Hybrid Sequential Function Charts

Johanna Nellen
RWTH Aachen University
Germany
johanna.nellen@cs.rwth-aachen.de

Erika Ábrahám
RWTH Aachen University
Germany
abraham@cs.rwth-aachen.de

Abstract

Sequential function charts are a popular formalism to specify programmable logic controllers. However, in the absence of the controlled system, verification of sequential function charts can only consider the controller's behavior, but cannot tell anything about the controlled system.

In this paper we propose an extension of the language to additionally model the continuous dynamics of the controlled system. We give syntax and semantics of this hybrid extension for sequential function charts and define a reachability-preserving transformation to hybrid automata.

1. Introduction

In automation *programmable logic controllers* (PLCs) are widely used to control the behavior of a plant. In the industry standard IEC 61131-3 [Int03] several languages are specified which can be used for the programming of a PLC. Commonly used in process control are *sequential function charts* (SFCs), a graphical language which allows the structuring of control sequences into several steps or into branches that are executed in parallel.

Since PLC-controlled plants are often safety-critical, SFC *verification* has been extensively studied [FL00]. There are several approaches which consider either an SFC in isolation or the combination of an SFC with a model of the plant [HKD98, BCMP]. The latter approaches usually define a timed or hybrid automaton that specifies the SFC, and a hybrid automaton that specifies the plant. Building the composition of these two models gives a hybrid model of the controller acting in the plant. Existing tools for hybrid automata analysis can be used for verification. Since the models are in general too large to analyze, also some CEGAR-based abstraction techniques were proposed in, e.g. [ELS05]. That work builds the composition of the SFC and the plant models, but abstracts away from parts of the continuous dynamics.

However, modeling a whole plant by a single hybrid automaton is a complex and erroneous procedure. Furthermore, the resulting hybrid automaton does not allow to extract behaviors of single plant components, which would be very helpful for abstraction and its refinement.

Instead of this global modeling approach, we propose to specify the dynamic behavior of plant components by sets of *conditional ordinary differential equation (ODE) systems*. Each conditional ODE system specifies the behavior of plant components by the ODE system in case its condition holds. The condition expresses assumptions about the current state of the system. For example, the dynamic change of the water level in a tank can be given as the sum of the flows through the

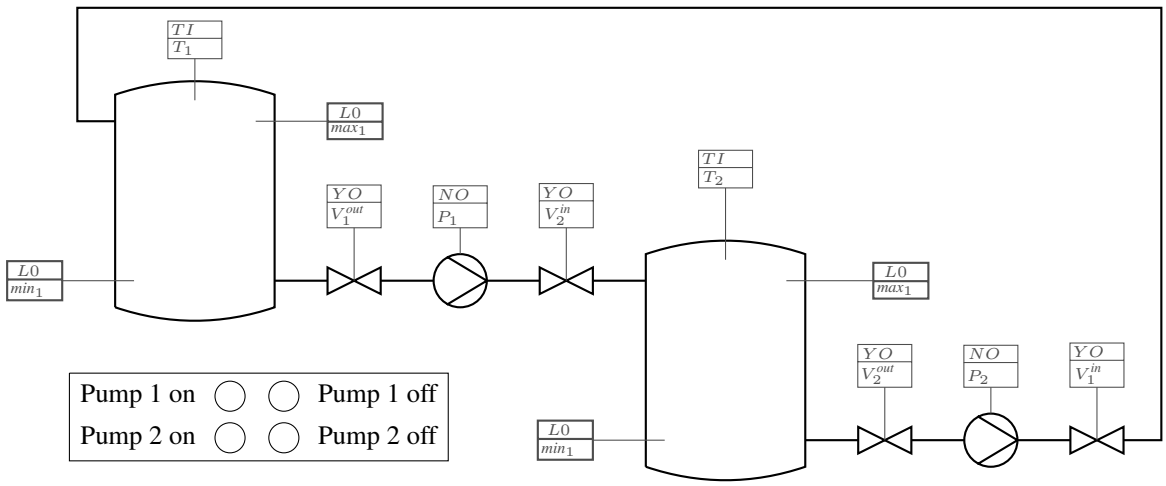

Figure 1: An example plant and its control panel.

pipes that fill and empty the tank. This sum may vary depending on valves being open or closed, pumps being on or off, and other tanks being empty or not.
We extend SFCs by attaching such conditional ODE systems to its steps, resulting in *hybrid SFCs (HSFCs)*. A hybrid SFC specifies, besides the controller behavior, also the plant behavior in dependence of the controller state. Such a specification is intuitive, and the synchronization between the plant and its controller needs not to be specified explicitly. Furthermore, our formalism allows a modular definition of the dynamic continuous behavior of plant elements with different levels of details for different control modes.
In this paper we formalize the semantics of HSFCs and give a transformation to hybrid automata. Using this transformation we can apply existing tools for hybrid automata analysis to check properties of the hybrid models.

The paper is structured as follows: We introduce SFCs and hybrid automata in Section 2. We extend SFCs to HSFCs in Section 3. The transformation from HSFCs to hybrid automata is given in Section 4. We conclude the paper in Section 5.

2. Preliminaries

Plants In this paper we focus on *chemical plants*. A simple example plant that we use as a running example is depicted in Figure 1.
There are two cylindrical tanks connected by pipes. Assume that the left tank T_1 has water level h_1 and the right tank T_2 water level h_2. Each tank T_i has two sensors min_i and max_i that observe the water level; the value *true* represents that the sensor senses water, the value *false* the opposite. When water reaches the upper sensor, the tank starts to flood and when the water level falls below the *min* sensor, the corresponding tank is empty. Each pump P_i can be turned on (P_i) or off ($\neg P_i$). When turned on, the pump P_1 pumps water from the left to the right tank with a flow that decreases the water level h_1 by c_1 per time unit. The second pump P_2 pumps a flow in the other direction causing a height increase of c_2 for the water level h_1 of the tank T_1 per time unit.
The filling and emptying of the tanks is controlled by the operator panel which allows to switch the pumps on or off manually. When the operator signals to switch a pump on, the internal control

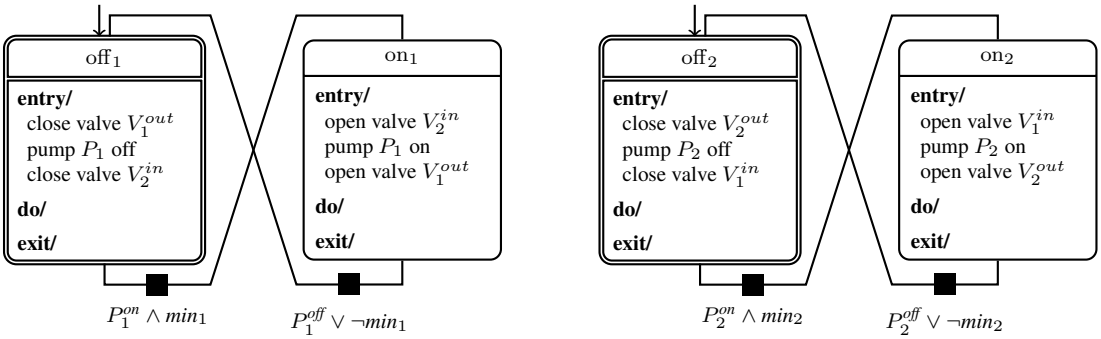

Figure 2: Parallel SFCs for the example tank system

program sends the corresponding command to the actuator of the pump. Our controller for this plant couples the pumps and the valves, e.g., turning a pump on opens the valves on both sides of the pump. Turning off works analogously. The controller also prevents the tanks from running dry: Pumps can be turned on only if the water level of the tank from which water should flow out is above the lower sensor. If the water level of a tank falls below the lower water sensor during pumping, the pump is switched off and its connected valves get closed.

SFC Syntax To specify internal control programs we use *sequential function charts (SFCs)*, defined in the industry norm IEC 61131-3 [Int03]. Our definitions follow [BHLE04]. The control program for our example plant is specified by the two parallel SFCs in Figure 2.

An SFC has a set of typed variables $Var = Var_I \uplus Var_O \uplus Var_L$, classified into input, output and local variables (the norm supports e.g. the elementary data types integer, real, boolean, string, time and date).

An SFC has a set of *steps* and a set of guarded *transitions*, connecting the bottom of a source step with the top of a target step. A distinguished initial step is active at start. A transition can only be taken if its source step is active and the transition *guard* evaluates to true; taking a transition makes its source step inactive and its target step active. We use G_{Var} for the set of guards over the variables *Var*, evaluating to true or false when fixing the values of the variables. A partial order on the transitions defines *priorities* for concurrently enabled transitions. Transitions are *urgent*, i.e., a step is active only as long as there are no enabled outgoing transitions. Apart from transitions that connect two steps, also parallel branching can be specified by defining multiple source/target steps.

Each step contains a set of *action blocks* specifying the actions that are performed during the activation period of the step. An action block $b = (q, a)$ is a tuple with an *action qualifier* q and an *action* a. The set of all action blocks using actions from the set *Act* is denoted by B_{Act}.

The action qualifier $q \in \{entry, do, exit\}$ specifies when the corresponding action is performed[1]. When control enters a step, its entry and do actions get executed. As long as control stays in a step, the do actions are executed repeatedly. The exit actions are executed at deactivation of the step.

An action a is either a variable assignment or an SFC. Executing an assignment changes the value of a variable, executing an SFC means activating it. In the latter case, if the *history* flag of the SFC is false then its initial step becomes active, otherwise its last active step is re-activated.

[1] In the IEC standard, the qualifiers $P1, N$ and $P0$ are used instead of *entry*, *do* and *exit*. The further qualifiers of the standard are not considered in this paper.

Definition 2.1 (SFC Syntax) *An* SFC $C = (\mathit{Var}, \mathit{Steps}, \mathit{Act}, s_0, \mathit{Trans}, \mathit{Blocks}, \sqsubset, \prec, \mathit{Hist})$ *is a tuple, where*

- Var *is a finite set of* variables*;*
- Steps *is a finite set of* steps*;*
- Act *is a finite set of* actions *referring to variables from Var in assignments and to SFCs whose variable and action sets are subsets of Var and Act resp. and whose action order is a subset of* $\sqsubset$*;*
- $s_0 \in \mathit{Steps}$ *is the* initial step*;*
- $\mathit{Trans} \subseteq (2^{\mathit{Steps}} \setminus \{\emptyset\}) \times G_{\mathit{Var}} \times (2^{\mathit{Steps}} \setminus \{\emptyset\})$ *is a finite set of* transitions *(transitions with multiple target/source steps define the begin/end of parallel branching);*
- $\mathit{Blocks} : \mathit{Steps} \to 2^{B_{\mathit{Act}}}$ *is a function which assigns a set of* action blocks *to each step;*
- $\sqsubset \subseteq \mathit{Act} \times \mathit{Act}$ *is a* total order on the actions*;*
- $\prec \subseteq \mathit{Trans} \times \mathit{Trans}$ *is a* partial order on the transitions*;*
- $\mathit{Hist} \in \{0, 1\}$ *is a* history *flag (*$\mathit{Hist} = 1$*: SFC with history,* $\mathit{Hist} = 0$*: SFC without history).*

Given a set of transitions T, we use $\mathit{source}(T)$ resp. $\mathit{target}(T)$ to denote the union of all source resp. target steps of transitions from T. We use $\overline{C}$ for the set containing the SFC C and all of its nested SFCs at all depths, $\mathit{Steps}(\overline{C})$ for the union of all steps of the SFCs in $\overline{C}$, and $\mathit{Trans}(\overline{C})$ for the union of all transitions of the SFCs in $\overline{C}$.

SFC Semantics In this section we recall the formal semantics of SFCs from [BHLE04, Luk05][2]. An SFC runs on a programmable logic controller (PLC) that performs the following steps in a cyclic way:

1. Get the input data from the environment and update the values of the variables accordingly.
2. Determine the set of transitions to be taken and execute them.
3. Determine the actions to be performed and execute them in priority order.
4. Send the output data to the environment.

Between two PLC cycles there is a time delay, which can be different for different cycles but globally bound by a lower bound δ_l and an upper bound δ_u. The first and last steps of the PLC cycle implement the communication with the environment. To specify the second and the third step of the cycle, we first define states and configurations.
In the following, let $C = (\mathit{Var}, \mathit{Steps}, \mathit{Act}, s_0, \mathit{Trans}, \mathit{Blocks}, \sqsubset, \prec, \mathit{Hist})$ be an SFC and let D be the union of all data type domains. A *state* $\sigma \in \Sigma$ of C is a function $\sigma : \mathit{Var} \to D$ that assigns a value from its domain to each variable $v \in \mathit{Var}$. A *state transformation* $f \in F$ is a function $f : \Sigma \to \Sigma$. A configuration $(\sigma, \mathrm{readyS}, \mathrm{activeS}, \mathrm{activeA}) \in \Sigma \times \mathit{Steps}(\overline{C}) \times \mathit{Steps}(\overline{C}) \times \mathit{Act}^*$ stores, besides the state σ of the SFC, the following elements (they are formally specified by the semantics of SFCs given below):

- The set readyS of *ready steps* contains all active steps of the top-level and the nested SFCs together with the last active steps of currently inactive nested SFCs with history. We say that control resides in these steps.

[2]The semantics of SFCs is partly PLC-dependent. We slightly adapted the semantics from [BHLE04, Luk05] according to a certain PLC.

- The set $\mathrm{activeS}$ contains the *active steps* of the top-level SFC and the active steps of those nested SFCs, to which an active action points.
- *Active actions* are the do actions of active steps and the exit/entry actions of the source/target steps of the taken transitions. The sequence $\mathrm{activeA}$ is the list of the active actions sorted according to decreasing action priorities. It specifies the actions that are executed in the next PLC cycle.

We use *Conf* for the set of all configurations. The initial configuration of an SFC is $(\sigma_0, \{s_0\}, \emptyset, \emptyset)$, where σ_0 assigns initial values to the variables (e.g. false to booleans and 0 to numerical variables if no initial value is specified) and s_0 is the initial step of the SFC.
For an SFC C and a configuration $c = (\sigma, \mathrm{readyS}, \mathrm{activeS}, \mathrm{activeA})$ of C, we define the set of enabled and taken transitions as follows:

$$\begin{array}{lcl} \textit{enabled}(C,c) & = & \{(S,g,S') \in \textit{Trans}(\overline{C}) \mid S \subseteq \mathrm{activeS} \wedge c \vDash g\} \\ \textit{taken}(C,c) & = & \{t = (S,g,S') \in \textit{enabled}(C,c) \mid \forall t_1 = (S_1,g_1,S_1') \in \textit{enabled}(C,c). \\ & & \qquad S \cap S_1 = \emptyset \vee t_1 \prec t\} \end{array}$$

We define the operational semantics of SFCs by configuration changes during PLC cycles. In this paper we do not formalize the communication with the environment (the first and the last step of the PLC cycle), but focus on the computation steps inbetween: computation steps start with a configuration $(\sigma, \mathrm{readyS}, \mathrm{activeS}, \mathrm{activeA})$ where σ has already been updated with the new input data from the environment. We show how to compute the configuration $(\sigma', \mathrm{readyS}', \mathrm{activeS}', \mathrm{activeA}')$ that is obtained after the execution of Steps 2. and 3. of the PLC cycle. The output data that is sent to the environment in the last step of the PLC cycle can be extracted from the updated state σ'.
If we remove the source step of all taken transitions and add the target step of those transitions afterwards, we obtain the new set of ready states. The new sets of active steps and active actions are recursively computed by the function computeActiveSets. The function starts to compute the active sets for the top-level SFC and then recursively adds the active sets for each nested active SFC. Afterwards, the set of active actions is sorted according to the given priorities and than performed in the specified order.

Definition 2.2 (SFC Semantics) *The semantics of SFCs is defined by the transition relation* $\rightarrow \subseteq$ *Conf* $\times$ *Conf with* $c = (\sigma, \mathrm{readyS}, \mathrm{activeS}, \mathrm{activeA}) \rightarrow (\sigma', \mathrm{readyS}', \mathrm{activeS}', \mathrm{activeA}') = c'$ *if and only if*

- $\mathrm{readyS}' = (\mathrm{readyS} \setminus \textit{source}(\textit{taken(C,c)})) \cup \textit{target}(\textit{taken(C,c)})$,
- $(\mathrm{activeS}', \mathrm{unsortedActiveA}') = \mathit{computeActiveSets}(\mathrm{readyS}', \emptyset, \emptyset, C, c, \mathrm{activeA} \cap \overline{C})$,
- $\mathrm{activeA}' = \textit{sort}(\mathrm{unsortedActiveA}', \sqsubset)$, *and*
- $\sigma' = (a_1 \circ \ldots \circ a_m)(\sigma)$ *where* $\mathrm{activeA}' = a_m \circ \ldots \circ a_1$.

The function computeActiveSets is listed on page 6.

Hybrid Automata We recall the definition of hybrid automata from [ACH+95]. Since we do not use the parallel composition of hybrid automata in this paper, we skip the components that are relevant for the composition only.

Definition 2.3 *A* hybrid automaton $H = (\mathit{Loc}, \mathit{Var}, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init})$ *is a tuple where*

Function computeActiveSets(readyS′, activeS, activeA, C, c, activeSFCs)

input : readyS′, activeS, activeA,
$C = (\mathit{Var}, \mathit{Steps}, \mathit{Act}, s_0, \mathit{Trans}, \mathit{Blocks}, \sqsubset, \prec, \mathit{Hist})$, c, activeSFCs
output: activeS′, activeA′

```
/* Add the local active steps of C to activeS'.                    */
```

if $\mathit{Hist} = 1 \vee C \in$ activeSFCs **then**
    activeS′ := activeS ∪ (*Steps* ∩ readyS′);
else
    activeS′ := activeS ∪ $\{s_0\}$;

```
/* Collect the local active actions and their qualifiers.          */
```

activeA′ := activeA;
foreach $s \in \mathit{Steps}, b = (q, a) \in \mathit{Blocks}(s)$ **do**
    if $(q{=}\mathit{exit} \wedge s{\in}\mathit{source}(\mathit{taken(C,c)})) \vee (q{=}\mathit{entry} \wedge s{\in}\mathit{target}(\mathit{taken(C,c)})) \vee (q{=}\mathit{do} \wedge s{\in}\text{activeS}')$
    then
        activeA′ := activeA′ ∘ a;
end

```
/* Compute activeA' and activeS' for each active nested SFC        */
```

foreach $s \in \mathit{Steps} \cap$ activeS′, $b = (q, a) \in \mathit{Blocks}(s)$ **do**
    if $a \in \overline{C}$ **then**
        (activeS′, activeA′) :=
        computeActiveSets(readyS′, activeS′, activeA′, a, c, activeSFCs);
end
return (activeS′, activeA′);

- *Loc is a finite set of* locations*;*
- *Var is a finite set of real-valued* variables*; A* valuation $\nu \in V$, $\nu : \mathit{Var} \to \mathbb{R}$ *assigns a value to each variable. A* state $s \in \mathit{Loc} \times V$ *is a location-valuation pair;*
- $\mathit{Edge} \subseteq \mathit{Loc} \times 2^{V^2} \times \mathit{Loc}$ *is a set of* edges*;*
- *Act is a function assigning a set of time-invariant* activities $f : \mathbb{R}_{\geq 0} \to V$ *to each location, i.e.,* $\forall l \in \mathit{Loc} : f \in \mathit{Act}(l)$ *implies* $(f + t) \in \mathit{Act}(l)$ *where* $(f + t)(t') = f(t + t')$ *for all* $t, t' \in \mathbb{R}_{\geq 0}$*;*
- $\mathit{Inv} : \mathit{Loc} \to 2^V$ *is a function assigning an* invariant *to each location;*
- $\mathit{Init} \subseteq \Sigma \times V$ *is a set of* initial states.

The activity sets are usually given in form of an ordinary differential equation (ODE) system, whose solutions build the activity set.
The semantics of hybrid automata distinguishes between *discrete steps (jumps)* and *time steps (flows)*. A discrete step follows an edge leading from one location to another with a standard semantics. Time steps model time elapse; the values of the variables evolve according to the activities in the current location, where the location's invariant must not be violated.

3. Plant specifications and HSFCs

To specify the dynamic continuous behavior of a plant, we use ordinary differential equation (ODE) systems. These equations can be derived from the hardware specifications and the set-up of the plant. Since the behavior of the plant may be state-dependent, we attach conditions to the ODE systems. We use these *conditional ODE systems* to annotate the steps of an SFC, resulting in *hybrid SFCs (HSFCs)*.
A *conditional ODE system* is a pair of a condition and a set of ODEs. The condition expresses assumptions about the overall system state. The ODE system describes the dynamic behavior of the plant (or a part of the plant) under the assumption that the condition holds. The semantics allows chaotic behavior for those variables whose evolution is not fixed by the ODE system.

Definition 3.1 (Conditional ODE System) *Let* ODE_{Var_C} *be the set of all ordinary differential equations over* Var_C *and* Conds *the set of all conditions. A* conditional ODE system *is a pair* $(\mathrm{cond} : \mathrm{ODEs})$ *with* $\mathrm{cond} \in \mathrm{Conds}$ *and* $\mathrm{ODEs} \subseteq \mathrm{ODE}_{Var_C}$.
The set of all conditional ODE systems over Var_C *is denoted by* CODE_{Var_C}.

We illustrate the use of conditional ODE systems on the water level of the tank T_1 in Figure 1. Let us assume that the pipes connecting the two tanks are designed for the delivery rate of the pumps, so that both pumps can work to their capacity. From the hardware specification of pump P_1 we can calculate that the water level of T_1 is decreased by c_1 units per time unit, if the pump is operating. The corresponding ODE is $\dot{h}_1 = -c_1$. Analogously, we derive the ODE $\dot{h}_1 = c_2$ if pump P_2 is working and T_1 is not full. Accordingly, the water level of T_1 changes by $\dot{h}_1 = c_2 - c_1$ units per time unit if both pumps are working and T_1 is not full, and finally the water level stays the same ($\dot{h}_1 = 0$) otherwise.
The conditions for the ODE systems of the tanks depend on the pumps being switched on or off and the sensor values that monitor the water level: P_1 pumps water from T_1 to T_2 if P_1 is switched on and the water level of T_1 is above the minimum. If P_1 is switched off or if T_1 is empty then no water can flow through P_1. The pump P_2 works analogously. In order to attach the right ODE system to the conditions, we have to differentiate which pump is working according to the condition.
The conditional ODE systems for tank T_1 is shown below:

$$\begin{array}{rrcll} & (\ P_1 \wedge & \mathit{min}_1) & \wedge\ (\neg P_2 \vee \neg \mathit{min}_2) & : \ \dot{h}_1 = -c_1 \\ \neg \mathit{max}_1 \wedge & (\neg P_1 \vee \neg \mathit{min}_1) & \wedge & (\ P_2 \wedge\ \mathit{min}_2) & : \ \dot{h}_1 = c_2 \\ \neg \mathit{max}_1 \wedge & (\ P_1 \wedge\ \mathit{min}_1) & \wedge & (\ P_2 \wedge\ \mathit{min}_2) & : \ \dot{h}_1 = c_2 - c_1 \\ \mathit{max}_1 \vee & ((\neg P_1 \vee \neg \mathit{min}_1) & \wedge & (\neg P_2 \vee \neg \mathit{min}_2)) & : \ \dot{h}_1 = 0 \end{array}$$

The dynamic behavior of the plant components can be integrated in the SFCs: Each step of the resulting HSFC has an associated set of conditional ODE systems which describes the continuous growth of the variables that model the plant behavior.

Definition 3.2 (Syntax of HSFCs) *A* hybrid SFC (HSFC) *is a tuple* $HC = (\mathit{Var}, \mathit{Steps}, \mathit{Act}, s_0, \mathit{Trans}, \mathit{Blocks}, \mathit{Dyn}, \sqsubset, \prec, \mathit{Hist})$*, where Steps, Act,* s_0*, Trans, Blocks,* $\sqsubset$*,* $\prec$ *and Hist are as defined for SFCs,*

- $\mathit{Var} = \mathit{Var}_I \uplus \mathit{Var}_O \uplus \mathit{Var}_L \uplus \mathit{Var}_C$*, where* Var_C *is the set of continuous variables, and*

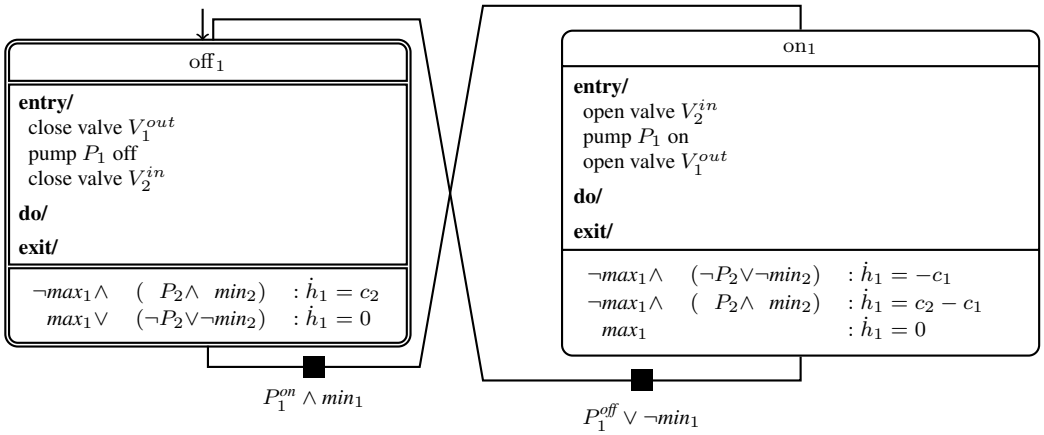

Figure 3: HSFC for the first pump of the example tank system

- *Dyn* : *Steps* $\rightarrow \mathrm{CODE}^*_{Var_C}$ *assigns a sequence of conditional ODE systems to each step.*

Figure 3 shows the HSFC of the left SFC from our example tank system. The water level in tank T_1 is added as a continuous variable. Since we assume the pump P_1 to be working in state on_1 and to be switched off in state off_1, only the relevant parts of the conditional ODE systems are added to the flow sections of the steps. If none of the conditions is fulfilled, we assume chaotic behavior.

A configuration of an HSFC is a tuple $c = (\sigma, \mathrm{readyS}, \mathrm{activeS}, \mathrm{activeA}, \mathrm{activeD}) \in \Sigma \times \mathit{Steps}(\overline{C}) \times \mathit{Steps}(\overline{C}) \times \mathit{Act}^* \times 2^{\mathrm{ODE}_{Var_C}}$. The new set activeD is the set of the *active ODEs* which are those ODEs that are attached to active steps. For each active step, only the differential equation system that belongs to the *first* condition that evaluates to true, is an element of activeD.
Let $\delta_l \leq t \leq \delta_u$ be the ellapsed time between two PLC cycles. During this time the dynamic part of the system evolves continuously as specified by the active ODEs. To represent this continuous dynamics in the semantics, we extend the formal semantics of SFCs by adding another step to the PLC cycle after Step 4. During this step we let time elapse and update the continuous variables. The semantics of this step is similar to the time steps of hybrid automata.
We collect the set of active ODEs and compute a solution of the active ODEs for the time elapse of $\delta_l \leq t \leq \delta_u$. Moreover, the values for all continuous variables in this solution at the time $t = 0$ must correspond to the values in $\sigma|_{Var_C}$.

Definition 3.3 (Semantics of HSFCs) *Runs of an HSFCs consist of the alternating execution of steps of the embedded SFCs and time steps* $(\sigma, \mathrm{readyS}, \mathrm{activeS}, \mathrm{activeA}, \mathrm{activeD}) \rightarrow (\sigma', \mathrm{readyS}, \mathrm{activeS}, \mathrm{activeA}, \mathrm{activeD}')$ *with*

- $\mathrm{activeD}' = \bigcup_{s \in \mathrm{activeS}} \{d \mid \exists i \in \{1, \ldots, n\} : \mathit{Dyn}(s) = ((\mathrm{cond}_1, O_1), \ldots, (\mathrm{cond}_n, O_n)) \wedge d \in O_i \wedge \sigma \models \mathrm{cond}_i \wedge \bigwedge_{j=1}^{i-1} \sigma \not\models \mathrm{cond}_j\}$,
- $\sigma'(v) = \sigma(v)$ *for all* $v \notin \mathit{Var}_C$ *and* $\sigma'|_{Var_C} = f(t)$ *for some* $\delta_l \leq t \leq \delta_u$ *and* f *a solution to the ODE systems in* activeD *with* $f(0) = \sigma|_{Var_C}$.

4. From HSFCs to Hybrid Automata

In this section, we describe the transformation of SFCs to hybrid automata and extend this procedure to HSFCs. We show the transformation for (H)SFCs without nested components and without

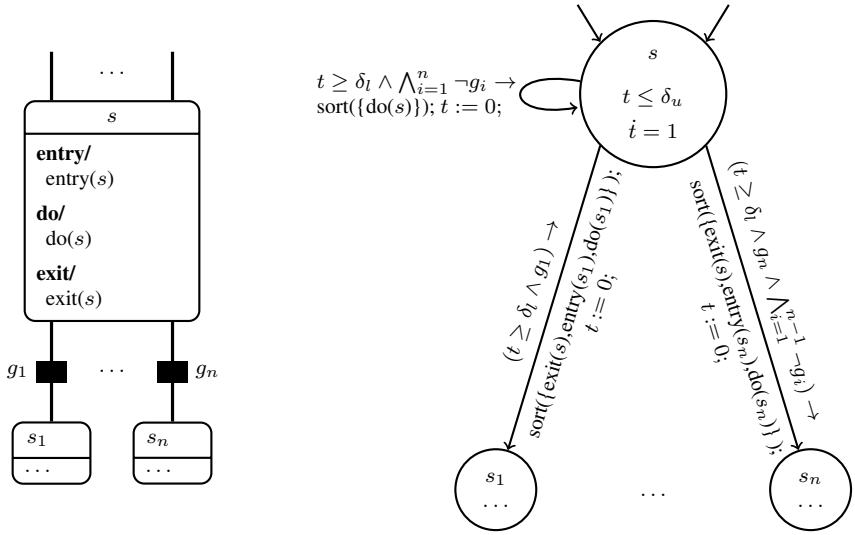

Figure 4: Transformation of an SFC to a hybrid automaton

parallel transitions, but our translation can be adapted to the general case.

4.1. From SFCs to Hybrid Automata

The transformation of an SFC into a hybrid automaton works as follows: For each step of the SFC there is a corresponding location in the hybrid automaton. The location that corresponds to the initial step defines the initial state of the hybrid automaton.
The time $\delta_l \leq t \leq \delta_u$ specifies the duration of the time elapse between two PLC cycles. We introduce a clock variable t and add the invariant $t \leq \delta_u$ to each location to force the automaton to leave the location latest at time δ_u. We prohibit the automaton from taking a transition before δ_l time has elapsed by adding the guard $t \geq \delta_l$ to each transition. The clock variable is reset when a transition is taken.
For each transition in the SFC we create a transition in the automaton. The transition connects the locations that correspond to the source and target steps of the transition in the SFC. The assignments of the transition model the execution of the active actions. The exit actions of the source step and both the entry and the do actions of the target step are sorted according to the action order $\sqsubset$. In order to model the execution of the do actions during the period of activation, we add a self-loop to all locations.
We use transition guards to manage the ordering of the outgoing transitions. The outgoing transition that corresponds to the SFC transition with the highest priority has the same guard as the SFC transition. For transitions with a lower priority according to $\prec$, we take the conjunction of its guard and the negated guards of the higher prioritized transitions. Finally, we add the conjunction of the negated guards of all outgoing transitions in the SFC to the self-loop in the automaton.
In Figure 4 the transformation of an SFC (left-hand side) into a hybrid automaton (right-hand side) is illustrated.

Definition 4.1 (Transformation of SFCs) *For an SFC* $C = (\mathit{Var}, \mathit{Steps}, \mathit{Act}, s_0, \mathit{Trans}, \mathit{Blocks}, \sqsubset, \prec, \mathit{Hist})$ *without nested components and parallel transitions, its* transformation to a hybrid au-

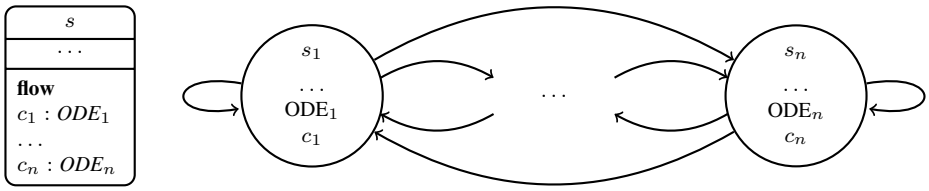

Figure 5: Transformation of an HSFC to a hybrid automaton

tomaton *is* $H = (Loc, Var_H, Edge, Act_H, Inv, Init)$ *with* $Loc = Steps$, $Var_H = Var \cup \{t\}$, *and*

- $Edge = \cup_{s \in Steps} Edge_s$, *where for each step* $s \in Steps$ *with outgoing transitions* $t_1, \ldots, t_n \in Trans$, $t_i = (s, g_i, s_i)$, *and ordering* $t_1 \prec \ldots \prec t_n$ *we define* $Edge_s = (\bigcup_{i=1}^{n} \{(s, \mu_i, s_i)\}) \cup \{(s, \mu_s, s)\}$ *with*
  - μ_i *the set of all valuation pairs* (ν, ν') *with* $\nu \models t \geq \delta_l \wedge g_i \wedge \bigwedge_{j=i+1}^{n} \neg g_j$ *and* ν' *results from* ν *by applying in decreasing priority order the exit actions of* s, *the entry and do actions of* s_i *and* $t := 0$ *and*
  - μ_s *is the set of all valuation pairs* (ν, ν') *with* $\nu \models t \geq \delta_l \wedge \bigwedge_{j=1}^{n} \neg g_j$ *and* ν' *results from* ν *by applying in decreasing priority order the do actions of* s *and* $t := 0$.
- $Act_H(s)$ *is specified for all* $s \in Loc$ *by* $\dot{t} = 1$ *and* $\dot{v} = 0$ *for all other variables,*
- $Inv(s)$ *is the set of all valuations satisfying* $t \leq \delta_u$ *for all* $s \in Loc$, *and*
- $Init = \{(s_0, \nu_0)\}$ *with* ν_0 *assigning initial values to the variables.*

4.2. From HSFC to Hybrid Automata

To transform an HSFC to a hybrid automaton, first we transform the SFC fragment of the HSFC to a hybrid automaton. Then for each step $s \in Steps$ with CODEs $(\text{cond}_1{:}ODE_1), \ldots, (\text{cond}_n{:}ODE_n)$ we replace the location s by $n+1$ copies $s_1, \ldots, s_{n+1}$ of it, and for each $i = 1, \ldots, n$ we add $\text{cond}_i \wedge \bigwedge_{j=1}^{i-1} \neg \text{cond}_j$ to the invariant and ODE_i to the activities of location s_i. The $(n+1)$th copy is extended with the invariant $\bigwedge_{j=1}^{n} \neg \text{cond}_j$. Moreover, we add transitions between each two copies s_i and s_j, $i \neq j$, to be able to switch from one copy to another during the period of activation when the evaluation of the conditions changes. The transformation scheme is illustrated in Figure 5.

Definition 4.2 (Transformation of HSFCs) *Given an HSFC* $C = (Var, Steps, Act, s_0, Trans, Blocks, Dyn, \sqsubset, \prec, Hist)$ *without nested components and parallel transitions, its* transformation to a hybrid automaton *is* $H = (Loc, Var, Edge, Act_H, Inv, Init)$ *with:*

- $Loc = \bigcup_{s \in Steps} Loc_s$ *with* $Loc_s = \{s_i \mid 1 \leq i \leq |Dyn(s)| + 1\}$ *and* $Var_H = Var \cup \{t\}$*;*
- *For each step* $s \in Steps$ *with* $Dyn(s) = ((\text{cond}_1 : \text{ODE}_1), \ldots, (\text{cond}_m : \text{ODE}_m))$ *and outgoing transitions* $t_1, \ldots, t_n \in Trans$, *transition ordering* $t_1 \prec \ldots \prec t_n$, *the set Edge contains the following edges:*
  - *For each* $t_i = (s, g_i, s^i)$, $1 \leq i \leq n$, *each* $s_j \in Loc_s$ *and* $s^i_k \in Loc_{s^i}$ *there is an edge* (s_j, μ, s^i_k) *with* μ *the set of all* (ν, ν') *with* $\nu \models t \geq \delta_l \wedge g_i \wedge \bigwedge_{j=i+1}^{n} \neg g_j$ *and* ν' *results from* ν *by applying the ordered sequence of the exit actions of* s *and the entry and do actions of* s^i *and* $t := 0$.
  - *For all* $s_k \in Loc_s$ *we have an edge* (s_k, μ, s_k) *with* μ *the set of all* (ν, ν') *with* $\nu \models t \geq \delta_l \wedge \bigwedge_{j=1}^{n} \neg g_j$ *and* ν' *results from* ν *by applying the ordered do actions of* s *and* $t := 0$.

  - *For each $s_k, s_l \in Loc_s$, $s_k \neq s_l$, we have edges (s_k, μ, s_l) and (s_l, μ, s_k) with μ the identity relation;*
- *For all $s \in Steps$, $Dyn(s) = ((\mathrm{cond}_1 : \mathrm{ODE}_1), \ldots, (\mathrm{cond}_m : \mathrm{ODE}_m))$ and $Loc_s = \{s_1, \ldots, s_{m+1}\}$, $Act(s_i)$ is given by $ODE_i \cup \{\dot{t} = 1\}$ for $i = 1, \ldots, m$ and $Act(s_{m+1})$ is specified by $\{\dot{t} = 1\}$;*
- *For all $s \in Steps$, $Dyn(s) = ((\mathrm{cond}_1 : \mathrm{ODE}_1), \ldots, (\mathrm{cond}_m : \mathrm{ODE}_m))$ and $Loc_s = \{s_1, \ldots, s_{m+1}\}$, $Inv(s_i)$ is given by $\mathrm{cond}_i \wedge (\bigwedge_{j=1}^{i-1} \neg \mathrm{cond}_j) \wedge t \leq \delta_u$ for $i = 1, \ldots, m$ and $Inv(s_{m+1})$ is specified by $(\bigwedge_{j=1}^{m} \neg \mathrm{cond}_j) \wedge t \leq \delta_u$;*
- *$Init = \{(s_0, \nu_0)\}$ with ν_0 assigning initial values to all variables.*

5. Conclusion

We extended SFCs to HSFCs to support the modeling of the dynamic behavior of plants. A transformation of HSFCs to hybrid automata allows the analysis of the plant model. As future work, we plan to use our approach in a CEGAR verification framework to allow to handle larger systems.

Acknowledgements

We thank David Kampert and Ulrich Epple for fruitful discussions.

References

[ACH+95] Alur, R., C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine: *The algorithmic analysis of hybrid systems*. Theoretical Computer Science, 138:3–34, 1995.

[BCMP] Baresi, L., S. Carmeli, A. Monti, and M. Pezzè: *PLC programming languages: A formal approach*.

[BHLE04] Bauer, Nanette, Ralf Huuck, Ben Lukoschus, and Sebastian Engell: *A unifying semantics for sequential function charts*. In *LNCS*, volume 3147, pages 400 – 418. Springer-Verlag, 2004.

[ELS05] Engell, S., S. Lohmann, and O. Stursberg: *Verification of embedded supervisory controllers considering hybrid plant dynamics*. Int. Journal of Software Engineering and Knowledge Engineering, 15(2):307–312, 2005.

[FL00] Frey, G. and L. Litz: *Formal methods in PLC programming*. In *Systems, Man and Cybernetics*, volume 4, pages 2431 – 2436. IEEEXplore, 2000.

[HKD98] Hassapis, G., I. Kotini, and Z. Doulgeri: *Validation of a SFC software specification by using hybrid automata*. In *Proc. of INCOM'98*, pages 65–70. Pergamon, 1998.

[Int03] Int. Electrotechnical Commission: *Programmable Controllers, Part 3: Programming Languages, 61131-3*, 2003.

[Luk05] Lukoschus, Ben: *Compositional Verification of Industrial Control Systems - Methods and Case Studies*. PhD thesis, Christian-Albrechts-Universität zu Kiel, 2005.