IMPROVING DEPENDABILITY OF LOGIC CONTROLLERS BY ALGORITHMIC VERIFICATION

O. Stursberg, S. Lohmann, and S. Engell

Process Control Laboratory (BCI-AST),
University of Dortmund, 44221 Dortmund, Germany.
Email: olaf.stursberg@uni-dortmund.de

Abstract: Functional safety, as addressed in the standard IEC 61508, is a key requirement for a high dependability of controlled systems. In order to guarantee that the function of programmable logic controllers (PLC) complies with given safety specifications, the use of verification has proven to be useful. This contribution builds upon a recently proposed approach to verify PLC programs with time specifications. It starts from a controller design given as sequential function chart (SFC), transforms the SFC into timed automata (TA), and applies model checking to verify (or falsify) functional safety. Since the explicit representation of the cyclic operation mode of PLC can lead to complex TA models, this paper investigates to which extent the cyclic mode can be omitted, to obtain simplified models for which the verification effort is considerably smaller.

Keywords: Automata, Analysis, Discrete Event Models, Safety, Timed Systems.

1. INTRODUCTION

A high level of dependability of logic controllers which supervise the operation of production plants is not only desirable to reduce the plant downtime, but is mandatory if the controller has to ensure that the plant does never encounter safety-critical situations. According to (Avizienis *et al.*, 2000), the term *dependability* involves the aspects availability, reliability, safety, maintainability, and - maybe less important for logic controllers - integrity and confidentiality. This paper focusses on the issue of *functional safety*, as defined in the standard IEC 61508 (Int. Electrotechn. Commission, 2002). Functional safety refers to the property that an electronic or programmable system operates together with its environment such that significant hazards (harming the personnel, equipment, or environment) are prevented. In the context of designing logic controllers for production systems, the consideration of functional safety is crucial for transferring the design specifications into the control code that is eventually implemented on programmable logic controllers (PLC).

To check if the control code complies with all safety-relevant requirements, testing for a chosen set of inputs to the PLC is the technique which is usually applied. As a complementary means, several academic groups have suggested formal verification. In particular *model checking*, which algorithmically searches the reachability tree of a transition system to decide if a formal property is satisfied (Clarke *et al.*, 1999), has been employed for the verification of logic controllers, see e.g. (Moon *et al.*, 1992; Stursberg, 2000). Since Sequential Functions Charts (SFC) become increasingly popular to specify industrial logic controllers, a number of recent investigations aim at making model checking applicable to SFC. While the approaches in (Bornot *et al.*, 2000; Lampérière and Lesage, 2000) transform the SFC into (purely discrete) automata to apply verification subsequently, the methods in (L'Her *et*

al., 1998; Remelhe *et al.*, 2004; Bauer *et al.*, 2004*b*) convert SFC into verifiable timed automata (TA), and thus make the inclusion of quantitative time into the analysis possible. The drawback of the latter method is, however, that the cyclic execution mode of PLC is explicitly transferred to the TA model, resulting in a rather complex model and thus a high verification effort (see Sec. 2). This motivates, as the main contribution of this paper, a modified transformation procedure into TA (Sec. 4). The key idea is that the TA model is driven by plant events (rather than being triggered in every cycle), while the delay of the control actions, which results from the cyclic execution, is still considered (Sec. 3). It is shown in Sec. 5 for an example that the modified procedure can significantly reduce the computational effort for the verification.

2. VERIFICATION BASED ON TRANSFORMATION OF SFC INTO TA

Figure 1 sketches the overall procedure proposed to design logic controllers for production systems: The set of design specifications and an understanding of the plant behavior are first converted into a design of the logic controller. The use of SFC for formulating the controller is justified by the fact that the charts intuitively represent sequential and parallel executions, and clearly establish which control actions are assigned to different steps (see Sec. 5 for an example).

In order to investigate whether the control design complies with the specifications, the SFC is translated into a set of communicating timed automata. The plant behavior is also modeled by timed automata – it is stressed at this point that the investigation of functional safety of the controlled system makes it indispensable to employ a plant model, because most safety properties are expressed for the plant part of the system. The TA model of the plant triggers the execution of the controller by *events*, and a *control action* is returned from the controller, possibly with a delay. For the parallel composition of the two parts of the model, model checking can reveal whether a formalized representation of the specifications is satisfied. In the context of functional safety, a

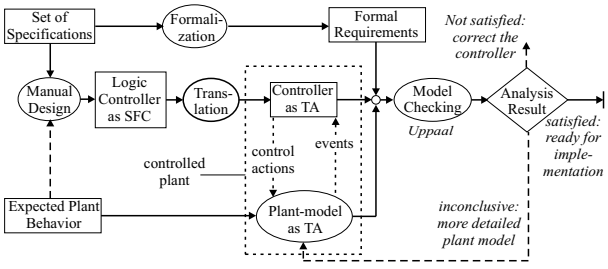

Fig. 1. Controller design procedure.

typical specification is to check whether any execution of the controlled systems leads to a safety-critical state. For timed automata, the answer can be obtained, e.g., by applying the tool UPPAAL (Pettersson and Larsen, 2000). If the verification shows that all relevant specifications are satisfied, the SFC controller can be implemented; otherwise it has to be corrected. If the violation of a specification is due to an insufficiently detailed plant model (what often can be inferred from the violating execution), the verification has to be repeated with a refined TA for the plant behavior. (If a TA model is not appropriate in general, one can resort to hybrid models and the techniques described in (Stursberg, 2000).)

A key step of the procedure is the transformation of the SFC controller into a TA model with equivalent behavior. As the basis of the transformation, the definition of SFC is first reviewed (see also (Int. Electrotechn. Commission, 2003) for an informal description and (Bauer *et al.*, 2004*a*) for a formal definition). For simplicity, the notion of hierarchy and history is omitted here:

Def. 1: A Sequential Function Chart is a tuple $SFC = (ST, s_0, X, G, T, A, \alpha, C)$ with:

- a finite set of steps ST and an initial step $s_0 \in ST$;
- a finite set X of variables, divided into input (X_{in}), output (X_{out}), and internal (X_{int}) variables;
- a set G of transition conditions;
- a finite set of transitions $T \subseteq (2^{ST} \setminus \{\emptyset\}) \times G \times (2^{ST} \setminus \{\emptyset\})$;
- a finite set A of actions a, each of which is a tuple $a = (q, \tau, o, f)$ consisting of a an action qualifier $q \in Q = \{N, S, R, P, P1, P0, L, D, SD, DS, SL\}$, a time quantifier $\tau \in \{\emptyset, t_v\}$ with $t_v \in \mathbb{Q}^{\geq 0}$, an execution flag f, and an operand o which is either an internal or output variable, or a function of an input variable;
- a function α: $ST \rightarrow B$ which assigns an action block $b \in B$ to each $s \in ST$, where $b(s)$ is an ordered subset of A;
- a finite set C of clocks which contains a clock c for each non-empty time qualifier.

The action qualifiers in Q denote : N - not stored, S - stored, R - reset, $P1$ - pulse when entering a step, $P0$ - pulse when leaving a step, P - pulse when entering and leaving a step, L - limited, D - delayed, SD - stored and delayed, DS - delayed and stored, and SL - stored and limited. A *configuration* of SFC is given by $\gamma = (V, ST_a, A_a, A_f, \Omega)$, where V are the current values of the variables in X, $ST_a \subset ST$ the set of active steps, $A_a \subset A$ the set of active actions, $A_f \subset A$ the set of final actions (i.e. actions that are executed a last time as they are associated with steps that were just deactivated),

and the valuations Ω of the clocks in C. A run $r_{SFC} = (\gamma_0, \gamma_1, \gamma_2, \ldots)$ of SFC is a sequence of configurations, with an initial configuration γ_0, and all subsequent γ_i are obtained from the cyclic execution of the PLC according to:

(1) receive inputs from the plant and store them as values of X_{in};
(2) program execution:
   (a) execute the actions in A_a and A_f according to a given total order (i.e., if $f = 1$ applies for the boolean execution flag); the result is an updated V;
   (b) determine which transitions can be executed depending on the conditions in G (which can involve time constraints for the clocks in C), on A_a, and on ST_a; for conflicting transitions, the execution is specified by a given priority; the result of this step is an updated set ST_a;
   (c) determine which actions become active or inactive, i.e., the sets A_a and A_f are updated; update the clock valuations Ω;
(3) emit the values of X_{out} as control actions to the plant;
(4) wait until a (possibly varying) cycle time $T_C \in [T_C^{min}, T_C^{max}]$ is elapsed since starting the cycle, then proceed with step (1).

The scheme for translating a controller given as SFC into TA according to (Bauer *et al.*, 2004*b*) can be summarized as follows: The SFC is first partitioned into syntactical units that represent either sequential chains (alternating sequences of steps and transitions) or parallel chains (two or more sequential chains that are enclosed by a parallel branching). The units are translated into separate TA which communicate via synchronization. The actions associated to the steps are modeled again by separate automata with clocks and time constraints if the corresponding action qualifiers involve time. The TA model of the controller is completed by a coordinator automaton which has the main function of establishing an execution corresponding to the cyclic execution of a PLC. Essentially, the coordinator triggers the different automata according to the order of the PLC cycle, i.e., each automaton synchronizes at least once per cycle with the coordinator. After termination of a cycle (the duration of which is modelled by a clock), the communication with the plant automata is scheduled. The following section proposes an alternative scheme.

3. AN EVENT-BASED REPRESENTATION OF THE PLC CYCLE

To discuss an event triggered controller model, the notation of the TA model is first clarified:

Def. 2: A timed automaton is defined as $TA = (Z, z_0, L, C, E, inv)$ with:

- the finite set Z of states with an initial state z_0;
- the set Lab of synchronization labels including an *empty* symbol ε;
- the finite set C of n clocks c;
- a finite set E of transitions $e = (z, z', l, g, \chi, \rho) \in E$ in which $z \in Z$ is the source state, $z' \in Z$ the target state, $l \in Lab$ a synchronization label, g a transition guard, $\chi \in \mathbb{R}^n$ a value assignment for all clocks in C, and ρ a function indicating which clocks are reset; g is a Boolean combination of inequalities $k_1 \cdot \chi(c) \sim k_2$ with $c \in C$, and $k_1, k_2 \in \mathbb{Q}$, and $\sim \in \{<, \leq, =, \geq, >\}$;
- and a function inv that assigns a Boolean combination of the same type as g to each state $z \in Z$.

A run of TA is a sequence $r_{TA} = ((z_0, \chi_0), (z_1, \chi_1), (z_2, \chi_2) \ldots)$, where (z_0, χ_0) is the initialization to z_0 and $\chi_0 := 0^n$, and a timed state (z_{i+1}, χ_{i+1}) follows from (z_i, χ_i) by executing $e = (z_i, z_{i+1}, l, g, \chi_i') \in E$ such that: TA synchronizes with another automaton if $l \neq \varepsilon$, $\chi_i' = \chi_i + \mu \cdot 1^n$ ($\mu \in \mathbb{Q}^{\geq 0}$) fulfills g, all χ_i'' with $\chi_i \leq \chi_i'' \leq \chi_i'$ fulfill $inv(z_i)$, and χ_{i+1} results from resetting the clocks indicated by ρ to zero and leaving the others unchanged.

Since the target of the model transformation discussed below is the type of TA used in the tool UPPAAL (Pettersson and Larsen, 2000), it is necessary to mention the special constructs available in UPPAAL: (a) additional variables can be defined; g can depend on these variables, and their values can be modified by transitions in E; (b) three different types of channels exist: (i) pairwise synchronization between a sender ($l!$) and a receiver ($l?$) for $l \in Lab$, (ii) urgent channels, for which the synchronized transitions of sender and receiver are taken at the earliest time possible, and (iii) broadcast communication, in which a sender synchronizes with an arbitrary number of receivers (if the latter are ready to synchronize); (c) the states $z \in Z$ can be marked as *urgent*, i.e. z must be left without delay, and as *committed*, meaning that z is left with highest priority and without delay.

A TA model according to this definition is obviously one, in which the execution is defined on dense time, and progress is triggered by communication among automata or transitions that depend on dense time constraints. However, the approach in (Bauer *et al.*, 2004*b*), maps the PLC cycle into a 'sampled time' construction – this is problematic because of very different timescales which correspond to the typical cycle time T_C of a PLC, and to the typical response time of production systems (in particular chemical processing systems) respectively. While T_C is usually chosen in the order of milliseconds, the frequency

of events in such plants is typically in the order of minutes, or even hours. As a consequence, a large number of cycles has to be processed between two consecutive events without any change of the sensor signals received by the controller, or the control actions. This means for verification, that the run r_{TA} contains long sequences of timed states, in which only the coordinator automaton changes its states, but not the automata representing the plant or the step-transition chains of the SFC. The resulting length of r_{TA} can make the verification of the controlled system inefficient or even infeasible with respect to computation time and memory consumption. Figure 2 shows this effect for an event signal $e(t)$ that marks the occurrence of two consecutive *plant events* e_{p1}, e_{p2}, i.e. transitions with synchronization labels are executed in the plant automata at event times t_{e1} and t_{e3}. For a controller TA that explicitly models the PLC cycle, the event e_{p1} is read at t_3, and a corresponding delayed *control event* (denoted by e_{c1}) at t_{e2} is sent to the plant (*write*) at t_7. The cyclic execution continues in the periods without events.

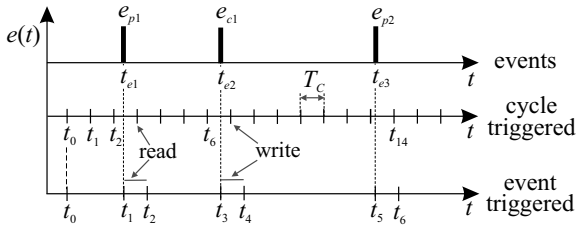

Fig. 2. Cycle and event-triggered execution.

As a more efficient way of modeling, the event driven scheme sketched on the lower abscissa in Fig. 2 is proposed: the event e_{p1} is processed by the controller in the time interval $[t_{e1}, t_{e1} + T_C]$, and the control action is returned to the plant for $t \in [t_{e2}, t_{e2} + T_C]$. The controller is triggered the next time only when another event e_2 occurs, i.e. all intermediate cycles are not modeled. Figure 3 shows how this concept can be realized with TA: In case of a plant event, the *trigger* automaton transitions into its right state, and emits a *trigger label* $\zeta?$ within the interval $[0, T_C]$ after the event. This label triggers a controller transition (leading to a control action, i.e. the direct manipulation of an $x \in X$, or the activation of an *action automaton*, see below). Furthermore, $\zeta!$ triggers an automaton which checks if the valuation (V') of the variables $x \in X$ has changed since the last triggering event. If so, the trigger automaton reacts to the control event e_c afterwards (υ marks an urgent state). This construction is an over-approximation of the cycle-triggered execution, since the update of the controller model is not fixed to the cycle time, but can occur at any $t \in [t_e, t_e + T_C]$ after an event.

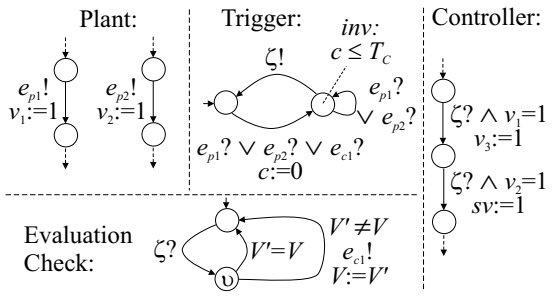

Fig. 3. Modeling the event-triggered execution.

4. EVENT-TRIGGERED TRANSFORMATION OF SFC INTO TA

Given an SFC according to Def. 1 and the target format according to Def. 2, the TA model with an event-based execution is obtained as follows:

(1) The SFC is partitioned according to the rules of the graph grammar in (Bauer *et al.*, 2004*b*).
(2) For each partition, one TA is introduced such that:
- each step of the partition, or each embedded partition respectively, is represented by a single state z of TA;
- the set Z is extended by an additional state that represents the inactivity of the respective partition;
- each transition (with a pair of ingoing / outgoing arcs) contained in the partition is mapped into one transition of E.

(3) The transition guards g are added such that:
- time conditions in SFC are translated into a pair of guard and invariant for the corresponding transition and state in TA;
- the input variables X_{in} of SFC are modeled by pairwise synchronization between a plant TA and the trigger automaton, and by variables as shown in Fig. 3.

(4) The actions A are first divided according to the action qualifiers into: *untimed* actions (*N,R,S,P,P1,P0*), *simple timed* actions (*L,D*), and *complex timed* actions (*SD,DS,SL*). The first two categories are directly embedded in the automaton for the respective partition, while complex timed actions are modeled by separate TA (see below). If an action manipulates a variable of X_{int} or X_{out} in SFC, a corresponding variable is introduced for TA, and it is set by the respective transition. If the action operand is an external function, the latter is modeled by a separate TA that synchronizes over channels.
(5) The trigger automaton and the evaluation check automaton are introduced as described before.

Figure 4 shows exemplarily how a parallel branching of the SFC is transformed into the TA model: When the transition condition g_1 is satisfied, and

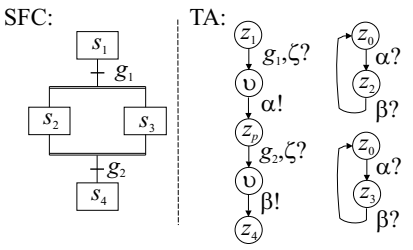

Fig. 4. Transformation of parallel branching.

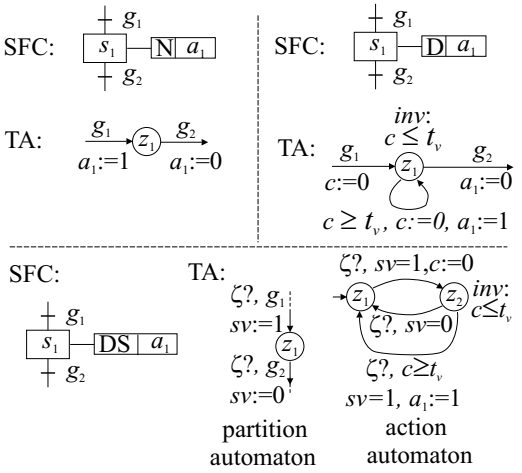

Fig. 5. Action qualifiers: N, D, DS.

the label $\zeta!$ is received from the trigger automaton, the left automaton first transitions into an urgent state, and then further into a state z_p, marking that the parallel branching is reached. The other two automata (one for each branch) synchronize with the latter transition upon receiving the label α, and leave the states z_0, which represent the inactivity of the two branches. Correspondingly, the synchronization over the label β means that the parallel branching is left, after the condition g_2 is satisfied. Figure 5 shows for examples of an untimed action (N), a simple timed action (D), and a complex timed action (DS) how the qualifiers are transferred into the TA model. (a_1 is an integer variable that is initialized to zero.)

5. APPLICATION TO AN EXAMPLE

The approach is illustrated for the example of a controlled batch evaporation system, as described in (Remelhe *et al.*, 2004). The system consists of two tanks T1 and T2 with heaters, a condenser, and pipes to fill T1, to empty T1 into T2, to drain T2, and to transfer vapor from T1 into the condenser. All pipes (except the last one) are equipped with on/off-valves. Furthermore, the tanks are equipped with sensors for monitoring if thresholds for the liquid levels, the temperatures, and the concentration of a dissolved substance are exceeded - this information defines the plant events received by the controller. An SFC controller for this system is shown in Fig. 6. The left branch models the nominal operation, i.e. T1 is filled with material, the latter is evaporated until a desired concentration is reached, then T1 is emptied into T2; this procedure is repeated twice, and finally T2 is emptied. The right branch models exception routines for the cases that either the heater in T1 or the condenser has a malfunction. The available control actions are to open or close the valves, and to switch the heaters and the condenser on or off.

The transformation of the controller using the method in Sec. 4, divides the SFC first into the partitions P0, P1, and P2, as indicated by the dashed boxes in Fig. 6. The subsequent transformation leads to 6 automata overall, three of which represent the partitions, and one the time-dependent action 'DS'. Figure 7 shows exemplarily the transformation result for the partition P2 (error handling).

The plant model includes automata for the liquid levels in T1 and T2, the heaters, the condenser, and the state of aggregation in T1. The automaton for the latter is shown in Fig. 8. Relevant verification tasks for this system are to check whether the following three formalized specifications are true for the parallel composition of all automata: (1) E⟨⟩ tank2.emptying, (2) A[ ] not liquid.crystallizing, and (3) A[ ] not liquid.overpressure. The first specification formulates that a run exists which leads to the state 'emptying' of T2, i.e. it checks whether the production goal can be reached. The second and third specifications formulate safety properties, namely that two critical states in the automaton in Fig. 8 are never reached. The TA model is complemented by an error automaton that emulates that either a heater malfunction (error1), or a condenser malfunction (error2) can occur at arbitrary times.

The verification with UPPAAL leads to the result that all three specifications hold true. In all three cases, the computation time is below 1 CPU-

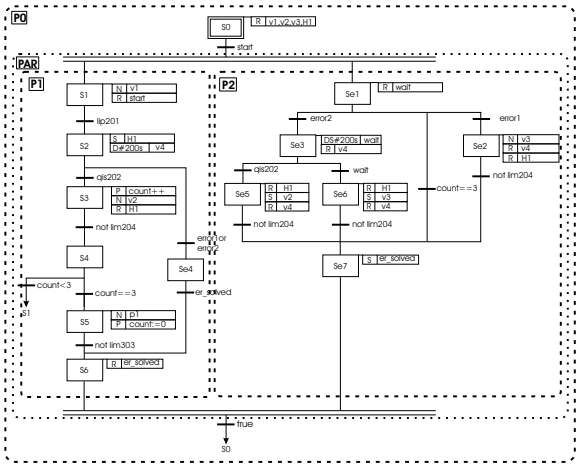

Fig. 6. Example: SFC controller.

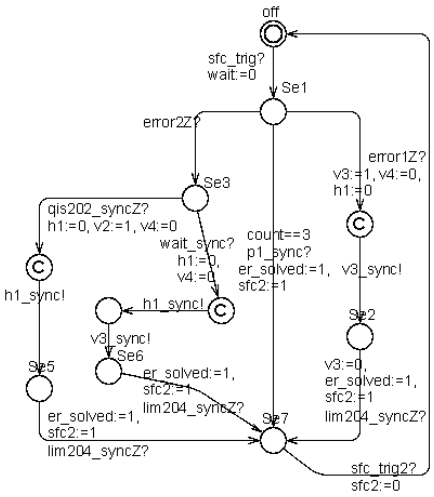

Fig. 7. TA model for the partition P2.

second, and the length of the run by which specification (1) was found to be true is 52 timed states. (A cycle time of 1 time unit was chosen.) For comparison, the verification of specification (1) for the same plant model but with the method described in (Bauer *et al.*, 2004*b*) terminated in 3.5 minutes on the same PC for a relatively large cycle time of $T_C = 50$. The corresponding run comprised 3248 timed states. (When using $T_C = 1$, the computation did not terminate within 1 hour.) In all cases, the outcome of the verification (i.e. the satisfaction of the specification) was identical.

6. CONCLUSION

The paper presented an approach by which SFC controllers with timed action qualifiers can be transformed into timed automata. The subsequent verification reveals whether the composition of the transformed controller and an appropriate plant model does meet dependability requirements like functional safety. As illustrated for an example, the gain in efficiency of the verification can be considerable if the method presented here is compared to the one in (Bauer *et al.*, 2004*b*). Future work includes to provide a tool for the automatic transformation of SFC into TA models.

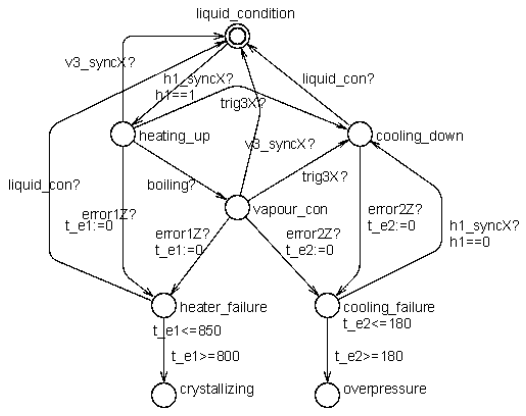

Fig. 8. TA model of the aggregation state of the liquid

7. REFERENCES

Avizienis, A., J.-L. Laprie and B. Randell (2000). Fundamental concepts of dependability. In: *3rd IEEE Workshop on Inform. Survivability*. pp. 7–12.

Bauer, N., R. Huuck, B. Lukoschus and S. Engell (2004*a*). A unifying semantics for sequential function charts. In: *Integr. Software Specif. Techn. for Applic. in Eng.*. Vol. 3147 of *LNCS*. Springer. pp. 400–418.

Bauer, N., S. Engell, R. Huuck, S. Lohmann, B. Lukoschus, M.P. Remelhe and O. Stursberg (2004*b*). Verification of PLC programs given as sfc. In: *Integr. Software Specif. Techn. for Applic. in Eng.*. Vol. 3147 of *LNCS*. Springer. pp. 517–540.

Bornot, S., R. Huuck, Y. Lakhnech and B. Lukoschus (2000). Verification of sequential function charts using SMV. In: *Proc. Int. Conf. on Parallel and Distr. Processing Techn. and Applic.*. pp. 2987–2993.

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

Int. Electrotechn. Commission (2002). *IEC 61508: Functional Safety of Progr. Electronic Safety-Related Systems*.

Int. Electrotechn. Commission (2003). *IEC 61131-3: Progr. Controllers - Progr. Languages*.

Lampérière, S. and J.J. Lesage (2000). Formal verification of the sequential part of PLC programs. In: *Discrete Event Systems*. Kluwer Acad. Publ.. pp. 247–254.

L'Her, D., P. Le Parc and L. Marce (1998). Proving sequential function chart programs using automata. In: *3rd Workshop on Automata Impl.*. Vol. 1660 of *LNCS*. Springer. pp. 149–163.

Moon, I., G. J. Powers, J. R. Burch and E. M. Clarke (1992). Automatic verification of sequential control systems using temporal logic. *AIChE Journal* **38**(1), 67–75.

Pettersson, P. and K.G. Larsen (2000). UPPAAL2K. *Bull. Europ. Assoc. for Theor. Comp. Science* **70**, 40–44.

Remelhe, M.P., S. Lohmann, O. Stursberg and S. Engell (2004). Algorithmic verification of logic controllers given as sequential function charts. In: *IEEE Conf. on Computer-Aided Control System Design*. pp. 53–58.

Stursberg, O. (2000). Analysis of switched continuous systems based on discretization. In: *Proc. 4th Int. Conf. Automation of Mixed Processes*. pp. 73–78.