Analyzing the Restart Behavior of Industrial Control Applications

Stefan Hauck-Stattelmann[1](✉), Sebastian Biallas[2], Bastian Schlich[1], Stefan Kowalewski[2], and Raoul Jetley[3]

[1] ABB Corporate Research Germany, Research Area Software, Ladenburg, Germany
stefan.hauck-stattelmann@de.abb.com

[2] Embedded Software Laboratory, RWTH Aachen University, Aachen, Germany

[3] ABB Corporate Research India, Research Area Software, Bangalore, India

Abstract. Critical infrastructure such as chemical plants, manufacturing facilities or tidal barrages are usually operated using specialized control devices. These devices are programmed using domain-specific programming languages for which static code analysis techniques are not widely used yet. This paper compares a sophisticated academic tool to a lightweight compliance check approach regarding the detection of programming errors that only occur after program restart. As this is a common problem in industrial control code, the paper proposes a way to improve the accuracy of analyses for this class of errors.

Keywords: Static Analysis · Abstract Interpretation · Programmable Logic Controllers

1 Introduction

Programmable Logic Controllers (PLCs) are widely used for industrial automation tasks, e. g., for controlling equipment or supervising production processes. Most PLC programs are written in programming languages defined in the IEC 61131-3 standard [1]. As these languages are rarely used in other domains, the number of available tools for static code analysis is quite limited in comparison to other languages. The authors previously investigated the use of static code analysis for PLC programs using abstract interpretation [2] and more lightweight techniques [3]. This work discusses the detection of problems that are only triggered after a PLC restart and proposes a way to improve this detection.

PLC programs have several interesting properties distinguishing them from standard applications. PLC programs are always executed cyclically, i. e., they are executed over and over again as long as the PLC is running. From an analysis perspective, this means that there is an implicit loop around the entry and exit point of a program. Interaction with the environment, e. g., sensors and actuators interfacing with machinery, is cleanly separated from program execution through the runtime system. Additionally, many PLCs have battery-backed memory regions, which means that certain program variables can *retain* their values even after the PLC is restarted. This is a very important capability of a PLC and required, e. g., to document the operating hours of machinery.

While **retain** variables are often necessary to implement the required functionality, they are also the source of problems in the code that are hard to detect. The reason for this is that the interaction between variables with and without the **retain** attribute is sometimes difficult to understand and even harder to test. A simplified example of this kind of problem is shown in Fig. 1. The variable **fs** is erroneously marked with the **retain** attribute and thus is only set to the initial value when the program is started the first time. All other variables are set to the value specified in their declaration whenever the PLC is restarted. This leads to a division by zero in the last assignment of the program after a restart, yielding unexpected results.

2 Comparison of Available Analysis Tools

To the best of our knowledge, Bornot et al. [4] were the first to describe static analysis of PLC programs based on abstract interpretation. More recently, Prahofer et al. [5] discuss the applicability of static code analysis for IEC 61131-3 languages and also assess the available commercial tools in this area. Existing commercial tools focus on syntactic checks, e. g., enforcing naming conventions for variables or looking for error-prone code patterns such as dividing by a variable that has not been compared to zero.

```
PROGRAM Program1
VAR RETAIN
   fs : BOOL := TRUE;
END_VAR
VAR
   a : INT := 0;
   b : INT := 0;
END_VAR
   IF fs THEN
      b := 2;
   END_IF;
   fs := FALSE;
   a := 1234 / b;
END_PROGRAM
```

Fig. 1. Example Program

The authors were involved in the development of different research tools for the static analysis of PLC programs. The Arcade.PLC tool[1] developed by RWTH Aachen University focuses on formal methods. A prototype tool developed by ABB corporate research [3] is a hybrid analysis combining abstract interpretation and syntactic checks. The example from Fig. 1 will be used to discuss the different approaches regarding **retain** variables.

Arcade.PLC can detect the division by zero problem shown in the example by first performing a value-set analysis and then using this information to perform further checks, e. g., detecting divisions where zero is part of the potential value range of the divisor. Since the value-set analysis is based on abstract interpretation, it can calculate a sound over-approximation of the value ranges without considering the semantics of the **retain** attribute. The analysis will simply deduce that **fs** can have the value *true* or *false* while **b** can have the value *0* or *2* at program entry. Thus, the division by zero cannot be ruled out and a warning is issued.

The ABB tool supports data flow analyses, but also can check purely syntactic compliance rules. One such rule, which is already used by ABB business units

[1] http://arcade.embedded.rwth-aachen.de, example can be tested there

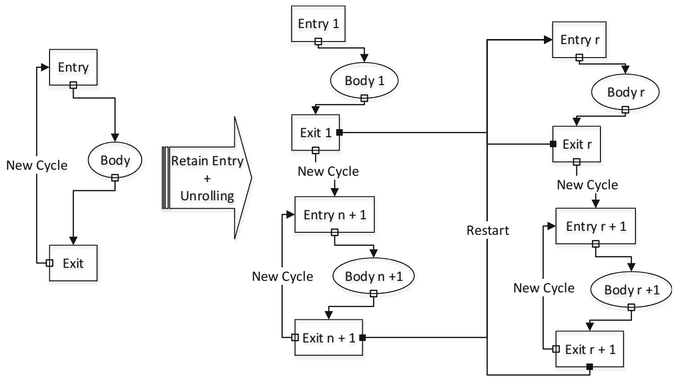

Fig. 2. Proposed Inlining and Unrolling of the Control Flow Graph for a PLC Program

for *manual* code reviews, is that every variable declaration has to specify the **retain** attribute (or a similar one). Applying this rule on the example program yields warnings for the variables **a** and **b**. Automating this check can help in detecting problematic statements in control code, in particular when combined with checks for programming errors like a potential division by zero. While the latter is also supported by the tool, the correlation between a potential error and the missing **retain** attribute still has to be done manually.

Neither of these tools cannot provide a developer with the information that certain problems are only triggered when a program has been running for some time and the PLC is restarted. Unexpected behavior due to incorrectly specified **retain** attributes, however, is a common problem in PLC programs. Detecting these errors manually is very difficult, in particular if the behavior of a program relates to certain characteristics of the equipment it is controlling.

3 Improving Accuracy Through Context Information

Detecting problems during restart can be automated by making the restart explicit during analysis. To achieve this, the analysis has to be made context-sensitive with respect to PLC restart behavior by adding disjoint analysis contexts for the execution cycles after a restart. This technique allows detecting initialization problems and problems that only manifest themselves after a restart. It is similar to the VIVU (Virtual Inlining and Virtual Unrolling) approach proposed in [6] which aims at improving the results for cache modeling.

Improved analysis accuracy can be achieved by building a supergraph from the regular control flow graph (CFG) of the program, as illustrated in Fig. 2. This is achieved through the following steps:

- Unroll the implicit loop around the program once (left hand side of the supergraph).
- Duplicate the unrolled CFG to consider the restart context (right hand side of the supergraph).

– Add edges to the graph so data flow information can be propagated to the entry of the subgraph for restart, but variables without the **retain** attribute are set to their initial value (Restart edges).

Performing data flow analysis on the supergraph makes the analysis of PLC programs more accurate in several ways. First of all, if certain problems such as a potential division by zero are only detected in one of the duplicated subgraphs, this information can be made available to the developer to ease debugging. Most importantly, the analysis results, e. g., value sets of variables, for the corresponding parts of the supergraph with and without a restart can be compared. Thus, divergent behavior between program execution with and without a restart can be detected automatically, which was not possible before. The proposed technique is applicable to all forms of data flow analysis.

4 Conclusion

This paper discussed the capabilities of formal static code analysis based on abstract interpretation and lightweight analysis using code compliance checks regarding errors in PLC programs rooted in PLC restart behavior. Both approaches can detect code smells hinting at these problems, but directly presenting this information to the developer has not been possible so far. To overcome this issue, this paper proposed handling the PLC restart in a separate analysis context by virtual inlining of the restart entry and virtual unrolling of the cyclic code execution. Considering the restart behavior of PLC in the analysis enables the automatic detection of divergent program behavior after a restart. This improvement has already been integrated into Arcade.PLC with little development effort and without significantly impacting the runtime of the analysis.

References

1. International Electrotechnical Commission, IEC 61131-3 Programmable Controllers Part 3: Programming languages (2003)
2. Stattelmann, S., Biallas, S., Schlich, B., Kowalewski, S.: Applying Static Code Analysis on Industrial Controller Code. In: Emerging Technology and Factory Automation (2014)
3. Nair. S., Jetley, R., Nair, A., Hauck-Stattelmann, S.: A Static Code Analysis Tool for Control System Software. In: 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (2015)
4. Bornot, S., Huuck, R., Lakhnech, Y., Lukoschus, B.: Utilizing Static Analysis for Programmable Logic Controllers. In: 4th International Conference on Automation of Mixed Processes (2000)
5. Prahofer, H., Angerer, F., Ramler, R., Lacheiner, H., Grillenberger, F.: Opportunities and Challenges of Static Code Analysis of IEC 61131-3 programs. In: Emerging Technology and Factory Automation (2012)
6. Martin, F., Alt, M., Wilhelm, R., Ferdinand, C.: Analysis of Loops. In: Koskimies, K. (ed.) CC 1998. LNCS, vol. 1383, pp. 80–94. Springer, Heidelberg (1998)