part of Fig. 3). For example, in Siemens S7 PLCs, the *TIME* data type is defined as a signed 32-bit integer with the same precision of 1 ms (see Siemens (1998)), having an upper limit of approximately +24 days and a lower limit of −24 days. However, in Schneider and Beckhoff PLCs, the *TIME* data type is an unsigned 32-bit integer with a precision of 1 ms. In this paper, we consider the signed time interpretation as defined in Siemens PLCs.

2.2 Case study

In the context of this paper, the industrial control system framework, developed and used by CERN, called UNICOS (Blanco et al. (2011)) is taken as a case study. UNICOS provides a library of base objects representing common industrial control instrumentation (e.g. sensor, actuators, subsystems). These objects are represented as function blocks in the PLC code, using the ST (Structured Text) language, which can call different functions, function blocks on the PLC. Currently UNICOS is implemented for standard PLCs, i.e. in which the cycle time is not fixed and depends on the overall application.

In this paper, we focus on the OnOff object provided by the UNICOS library for Siemens PLCs. This object is used to represent a physical equipment, as actuators driven by digital signals (e.g. valves, heaters, motors). With 60 input variables (of which 13 are parameters), 62 output variables, 600 lines of ST code, and 3 timer instances, the OnOff object is representative of other UNICOS objects in terms of size and complexity.

3. MODELLING PLC PROGRAMS

This section gives a brief overview of the general methodology for modelling PLC programs without considering timing aspects. For more details, readers should refer to Darvas et al. (2013). Also, the applied reductions and abstractions are introduced.

3.1 Methodology

This methodology aims to generate formal models out of ST code. It provides an intermediate model and a set of transformation rules to produce the input models for different verification tools from the "non-formal world" of control systems (see Fig. 2). This "non-formal world" is basically composed by the ST program and environmental factors related to the execution platform (for example the PLC scan cycle).

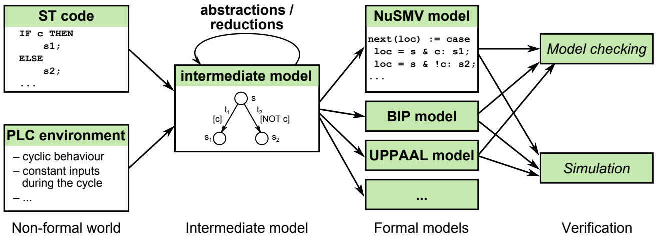


Fig. 2. Overview of generic methodology producing formal models out of ST code.

This intermediate model based on automata simplifies the transformation rules to any formal model used by a model checker, whose modelling language is close to an automata-based formalism. It allows to easily add new model checkers to the methodology and verify the same model with different verification tools, thus benefiting from the different advantages of the verification tools in terms of verification performance, simulation and expressiveness of properties specification.

Our automata-based formalism is strong enough to model all relevant features of a PLC control system. The semantics of the intermediate model is similar to the network of timed automata formalism defined in Behrmann et al. (2004) but without explicit logical clock representation. In addition, an automatic generation tool has been developed supporting this methodology, which allows to generate models for NuSMV, UPPAAL and BIP out of ST code.

3.2 Reduction and abstraction techniques

Modelling PLC programs for verification purposes implies the creation of models with huge state spaces, as it happens for any other real-life software. If these are timed systems, then modelling of time is required, so the problem becomes even bigger. Although the main goal of this paper is not to present the abstraction techniques applied on the generated models in details, we introduce them as they are needed to understand our experimental results on timed PLC programs. These techniques are included in the methodology and applied automatically to the intermediate model, therefore all the generated final models benefit from them. The main abstraction and reduction techniques applied:

- *General rule-based reductions* are used to simplify the generated automata by taking advantage of the characteristics of the PLC's execution model. For example, variable assignments on succeeding transitions can be merged if they are not affecting each other, as the value of the variables are not checked during the execution of the PLC cycle but only at the end. More than 20 rules have been defined to simplify and reduce the model.
- When the property to verify is provided, we apply the *Cone of Influence* (COI) reduction (c.f. Clarke et al. (1999)), which consists in removing all variables that do not affect the property. As it is depending on the requirement to be checked, its impact on the size of the state space highly varies, but for many examples, we can observe a large state space reduction.
- In some cases, predicate abstraction can be also applied to PLC applications. It consists in substituting non-Boolean variables by Boolean-valued functions.

The reader find detailed information about the applied techniques in Darvas et al. (2014).

4. MODELLING TIMING ASPECTS OF PLC PROGRAMS

This section focuses on the modelling aspects of PLC time and timers extending the described methodology in Section 3. The TON timer, presented in Section 2, is used to illustrate the two proposed approaches to model timing aspects, but the same methodology can be applied to any other PLC timer block. In the proposed methodology extension, a PLC timer is represented by a separated automaton which is synchronized with the main program. Modelling PLC timers also implies to model the *TIME* data type. However, as mentioned previously, timed models usually contain a huge state space, which cannot be handled by model checkers. Based on this observation, two different approaches are proposed in this section. The first is a realistic timer representation, which is close to the reality, enabling precise modelling and verification. The second approach proposes an abstract time representation, which is less accurate, but drastically reduces the state space of the model.

Both approaches are analyzed hereafter paying attention to time and timer representation, property specification, and experimental results. In addition, we are presenting the proof that the abstract approach simulates the realistic one, thus guaranteeing that any property verified by the abstract approach also holds in the realistic one, even if is much simpler.

4.1 Realistic approach

This first approach presents a realistic model of a TON timer and the time handling. This approach allows to specify properties with explicit time in it. Having this realistic representation of the timer implies that *time* needs also to be modelled realistically.

Time representation Three main characteristics are considered to model time for this approach:

(1) **Time is modelled as a finite variable**: it represents with high fidelity the *TIME* data type in a PLC. However, instead of having a signed 32-bits integer variable (like in Siemens PLCs), a 16-bits variable is used to represent this data type. This range reduction is possible as the behaviour of a PLC is cyclic and the cycle time, the delays of the timers, and the delays in the requirements are much smaller than the range of this 16 bit variable. The accuracy of this variable is 1 ms, as it is in real PLCs. Because of this time representation, overflow of time has to be considered when the timer is modelled, also when the requirement to be checked is expressed (e.g., the current time can be smaller for a later event, see Fig. 3).

(2) **Time is incremented by adding the cycle time**: In this representation, time is not incremented by individual units of time. It is instead incremented by the duration of the last PLC cycle at the end of it. This assumption obviously simplifies the global model and there is not any loss of accuracy when comparing with the real implementation in a PLC, considering that the timers are called at most once in each PLC cycle, which holds for our real cases.

(3) **Cycle time is chosen non-deterministically**: in order to represent standard PLC with a varying cycle time, a random value is generated at the end of each cycle to represent this. The selected random values are between 5 ms and 100 ms, which is a valid assumption based on the PLC systems at CERN.

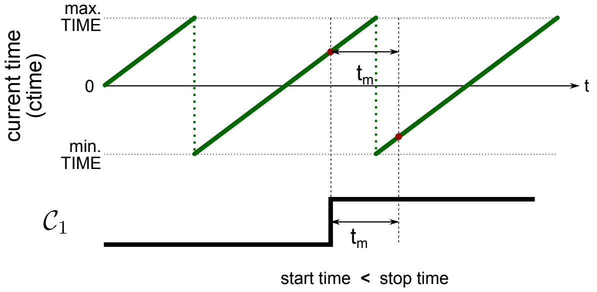


Fig. 3. Consequences of finite time representation

Timer representation Given this finite time representation, the behaviour of the timer is represented as it is shown on the ST code (see Fig. 4). In this code, the input variables are *IN* and *PT*, and the output variables are *Q* and *ET*. The variable *ctime* represents the current time and it is modelled as previously explained. In addition, two variables are added: *running* and *start*, where *running* is a Boolean variable representing when the timer is working after a rising edge on *IN* and *start* contains the value of *ctime* when *IN* has a rising edge.

By applying the extended methodology, the corresponding automaton of the TON ST code was produced and the equivalent state machine is shown in Fig. 5. (Note that the assignments of ET are omitted from the state machine to simplify the figure.) This state machine contains 3 states corresponding to the 3 original states of the TON: *NR* (not running; running=false, Q=false), *R* (running; running=true, Q=false) and *TO* (timeout; running=true, Q=true). The transitions in the state machine (labelled as t_1, t_2, t_3, t_4) correspond to the conditional statements in the ST code.

```
IF in = false THEN
    Q := false; ET := 0;
    running := false;                // t1
ELSIF running = false THEN
    start := CTIME;
    running := true;                 // t2
ELSIF CTIME - (start + PT) >= 0 THEN
    Q := true; ET := PT;             // t3
ELSE
    IF not Q THEN ET := CTIME - start; END_IF;  // t4
END_IF;
```

Fig. 4. ST code of TON

Only one transition can happen in this state machine for every call of the timer. Therefore the timer cannot go from state NR to state TO with one function call, which is valid if we assume that delays are always greater than zero ($PT > 0$). According to the specification of the Siemens TON implementation (Siemens (1998)), the parameter PT should be positive.

The potential state space (PSS) size of this timer representation is $3.38 \cdot 10^{16}$ and its reachable state space (RSS) is $5.91 \cdot 10^{15}$ without counting the variable ctime (as it is a global variable used by all the timers).

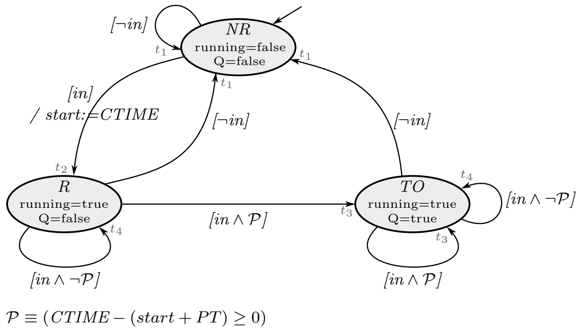
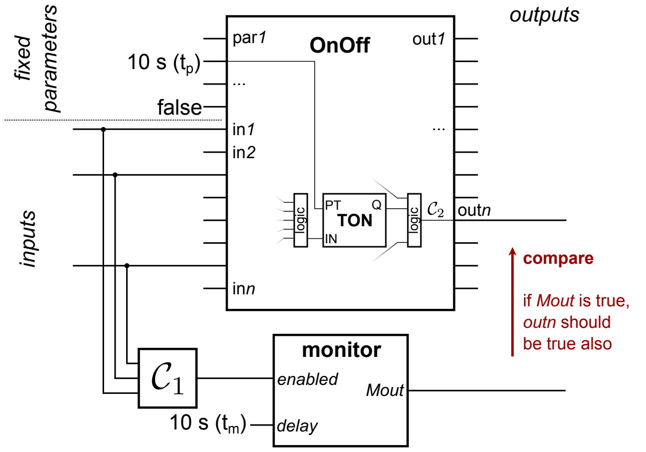


Fig. 5. State machine of the realistic TON representation

Property specification Using the methodology and the developed tool, input models for NuSMV are automatically generated. The experimental results presented here are produced using this model checker. NuSMV provides CTL (Computational Tree Logic) and LTL (Linear Temporal Logic) for property specification. Our goal is to verify properties with explicit time in it, like "if $\mathcal{C}_1$ is true, after t_m time $\mathcal{C}_2$ will be true, if $\mathcal{C}_1$ remained true" (where $\mathcal{C}_1$ and $\mathcal{C}_2$ are Boolean expressions, $\mathcal{C}_1$ contains input variables and parameters and $\mathcal{C}_2$ contains output variables).

CTL and LTL do not provide this expressiveness, for this reason a *monitor* or observer automata is added to the model. The goal of the monitor is to check $\mathcal{C}_1$, and if it is true for at least t_m time, the monitor output value *Mout* is set to true. By using such monitor, the requirement is simplified as "if *Mout* is true, then $\mathcal{C}_2$ should be true". This requirement can be formalized easily in CTL: $AG(Mout \rightarrow \mathcal{C}_2)$. An example for this monitor usage can be seen on Fig. 6. As the computed values are assigned to the real outputs of the PLC only at the end of the PLC cycle, the requirements should be checked only at this point. The general CTL expression extended with it is the following: $AG(PLC\_END \rightarrow (Mout \rightarrow \mathcal{C}_2))$, where PLC_END is true, iff the execution is at the end of a PLC scan cycle.

The behaviour of this monitor is similar to the TON timer, but it is independent of the rest of the program logic, therefore *Mout* will be true after t_m time, if $\mathcal{C}_1$ is true and it can be used to verify if *outn* holds the property. It has to be noticed, that it is not enough to save a "timestamp" $t_{\mathcal{C}_1}$ when $\mathcal{C}_1$ is true, and formalize the requirement as $AG((ctime \geq t_{\mathcal{C}_1} + t_m) \rightarrow \mathcal{C}_2)$, because the *ctime* variable is a finite integer, thus it can overflow, as it is illustrated by Fig. 3.



if *Mout* is true, *outn* should be true also

Fig. 6. Verification configuration for OnOff model extended with monitor

Experimental results This approach has been applied to the UNICOS framework. In particular, the experiments showed here have been applied to the OnOff object from the UNICOS library. Table 1 presents different state spaces depending on which abstraction and reductions techniques are applied. On the OnOff model, we made our experiments with a requirement of the previously introduced structure: "if $\mathcal{C}_1$ is true, after t_m units of time $\mathcal{C}_2$ will be true, if $\mathcal{C}_1$ remained true".

To be able to verify this property, the original model was shrink from a PSS of $1.6 \cdot 10^{218}$ states to a PSS of $1.1 \cdot 10^{36}$ as can be seen in Table 1. In order to do so, reductions, fixed parameters (focusing on a specific scenario), and COI technique (which eliminated 2 timers out of 3) have been applied. As it was mentioned before, time values were modelled as 16-bits variables instead of 32-bits.

Table 2 presents verification results for the OnOff model after applying all the previously mentioned reduction techniques. We checked the introduced requirement with three different configurations in terms of timer delay (t_p) and monitor delay (t_m). The first uses $t_p = t_m = 10\ s$ values, which means the parameter t_p of OnOff and the delay parameter t_m of the monitor were both 10 s. In this case, the result is true which is given by NuSMV in 11.4 s. The other two configurations use monitor delay times smaller than the parameter t_p, therefore the output is expected too early, thus the result should be false. As it can be seen in Table 2, in both cases the verification time was around 5–20 s, but the generation of the counterexample (C.ex. gen. time) took significantly more time. In the case when we used $t_m = 9\,s$, the counterexample was longer (3 876 steps), and its generation time was around 15 times bigger than when we used $t_m = 1\,s$. Notice that these verification results can have a timing error not bigger than the maximal cycle time, 100 ms in this case, because the current time is incremented in quanta.

Table 1. State space of the OnOff model with realistic time representation

Time	Monitor	Reductions	PSS	#Vars
32 bit	–	none	$1.6 \cdot 10^{218}$	255
16 bit	–	none	$2.5 \cdot 10^{160}$	255
16 bit	–	general	$1.6 \cdot 10^{134}$	185
16 bit	+	general	$8.5 \cdot 10^{139}$	189
16 bit	+	general, COI	$1.1 \cdot 10^{85}$	143
16 bit	+	gen., fix params, COI	$1.1 \cdot 10^{36}$	86

Table 2. Verification time on OnOff model with realistic time representation

Timer delay (t_p)	Monitor delay (t_m)	Result	Run time	C.ex. gen. time	C.ex. length
10 s	10 s	true	11.4 s	—	—
10 s	9 s	false	19.5 s	2513.2 s	3 876
2 s	1 s	false	5.2 s	123.0 s	510

4.2 Abstract approach

As it was shown in the previous experimental results, the models created by realistic approach have a big state space even if only one TON is modelled. If this approach is applied to larger models than the UNICOS OnOff object probably verification would not be even possible. For that reason, we propose a second approach based on a data abstraction of the first approach.

Time representation In this case, time is not represented explicitly in the model, the variable *ctime* representing the current time is not maintained. Therefore properties with explicit time cannot be validated.

Timer representation This timer representation approach consists in a non-deterministic model produced as an abstraction of the realistic approach. The corresponding state machine is presented in Figure 7.

Similarly to the realistic model, this model has three possible internal state: *NR* (not running), *R* (running) and *TO* (timed out). If the input *IN* is true, the TON will start to run (goes to state *R*). After that, the TON can stay in the state *R* or go to *TO* non-deterministically, which means, we do not know when the timer will stop. However, by adding a *fairness constraint* to the model, it is ensured that the timer cannot stay in state *R* for infinite time. In one PLC cycle only one transition can be fired, i.e., the timer cannot go from state *NR* to state *TO* with one call. This corresponds to the previously introduced $PT > 0$ constraint. Figure 8 shows the timing diagram of the abstract approach compared with the realistic one. The size of the PSS and RSS for this model is 6 comprising the Boolean input variable, reducing significantly the size of the realistic approach but introducing some limitations of the property specification.

This representation can result false positives, meaning that verification tools can give spurious counterexample that cannot occur in reality. However false negatives can never occur, therefore if a property holds in the abstract model, it holds in the real system.

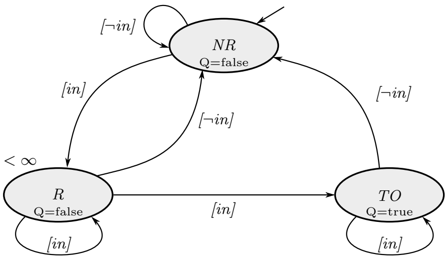


Fig. 7. State machine of the abstract TON representation

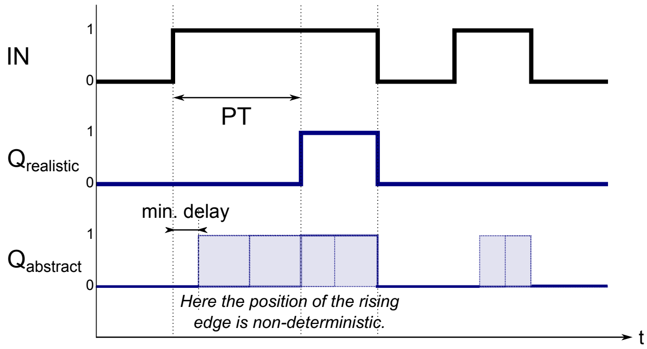


Fig. 8. Timing diagram of TON modelled using different approaches

Property specification Obviously, this abstract model implies certain limitations in the specification properties. Explicit delay times cannot be expressed in the requirement, but safety or liveness properties can be expressed.

Experimental results This approach has also been applied to the OnOff object from the UNICOS library. For Experiment 1, we used the same reduction techniques and the same fixed parameters as for the realistic approach. The PSS of the OnOff model became $5.5 \cdot 10^{24}$ (instead of $1.1 \cdot 10^{36}$). Representing the requirement checked on the realistic model is not possible in this approach as the time is not counted explicitly. Instead, we can check the following liveness property: "if $\mathcal{C}_1$ is sometime true and remains true forever, eventually $\mathcal{C}_2$ will be true". The equivalent LTL property is: $F(G(PLC\_END \rightarrow \mathcal{C}_1) \rightarrow F(\mathcal{C}_2))$, where PLC_END represent the end of a PLC cycle. In this case, no monitor is needed.

We made two other experiments on the model with abstract time representation (Exp. 2, 3). In these cases, no parameters were fixed and all the three timers had effect on the requirement. The requirement 2 was a simple safety requirement in CTL ($AG(\mathcal{C}_3 \rightarrow \mathcal{C}_4)$), while requirement 3 was a bit more complex: $AG(\mathcal{C}_5 \rightarrow AF(\mathcal{C}_6))$. Our experiments (c.f. Table 4) showed that these requirements can be checked using the abstract time representation, even without fixing any parameters. With the realistic time representation, the state space would be too large to be verified using NuSMV.

Table 3 presents different state spaces depending on which reductions techniques are applied and what requirement was verified. Table 4 shows the measured run times.

Table 3. State space of the OnOff model with abstract time representation

Reductions	PSS	#Vars
none	$1.1 \cdot 10^{131}$	243
general	$4.7 \cdot 10^{112}$	164
general, COI, fix params (Exp. 1)	$5.5 \cdot 10^{24}$	77
general, COI (Exp. 2, 3)	$1.1 \cdot 10^{40}$	126

Table 4. Verification time on OnOff model with abstract time representation

Exp.	#Timers	Result	Run time	C.ex.
(1)	1	true	6.1 s	—
(2)	3	false	294.2 s	18.6 s
(3)	3	false	4 201.9 s	12 020.9 s

4.3 Refinement between the two approaches

We can verify that the realistic approach indeed refines the abstract approach. Using this proof, we are able to guarantee that requirements verified on the abstract model – where verification is easier – also hold on the realistic model – where automatic verification would be harder and more time consuming. However, the false result on the abstract model does not imply false result on the realistic model. Using more abstract models for verification purposes and lifting results to realistic models is a well known technique (Clarke et al. (1999); Loiseaux et al. (1995)) and we apply it to the PLC domain. Specifically to PLC timer models and we perform a proof in a similar fashion as the refinement proofs described in Blech and Grégoire (2011).

The proof is done by first establishing a simulation relation S between realistic (Fig. 5) and abstract automaton (Fig. 7) that relates the states of the different automata with each other such that only the following holds:

$$S(NR_{realistic}, NR_{abstract})$$
$$S(R_{realistic}, R_{abstract})$$
$$S(TO_{realistic}, TO_{abstract})$$

Furthermore, the simulation relation ensures that Q has the same value in all states. In order to finish the proof, we show that:

- The initial states are in the simulation relation: $NR_{realistic}$ and $NR_{abstract}$ are in the simulation relation and the value of Q is the same, false.
- Each pair of realistic and abstract model state transitions with a corresponding condition – regarding the value of in – from possible states in the simulation relation S lead to a pair of states for which S holds again. Most cases are trivial expect: the transition of the $[in]$-guarded transition in the abstract model in the $(TO_{realistic}, TO_{abstract})$ state pair has two corresponding transitions in the realistic model since we do not regard the value of $\mathcal{P}$ in the abstract model.

Due to the fairness constraint in the abstract model, its state cannot be $R_{abstract}$ for infinite time. However, this is also true for the corresponding $NR_{realistic}$ state of the realistic model if it is called sufficiently often [1], because the delay time PT is finite.

[1] It is not defined in the standard or in the manual, but our implementation works properly, if the elapsed time between two calls is less than the maximal value of the TIME data type, which is about

Based on the definition of the simulation relation, safety properties making use of the atomic elements – like conditions on the used variables Q and in – that are preserved in S are also preserved between abstract and realistic model. However, the abstract model can impose false positives, i.e. the given abstract counterexamples can never occur in the in the real system.

5. ANALYSIS AND CONCLUSIONS

This paper proposes two different approaches for modelling time and timers in PLC-based control systems. Experimental results applied to a real PLC program developed at CERN and an analysis of both approaches have been presented. While the first approach provides a realistic model of a PLC time and timers, the second provides an abstract representation of them.

The first approach represents with high fidelity the TON implementation on a real PLC. This TON representation implies also to model time. The accuracy of the model is high enough, using as unit of logic time 1 ms for timers (as in a real PLC). In terms of specification, this approach allows to express properties with explicit time by using CTL and a monitor. However the resulting state space is very big. Even if some abstraction techniques are applied verification time can become very large and in some cases it may not be handled by model checkers.

The second approach solves the problem of state space explosion by proposing a simplified model of the first approach. The resulting model has a non-deterministic nature, so the accuracy is reduced and it can produce false positives. In terms of specification, it is not possible to verify properties with explicit time, but it can verify properties that guarantee that the timer gives a response (liveness property) although we can not verify when the response will be given.

Table 5 summarizes the main differences of both approaches.

Table 5. Overview of the timer representation approaches

	Realistic	**Abstract**
Size of PSS	huge (e.g. $5.91 \cdot 10^{15}$ for the TON model)	moderate (e.g. 6 for the TON model)
Requirement expressivity	rich requirements with explicit time, CTL/LTL + monitors	requirements without explicit time, CTL/LTL is enough
Constraints	time incremented by PLC cycle time	no explicit time, false positives

The first approach is thus needed when properties with explicit time have to be verified but there is a higher chance of having a state explosion problem. The second approach is suitable for timed properties without explicit time (for example certain before/after properties) and for non-timed properties (for example safety or liveness properties), where the variables linked to the property are affected by a timer in the real system and this timer cannot be eliminated by using abstraction techniques, such as cone of influence. Although, false positive results can occur using this approach, false negative can never occur.

Both approaches are supported by the suggested general methodology for verifying PLC programs. A tool is also provided for the automatic generation of formal models for different verification tools.

Future plans comprise the integration of the tool and methodology in the UNICOS development process. In addition, extending and optimizing the abstraction techniques and the proof of the transformation to guarantee its correctness are ongoing work.

30 s represented on 16 bits or 24 days represented on 32 bits. Usually, the timers are called in every PLC cycle, thus this is not a limitation. This is also needed for the timers to work properly.

REFERENCES

Amnell, T. et al. (2001). UPPAAL – now, next, and future. In *Modeling and Verification of Parallel Processes*. Springer Berlin Heidelberg.

Basu, A. et al. (2011). Rigorous component-based system design using the BIP framework. *IEEE Sw.*, 28, 41–48.

Behrmann, G., David, A., and Larsen, K.G. (2004). A tutorial on UPPAAL. In *SFM-RT 2004., Revised Lectures*, volume 3185 of *LNCS*, 200–237. Springer Verlag.

Blanco, E. et al. (2011). UNICOS evolution: CPC version 6. In *12th ICALEPCS*.

Blech, J.O. and Grégoire, B. (2011). Certifying compilers using higher-order theorem provers as certificate checkers. *Formal Methods in System Design*, 38(1), 33–61.

Cimatti, A. et al. (2002). NuSMV 2: An opensource tool for symbolic model checking. In *14th CAV*.

Clarke, E.M., Grumberg, O., and Peled, D.A. (1999). *Model Checking*. The MIT Press.

Darvas, D., Fernández, B., and Blanco, E. (2013). Transforming PLC programs into formal models for verification purposes. Internal note, CERN. CERN-ACC-NOTE-2013-0040.

Darvas, D., Fernández, B., Vörös, A., Bartha, T., Blanco, E., and González, V.M. (2014). Formal verification of complex properties on PLC programs. In *Proc. of 34th Conf. on Formal Techniques for Distributed Objects, Components and Systems*, LNCS. Springer. To appear.

IEC 61131 (2013). IEC 61131: Programming languages for programmable logic controllers.

Loiseaux, C., Graf, S., Sifakis, J., Bouajjani, A., Bensalem, S., and Probst, D. (1995). Property preserving abstractions for the verification of concurrent systems. *Formal methods in system design*, 6(1), 11–44.

Mader, A. and Wupper, H. (1999). Timed automaton models for simple programmable logic controllers. In *11th Euromicro Conference on Real-Time Systems*.

Mokadem, H.B., Brard, B., Gourcuff, V., et al. (2010). Verification of a timed multitask system with UPPAAL. *IEEE Transactions on Computers Transactions on Automation Science and Engineering*, 7, 921 – 932.

Perin, M. and Faure, J.M. (2013). Building meaningful timed models of closed-loop DES for verification purposes. *Control Engineering Practice*, In press.

Siemens (1998). *Structured Control Language (SCL) for S7-300/S7-400 Programming*. Siemens.

Wang, R., Guan, Y., Liming, L., Li, X., and Zhang, J. (2013). Component-based formal modeling of PLC systems. *Journal of Applied Mathematics*, 2013.