Algorithmic Verification of Logic Controllers
given as Sequential Function Charts

M.P. Remelhe, S. Lohmann, O. Stursberg, S. Engell
Process Control Laboratory (BCI-AST), University of Dortmund
44221 Dortmund, Germany. Email: olaf.stursberg@uni-dortmund.de

N. Bauer
BASF AG, WLE/ED - L440,
67056 Ludwigshafen, Germany.

Abstract—The a-posteriori analysis of logic controllers can be a suitable means to detect design flaws if the controller was not developed by a synthesis algorithm that correctly considered all relevant requirements. This paper advocates the verification of logic controllers with a special focus on the following three issues: (a) the control code is given as a Sequential Function Chart (SFC), a description language becoming increasingly popular for industrial controllers; (b) the cyclic operation mode of the hardware on which the controllers is implemented is taken into account; (c) specifications of the control logic that include timers and the real-time behavior of the controlled plant are considered. We propose an approach in which the SFC controller is first translated into a timed automaton using an algorithm that explores a special graph grammar. The automaton can then be composed with a timed automaton modeling the plant behavior, and model-checking of the composition reveals whether a given set of requirements is fulfilled. All steps of the procedure are illustrated for the example of a controlled evaporation system.

I. INTRODUCTION

In manufacturing and processing systems, supervisory and logic controllers often represent the dominating part of the automation equipment. The principal task of logic controllers is to ensure a specific sequence of process steps and to initiate exception routines if a deviation from normal operation is monitored. Considering the importance of logic controllers for running many processes correctly, the low degree to which rigorous or algorithmic design procedures are used in industrial practice is somewhat undesirable. In many cases the design is simply an alternation between manually writing pieces of control code and testing by simulation or even at the real plant whether the desired behavior is obtained. As an alternative to this relatively error-prone and time-consuming procedure, the authors of this paper aim at developing a framework that realizes the complete procedure from collecting all relevant design specifications up to obtaining the control code in an as systematic as possible way, and to use formal models and algorithmic procedures wherever possible. The algorithmic verification, as a technique that proves for a formally specified system that properties of its dynamic behavior hold true or not, is a key component of this framework. Techniques for algorithmic verification have been studied extensively in computer science in recent years [1], [2], and have been successfully applied to analyze, for example, digital circuits and software [3]. In the context of controller design, the obvious purpose of verification is to verify that the controlled plant satisfies a given set of requirements.

Several approaches that employ verification for controller design can be found in literature, see for example [4], [5], [6], [7]. They differ with respect to the representation of the plant and controller, the properties that can be checked, and the computational techniques. Prominent classes of techniques are theorem proving, satisfiability checking, and model checking. We restrict our consideration in this paper to the last one, which proves, in a completely algorithmic way, whether a formal model that is specified as a state transition system satisfies a property written in computational tree logic (CTL). Important properties for controller design are: (i) no evolution encounters a state that is contained in an unsafe set (safety property), (ii) for each evolution of the model it is guaranteed that a set of goal states is eventually reached (possibly within a given period of time) or is periodically reached.

The type of transition system that is chosen to represent the controller and the plant dynamics determines the model checking technique. Existing techniques range from purely discrete models, over real-time systems to complex hybrid systems with nonlinear continuous dynamics. In this paper, we focus on the case that a timed transition system is sufficient to model the controller as well as the plant dynamics. For the controller model, we start from Sequential Function Charts (SFC) which graphically encode the logic and become increasingly popular in industry. In order to make this description amenable to model checking, we propose an algorithmic procedure to transform SFCs into Timed Automata (TA). The underlying idea of this transformation is to explore a graph grammar for the SFC and to create a structure of communicating automata from it. A particular issue in this transformation is to account for the cyclic scanning mode in which the control code is run on PLC hardware. The obtained set of TA can then be composed with a description of the plant behavior, also formulated by timed automata. Efficient model checking tools for TA are available and can be used to verify that the desired properties are satisfied [8].
II. CONTROLLER DESIGN BASED ON SEQUENTIAL FUNCTION CHARTS

The flowchart in Fig. 1 summarizes the overall design procedure considered in this paper. The starting point is a set of specifications for the operation of a plant, i.e., usually a sequence of steps that constitute the desired operation, as e.g. the production steps of a manufacturing plant. In addition, the specifications usually include a set of exception routines that specify control actions for the case that the nominal operation is disturbed, e.g., due to a device malfunction. Manually designing a controller for such specifications means to determine appropriate control actions that establish the desired steps for the nominal operation as well as for the exception modes. The designer must necessarily have an intuitive understanding of how the plant reacts to the control actions or to disturbances in the corresponding situations.

In order to represent the controller, SFCs are a suitable (and industrially accepted) means since they show sequential and parallel operations including control actions in a very transparent manner [9]. We briefly explain the modeling by SFCs for the example of a batch evaporation plant, which is taken from [10]. As sketched in Fig. 2, the plant consists of two tanks $T_1$ and $T_2$ with heating devices $H_1$ and $H_2$, a condenser $C_1$, a pump $P_1$, four on-off valves $V_1$ to $V_4$, and sensors for monitoring if thresholds for the liquid levels ($L_1$), the temperature ($T_1$), the concentration ($Q_1$), and the flow ($F_1$) are exceeded. The desired operation of this system is the following sequence: $T_1$ is first filled through $V_1$ with a liquid that contains a dissolved substance. The liquid is heated up in $T_1$ by the heater $H_1$ until the boiling point is reached. By further supplying heat, a certain amount of solvent is evaporated until the concentration of the liquid has reached a certain concentration. During the evaporation, vapor is condensed in $C_1$ which is cooled by a cooling agent that is supplied through $V_4$. When the evaporation is finished, the liquid is transferred from $T_1$ into $T_2$ by opening $V_2$. This procedure is repeated twice until $T_2$ is filled by three batches. The content of $T_2$, the product, is then pumped out of $T_2$ through $P_1$, and afterwards the complete operation can start again. Two different disturbance scenarios shall be considered for the process: (a) In case of a cooling failure (detected by $F_{IS101}$) the evaporation is continued for a short period of time and, if the concentration goal is not reached by then, the remaining content of $T_1$ is disposed through $V_3$; if the cooling failure occurs during heating up, the content is immediately disposed. (b) In case of a heating failure $T_1$ is also emptied immediately through $V_3$. In both cases the nominal operation can be resumed if the faulty devices are repaired or replaced.

A possible logic controller, given by an SFC, which is assumed to realize the described behavior, is shown in Fig. 3. Each step is denoted by a rectangle and a step identifier ($S_0$ is the initial step). If an action has to be carried out when a certain step becomes active, a so-called action block is associated with the step. It consists of a qualifier and a Boolean variable, where the qualifier defines in which way the variable is updated ($R$ - variable is reset when the step is reached, $S$ - set if the step is reached, $N$ - not stored, i.e. set to one only as long as the step is active, $P$ - pulse, $D_{#200s}$ - delayed set to one after 200 sec if the step is still active, $DS_{#200s}$ - delayed set after 200 sec irrespectively of the activity of the step). Two consecutive steps are separated by a transition which is marked by a bold horizontal line and labeled by a Boolean expression. If the latter evaluates to true the transition can be taken and the lower step becomes active. The variables that appear in the expressions assigned to transitions are either input variables (i.e. their values are received from sensors) or
internal variables. As an example for the latter, the internal variable count is incremented in step S3 and reset in S5, and its value determines by which transition S4 is left. The variables that are manipulated by action blocks are either internal variables or output variables, i.e., in the latter case they represent the control actions that are applied to the plant. If a step can be left by more than one transition, a so-called alternative branching, only one of the transitions is taken. If a transition is followed by a horizontal double line, as after the transition labeled by start, both following steps are reached. Such a parallel branching must be closed as shown by the additional horizontal double line below the steps S6 and Se7.

The SFC encodes the desired operation of the evaporator system as follows: in the initial step the valves V1, V2, and V3 are closed and the heater is switched off by resetting the corresponding boolean variables. The step is left when an operator sets an input variable start, and the parallel branches starting with S1 and Se1 are both activated. While the left branch accounts for the nominal operation, the right one encodes the behavior when one of the two error cases is encountered. In nominal operation, the system cycles three times through the sequence from S1 to S4. The tank T2 is filled with three batches of T1 at this stage, and is subsequently emptied in S5. If one of the errors is encountered during the heating or the evaporation phase the left branch leads to Se4. The right branch leaves Se1 through one of the transitions labelled by error1 (heating failure) or error2 (cooling failure). The actions assigned to Se2, or Se3, Se5 and Se6 respectively, correspond to the exception procedures described above. If the left branch has reached S6 and the right one has reached Se7, the parallel branching is left and the initial state is reached again.

Assuming that such an SFC controller is the result of manual design, the question is whether the controlled plant fulfills all requirements that one would pose for this system. The following two safety-related requirements are mentioned in [10]: (a) Can over-pressure occur for the vapor phase of T1 in case of a cooling breakdown? (b) For a malfunction of H1, is it possible that the temperature of the liquid in T1 drops below a critical value (determined by crystallization of the dissolved component) before T1 is emptied? In order to explain that model checking can be a suitable means to answer these questions, we resume the explanation of Fig. 1. Given the SFC controller, the design approach continues with an automatic transformation into a timed automaton. This step, as the focus of this paper, is described in detail in the next section. The motivation for the transformation is twofold: (i) There are currently no model checking tools available that operate directly on SFCs, whereas such tools exist for TA. (ii) The objective to analyze the behavior of the controlled plant requires to consider the behavior of the plant itself in an appropriate form. In case that the controller model does not contain any timed function, it can be sufficient to model the controller and the plant by a purely discrete automaton and to verify
the composition of both. This approach is taken in [11], [12]. The controller of the evaporator contains the time-dependent action qualifiers $DS \#200s$ and $D \#200s$. In order to consider the interaction of the controller and the plant correctly, the plant evolution has to be modelled with a quantitative notion of time, too. Thus timed automata, as originally introduced in [13], are suitable here.

Assuming that a TA controller was obtained from the transformation and that a TA model sufficiently describes the plant behavior, both parts can be composed using standard composition operators. The set of initial specifications is formalized such that it corresponds to requirements formulated for the evolution of the composed TA model. Tools like UPPAAL can then be used to verify or to falsify these requirements [8]. The outcome of this step is one of the following (Fig. 1): either the model satisfies all requirements such that also the SFC controller complies with all initial specifications, or at least one requirement is violated. If the latter is due to a design failure the control logic has to be corrected, and the verification is repeated until all requirements are satisfied. If the requirement violation is due to an insufficiently detailed plant model, the latter has to be refined.

### III. TRANSFORMATION INTO TIMED AUTOMATA

The transformation of the SFC controller into a TA model requires a precise syntax and semantics of the source format. Since the standard [9] defines the semantics of SFCs vaguely in some points [14], we refer to the formal definition in [14]. As for the formal definition of the target format (TA), we refer to the one in [8], since we employ the tool UPPAAL for model checking. In comparison to the standard definition of TA consisting of locations, clocks, transitions (including guards and clock resets), synchronization labels, and invariants, the UPPAAL model is extended by the following elements: (i) The communication between several TA can be established through channels for pairwise synchronization, through broadcasting to synchronize the transition of a sender with the transitions of an arbitrary number of receivers, and by shared variables, i.e., Boolean or integer variables that are set with the transition of one TA and occur in the transition guard of another. (ii) The locations can be labeled as urgent (u) or committed (c). While both types of locations must be left without delay after being reached, committed locations have higher priority for being left than the other locations.

A procedure for transforming SFCs into TA is described in [15]. Building on this procedure, we here start from an object-oriented representation of the SFC and use the principles of graph grammars to algorithmically convert the graphical structure into a set of communicating TA. The starting point is an SFC-model that has been modeled in a special graphical editor for SFCs. The editor uses a datastructure which essentially is a bipartite graph that links the relevant graphical elements of the SFC, i.e., symbols for steps, the initial step, transitions, action blocks, double horizontal lines for parallel branching, single horizontal lines for alternative branching, directional arrows, and optionally priority numbers for transitions. Two of these objects can be linked if and only if the result is a proper SFC (e.g., two transitions are not compatible). Each action block consists of an action qualifier (from the set N, R, S, L, D, P, DS, SD, SL; see [9] for explanation), an integer number specifying a duration if the qualifier contains an L or D, and an action. The latter is either the name of a variable (the value of which is newly assigned), or the call of a function or of another SFC. Using these conventions, an SFC-graph is established as a network of steps, transitions, branching elements, and action boxes. The complete SFC-model can be a collection of SFC-graphs, one of which is defined as the main graph. The other graphs of an SFC-model are subordinated and are executed as actions. Each graph can (de-)activate another one – it is important, however, that no cyclic dependencies occur for the (de-) activation. The main graph itself cannot be deactivated by any subgraph.

The transformation of the SFC into timed automata is performed by parsing the graphic representation. This provides a check of the SFC for syntactical correctness and the possibility to easily construct the automata from the result of the parsing procedure. In detail, the transformation comprises the following steps:

1. checking the identifiers of steps and actions for syntactical correctness;
2. analyzing whether the SFC graphs comply syntactically with the conventions mentioned above;
3. converting the SFC graphs into a parse graph according to the reduction rules of a graph grammar;
4. generation of the TA-model based on the parse graph.

In the third step, the SFC model is parsed in a top-down manner and is divided into syntactical units, in which at most one step is active at a time. These units, called partitions, consist of an alternating sequence of steps and transitions but do not contain any parallel branching. Parallel partitions are replaced by a new step (named parallel block step) that represents the activity of the enclosed parallel branches. These transformations are specified by the graph grammar rules listed in Tab. 1. The meaning of these rules is as follows:

- (R1) forming of an initial partition,
- (R2) introduction of parallel partitions,
- (R3) extension of a partition by a transition and a subsequent step,
- (R4) extension of a transition by a self-loop transition,
- (R5) forming of a parallel block step combining all subsequent parallel partitions of one transition,
- (R6) introducing a parallel block step combining all parallel partitions that lead to one subsequent transition.

The iterative application of these rules with a decreasing priority from (R1) to (R6) reduces any regular SFC into a single partition. A regular SFC is one in which the branches
TABLE I
REDUCTION RULES OF THE GRAPH GRAMMAR.

<table>
<thead>
<tr>
<th>Rule No.</th>
<th>Graph element</th>
<th>replaced by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S0</td>
<td>Partition</td>
</tr>
<tr>
<td>2</td>
<td>Step1, StepN</td>
<td>T1a, T1b, Tnb</td>
</tr>
<tr>
<td>3</td>
<td>Partition</td>
<td>T0, T1, T2, T3, T4</td>
</tr>
<tr>
<td>4</td>
<td>ParallelBlockStep</td>
<td>T1, T2, T3, T4</td>
</tr>
<tr>
<td>5</td>
<td>Partition</td>
<td>T0, T1, T2, T3, T4</td>
</tr>
<tr>
<td>6</td>
<td>ParallelBlockStep</td>
<td>T1, T2, T3, T4</td>
</tr>
</tbody>
</table>

originating from two different parallel branchings do never interact (i.e., a branch starts and ends in those two double horizontal lines that open and close one parallel branching).

In Fig. 4 the partitions generated by this procedure for the controller example are shown: P0 is the initial and final partition containing the complete SFC; the parallel block step containing the parallel branching is denoted by PAR, and it contains the partitions P1 and P2, which represent the two parallel branches.

The parse graph obtained from applying the reduction rules is the basis for constructing the TA model in the input format of the tool UPPAAL. (Technically, the model is generated in XML-format which can be read by UPPAAL.) The essential steps for the construction are as follows:

- a separate TA is introduced for each partition; each of these TAs (except of the one for the final partition) contains a location ‘off’ representing deactivation of the partition;
- if a partition contains a subordinated one, the latter is represented by a single location in the TA of the first partition;
- steps and transitions inside of one partition are correspondingly modeled by locations and transitions in the TA such that the logics is preserved;
- transitions that are connected to a parallel branching of the SFC are modeled in the TA by the sequence of urgent locations with intermittent transitions. The first transition is labeled by the guard which occurs in the SFC, and subsequent transitions realize the synchronization with the automata that model the parallel branches;
- for each action block a separate TA is introduced which models the functionality of the action qualifier.

Fig. 4 shows the automata obtained for the partitioned SFC in Fig. 3. The location PAR in P0 models that the two parallel branches are activated, i.e., P1 and P2 have left 'off'.

An important point in the TA representation of the controller is the consideration of the operation mode of the hardware on which SFC-programs are usually run. Programmable Logic Controllers (PLCs) work in a cyclic scanning mode where a cycle consists of the following steps: (i) read the values of the input variables (the sensor information), (ii) execute the SFC, i.e., execute the actions of active steps, update the values of internal variables, and execute enabled transitions, (iii) update the output variables, (iv) wait until the cycle time is over and proceed with (i).
In order to convey this principle to the TA model, the following construction is chosen: An additional TA, called **coordinator**, is introduced. It triggers the other controller automata according to the cyclic fashion described above. The coordinator is essentially a deterministic automaton that first runs through a sequence of transitions that synchronize with the partition automata and the action automata, and it waits then until the cycle time is elapsed. Most of the locations are of the **committed** type to enforce that the synchronizing automata progress in their evolution. As an example for a coordinator automaton, Fig. 5 shows the coordinating TA for the evaporator controller.

**IV. VERIFICATION OF THE EXAMPLE**

To accomplish the verification task posed in Sec. II for the evaporator system, the result of the transformation (the deterministic automata for the partitions, the coordinator, and the action control blocks) are composed with a non-deterministic plant model. The latter comprises one TA each for the two tanks, the heater of tank \( T_1 \), the condenser, and the state of aggregation of the fluid in \( T_1 \). Exemplarily, Fig. 6 shows the TA model of \( T_2 \). The communication of all components is established through channels, broadcast communication, and shared variables.

The two plant situations named as critical in Sec. II (over-pressure and crystallization of material in \( T_1 \)) are explicitly modeled as locations in the TA for the state of aggregation in \( T_1 \). A safe operation of the plant is formulated by a CTL formula specifying that no evolution of the TA does lead into these locations. A cooler breakdown (as a precondition for the occurrence of the two critical plant situations) is included in the condenser model such that it can happen within any time interval. The verification with Uppaal leads to the results that the critical states are unreachable, i.e. the SFC controller shown in Fig. 3 meets the requirements. The analysis was completed within two minutes on a 1.5 GHz Pentium-IV PC.

**V. CONCLUSIONS**

The approach presented here establishes a link between the industrially relevant controller language SFC and the verification tool Uppaal, arguably the currently most efficient one for TA. This link enables the completely algorithmic analysis of logic controllers with timers once an SFC design and a plant model are available.

For a successful application of the method, two issues are important: (i) The explicit transformation of the cyclic PLC-mode into TA can considerably enlarge the size of the composed model and thus the complexity of verification. While the consideration of the cyclic mode is apparently necessary when the plant runs on a similar time scale as the PLC-cycle, it seems possible to abstract away from it if the plant behavior is much slower. (ii) The accuracy of the plant model is obviously crucial for successfully verifying or falsifying a system property. If a plant model given as TA is not sufficiently detailed, a reasonable approach is to model the plant by a more complex hybrid system, and to use the techniques described in [16] to automatically derive a TA from the hybrid model, or to use one of the verification methods for hybrid systems referred to in [17].

**REFERENCES**


![Fig. 6. The TA model for the tank \( T_2 \).](image-url)