ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV042

APPLYING MODEL CHECKING TO HIGHLY-CONFIGURABLE SAFETY CRITICAL SOFTWARE: THE SPS-PPS PLC PROGRAM

B. Fernández*, I. D. Lopez-Miguel, J-C. Tournier, E. Blanco,
T. Ladzinski, F. Havart, CERN, Geneva, Switzerland

Abstract

An important aspect of many particle accelerators is the constant evolution and frequent configuration changes that are needed to perform the experiments they are designed for.

This often leads to the design of configurable software that can absorb these changes and perform the required control and protection actions. This design strategy minimizes the engineering and maintenance costs, but it makes the software verification activities more challenging since safety properties must be guaranteed for any of the possible configurations.

Software model checking is a popular automated verification technique in many industries. This verification method explores all possible combinations of the system model to guarantee its compliance with certain properties or specification. This is a very appropriate technique for highly configurable software, since there is usually an enormous amount of combinations to be checked.

This paper presents how PLCverif, a CERN model checking platform, has been applied to a highly configurable Programmable Logic Controller (PLC) program, the SPS Personnel Protection System (PPS). The benefits and challenges of this verification approach are also discussed.

INTRODUCTION

Model checking is a popular verification technique that has been applied in many industries to guarantee that a critical system meets the specifications. Model checking algorithms explore all possible combinations of a system model, trying to find a violation of the formalized specification in the model. This technique is very appropriate for highly configurable projects, since it is necessary to guarantee the safety of the system for all possible configurations.

When model checking shows a discrepancy between the PLC program and the specification, it means that either the PLC program has a bug or the specification is incomplete or incorrect.

In the domain of critical PLC programs, several researchers and engineers published their experiences in the field. To name but a few, in [1] the authors translate the PLC program of an interlocking railway system, written in the FBD (Function Block Diagrams) language, into the input format language of NuSMV to verify their specification written as CTL (Computation Tree Logic) properties. In [2], the PLC program that controls the doors' opening and closing in the trains from the Metro in Brasília, Brazil, was formally verified. In this case, a B model [3] is created automatically from the PLC code. This model is formally verified using the model checker ProB [4].

When applying model checking to PLC programs, three main challenges are faced: (1) building the mathematical model of the program, (2) formalizing the requirements to be checked and (3) the state-space explosion, i.e. the number of possible input combinations and execution traces is too big to be exhaustively explored. In our case, we use the open-source tool PLCverif[5], developed at CERN. It creates automatically the models out of the PLC program and integrates three state-of-the-art model checkers: nuXmv [6], Theta [7] and CBMC [8]. It also implements some reduction and abstraction mechanisms to reduce the number of states to be explored and to speed up the verification. Therefore, challenges 1 and 3 are transparent for the user. There are certainly still limitations and large state-space PLC program models cannot be verified. Regarding challenge 2, PLCverif also provides mechanisms to help the users to formalize their requirements and provide a precise specification. However, this is normally a difficult task, specially for configurable programs. In this paper, we will show examples of functional requirements formalization with PLCverif. More details about PLCverif can be found in [5, 9].

This paper aims to show the benefits of applying model checking to verify highly configurable PLC programs. In such systems, it is usually unfeasible to check all possible combinations by traditional testing methods and model checking is a good complement to these methods, especially for module verification.

In particular, this paper shows how PLCverif was applied to the PLC programs of the SPS Personnel Protection System (PPS) [10] and how it helped to improve their original specification and correct PLC bugs before the commissioning of the system.

SPS PERSONNEL PROTECTION SYSTEM

The SPS-PPS is a large distributed control system in charge of the access control and the personnel protection of the SPS accelerator.

The SPS has 16 access zones divided in different sectors and each access zone has an access point. Several access zones are always interlocked with the same elements inhibiting operation with beam when a hazardous event is detected. This is the concept of a safety chain. Each safety chain contains the "important safety elements" to stop the beam (EISb) when a hazardous event is detected or to avoid access (EISa) when the accelerator is in Beam mode.

All details of the system can be found in [10].

* borja.fernandez.adiego@cern.ch

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV042

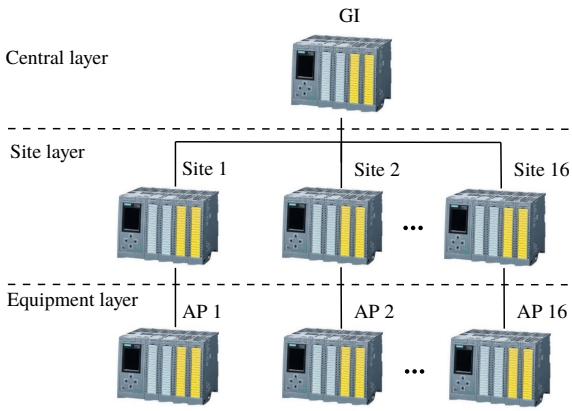

Figure 1: Simplified SPS-PPS PLC interlock architecture.

The SPS evolves with time as new access zones are added or existing ones modified. For that reason, a configurable system was designed.

Hardware Architecture

The SPS-PPS safety-interlock part is based on Siemens S7-1500F series of PLC controllers. It is formed by the following 3 layers as shown in Fig. 1:

- The central layer is composed by the Global Interlock (GI) PLC. This PLC receives the information of all the Site PLCs and computes, for example, the conditions to allow Beam or Access modes to each safety chain and the conditions to apply veto to all the EISa and EISb associated to the safety chain.
- The site layer is composed by 16 Site PLCs (1 PLC per access zone). These PLCs monitor the EISa devices (doors, emergency handles, etc.), receive the status from the AP PLCs, perform some automatic control procedures (e.g. patrol management) and apply the safety interlock actions (e.g. stop the beam) with the commands received from the GI PLC for each access zone.
- The equipment layer is composed by 16 Access Point (AP) PLCs (1 PLC per access point). They monitor and control the PAD (Personnel Access Device) and the MAD (Material Access Device) of the access point.

Software Architecture

All Site PLCs run the same generic PLC program, a common logic for all of them with a specific configuration for each access zone. The logic is composed by a set of Functions (FC) and Function Blocks (FBs) executed sequentially from a cyclic interrupt every 300 ms. Each access zone has a different configuration, indicating for example how many EISa and EISb are installed in each of them, in which sector they are located and to which safety chain they belong.

A similar approach is in place for the AP PLCs.

The GI PLC contains less configurable parameters since it communicates with all Site PLCs, having a global view of the system.

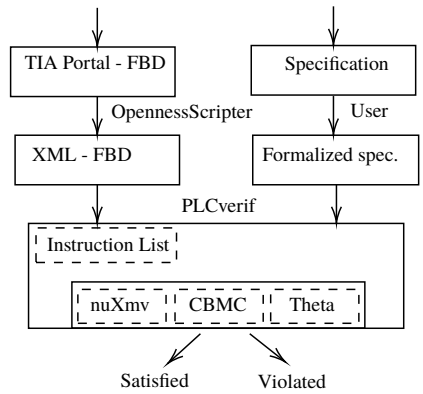

Figure 2: PLCverif workflow to verify TIA portal Safety programs.

Software Modules Specification

The different program modules (FC and FBs) are specified in a detailed design document. It contains 106 formulas that describe the requirements for the 29 FC and FBs.

The chosen formalism for the specification is a simple *if-else* conditional statement that contains Boolean formulas, as shown in the example Specification 1. The goal was to have a simple pseudo-formal specification providing precise and unambiguous requirements for each module of the system.

Specification 1 Template for specifying protection actions of the Safety program.

if (*Boolean expression condition*) **then**
  result ← 0
else if (*Boolean expression condition 2*) **then**
  result ← 1
else
  result ← *result*
end if

MODULES VERIFICATION WITH PLCVERIF

PLCverif was the tool to apply model checking to the SPS-PPS modules (FC and FBs). All the necessary model transformations to create the formal model of the PLC program are done automatically. The user only needs to provide the exported PLC program from the programming environment tool, Siemens TIA Portal[1], import it into PLCverif and finally formalize the requirements to be verified.

Figure 2 shows the workflow to verify a FC or FB from a TIA Portal safety PLC project with PLCverif:

1. The project is hosted in TIA Portal and the PLC program is written in the FBD (Function Block Diagram) programming language. An example of how this language looks like is shown in Fig. 3. Here an AND gate and an OR gate are displayed.

[1] https://new.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal/software.html

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV042

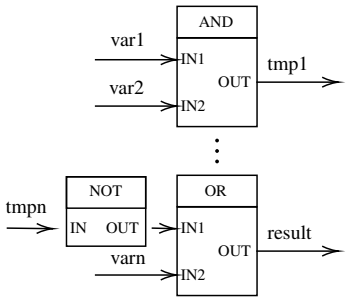

Figure 3: FBD program example.

2. Taking advantage of the Siemens OpennessScripter tool[2], the program is exported to XML (*F_FBD* XML format). An example of how an AND gate is represented in this language is shown in Listing 1.

3. PLCverif can then import that XML code, translating it into STL code, which can be understood by PLCverif. Subsequently, it creates the formal models.

4. From the specification created by the process engineers, the user needs to formalize the requirements, such as the assertion shown in Listing 2. This can be direct input to PLCverif.

5. Finally, PLCverif executes one of the model checking backends and generates a report, stating whether the property is satisfied or violated. In case it is violated, it shows the trace leading to that property failure (the counterexample).

Listing 1: XML code exported from TIA portal representing an AND gate

```
<!-- Input variables -->
<Access Scope="LocalVariable" UId="21">
<Symbol>
<Component Name="var1"/>
</Symbol>
</Access>
<Access Scope="LocalVariable" UId="22">
<Symbol>
<Component Name="var2"/>
</Symbol>
</Access>

<!-- AND block -->
<Part Name="And" UId="97">
<TemplateValue Name="Card" Type="Cardinality">2</
    TemplateValue>
<TemplateValue Name="SrcType" Type="Type">Word</
    TemplateValue>
</Part>

<!-- Connecting the inputs with the block -->
<Wire UId="134">
<IdentCon UId="21" />
<NameCon UId="97" Name="in1" />
</Wire>
<Wire UId="135">
<IdentCon UId="22" />
<NameCon UId="97" Name="in2" />
</Wire>

<!-- Output variable -->
<Wire UId="136">
<NameCon UId="97" Name="out" />
<IdentCon UId="23" />
</Wire>
```

[2] https://support.industry.siemens.com/cs/us/en/view/109742322

Listing 2: Example of a PLCverif assertion

```
//#ASSERT (var1 AND var2) = result;
```

VERIFICATION RESULTS AND ANALYSIS

Although various safety functions were verified during this project, due to space limitations for this paper, only two of them will be shown here: the *SIF-X1* Function from the Site PLCs and the *SIF-2* Function from the GI PLC. Each of these PLC Functions implements several requirements of a similar nature and each of these requirements was verified with PLCverif by including an assertion which represents the mathematical Boolean formula given for that requirement. After executing a verification case with PLCverif, the following outcomes can occur:

- Satisfied property. It will be shown in Example 1 from SIF-X1.
- Violated property. The reasons why this happens need to be further investigated, leading to the following two possibilities:
  - Incomplete specification. Example 2 from SIF-2.
  - Bug in the program. Example 3 from SIF-2.

SIF-X1 Function

This function is in charge of monitoring all the EISa of the access zone and computing the status (safe or unsafe) for each safety chain.

Example 1 - Satisfied property. The selected requirement for this use case is the following: For each Safety Chain, this Function monitors all the EISa that are installed in the access zone and belong to the Safety Chain. It also receives the status from its access point (PAD, MAD and token distributor status). If all these elements are in a safe state, it returns a Boolean variable indicating that the Safety Chain is safe.

The formalized requirement is shown in Specification 2, where

i is the EISa index,

j is the safety chain index,

N_i is the number of EISa assigned to the safety chain j (the maximum number of EISa per access zone is 32),

$I\_EISa\_Pos[i]\_Stat$ represents the status of the EISa i: true if i is closed,

$I\_EISa\_PU[i]\_Stat$ represents the status of the emergency handle of the EISa i: true if the emergency handle is armed, $I\_EISa\_Pos[i]$ is a configuration variable: true if the EISa i is installed in the access zone,

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV042

$I\_EISa\_PU[i]$ is configuration variable: true if the EISa i has an emergency handle,

$SC - Sj[i]$ is configuration variable: true if the EISa i belongs to the Safety Chain j,

$S0\_AP\_Pos[j]$, $S0\_AP\_PU[j]$ and $S0\_AP\_Key\_Distrib[j]$ represent the status of the access point: true if the access point is closed, the emergency handle is armed and the token distributor is safe respectively,

$N\_EISa\_Safe[j]$ is the output variable of the Function and represents the status of the safety chain j: true if j is safe.

Specification 2 Partial, simplified requirement of SIF-X1.

if (

$$\prod_{\substack{i=1 \\ \forall i \in [1,N_I]:(I\_EISa\_Pos[i]=1 \wedge SC-S\{j\}[i]=1}}^{N_i} (I\_EISa\_Pos\_Stat[i] \vee I\_EISa\_Bypass[i])$$

$$\wedge$$

$$\prod_{\substack{i=1 \\ \forall i \in [1,N_I]:(I\_EISa\_PU[i]=1 \wedge SC-S\{j\}[i]=1}}^{N_i} (I\_EISa\_PU\_Stat[i] \vee I\_EISa\_Bypass[i])$$

$$\wedge$$

$$S0\_AP\_Pos[j] \wedge S0\_AP\_PU[j] \wedge S0\_AP\_Key\_Distrib[j]$$

) **then**

$N\_EISa\_Safe[j] \leftarrow 1$

else

$N\_EISa\_Safe[j] \leftarrow 0$

end if

The behaviour of this relatively complex specification is shown in Table 1 with an example. Here, 2 doors (EISa) are installed in this access zone (indexes 1 and 4 of $EISa\_Pos[i]$), only one of them has an emergency handle (index 1 of $EISa\_PU[i]$). One of the EISa belongs to the Safety Chain 0 (index 1 of $SC - S\_0[i]$) and the other one to the Safety Chain 1 (index 4 of $SC - S\_1[i]$). For each safety chain j, the resultant $N\_EISa\_Safe[j]$ is true if the installed doors and emergency handles are in the safe state ($EISa\_Pos\_Stat[i]$ and $EISa\_PU\_Stat[i]$).

Table 1: SIF-X1 Expected Behaviour

Source	Variable	Value
Input	$EISa\_Pos\_Stat$	0000 0000 0000 1001
Input	$EISa\_PU\_Stat$	0000 0000 0000 0001
...	...	...
Input (AP PLC)	$S0\_AP\_Pos$	0000 0000 0000 1001
...	...	...
Configuration	$EISa\_Pos$	0000 0000 0000 1001
Configuration	$EISa\_PU$	0000 0000 0000 0001
...	...	...
Configuration	$SC - S\_0$	0000 0000 0000 0001
Configuration	$SC - S\_1$	0000 0000 0000 1000
...	...	...
Output	N_EISa_Safe	0000 0000 0000 1001

The specification was translated into 16 verification cases (one per safety chain), in order to guarantee that this property is respected in all safety chains for all possible combinations of the input and configuration variables. The example for the safety chain 0 can be seen in Listing 3 (simplified assertion).

The corresponding PLC program has 94 WORD (16-bit variable) and 4 BOOL configuration variables, and 21 WORD and 2 BOOL input variables to implement all the requirements. This implies approximately $2^{16\cdot(94+21)+6} = 2^{1846} \approx 5.0 \cdot 10^{555}$ combinations to be checked. The equivalent STL code of the original F-FBD version is around 700 lines of code.

PLCverif was able to validate the 16 verification cases in 41 seconds. The result of all properties (assertions) was satisfied. This means that no counterexamples were found and, hence, the properties always hold true.

Listing 3: SIF-X1 formalized requirement for PLCverif.

```
//#ASSERT
(((I_EISa_Pos_Stat OR I_EISa_Bypass) AND
(SC-S_0 AND I_EISa_Pos))
= (SC-S_0 OR I_EISa_Pos)) AND
(((I_EISa_PU_Stat OR I_EISa_Bypass) AND
(SC-S_0 AND I_EISa_PU))
= (SC-S_0 OR I_EISa_PU)) AND
(S0_AP_Key_Distrib.%X0 AND
S0_AP_PU.%X0 AND S0_AP_Pos.%X0)
= N_EISa_Safe.%X0
```

SIF-2 Function

The following two examples belong to the verification of the SIF-2 Function. In this function, PLCverif was able to find several discrepancies between the PLC program and the specification and two of them are presented here.

This function contains the transitions rules between the Beam and Access modes for each safety chain j. In order to allow the transition between modes, the function checks the status of the safety chain (computed variable calculated by another GI PLC function) and the requests from the operator, via the OKP (Operator Key Panel). When all safety conditions are met, this function authorizes the change of mode. The function also contains the conditions to lock or release the keys from the OKP.

The implementation of the SIF-2 Function Block has 19 WORD and 1 BOOL input variables to implement all the requirements. This implies approximately $2^{19\cdot16+1} = 2^{305} \approx 6.5 \cdot 10^{91}$ combinations to be checked.

The equivalent STL code of the original F-FBD version is around 1200 lines of code.

Example 2 - Incomplete specification. This specification was related to the conditions to switch to Beam mode. The property was formalized with the assertion shown in Listing 4:

Listing 4: SIF-2 Beam mode activation formalized requirement for PLCverif.

```
//#ASSERT
((NOT((N_EXT_ACCE_OK AND N_NO-SAFETY_ERR) AND
((I_PB_TEST_ON AND I_Key_TEST) OR
(I_PB_Acce_ON AND I_Key_Acce)))
OR (NOT N_MODE_BEAM)) = 16#FFFF)
```

This assertion was verified with PLCverif in 93 seconds and a counterexample was found, meaning that the property was violated. The reason for this violation was a missing variable in the formula from the documentation.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV042

Example 3 - Bug in the PLC program. In this case, the conditions to release an access key from the OKP for one of the safety chains were used as specification. The formalized assertion was the one shown in Listing 5:

Listing 5: SIF-2 extractors access key release formalized requirement for PLCverif.

```
//#ASSERT
((N_SECU_NO-REQ_Down.%X15 AND N_EXT_BEAM_OK.%X15)
= (SIF-2_TAGS.O_RLS_ACCESS.%X15))
```

PLCverif was able to verify this assertion in 147 seconds. The result was not satisfied and a counterexample was provided. After analysing the counterexample and discussing with the SPS-PPS experts, it was concluded that there was a problem in the PLC program. The program was modified accordingly to include the missing variables before the commissioning of the system.

Analysis

PLCverif hides most of the complexity of applying model checking to PLC programs. Nevertheless, a few manual steps were still needed in this project. In particular to import the program in PLCverif and to create the formal properties from the pseudo-formalized specification.

We also encounter the state-space explosion problem in other functions, where the number of configurations and input variables was much bigger.

Overall, PLCverif helped to validate a few critical functions from the SPS-PPS PLC program and it was a good complement to the testing activities performed in the lab by the access control experts at CERN (e.g. the SIF-X1 function).

PLCverif was able to find discrepancies between some of the specifications and the PLC program before the commissioning of the system, sometimes due to an incomplete or erroneous specification, other times due to a bug in the program (e.g. the SIF-2 function). Due to the complexity of some of the specifications, it was sometimes very hard for experts to validate the desired behaviour. By using model checking, the experts were able to find undesirable corner cases, helping them to understand the behaviour of the system and modifying the specification when necessary.

CONCLUSIONS

This paper presented a highly configurable PLC program and the benefits and challenges of applying model checking to verify it. In general, we can conclude that model checking contributed to this project to (1) detect bugs in the PLC program before the commissioning, (2) identify deficiencies in the specification, and (3) help experts to better understand the behaviour of the program for all possible configurations.

There are still several challenges to overcome for model checking to become a common practice in the industrial automation domain. Probably the more important ones are: (1) the integration of model checking in PLC programming environments (e.g. TIA portal) to minimize the number of manual steps and (2) the improvement of verification performance with better algorithms and abstraction techniques.

In the context of the PLCverif project, the future work will focus on the following directions: (1) improve PLCverif to reduce the verification time, (2) support more PLC manufacturers – we are currently working on supporting Schneider ST and IL programs. More information about the current and future work of PLCverif can be found in [9].

REFERENCES

[1] O. Pavlovic and H.-D. Ehrich, "Model checking plc software written in function block diagram," in *2010 Third International Conference on Software Testing, Verification and Validation*, 2010, pp. 439–448. DOI: 10.1109/ICST.2010.10.

[2] H. Moreira Barbosa, "Formal verification of PLC programs using the B Method," M.S. thesis, Universidade Federal do Rio Grande do Norte, Brazil, Oct. 2012.

[3] J.-R. Abrial, *The B-book - assigning programs to meanings*. Jan. 2005, ISBN: 978-0-521-02175-3.

[4] M. Leuschel and M. Butler, "The ProB Animator and Model Checker for B A Tool Description," in *International Symposium of Formal Methods Europe*, Nov. 2003, ISBN: 978-3-540-40828-4. DOI: 10.1007/978-3-540-45236-2_46.

[5] E. Blanco Viñuela, D. Darvas, and V. Molnár, "PLCverif Re-engineered: An Open Platform for the Formal Analysis of PLC Programs," 21. 7 p, 2019. DOI: 10.18429/JACoW-ICALEPCS2019-MOBPP01. https://gitlab.com/plcverif-oss/cern.plcverif

[6] R. Cavada *et al.*, "The nuxmv symbolic model checker," in *CAV*, 2014, pp. 334–342. https://nuxmv.fbk.eu/

[7] T. Tóth, Á. Hajdu, A. Vörös, Z. Micskei, and I. Majzik, "Theta: A framework for abstraction refinement-based model checking," in *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, D. Stewart and G. Weissenbacher, Eds., 2017, pp. 176–179, ISBN: 978-0-9835678-7-5. DOI: 10.23919/FMCAD.2017.8102257. https://github.com/FTSRG/theta

[8] E. Clarke, D. Kroening, and F. Lerda, "A tool for checking ANSI-C programs," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, K. Jensen and A. Podelski, Eds., ser. Lecture Notes in Computer Science, vol. 2988, Springer, 2004, pp. 168–176, ISBN: 3-540-21299-X. https://www.cprover.org/cbmc/

[9] I. D. Lopez-Miguel, J.-C. Tournier, and B. Fernandez Adiego, "PLCverif: status of a formal verification tool for Programmable Logic Controller," in *18th International Conference on Accelerator and Large Experimental Physics Control Systems*, 2021.

[10] T. Ladzinski, B. Fernández Adiego, and F. Havart, "Renovation of the SPS Personnel Protection System: A Configurable Approach," in *17th International Conference on Accelerator and Large Experimental Physics Control Systems*, 2019, 395. 5 p. DOI: 10.18429/JACoW-ICALEPCS2019-MOPHA078. https://cds.cern.ch/record/2777797