Translating Sequential Function Charts to the Compositional Interchange Format for Hybrid Systems

Christian Sonntag and Stephan Fischer

Abstract—Sequential Function Chart (SFC) is a powerful graphical formalism for the specification of logic controllers that is well recognized and very successful in industry. In this paper, a scheme for the automatic translation of SFCs into the Compositional Interchange Format (CIF) for hybrid systems is presented that enables the straightforward connection of SFC controllers to CIF models of the controlled system which enables the application of a wide range of modeling, design, and analysis tools that are currently connected to the CIF. The translation procedure is based on the reformulation of the SFC as a CIF automaton that explicitly considers the cyclic execution mode of programmable logic control (PLC) hardware and, thus, provides a realistic representation of the behavior of the logic controller on a PLC. The translation scheme is demonstrated on a two-tank system that is modeled in the $g$PROMS process simulation environment, automatically translated to the CIF, and simulated after the connection of a SFC controller.

I. INTRODUCTION

In recent years, the graphical formalism Sequential Function Chart (SFC) has become increasingly popular in industry for the design of discrete-event logic control systems. Besides its intuitive usability and expressional power, the major reason for the industrial success of SFC is that it has been standardized [1] and is supported by a wide range of programmable logic control (PLC) hardware. Since the industry-oriented SFC formalism significantly differs from formalisms that are amenable to model-based analysis, several approaches have been developed in recent years for the translation of SFC, mostly with the goal of subsequent automated correctness verification using e.g. model checking techniques. In [2], [3], [4], approaches for the translation and analysis of untimed SFCs are presented, while [5], [6], [7], [8] consider timed SFCs, i.e. SFCs that contain control actions that are limited to a certain period of time or are applied after a delay. In this paper, an approach for the algorithmic translation of timed SFCs to the Compositional Interchange Format (CIF) for hybrid systems ([9], [10], [11]) is presented that enables the straightforward connection of SFCs to CIF models of the controlled system. The CIF is a powerful modeling formalism that facilitates the exchange of hybrid models between tools that employ differing modeling formalisms, and that supports concepts such as fully implicit high-index differential-algebraic equations (DAEs), different classes of variables (discrete, continuous, and algebraic), parallel composition with synchronization using variables and action labels, different urgency concepts, as well as hierarchy and modularity. Currently, a large number of model-based tools and languages, such as e.g. $g$PROMS, Modelica, Matlab/Simulink, UPPAAL, or PHAVer, are connected to the CIF within the scope of several EU-funded research projects such as the MULTIFORM project [12]. Thus, in comparison with the approaches described above that focus on verification, the connection of SFC to the CIF that is presented in this work enables a wider range of analysis and design possibilities to the control engineer, e.g. simulation of SFCs with complex hybrid system models, verification, or dynamic optimization. Many of the existing translation approaches focus only on small subsets of the SFC standard or overapproximate the behavior of SFC controllers for verification purposes. The goal of this work is to provide an accurate and complete model of non-hierarchical SFC controllers. The translation scheme is based on a compact, formal SFC syntax that does not allow for unreachable or deadlock SFCs. The translation supports the cyclic execution mode that is defined in the SFC standard and that can be implemented either cycle- or event-based, as well as a complete model of the action control concept of SFC and arbitrary transition priorities. In addition, it is easily adaptable to different execution modes, such as lock-step semantics, maximal progress semantics, and different cycle-execution models. The need for different execution models arises from ambiguities in the SFC specification that has led to different interpretations by tool vendors (see e.g. [6], [13]).

II. A FORMAL DEFINITION OF SEQUENTIAL FUNCTION CHARTS

The main building blocks of the Sequential Function Chart formalism are steps $s_i$, transitions with associated transition conditions $q_j$, alternative executions / parallel executions that are enclosed by single/double horizontal lines, and jump transitions (see Fig. 1)$^1$. Each step $s_i$ is equipped with a Boolean variable $s_i.X$ that evaluates to 1 if the step is active, and with a clock $s_i.T$ that measures the activation time of a step. These variables can be used in transition conditions. The execution proceeds from top to bottom along the vertical lines (except in jumps). A step may contain an action block that encompasses one or more actions, and each action is represented by an action qualifier $q_a$ that defines the mode of execution, an action operand $a_o$, and a time quantifier

$^1$The SFC standard only includes the concept of loop transitions. In this work, loop transitions are replaced by the more general concept of jump transitions since they are supported by many SFC implementations.
The following grammar specifies the SFC structure that is considered in this paper. It is based on [8], but has been extended to include jump transitions and transition priorities.

Definition 1 (SFC grammar): The grammar $\Gamma_{SFC} = (\Sigma_{SFC}, \Psi_{SFC}, \Psi_{SFC}, \Phi_{SFC})$ of a SFC consists of the alphabet $\Sigma = \{\parallel, \perp, \parallel_p, \forall, \Delta, \nabla, \top, \top^i, \cdot\}$, a set of non-terminals $\Psi_{SFC} = \{\text{seq}, \text{alt}, \text{par}\}$, the start symbol $\psi_{SFC}$, and the set $\Phi_{SFC}$ of production rules as follows:\footnote{\{a\}^\ast \text{ indicates that the string } a \text{ occurs zero or more times, } \{b\} \text{ denotes that } b \text{ is optional (i.e. occurs zero times or once), and } (a|b) \text{ indicates that either } a \text{ or } b \text{ occurs.}}

\[
\psi_{SFC} := '\parallel' \ 	ext{seq} \tag{1}
\]

\[
\text{seq} := '1.1.1' \parallel \text{seq} | '1.1.1' ∨ \Delta \top \parallel' \text{seq} |∅ \tag{2}
\]

\[
\text{alt} := \{@[p]@[\nabla] \text{seq} '@' ∨ ('1.1.1')^\ast \{ (\square) ∨ (\square) \} '1.1 Now\}
\]

\[
\text{par} := \{@[\top]@[\top] \text{seq} '@' ∨ ('1.1.1')^\ast \{ (\square) ∨ (\square) \} '1.1\}
\]

\[
\text{for the example in Fig. 1, } \Gamma_{SFC} \text{ produces the string } Υ := [\square(\square)] ∨ [\square(\square)] ∨ [\square(\square)] ∨ [\square(\square)].\]"
**Definition 3 (Semantics of SFC):** The state of a SFC in cycle $i$, $V_i = (St_{a,i}, X_i, C_i, Ac_{a,i})$, consists of the set of active steps $St_{a,i} \subseteq St$, the values $X_i$ ($C_i$, $St_{a,i}$), and a set of Boolean flags $Ac_{a,i} \in \{0, 1\}^{|Ac|}$ that determine if an action $(qu, o, \tau_a) \in Ac$ affects its operand $o$ in the current cycle. The state $V_{i+1}$ after one cycle is derived from $V_i$ according to:

1. The values of the input variables $X_{in}$ are updated in $V_{i+1}$ from sensor readings.
2. All actions $a \in Ac$ are executed depending on $Ac_{a,i}$. The effect of the execution of an action $a = (qu, o, \tau_a) \in Ac$ with $a \in \alpha(s)$ on its operand $o \in \{0, 1\}$ is determined by its qualifier $qu$ according to:
   
   a) $qu = N$ (not stored): $o := 1$ while $s \in St_{a,i}$.
   b) $qu = S$ (stored): $o := 1$ when $s \in St_{a,i}$, and $o := 1$ until another action resets $o$ by $qu = R$.
   c) $qu = R$ (reset): $o := 0$ when $s \in St_{a,i}$.
   d) $qu = P/P1$ (pulse): $o := 1$ only in one cycle when $s$ is entered. If $qu = P0$, $o := 1$ only in one cycle when $s$ is left.
   e) $qu = L$ (limited): $o := 1$ for the time $\tau_a$, but at most for the duration in which $s \in St_{a,i}$.
   f) $qu = D$ (delayed): After a time delay of length $\tau_a$, $o := 1$, but only until $s \notin St_{a,i}$.
   g) $qu = SD$ (delayed stored): $o := 1$ after the delay $\tau_a$ even if $s$ is left before $\tau_a$ has elapsed. $o$ remains true until another action resets $o$ by $qu = R$.
   h) $qu = DS$ (delayed stored): $o := 1$ after the delay $\tau_a$ if $s \in St_{a,i}$, and $o := 1$ until another action resets $o$ by $qu = R$.
   i) $qu = SL$ (stored limited): $o := 1$ for the time $\tau_a$ even if $s \notin St_{a,i}$ before $\tau_a$ has elapsed. $o := 1$ until another action resets $o$ by $qu = R$.
3. If an action $a = (qu, o, \tau_a)$ has a time-dependent qualifier (i.e. if $qu \in L, D, SD, DS, SL$) the corresponding clock $c \in C$ is reset to zero in $C_{i+1}$ if the associated step $s$ was activated in cycle $i$. The values of all other clocks are copied to $C_{i+1}$.
4. The set of active steps $St_{a,i+1}$ is obtained from $St_{a,i}$ by executing all enabled transitions $e = (S, g, S', p) \in E$ for which $S \subseteq St_{a,i}$, except for enabled transitions that lead into an alternative execution and that are of lower priority than other enabled transitions into the same alternative execution. Then, $St_{a,i+1}$ contains all states $s \in St_{a,i}$ except all source states $S$ of executed transitions. Furthermore, all steps $S' \subseteq S \setminus St_{a,i}$ that are target states of the executed transitions are added to $St_{a,i+1}$. All clocks $c_{St} \in C_{St}$ that correspond to steps $s \in St_{a,i+1}$ are reset to zero.
5. The set of flags $Ac_{a,i+1}$ is determined for all actions $(qu, o, \tau_a) \in Ac$ based on $qu$ according to the rule (2) above and depending on the states in $St_{a,i+1}$.

A cycle can be executed according to different semantics:

- **Lock-step semantics:** The SFC cycle (1) - (5) is executed only once, even if after the execution, transitions can still be executed.
- **Maximal progress semantics:** The SFC cycle steps (2) - (5) are repeated until no transition can be executed.

### III. THE COMPOSITIONAL INTERCHANGE FORMAT FOR HYBRID SYSTEMS

CIF models can be specified in an abstract format that is suitable for the formal definition of the semantics ([9]), and in a textual concrete format ([10]) that facilitates modeling and model readability. The formal semantics of the concrete format is defined by a mapping to the abstract format. In this paper, a subset of the concrete format is used that omits several CIF constructs that are not required for the translation. CIF models consist of closed automata and atomic automata. Closed automata contain a declaration section for variables, clocks, and a set of parallel closed or atomic automata and thus support hierarchical structures. Open automata define the discrete and continuous model dynamics.

**Definition 4 (Closed CIF automaton):** A closed CIF automaton is a tuple $A_{cl} = (X_e, X_i, I_X, C_e, C_i, q, \nu, A, J_X, J_C, \gamma_ID)$ where:

- $X_e$ is a finite set of external variables, and $X_i$ is the set of internal variables. The set of all variables is given by $X = X_e \cup X_i$.
- $I_X$ is a mapping $X^\prime \rightarrow \text{expr}(X)$ that assigns initial values to a subset of variables $X^\prime \subseteq X_e \cup X_i$.
- $C_e$ is a finite set of external clocks, and $C_i$ is the set of internal clocks. The set of all clocks is given by $C = C_e \cup C_i$. All clocks $c \in C$ are initialized as $c := 0$.
- $\nu$ is a function $X \rightarrow \{\text{disc, cont, alg}\}$ that assigns a dynamic type to each variable $x \in X$.
- $\gamma$ is a function $X \rightarrow \{\text{bool, nat, int, real}\}$ that assigns a static type to each variable $x \in X$.
- $A$ is either a set of closed automata (in which case $A_{cl}$ is called non-terminal) or a set of atomic automata (in which case $A_{cl}$ is terminal).
- The sets $J_X (J_C)$ define connections between variables (clocks) of closed automata. All connected variables/clocks are equal at all times.

**Definition 5 (Atomic CIF automaton):** Given the set of variables $X_{cl}$ of the enclosing terminal closed automaton $A_{cl}$, an atomic CIF automaton in $A_{cl}$ is defined as a tuple $A_{atom} = (V, v_0, E)$ where:

- $V$ is a non-empty and finite set of discrete locations, and $v_0 \in V$ is the initial location.
- $E$ is a finite set of edges $(v, g, (W, r), v') \in V \times \text{pred}(X_{cl}) \times (2^{V_{cl}} \times \text{pred}(X_{cl} \cup X_{cl,\ldots})) \times V$. Here, $v$ is the source location, $v'$ is the target location, $g$ is a transition condition, $W$ is a set of jumping variables.

The definition of these sets is not given here due to space limitations.
that may change values in an action transition, and \( r \) is the reset map\(^{10}\). Here, \( X_{cl, \neg} \) denotes a set of variable valuations that represents the values of \( X_{cl} \) before the execution of the edge. In the concrete CIF format, each edge is defined as: when \( g \) now do \( W := r \) goto \( v' \).

IV. THE TRANSLATION PROCEDURE

The translation procedure transcribes a SFC \( M_{SFC} \) into a closed automaton \( \phi_{SFC} \). In the following, the model employing maximal progress semantics is described\(^ {11}\). Fig. 3 depicts the structure of a CIF model of a process (automaton \( \phi_p \)) that is controlled by an automaton \( \phi_{SFC} \), where

- \( X_{in} \) are the (continuous or discrete) sensor readings,
- \( X_{op} = X_{int} \cup X_{out} \), \( X_{op,t} \), and \( X_R \) each contain one variable of type \( \text{bool} \) for each operand \( o \) of the actions \( a \in Ac \) of \( M_{SFC} \).
- \( X_{st} (C_{sth}) \) contains one Boolean variable (clock variable) for each state \( s \in St \) of \( M_{SFC} \) that represents \( s \cdot X (s, T) \).
- \( X_m \) contains one Boolean variable for each distinct transition condition \( g \in G \) of \( M_{SFC} \).
- \( X_a (C_a) \) contains one Boolean variable (clock variable) for each action \( a = (\{\text{qu}, o, \tau_a\}) \in Ac \) of \( M_{SFC} \) for which \( qu \in \{L, D, SD, DS, SL\} \).
- \( a_{tr} \) and \( a_{mon} \) are the trigger automaton and the monitor automaton that model the cyclic execution of the SFC using the clock \( c \) and the real-valued variables \( t_c \) (cycle time) and \( t_{rem} \) (remaining time to next cycle).
- Each atomic automaton \( a_{ac} \in A_{ac}, |A_{ac}| = |Ac| \) models the execution of an action \( a \in Ac \).
- The atomic automata \( a_{str} \in A_{str} = \{a_{str,1}, a_{str,2}, \ldots, a_{str,k}\} \) represent the SFC structure according to \( \Gamma_{SFC} \), where \( a_{str,m} \) is the main structure automaton, and \( a_{str,1}, a_{str,2}, \ldots, a_{str,k} \) model the branches of all parallel execution of \( M_{SFC} \).

\(^{10}\) All edges are urgent, i.e. must be taken instantly when \( g \) becomes 1.

\(^{11}\) Comments will be given throughout the section on how the model can be adapted to different execution semantics.

- \( l_u, L_R, L_{act}, L_{str}, \) and \( L_{par} \) are binary variables that implement a directed communication between the automata \( a_{tr}, a_{m}, A_{ac} \), and \( A_{str} \), as shown in Fig. 3, with \( |L_R| = |\{a \in Ac | qu = R\}| \), \( |L_{act}| = |\{a \in Ac | qu = 1\}| \), \( |L_{str}| = |A_{str}| \), \( |L_{par}| = |A_{str} - 1| \).
- \( F_n \) contains one Boolean variable for each automaton in \( A_{str} \) that determines if a structure automaton has finished its execution in the current SFC cycle, and
- \( I_{X_{SFC}} \) sets the variable \( x_{st} \in X_{st} \) that corresponds to the initial state \( s_0 \) of \( M_{SFC} \), to 1. All other variables except those in \( X_{in} \) are initialized to the value 0.

The connections of variables between the plant and SFC automata are realized by a suitably chosen connect set \( J_{X_{SFC}} \).

Trigger and monitor automata \( a_{tr} \) and \( a_{m} \) : The trigger automaton is an atomic automaton \( a_{tr} = \{\{v_{tr,1}, v_{tr,2}, v_{tr,3}, v_{tr,4}, v_{tr,5}\}, v_{tr,1}, E_{tr}\} \) that models the cyclic SFC execution (see Fig. 4). It triggers the execution of the action control automata and of the structure automata via the communication variables in \( L_R, L_{act}, \) and \( L_{str} \). A cycle execution is triggered by the communication variable \( l_u \) that is set by the monitor automaton \( a_{m} \).

If the cycle time \( t_c \) is very small compared to the time constants of the controlled process, a cycle-driven execution model leads to a large computational effort (\([8]\)). This problem can be avoided if an event-based execution model is employed. To this end, it must be determined at the beginning of each cycle \( i \) if an event (i.e. the change of the valuation of a transition condition \( g \in G \)) occurred in cycle \( i - 1 \) (see Fig. 2). In addition, a cycle execution must be triggered if any timing predicates of action automata for timed actions change valuation. The atomic monitor automaton \( a_{m} = \{(v_{m1}, v_{m2}), v_{m1}, E_{m}\} \) monitors the values of all transition conditions \( g \in G \) using the memory variables \( X_m \) and triggers a cycle simulation via the edge \((v_{m1}, X_m \neq G \lor G_a, (X_m, l_u), (G, 1), v_{m2}) \in E_{m} \) if the valuations of any \( g \in G \) change (see Fig. 4), and if the timing conditions of active timed actions have changed value. To this end, a set \( G_a \) of predicates is monitored that contains a predicate of the form \((x_{a,i} \land \text{pred}(c_{a,i})) \) if \( a_i \in Ac \) of \( \text{pred}(c_{a,i}) \) corresponds to the timing condition of \( a_i \), and the variable \( x_{a,i} \) is set or reset by \( a_i \) to indicate the activity of its timing condition. The variables in \( X_m \) are reset to the new valuations of all \( g \in G \) at \( a_{m} \) returns to the monitoring state \( v_{m1} \) via \((v_{m2}, l_u = 0, (\emptyset, \emptyset), v_{m1}) \in E_{m} \) after \( a_{tr} \) has reset the communication variable \( l_u \).

\( a_{tr} \) implements Def. 3 for cycle \( i \) as follows: via the edge \((v_{tr,1}, l_u, (c_c, t_{rem}), (0, t_c - t_{sim} \text{ mod } t_c), v_{tr,2}) \), \( a_{tr} \) synchronizes with \( a_{m} \), resets the clock cycle \( c_c \), and computes the remaining time \( t_{rem} = t_c - t_{sim} \text{ mod } t_c \) (see Fig. 2). Here, \( t_{sim} \) is the current simulation time, and \( \text{mod} \) is the modulo operator. Step (2) of Def. 3 is modeled by two edges: the first edge \((v_{tr,2}, c_c \geq \)

The dynamic type of a variable is \( \text{disc} \) if not specified otherwise.

If a cycle-driven execution is desired, the monitor automaton \( a_{m} \) must be omitted, and \( l_u \) must be set to 1 at all times.

Fig. 3. Structure of the CIF model of a controlled plant.
\( t_{rem} = (X_{op,t}, X_R, l_a, L_{tr}, c_e) = \{0, 0, 0, 1, 0\} \), \( v_{tr,3} \in E_{tr} \) resets all variables in \( X_{op,t} \) (which are used as temporary operands for the action control automata \( A_{ac} \)) and \( X_R \) (which store if an operand \( x_{op,t} \in X_{op,t} \) is reset in cycle \( i \) by an action with qualifier \( R \)). By setting the communication variables in \( L_R \), \( \alpha_{tr} \) triggers the execution of the action control automata that model actions with \( \{0\} \). The steps (2), (3), and (5) of Def. 3 are encoded in the action control automata. Each action \( a = (\{0\}, \omega, \tau_a) \in A_c \) is modeled by an atomic automaton \( A_{ac} = (V_a, v_a, 0, E_a) \), where \( V_a = \{v_a, 0\} \) if \( qu \in \{R, N\} \), \( V_a = \{v_a, 0, v_a\} \) if \( qu \in \{S, P, P_1, P_0, L, D, SD, SL, DS\} \), and \( V_a = \{v_a, 0, v_a, 1, v_a, 2\} \) if \( qu = DS \). The actions with qualifier \( R \) set the flag variables \( x_{o,R} \in X_R \) to indicate if any of the temporary operands \( x_{op,t} \in X_{op,t} \) are reset in the current cycle \( i \). Only if \( x_{o,R} = 0 \), the action control automata that model actions with \( qu \in \{N, S, P, P_1, P_0, L, D, SD, SL, DS\} \) set the corresponding operand \( x_{op,t} \) according to the rules described in step (2) of Def. 3. Given a mapping \( \beta : A_c \rightarrow St \) that returns the step \( s = \beta(a) \) that the action \( a \) is associated with, unique clocks and variables \( c_a, x_a \in X_a \forall a \in A_c \), \( qu \in \{L, D, SD, SL, DS\} \), a communication variable \( l_R \in X_R \) (if \( qu = R \) or \( l_{act} \in X_{act} \) (if \( qu \neq R \)), auxiliary variables \( g_{a}, w_{a}, r_{a} \), and predicates \( p_1 = (\beta(a) \in St_{a,i}) \), \( p_2 = (\beta(a) \in St_{a,i}) \), \( p_3 = (\tau_a, \beta(a) \notin St_{a,i}) \). \( E_a \) is constructed for each action control automaton as:

- If \( qu = R \), add two edges \( (v_{a,0}, p_1, l_R, (l_R, 0), v_{a,0}) \) and \( (v_{a,0}, p_1, l_R, ((l_R, x_{o,R}), (0, 1)), v_{a,0}) \) to \( E_a \).

\( \alpha_{tr} \) by rearranging the states and transitions.

4254
If $qu \neq R$, add $(v_{a,0},g_{a},(l_{act},0),v_{a,0})$ to $E_{a}$, where $g_{a} = \overline{p_{l}} \land l_{act}$ if $qu \in \{P, P1, P0, D, L, DS\}$ and $g_{a} = p_{3} \land l_{act}$ if $qu \in \{S, N, SD, SL\}.$ If $qu \neq R$, add $(v_{a,0},g_{a},(w_{a},r_{a}),v_{a,1})$ to $E_{a}$, where $g_{a} = p_{2} \land l_{act}$ if $qu \in \{S, N, P, P1, L, SD, SL\}$ and $g_{a} = p_{1} \land l_{act}$ if $qu \in \{P0, D, DS\}.$ Here, $w_{a} = l_{act}, r_{a} = 0$ if $qu = P0$, $w_{a} = (x_{a}, l_{act}), r_{a} = (1,0)$ if $qu \in \{S, N, P1\}$, $w_{a} = (c_{a}, l_{act}, x_{a}), r_{a} = (0,0,1)$ if $qu \in \{D, DS, SD\}$, and $w_{a} = (x_{a}, c_{a}, l_{act}, x_{a}), r_{a} = (1,0,0,1)$ if $qu \in \{L, SL\}.$

If $qu \in \{P, P1, L\}$, add $(v_{a,0},x_{a},o \land p_{1} \land l_{act}, (w_{a},r_{a}),v_{a,1})$ to $E_{a}$, where $w_{a} = l_{act}, r_{a} = 0$ if $qu \in \{P, P1, P0\}$ and $w_{a} = (c_{a}, l_{act}, x_{a}), r_{a} = (0,0,1)$ if $qu = L.$

If $qu \in \{S, P1, P0\}$, add $(v_{a,1}, o \land p_{1} \land l_{act}, (w_{a},r_{a}),v_{a,1})$ to $E_{a}$, where $g_{a} = \overline{p_{x}} \land l_{act}$ if $qu \in \{P, P1, D, L\}$, $g_{a} = \overline{p_{x}} \land l_{act}$ if $qu = P0$, and $g_{a} = (\overline{p_{x}} \land c_{a} < \tau_{a}) \lor (x_{a} \land c_{a} \geq \tau_{a}) \land l_{act}$ if $qu = DS.$ Here, $w_{a} = l_{act}, r_{a} = 0$ if $qu \in \{S, P1, P0\}$ and $w_{a} = (l_{act}, x_{a}), r_{a} = (0,0)$ if $qu \in \{D, L, SD, SL, DS\}.$

If $qu = P0$, add $(v_{a,1}, \overline{x_{a}} \land p_{1} \land l_{act}, (x_{a}, l_{act}, (0), (1,0)), v_{a,0})$ to $E_{a}.$

If $qu \in \{D, L\}$, add three edges $(v_{a,1}, x_{a}, o \land p_{1} \land c_{a} \cdot \tau_{a} \land l_{act}, (l_{act}, 0), v_{a,1}), (v_{a,1}, x_{a}, o \land p_{1} \land c_{a} \cdot \tau_{a} \land l_{act}, (x_{a}, l_{act}, (1,0), v_{a,1}), (v_{a,1}, x_{a}, o \land p_{1} \land c_{a} \cdot \tau_{a} \land l_{act}, (l_{act}, 0), v_{a,1})$, to $E_{a},$ where $x_{1} = \leq, x_{2} = \leq$ if $qu = D$ and $x_{1} = \leq, x_{2} = \leq$ if $qu = L$. Add the assignment $x_{a} := 0$ to an edge if $x_{1} = x_{2} = \leq$.

If $qu \in \{SD, SL\}$, add two edges $(v_{a,1}, x_{a}, o \land c_{a} \cdot \tau_{a} \land l_{act}, (x_{a}, l_{act}, (1,0), v_{a,1}), (v_{a,1}, x_{a}, o \land c_{a} \cdot \tau_{a} \land l_{act}, (l_{act}, 0), v_{a,1})$, to $E_{a},$ where $x_{1} = \leq, x_{2} = \leq$ if $qu = SD$ and $x_{1} = \leq, x_{2} = \leq$ if $qu = SL$. Add the assignment $x_{a} := 0$ to an edge if $x_{1} = x_{2} = \leq$.

If $qu = DS$, add three edges $(v_{a,2}, x_{a}, o \land c_{a} \geq \tau_{a} \land l_{act}, (x_{a}, l_{act}, (0,0), v_{a,2}), (v_{a,2}, x_{a}, o \land c_{a} \geq \tau_{a} \land l_{act}, (x_{a}, l_{act}, (1,0), v_{a,2}), and $(v_{a,2}, x_{a}, o \land c_{a} \geq \tau_{a} \land l_{act}, (l_{act}, 0), v_{a,0})$ to $E_{a}.$

Fig. 4 exemplarily shows the action control automata that result from the translation of the SFC of Fig. 1.

**Structure automata $A_{str}$**: The structure automata model the SFC structure according to the SFC grammar (Def. 1). Assuming that each step $s_{i} \in St$ of a SFC $M_{SFC}$ is represented by a unique variable $x_{s_{i}} \in X_{S_{i}}$ and clock $c_{s_{i}} \in C_{S_{i}}$ and that each structure automaton $a_{str} \cdot \cdot \cdot = (V_{s}, v_{s_{0}}, E_{s})$ is associated with a unique communication variable $l_{str} \cdot \cdot \cdot \in L_{str}$ and a unique maximal-progress variable $f_{n} \cdot \in F_{n}$, the structure automata are constructed as:

- If the variable $s_{i}, T$ of a step $s_{i}$ is not used in $M_{SFC}$, any assignments to $c_{s_{i}}$ can be omitted.

- The identifiers of steps $s$ and transition conditions $g$ coincide with the identifiers used in the corresponding rules (1)-(6) in Def. 2.
is realized as follows (see Def. 2, rules (4) and (5)): two locations \( v_{pe} \) and \( v_n \) are added to \( V_\bullet \), and three edges \((v_{pe}, l_{str,\bullet} \wedge l_{par,x}, f_{n,\bullet}),(0,0,1,0), v_{pe})\),
\((v_{pe}, l_{str,\bullet} \wedge (g_n \lor \overline{L_{par,x}} = 0 \land L_{str,x} = 0)),((l_{str,\bullet}, f_{n,\bullet}),(0,0), v_{pe})\), and
\((v_{pe}, l_{str,\bullet} \wedge g_n \land L_{par,x} = 0 \land L_{str,x} = 0, ((X_{pe,end}, x_{pe}, l_{str,\bullet}, f_{n,\bullet}, c_{pe}), (0,1,0,1)), v_{n})\) are added to \( E_\bullet \). Here, \( X_{pe,end} \in X_\delta \) is the set of the final states of all automata \( a_{str,j,y} \).

3) For each location \( v_i \) of all structure automata \( a_{str,j} \), \( j \in 1, \ldots, |A_{str}| \), (except for locations \( v_{in} \) and \( v_{pe} \)), add an edge \((v_i, \overline{\gamma_0 \land \ldots \land \overline{\gamma_n} \land M_{str,j}}, ((l_{str,j}, f_{n,j}),(0,0), v_j)\) to \( E_j \), where \( g_1, \ldots, g_n \in G \) of \( M_{SFC} \) are the SFC transition conditions of all outgoing edges of \( v_i \).

Fig. 4 exemplarily shows the structure automata that result from a translation of the SFC in Fig. 1.19

V. APPLICATION EXAMPLE

The SFC controller shown in Fig. 5 was translated to the CIF using maximal progress semantics and event-driven execution and was connected to a CIF model of a two-tank system. The process model was obtained by translating the hybrid gPROMS model described in [15] to the CIF using an algorithmic technique that was developed in [12]. \( Q_1 \), \( Q_2 \), \( Q_{1,2,1} \), and \( Q_{1,2,0} \) are the liquid flows within the system. The controller aims at keeping the liquid levels \( h_1, h_2 \) in the tanks within a range \( [t_1, t_0] = [0.2 \text{ m}, 0.5 \text{ m}] \) using the valves \( V_1 \) and \( V_2 \). The controlled system was

The translation procedure has been implemented in a software tool for the systematic refinement of informal specifications that is based on [14].

simulated using the freely available CIF tool set20 with a large cycle time to illustrate the effect of the cyclic execution.

VI. CONCLUSIONS AND OUTLOOK

This paper describes an algorithm for the translation of SFC controllers to the Compositional Interchange Format for hybrid system. The algorithm produces an accurate model of the SFC semantics and supports different types of execution models, such as lock-step and maximal progress semantics, as well as different cycle-execution models. This algorithm will greatly facilitate the simulation, analysis, and optimization of SFC controllers, particularly since several suitable tools are currently being connected to the CIF within the scope of the European research projects such as MULTIFORM. Current work focuses on the application of the translation algorithm in complex design and analysis tool chains.

REFERENCES


http://se.wtb.tue.nl/sewiki/cif

Fig. 5. The two-tank system and its SFC controller as well as simulation results for a SFC cycle time of \( t_{c} = 5 \text{ s} \). The dashed components of the SFC might be hidden by a SFC tool support standard-compliant SFCs.