Joel Greenyer, Malte Lochau, and Thomas Vogel (Hrsg.): Explainable Software for Cyber-Physical Systems (ES4CPS), GI-Dagstuhl Seminar 19023, January 2019, Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2019

Explainable Restart-Behaviour of Reactive Systems Software

Dimitri Bohlender[1], Stefan Kowalewski[2]

1 Restart-robustness of Control Software

Control software is at the heart of many complex systems, ranging from assembly lines to space probes, and must meet the safety and reliability requirements of its embedding system. To this end, a lot of research has gone into formal methods for proving a program's compliance w. r. t. the properties of interest, such as absence of runtime errors. However, such a proof of correctness only holds w. r. t. the considered specification, and only under the assumption that the program's semantics were characterised faithfully. With controllers that are particularly tailored to certain safety-critical domains, such as programmable logic controllers (PLCs) for industrial automation, the actual hardware specifics may also affect the program's semantics in unexpected ways.

One such feature that is widespread in controllers for safety-critical applications, is non-volatile memory, or battery-backed memory areas in the case of PLCs, as this allows for controlled handling of crashes and restarts through implementation of resumption strategies. And its not far-fetched having to account for restarts even in such applications. For example to detect exceptional behaviour like hang, most controllers come with a *watchdog timer* which restarts the hardware once it elapses, and therefore has to be reset regularly. Furthermore, if a chemical plant experiences a power outage and switches to emergency power, to the PLCs this will look like a restart, too.

However it is typically up to the author of the control software to decide which variables to *retain* in case of a restart, and make sure that the program complies with the specifications even in the context of restarts. It turns out difficult to keep track of all the corner cases even for small examples, and justify why a choice of *retain variables* is safe – even retaining all variables is no remedy. Accordingly, in practice, errors that only occur after program restart are a common problem in industrial control code [Ha15]. For example consider the use case of automated drilling of holes in workpieces. If the drill's position or mode of operation are not retained, a restart may result in unintended movement of the drill and damage to the

[1] RWTH Aachen University, Informatik 11 – Embedded Software, Ahornstraße 55, 52074 Aachen, Germany bohlender@embedded.rwth-aachen.de
[2] RWTH Aachen University, Informatik 11 – Embedded Software, Ahornstraße 55, 52074 Aachen, Germany kowalewski@embedded.rwth-aachen.de

system itself, the payload, or persons within reach – even if such a malfunction were not possible in a restart-free operation.

To this end, we devised an approach for augmenting the characterisation of PLC program semantics with prominent implementations of writing to non-volatile memory, enabling checking a program's *restart-robustness* w. r. t. a property of interest, given a configuration of retain variables. To aid in the design of restart-robust control software, we also proposed a technique for synthesising configurations of retain variables that are provably safe to use [BK18]. Since this approach is based on *symbolic model checking*, we also get counterexamples, when a violation is found, or a certificate in terms of invariants, otherwise.

2 Challenges

While checking a control program's restart-robustness w. r. t. a property is already helpful, several conceptual challenges remain to make analysis of restart-behaviour more applicable in the context of cyber-physical systems.

In practice, one is often interested in updating existing control software to operate "correctly" even in the context of restarts. In that case, the specification is not given in some logic but as a program, and restart-robustness needs to be defined as a relational property between the nominal and the restart-augmented program behaviour.

A naïve way to define this is to require the *observable* states of the restart-augmented program to be a subset of the observable states of the original program, or that this relation must be established within a finite number of execution-cycles, as this would indicate whether restarting allows the program to exhibit behaviour not possible otherwise. In contrast to this rather weak constraint, the requirement of [KY16], that eventually a state will be reached which is observationally equivalent to a state visited before the restart, might be too strict in practice.

So while we have the background for developing verification procedures, insights from the automation and systems engineering perspective on the feasibility of our undertaking, and reasonable definitions for relational restart-robustness, would be helpful.

References

[BK18] Bohlender, Dimitri; Kowalewski, Stefan: Design and Verification of Restart-Robust Industrial Control Software. In: IFM 2018. pp. 47–68, 2018.

[Ha15] Hauck-Stattelmann, Stefan; Biallas, Sebastian; Schlich, Bastian; Kowalewski, Stefan; Jetley, Raoul: Analyzing the Restart Behavior of Industrial Control Applications. In: FM 2015. pp. 585–588, 2015.

[KY16] Koskinen, Eric; Yang, Junfeng: Reducing crash recoverability to reachability. In: POPL 2016. pp. 97–108, 2016.