# FORMAL VERIFICATION OF CHI MODELS USING PHAVer

D.A. van Beek[1], K.L. Man[2], M.A. Reniers[2], J.E. Rooda[1], and R.R.H. Schiffelers[1]

[1] Department of Mechanical Engineering, Eindhoven University of Technology,
P.O.Box 513 5600 MB Eindhoven, The Netherlands
[2] Department of Computer Science, Eindhoven University of Technology,
P.O.Box 513 5600 MB Eindhoven, The Netherlands
{d.a.v.beek, k.l.man, m.a.reniers, j.e.rooda, r.r.h.schiffelers}@tue.nl

**Abstract.** The hybrid Chi ($\chi$) language is a formalism for modeling, simulation and verification of hybrid systems. One of the most widely known hybrid system formalisms is that of hybrid automata. The formal translation of $\chi$ to hybrid automata enables verification of $\chi$ specifications using existing hybrid automata based verification tools. In this paper, we describe the translation from $\chi$ to hybrid automata, and the relation between hybrid automata and the linear hybrid I/O automata that are used for the verification tool PHAVer (Polyhedral Hybrid Automaton Verifyer). In the case study, we translate a $\chi$ specification to a linear hybrid I/O automaton, and use PHAVer to verify properties.

## 1 Introduction

Hybrid systems related research is based on two, originally different, world views: on the one hand the dynamics and control (DC) world view, and on the other hand the computer science (CS) world view. Clearly, hybrid systems represent a domain where the DC and CS world views meet, and we believe that a formalism that integrates the DC and CS world views is a valuable contribution towards integration of the DC and CS methods, techniques, and tools. The hybrid $\chi$ formalism [1, 2] is such a formalism. On the one hand, it can deal with continuous-time systems, PWA/MLD/LC systems, and hybrid systems based on sets of ordinary differential equations using discontinuous functions in combination with algebraic constraints (the DC approach). On the other hand, it can deal with discrete-event systems, without continuous variables or differential equations, and with hybrid systems in which discontinuities take place (mainly) by means of actions (the CS approach). The intended use of hybrid $\chi$ is for modeling, simulation, verification, and real-time control. Its application domain ranges from physical phenomena, such as dry friction, to large and complex manufacturing systems. The history of the $\chi$ formalism dates back quite some time. It was originally designed as a modeling and simulation language for specification of discrete-event, continuous-time or combined discrete-event/continuous-time models. The first simulator [3], however, was suited to discrete-event models only. The simulator was successfully applied to a large number of industrial cases, such as an integrated circuit manufacturing plant, a brewery, and process industry plants [4]. Later, the hybrid language and simulator were developed [5–7]. For the purpose of verification, the discrete-event part of the language was mapped onto the process algebra $\chi_\sigma$ by means of a syntactical translation. The semantics of $\chi_\sigma$ was defined using a structured operational semantics style (SOS), bisimulation relations were derived, and a model checker was built [8]. In this way, verification of discrete-event $\chi$ models was made possible [9]. The hybrid $\chi$ formalism defined in [1, 2] integrates the modeling language and the verification formalism. It resulted in a formalism with the straightforward and elegant syntax and semantics that is also highly suited to non-computer scientists.

One of the most successful formalisms for hybrid system verification is the theory of hybrid automata. In [2], formal translations between $\chi$ and hybrid automata (in both directions) have been defined. The translation from hybrid automata to $\chi$ aims to show that the $\chi$ formalism is at least as expressive as the theory of hybrid automata. The translation from (a subset of) $\chi$ to hybrid automata enables verification of $\chi$ specifications using existing hybrid automata based verification tools.

To the best of our knowledge, none of the hybrid automata definitions from literature is expressive enough to be used as the target for the translation of hybrid $\chi$. Therefore, the translation uses a target hybrid automata definition, called $HA_{\mathrm{u}}$ automata, where the u stands for urgency, that uses features from different hybrid automata definitions. In particular, the definition of the jump predicate in combination with a set of changeable variables is based on [10], the solution concept that allows piecewise differentiable functions is based on [11], and the definition of urgent transitions was inspired by [12].

In this paper, the hybrid automata definition $HA_{\mathrm{u}}$ will be given as well as the subset of the $\chi$ language that is translated. We use the verification tool PHAVer (Polyhedral Hybrid Automaton Verifyer) [13] to show that it is indeed possible to verify properties of $\chi$ specifications using an existing hybrid automata based verification tool. For this purpose, we show that the linear hybrid I/O automata that are used as the underlying mathematical model for the verification tool PHAVer are a subclass of the $HA_{\mathrm{u}}$ automata.

The verification of properties of a $\chi$ specification using PHAVer is illustrated by means of an case study: the water-level monitor, which is taken from [14]. First, the water-level monitor is modeled using $\chi$. Then we translate the $\chi$ specification to a hybrid automaton $HA_{\mathrm{u}}$. Since the obtained hybrid automaton $HA_{\mathrm{u}}$ is a linear hybrid I/O-automaton, it is possible to verify properties of this automaton model using PHAVer. Since we proved that any transition of a $\chi$ specification can be mimicked by a transition in the corresponding hybrid automaton $HA_{\mathrm{u}}$ and vice versa, it can be concluded that these properties also hold for the $\chi$ specification of the water-level monitor.

This paper is organized as follows: In Section 2, (a subset of) the $\chi$ language is described. The translation from $\chi$ to hybrid automata is defined in Section 3. The relation between hybrid automata and the linear hybrid I/O automata that are used for the verification tool PHAVer and the case study are described in Section 4.

## 2 The hybrid $\chi$ language

In this section, the syntax and semantics of (a subset of) the $\chi$ language are discussed. A more detailed explanation of $\chi$ can be found in [1, 2].

A $\chi$ process is a triple $\langle p, \sigma, E\rangle$, where $p$ denotes a process term, $\sigma$ denotes a valuation, and $E$ denotes an environment. A valuation is a partial function from variables to values. Syntactically, a valuation is denoted by a set of pairs $\{x_0 \mapsto c_0, \ldots, x_n \mapsto c_n\}$, where $x_i$ denotes a variable and $c_i$ its value. The environment $E$ is a tuple $(C, J, L, H, R)$, where $C$, $J$, $L$ denote the set of continuous variables, the set of jumping variables, and the set of algebraic variables, respectively, $H$ is a set of channels, and $R$ denotes a recursive process definition (partial function from recursion variables to process terms). The valuation $\sigma$ and the environment $E$, together define the variables that exist in the $\chi$ process and the variable classes to which they belong. In $\chi$, there is the predefined variable $\mathsf{time} \in \mathrm{dom}(\sigma)$, that denotes the current (model) time.

The valuation $\sigma$ captures the values of those variables that are relevant for determining the future behaviors of a process. The dotted continuous variables and the algebraic variables are not included in the domain of $\sigma$, because their values depend only on the process term $p$, possibly together with the values of the other variables. The values of the dotted continuous and algebraic variables are included in the so called 'extended valuation'. This extended valuation is required, among others, to ensure consistency of $\chi$ processes. Consistency is related to extended valuations in the following way: a $\chi$ process $\langle p, \sigma, E\rangle$ is consistent with extended valuation $\xi$, where $\xi$ is the valuation $\sigma$ extended with the (valuation for the) algebraic and dotted variables as defined by environment $E$, if the delay predicates $u$ in $p$ hold when evaluated in extended valuation $\xi$.

### 2.1 The $\chi_{\mathrm{sub}}$ language

**Syntax** The subset $\chi_{\mathrm{sub}}$ of the $\chi$ language that is translated consists of processes $\langle p, \sigma, (\mathrm{dom}(\sigma) \setminus \{\mathsf{time}\}, J, \emptyset, H, \emptyset)\rangle$. where $p \in P_{\mathrm{sub}}$. In [2] the complete $\chi_{\mathrm{sub}}$ language is defined, including the parallel composition operator, and send/receive process terms. In this paper, the set of process terms $P_{\mathrm{sub}}$ is defined by the following grammar:

$$p_{\mathrm{sub}} ::= u \quad | \quad b \to W : r \gg l_{\mathrm{a}} \quad | \quad [p_{\mathrm{sub}}] \quad | \quad *p_{\mathrm{sub}} \quad | \quad p_{\mathrm{sub}};\ p_{\mathrm{sub}} \quad | \quad p_{\mathrm{sub}}\ []\ p_{\mathrm{sub}}$$

Here, $r$ is a predicate over variables (including the variable $\mathsf{time}$), dotted continuous variables, and '$^-$' superscripted variables (including the dotted variables, e.g. $\dot{x}$). The action label $l_{\mathrm{a}}$ is taken from a given set $A_{\mathrm{label}}$ which at least contains the special action label $\tau$ representing the internal or silent step. Furthermore, $u$ and $b$ are both predicates over variables (including the variable $\mathsf{time}$) and dotted continuous variables, and $W$ is a sets of (non-dotted) variables such that $\mathsf{time} \notin W$. The operators are listed in descending order of their binding strength as follows $\{\to\}, ;, \{\|, []\}$. The operators inside the braces have equal binding strength. In addition, operators of equal binding strength associate to the right, and parentheses may be used to group expressions. For example, $p;\ q;\ r$ means $p;\ (q;\ r)$. An *guarded action predicate* $b \to W : r \gg l_{\mathrm{a}}$ denotes instantaneous changes to the variables from set $W$, by means of an action labeled $l_{\mathrm{a}}$, such that predicate $r$ is satisfied, if guard $b$ holds. The guarded action predicate can perform arbitrary delays under the condition that for the intermediate valuations during the delay,

possibly excluding the first and last valuation, the guard $b$ does not hold [3]. A *delay predicate* $u$, usually in the form of a differential algebraic equation, restricts the allowed behavior of the continuous and algebraic variables in such a way that the value of the predicate remains true over time. Arbitrary delay behavior can be described by means of the *any delay operator* $[p]$. *Sequential composition operator term* $p\,;\,q$ behaves as process term $p$ until $p$ terminates, and then continues to behave as process term $q$. The *alternative composition operator* $p\,[]\,q$ models a non-deterministic choice between different actions of a process. With respect to time behavior, the participants in the alternative composition have to synchronize (i.e. a strong time-deterministic choice). The *repetition operator* $*p$ represents the infinite repetition of process term $p$.

In $\chi_{\text{sub}}$ processes, there are no discrete variables ($\text{dom}(\sigma) = C \cup \{\text{time}\}$), no algebraic variables ($L = \emptyset$), and no recursion variables ($R = \emptyset$). This subset is further restricted such that dotted continuous variables are not allowed to occur in action predicates or guards.

### 2.2 Semantics

The semantics of $\chi$ is defined by means of deductions rules in SOS style [15] that associate a hybrid transition system with a $\chi$ process as defined in [2]. Such a hybrid transition system has three different kinds of transition relations:

- *action transitions*; The intuition of an action transition $\langle p, \sigma, E\rangle \xrightarrow{\xi,a,\xi'} \langle p', \sigma', E'\rangle$ is that the process $\langle p, \sigma, E\rangle$ executes the discrete action $a \in \mathcal{A}$ with extended valuations $\xi$ and $\xi'$ and thereby transforms into the process $\langle p', \sigma', E'\rangle$, where $\sigma'$ and $E'$ denote the accompanying valuation and environment of the process term $p'$, respectively, after the discrete action $a$ is executed.
- *termination transitions*; The intuition of a (termination) transition $\langle p, \sigma, E\rangle \xrightarrow{\xi,a,\xi'} \langle \checkmark, \sigma', E'\rangle$ is that the process $\langle p, \sigma, E\rangle$ executes the discrete action $a$ with extended valuations $\xi$ and $\xi'$ and thereby transforms into the terminated process $\langle \checkmark, \sigma', E'\rangle$.
- *time transitions*; The intuition of a time transition $\langle p, \sigma, E\rangle \overset{t,\rho}{\longmapsto} \langle p', \sigma', E'\rangle$ is that during the time transition, the extended valuation at each time-point $s \in [0, t]$ is given by $\rho(s)$. At the end-point $t$, the resulting process is $\langle p', \sigma', E'\rangle$.

## 3 Translation of Chi to hybrid automata

In literature, many different hybrid automata definitions exist. Some definitions require solutions for the continuous variables to be differential functions, e.g. [16, 10]. Other definitions allow the more general case of piecewise differential functions, e.g. [11]. Most hybrid automata definitions do not define urgent transitions, or they define urgent transitions in a restrictive way, as in [17]. In [12], urgent transitions are defined in a general way, using a predicate that defines the maximum sojourn time in a location, but instead of invariants and flow clauses, evolution functions are used. With respect to the meaning of jump clauses, that define the behavior of the variables in action transitions, differences also occur: where in [16] the variables can in principle perform arbitrary jumps unless restricted by the jump predicate, in [17], variables in principle remain unchanged unless changes are enforced by the jump predicate.

None of these hybrid automata definitions is expressive enough to be used as the target for the translation of hybrid $\chi$. Therefore, the translation uses a target hybrid automata definition $HA_{\text{u}}$, which will be defined in Section 3.1, that uses features from different hybrid automata definitions. In particular, the definition of the jump predicate in combination with a set of changeable variables is based on [10], the solution concept that allows piecewise differentiable functions is based on [11], and the definition of urgent transitions was inspired by [12].

### 3.1 Hybrid automata definition $HA_{\text{u}}$

**Syntax** A hybrid automaton $HA_{\text{u}}$ consists of the following components:

[3] In $\chi$, the guard operator can be applied to arbitrary process terms. Since it is not possible to translate the guard operator in a general way, the process terms to which the guard operator can be applied are restricted to, among others, the action predicate.

- A finite set of (real-valued) variables $X = \{x_1, \ldots, x_n\}$, the set $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ which denotes the first derivatives of the variables w.r.t. time, and the set $X' = \{x'_1, \ldots, x'_n\}$ which denotes the primed variables that represent values at the conclusion of a discrete change.
- A finite directed multi-graph $(V, E)$, where $V$ denotes a set of vertices (also referred to as control modes or locations) and $E$ denotes a set of edges (control switches).
- Three vertex labeling functions init, inv, and flow that assign to each location $v \in V$ a predicate for initial conditions, invariants and flow conditions, respectively. The free variables of the initial and invariant predicates are from $X$. The free variables of the flow predicates are from $X \cup \dot{X}$.
- An edge labeling function jump, that assigns to each edge $e \in E$ a set of variables ($\subseteq X$) which are allowed to change and a jump condition which is a predicate whose free variables are from $X \cup X'$.
- An edge labeling function guard, that assigns to each edge $e \in E$ a guard which is a predicate whose free variables are from $X$.
- An edge labeling function urgent : $E \to \{\text{true}, \text{false}\}$, that assigns to each edge a boolean: true for an urgent edge, and false for a non-urgent edge.
- A finite set $\Sigma$ of events, and an edge labeling function event : $E \to \Sigma$ that assigns to each edge an event.

Usually, an edge $e$ is represented as $e = (v, v')$, which identifies a source location $v \in V$ and a target location $v' \in V$. This representation cannot be used in case of multi-edges (multiple edges with the same source location and target location). To deal with these, two additional functions are defined: function source : $E \to V$ returns the source location of a given edge, and function target : $E \to V$ returns the target location of a given edge. This results in the following hybrid automata definition: $HA_{\text{u}} = (X, V, \text{init}, \text{inv}, \text{flow}, E, \text{source}, \text{target}, \text{urgent}, \text{guard}, \text{jump}, \Sigma, \text{event})$.

**Semantics** The semantics of a hybrid automaton is defined in terms of a timed transition system. In this transition system, two kinds of transition relations are defined: action transitions and time transitions.

- an action transition is labeled with an action label from an action label set to indicate that the transition may take place on performing that action;
- a time transition is labeled with an non-negative real number to indicate that the transition takes place on idling for that number of time units.

Let $HA_{\text{u}} = (X, V, \text{init}, \text{inv}, \text{flow}, E, \text{source}, \text{target}, \text{urgent}, \text{guard}, \text{jump}, \Sigma, \text{event})$ be a hybrid automaton. Then a *state* of $HA_{\text{u}}$ is a pair $(v, \alpha) \in V \times (X \mapsto \Lambda)$. A state $(v, \alpha)$ of $HA_{\text{u}}$ is *admissible* if $\alpha \models \text{inv}(v)$, and a state $(v, \alpha)$ of $HA_{\text{u}}$ is *initial* if it is admissible and $\alpha \models \text{init}(v)$. Here, notation $\alpha \models \varphi$ denotes the truth value obtained by evaluating the predicate $\varphi$ in $\alpha$, i.e. replacing in $\varphi$ all occurrences of all variables $x \in X$ by $\alpha(x)$.

The transition system interpretation of $HA_{\text{u}}$, written $[\![ HA_{\text{u}} ]\!]$, is the timed transition system $(Q, Q^0, \Sigma, \to, \longmapsto)$, where

- $Q$ is the set of admissible states of $HA_{\text{u}}$;
- $Q^0$ is the set of initial states of $HA_{\text{u}}$;
- $\Sigma$ is the set of events;
- $\to \subseteq Q \times \Sigma \times Q$ is the action transition relation. For $l \in \Sigma$, $(v, \alpha), (v', \alpha') \in Q$,

$$(v, \alpha) \xrightarrow{l} (v', \alpha') \Leftrightarrow \exists_{e \in E} \ (\text{source}(e) = v, \text{target}(e) = v', \text{event}(e) = l, \\ \alpha \models \text{guard}(e), (\alpha, \alpha') \models \text{jump}(e))$$

- $\longmapsto \subseteq Q \times \mathbb{R}_{\geq 0} \times Q$ is the time transition relation. For $r \in \mathbb{R}_{\geq 0}$, $(v, \alpha), (v', \alpha') \in Q$: $(v, \alpha) \xmapsto{r} (v', \alpha')$ iff
  - $v = v'$,
  - $\exists_{\rho:[0,r] \to (X \cup \dot{X} \mapsto \Lambda)}$ such that
    - $\rho_\alpha(0) = \alpha$, $\rho_{\alpha'}(r) = \alpha'$,
    - $\forall_{t \in [0,r]} \begin{cases} \rho(t) \models \text{inv}(v) \wedge \text{flow}(v) \\ \forall_{x \in X} \ (\rho \downarrow x)(t) = (\rho \downarrow x)(0) + \int_0^t (\rho \downarrow \dot{x})(s) ds \end{cases}$
    - $\forall_{e \in E} \ (\text{source}(e) = v \wedge \text{urgent}(e)) \implies \forall_{t \in [0,r)} \ \rho(t) \models \neg \text{guard}(e)$,

where notation $(\alpha, \alpha') \models \text{jump}(e)$ is defined as follows: Let $\text{jump}(e) = (W, \text{pred})$, then notation $(\alpha, \alpha') \models \text{jump}(e)$ is defined as $(\alpha, \alpha') \models \text{pred} \wedge \alpha \restriction (\text{dom}(\alpha) \setminus W) = \alpha' \restriction (\text{dom}(\alpha') \setminus W)$, where notation $(\alpha, \alpha') \models \varphi$ denotes the truth value obtained by evaluating the predicate $\varphi$ by replacing in $\varphi$ all occurrences of all variables $x \in X$ by $\alpha(x)$, and by replacing in $\varphi$ all occurrences of all variables $x \in X'$ by $\alpha'(x)$.

### 3.2 The translation

The translation function $\mathcal{HA} : \chi_{\text{sub}} \to \text{HA}$ is defined in terms of function $\mathcal{T} : \mathcal{P}(\mathcal{V}) \times P \to HA_{\text{frag}}$ that translates a $\chi_{\text{sub}}$ process term $p$ with a set of jumping continuous variables $J$ to a corresponding hybrid automaton fragment $HA_{\text{frag}}$. Function $\mathcal{T}$ is further defined below.

*Hybrid automaton fragment* A hybrid automaton fragment $HA_{\text{frag}}$ is a tuple ($V$, $v_0$, inv, flow, done, $E$, source, target, urgent, guard, jump, $\Sigma$, event), where $V$, inv, flow, $E$, source, target, guard, urgent, jump, $\Sigma$, and event are defined as in the hybrid automation definition. Location $v_0 \in V$ is the initial location of the hybrid automaton fragment, and function done : $V \to \{\text{true}, \text{false}\}$ assigns to each location $v \in V$ a status (also known as done condition) that partitions the locations into terminating locations (status is true), and non-terminating locations (status is false). The distinction between terminating and non-terminating locations is needed in the definition of the translation of some $\chi_{\text{sub}}$ operators (e.g. sequential composition and repetition). Note that a hybrid automaton fragment is defined at the level of $\chi_{\text{sub}}$ process terms, so that no transition system is associated with a hybrid automaton fragment.

*Auxiliary functions on hybrid automaton fragments* In the translation of hybrid chi process terms, with a set of jumping continuous variables, to hybrid automaton fragments, we frequently combine hybrid automaton fragments for which the sets of node names are not disjoint and for which the sets of edge names are not disjoint. This presents us with a technical problem in case we simply wish to use such nodes in the combination. To overcome this technicality we introduce functions $L_*$ and $R_*$ that rename nodes and edges in such a way that for any two hybrid automata fragments $HA_p$ and $HA_q$ we have that the nodes of $L_*(HA_p)$ and $R_*(HA_q)$ are disjoint and that the edges of these hybrid automaton fragments are disjoint.

Let $HA = (V, v_0, \text{inv}, \text{flow}, \text{done}, E, \text{source}, \text{target}, \text{urgent}, \text{guard}, \text{jump}, \Sigma, \text{event})$ be a hybrid automaton fragment. Then we define $L_*(HA) = (V_l, v_{l0}, \text{inv}_l, \text{flow}_l, \text{done}_l, E_l, \text{source}_l, \text{target}_l, \text{urgent}_l, \text{guard}_l, \text{jump}_l, \Sigma_l, \text{event}_l)$ where $V_l = \{(l, v) \mid v \in V\}$, $v_{l0} = (l, v_0)$, for all $v \in V$, $\text{inv}_l(l, v) = \text{inv}(v)$, $\text{flow}_l(l, v) = \text{flow}(v)$, and $\text{done}_l(l, v) = \text{done}(v)$, $E_l = \{(l, e) \mid e \in E\}$, for all $e \in E$, $\text{source}_l(l, e) = (l, \text{source}(e))$, $\text{target}_l(l, e) = (l, \text{target}(e))$, $\text{urgent}_l(l, e) = \text{urgent}(e)$, $\text{guard}_l(l, e) = \text{guard}(e)$, $\text{jump}_l(l, e) = \text{jump}(e)$, $\Sigma_l = \Sigma$ and $\text{event}_l(l, e) = \text{event}(e)$. The function $R_*$ is defined similarly.

*Graphical representation* In our graphical representation of hybrid automaton fragments, only the initial location of the hybrid automaton fragment has an incoming arrow. The terminating locations are drawn with double circles. Single arrows represent non-urgent edges, and double arrows represent urgent edges.

*Variables used in the hybrid automaton* A channel has a type. This defines a number of expressions to be sent or a number of variables to be received in (e.g. $h\,!!\,1, 2$ and $h\,??\,x, y$) via a channel. We refer to the number of expressions to be sent or the number of variables to be received via a channel as the number of arguments of the channel. For simplicity, the set of channel names $H$ with the number of arguments of each channel (denoted by $\text{ar}(h)$ for $h \in H$) which is used in the $\chi$ specification under consideration is assumed. Using this set, valuation $\sigma$ and the set of continuous variables $C$, the set of variables of the hybrid automaton is defined as $X = \text{dom}(\sigma) \cup DC \cup H_{\text{var}}$. The set of variables $DC$ consists of the variables of $C$ prefixed with '$d$': $DC = \{dc \mid c \in C\}$. The set $H_{\text{var}}$ consists of $\text{ar}(h)$ additional variables for each channel from $H$: $H_{\text{var}} = \{h'_1, \cdots, h'_{\text{ar}(h)} \mid h \in H\}$. It is assumed that the sets $\text{dom}(\sigma)$, $DC$, and $H_{\text{var}}$ are pairwise disjoint.

*Translation function $\mathcal{HA}$* Function $\mathcal{HA}$ is now defined as follows: Let $\mathcal{T}_J(p) = (V_p, v_{0_p}, \text{inv}_p, \text{flow}_p, \text{done}_p, E_p, \text{source}_p, \text{target}_p, \text{urgent}_p, \text{guard}_p, \text{jump}_p, \Sigma_p, \text{event}_p)$ be the hybrid automaton fragment of $p$ with the set of jumping continuous variables $J$, and $X$ as defined before. Then the hybrid automaton corresponding to the $\chi_{\text{sub}}$ process $\langle p, \sigma, (C, J, \emptyset, H, \emptyset)\rangle$ is $\mathcal{HA}(\langle p, \sigma, (C, J, \emptyset, H, \emptyset)\rangle) = (X, V_p, \text{init}, \text{inv}_p, \text{flow}, E_p, \text{source}_p, \text{target}_p, \text{urgent}_p, \text{guard}_p, \text{jump}_p, \Sigma_p, \text{event}_p)$, where

$$\forall_{v \in V_p} : \quad \text{init}(v) = \begin{cases} \mathcal{M}(\sigma) & \text{if } v = v_{0_p}, \\ \text{false} & \text{otherwise}, \end{cases}$$
$$\forall_{v \in V_p} : \quad \text{flow}(v) = \text{flow}_p \wedge \dot{\text{time}} = 1.$$

Function $\mathcal{M}$ maps a valuation $\{x_0 \mapsto c_0, \ldots, x_n \mapsto c_n\}$ to a predicate $x_0 = c_0 \wedge \cdots \wedge x_n = c_n$. E.g. $\mathcal{M}(\{x \mapsto 1, y \mapsto 2\}) = (x = 1 \wedge y = 2)$.

**Translation of atomic process terms of $\chi_{\text{sub}}$** In this section, the translation of the atomic process terms of $\chi_{\text{sub}}$ to the corresponding hybrid automaton fragments is defined. Notation $X_{\text{aux}}$ is defined as $X_{\text{aux}} = DC \cup H_{\text{var}}$.

*Delay predicate u* A delay predicate $u$ restricts the allowed behavior of the variables in such a way that the value of the predicate remains true over time. Since $u$ can only perform time transitions, $\mathcal{T}_J(u)$ has only one location without outgoing edges.

$$\mathcal{T}_J(u) = (\{v_0\}, v_0, \text{inv}, \text{flow}, \text{done}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),$$

where $\text{inv}(v_0) = u[DC/\dot{C}]$, $\text{flow}(v_0) = u$, and $\text{done}(v_0) = \text{false}$. Predicate $u[DC/\dot{C}]$ is the predicate $u$ where all occurrences of $\dot{c}$ are replaced by a variable $dc$ from $DC$. E.g. $(\dot{x} = -x + 1 \wedge x \geq 0 \wedge y \in [0, 1])[DC/\dot{C}] = (dx = -x + 1 \wedge x \geq 0 \wedge y \in [0, 1])$.

In $\chi$, it is not possible to reach a state in which the delay predicate evaluates to false, while in hybrid automata it is possible to reach a state in which the flow condition does not hold. For example, in the semantics of $\chi_{\text{sub}}$, the delay predicate $\dot{x} = 0 \wedge \dot{x} = 1$ denotes an inconsistent process, i.e., a process that cannot be reached. To overcome this semantical difference, the invariant is used to prevent entrance in case there is no solution for the delay predicate. As invariants cannot contain dotted variables, these dotted variables are replaced by variables from $DC$ ($\text{inv}(v_0) = u[DC/\dot{C}]$), see Figure 1.

*Guarded action predicate* Let $b$ denote a guard and let $W : r \gg l_{\text{a}}$ denote an action predicate. Then guarded action predicate $b \rightarrow W : r \gg l_{\text{a}}$ behaves as the action predicate $W : r \gg l_{\text{a}}$ if $b = \text{true}$. If $b = \text{false}$, it can perform arbitrary time transitions. An action predicate $W : r \gg l_{\text{a}}$ allows instantaneous changes to the variables from the set $W \cup J$ in such a way that the predicate $r$ is satisfied. The values of variables not in the set $W \cup J$ remain unchanged.

$$\begin{aligned}\mathcal{T}_J(b \rightarrow W : r \gg l_{\text{a}}) = (&\{v_0, v_1\}, v_0, \text{inv}, \text{flow}, \text{done},\\ &\{e\}, \text{source}, \text{target}, \text{urgent}, \text{guard}, \text{jump}, \{l_{\text{a}}\}, \text{event}),\end{aligned}$$

where $\text{inv}(v_0) = \text{true}$, $\text{inv}(v_1) = \text{true}$, $\text{source}(e) = v_0$, $\text{target}(e) = v_1$,
$\text{flow}(v_0) = \text{true}$, $\text{flow}(v_1) = \text{false}$, $\text{urgent}(e) = \text{true}$, $\text{guard}(e) = b$,
$\text{done}(v_0) = \text{false}$, $\text{done}(v_1) = \text{true}$, $\text{jump}(e) = (X_{\text{aux}} \cup W \cup J, \zeta_{W \cup J}(r))$, $\text{event}(e) = l_{\text{a}}$.

A guarded action predicate can only delay while its guard is false. Only for the end-point of the trajectory the guard may become true. Therefore, the flow condition of location $v_0$ is true, and the urgent edge $e$ is guarded with $b$ (see Figure 2).

The difference in notation of a jump predicate in the $\chi_{\text{sub}}$ language and the syntax of hybrid automata defined in Section 3.1 requires function $\zeta_{W \cup J}$. Function $\zeta_{W \cup J}$ renames variables of $W \cup J$ in $r$ to variables with superscript "′", and replaces variables occurring with a "−" superscript in $r$ to variables without any superscript. E.g. $\zeta_{\{x\}}(x + y = x^- + y^- + 5)$ becomes $x' + y = x + y + 5$. The set of variables which is allowed to change is given by $X_{\text{aux}} \cup W \cup J$. Edge $e$ is labeled with the action label $l_{\text{a}}$.

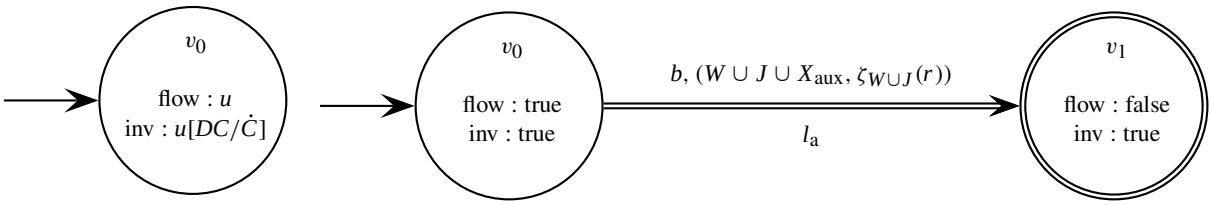


**Fig. 1.** $\mathcal{T}_J(u)$.




**Fig. 2.** $\mathcal{T}_J(b \rightarrow W : r \gg l_{\text{a}})$.

**Translation of the operators of $\chi_{\text{sub}}$** In this section, the translation of the $\chi_{\text{sub}}$ operators to hybrid automaton fragments is defined. We let $p, q$ be closed process terms, and we use $\mathcal{T}_J(i) = (X_i, V_i, v_{0_i}, \text{inv}_i, \text{flow}_i, \text{done}_i, E_i, \text{source}_i, \text{target}_i, \text{urgent}_i, \text{guard}_i, \text{jump}_i, \Sigma_i, \text{event}_i)$ to denote the hybrid automaton fragment of $i$, for $i \in \{p, q\}$.

In the translations, notation $f_{pq}$ is used as an abbreviation for $f_p \cup f_q$, where $f \in \{\text{inv}, \text{flow}, \text{done}, E, \text{source}, \text{target}, \text{urgent}, \text{guard}, \text{jump}, \Sigma, \text{event}\}$.

*Any delay operator* By means of the any delay operator $[p]$, time transitions of arbitrary duration are allowed for the behavior of $p$. Time transitions of $p$ itself are neglected. The any delay operator does not affect the action behavior of $p$. Let $L_*(\mathcal{T}_J(p)) = (V_p, v_{0_p}, \text{inv}_p, \text{flow}_p, \text{done}_p, E_p, \text{source}_p, \text{target}_p, \text{urgent}_p, \text{guard}_p, \text{jump}_p, \Sigma_p, \text{event}_p)$, then

$$\begin{aligned}\mathcal{T}_J([p]) = (&\{v'_{0_p}\} \cup V_p, v'_{0_p}, \text{inv} \cup \text{inv}_p, \text{flow} \cup \text{flow}_p, \text{done} \cup \text{done}_p,\\ &E \cup E_p, \text{source} \cup \text{source}_p, \text{target} \cup \text{target}_p, \text{urgent} \cup \text{urgent}_p,\\ &\text{guard} \cup \text{guard}_p, \text{jump} \cup \text{jump}_p, \Sigma_p, \text{event} \cup \text{event}_p),\end{aligned}$$

where $\text{dom}(\text{inv}) = \text{dom}(\text{flow}) = \text{dom}(\text{done}) = \{v'_{0_p}\}$, $\text{inv}(v'_{0_p}) = \text{true}$, $\text{flow}(v'_{0_p}) = \text{true}$, $\text{done}(v'_{0_p}) = \text{done}_p(v_{0_p})$,
$E = \{e' \mid e \in E_p, \text{source}_p(e) = v_{0_p}\}$, $E \cap E_p = \emptyset$,
$\text{dom}(\text{source}) = \text{dom}(\text{target}) = \text{dom}(\text{urgent}) = \text{dom}(\text{guard}) =$
$\text{dom}(\text{jump}) = \text{dom}(\text{event}) = E$,
$\forall_{e' \in E} : \text{source}(e') = v'_{0_p}$, $\text{target}(e') = \text{target}_p(e)$,
$\text{urgent}(e') = \text{false}$, $\text{guard}(e') = \text{guard}_p(e)$,
$\text{jump}(e') = \text{jump}_p(e)$, $\text{event}(e') = \text{event}_p(e)$.

The automaton fragment $\mathcal{T}_J([p])$ may start with an arbitrary delay. We introduce an additional location $v'_{0_p}$ to $\mathcal{T}_J(p)$ to play the role of the original initial location but now with arbitrary initial delay. Therefore, the invariant and the flow condition of $v'_{0_p}$ are set to true. Also $v'_{0_p}$ becomes the initial location of $\mathcal{T}_J([p])$. Observe that any transition from $v'_{0_p}$ ends up in a location of $\mathcal{T}_J(p)$. Also, any urgent edge from $v'_{0_p}$ is turned into a non-urgent one.

*Sequential composition operator* The sequential composition of process terms $p$ and $q$ behaves as process term $p$ until $p$ terminates, and then continues to behave as process term $q$. Let $L_*(\mathcal{T}_J(p)) = (V_p, v_{0_p}, \text{inv}_p, \text{flow}_p, \text{done}_p, E_p, \text{source}_p, \text{target}_p, \text{urgent}_p, \text{guard}_p, \text{jump}_p, \Sigma_p, \text{event}_p)$, $R_*(\mathcal{T}_J(q)) = (V_q, v_{0_q}, \text{inv}_q, \text{flow}_q, \text{done}_q, E_q, \text{source}_q, \text{target}_q, \text{urgent}_q, \text{guard}_q, \text{jump}_q, \Sigma_q, \text{event}_q)$, then

$$\begin{aligned}\mathcal{T}_J(p;\, q) = (&V, v_{0_p}, \text{inv}_{pq} \upharpoonright V, \text{flow}_{pq} \upharpoonright V, \text{done}_{pq} \upharpoonright V, E_{pq}, \text{source}_{pq},\\ &\text{target} \cup \text{target}_q, \text{urgent}_{pq}, \text{guard}_{pq}, \text{jump}_{pq}, \Sigma_{pq}, \text{event}_{pq}),\end{aligned}$$

where $V = \{v \mid v \in V_p, \neg\text{done}_p(v)\} \cup V_q$, $\text{dom}(\text{target}) = E_p$

$$\forall_{e \in E_p} : \text{target}(e) = \begin{cases} v_{0_q} & \text{if } \text{done}_p(\text{target}_p(e)),\\ \text{target}_p(e) & \text{otherwise.}\end{cases}$$

The initial location of $\mathcal{T}_J(p;\, q)$ is the initial location of $\mathcal{T}_J(p)$. The end-points of the edges that go to terminating locations of $\mathcal{T}_J(p)$ are reconnected to the initial location of $\mathcal{T}_J(q)$ (i.e. $v_{0_q}$). The terminating locations of $\mathcal{T}_J(p)$ are removed. This is safe since we never create hybrid automaton fragments with outgoing edges in terminating locations. The behavior of $\mathcal{T}_J(p;\, q)$ is straightforward: first $p$ is executed, then $q$. Upon termination of $p$, the invariants of the initial location of $q$ must hold.

From the above definition it follows that if the hybrid automaton fragment for $p$ has no terminating locations, i.e. $\forall_{v \in V_p} : \neg\text{done}_p(v)$, then the locations from $\mathcal{T}_J(q)$ are unreachable. Hence, $\mathcal{T}_J(p;\, q)$ and $\mathcal{T}_J(p)$ are identical apart from the unreachable locations/edges.

*Alternative composition operator* The delay behavior of $\mathcal{T}_J(p \,[]\, q)$ is the intersection of the respective delay behaviors of $p$ and $q$. The action behavior is a non-deterministic choice between the first action allowed by $p$ and the first action allowed by $q$. Let $L_*(\mathcal{T}_J(p)) = (V_p, v_{0_p}, \text{inv}_p, \text{flow}_p, \text{done}_p, E_p, \text{source}_p, \text{target}_p, \text{urgent}_p, \text{guard}_p, \text{jump}_p, \Sigma_p, \text{event}_p)$, $R_*(\mathcal{T}_J(q)) = (V_q, v_{0_q}, \text{inv}_q, \text{flow}_q, \text{done}_q, E_q, \text{source}_q, \text{target}_q, \text{urgent}_q, \text{guard}_q, \text{jump}_q, \Sigma_q, \text{event}_q)$, then

$$\begin{aligned}\mathcal{T}_J(p \,[]\, q) = (&\{v_0\} \cup V_{pq}, v_0, \text{inv} \cup \text{inv}_{pq}, \text{flow} \cup \text{flow}_{pq}, \text{done} \cup \text{done}_{pq},\\ &E \cup E_{pq}, \text{source} \cup \text{source}_{pq}, \text{target} \cup \text{target}_{pq},\\ &\text{urgent} \cup \text{urgent}_{pq}, \text{guard} \cup \text{guard}_{pq}, \text{jump} \cup \text{jump}_{pq}, \Sigma_{pq},\\ &\text{event} \cup \text{event}_{pq}),\end{aligned}$$

where $\mathrm{dom}(\mathrm{inv}) = \mathrm{dom}(\mathrm{flow}) = \mathrm{dom}(\mathrm{done}) = \{v_0\}$,
$\mathrm{inv}(v_0) = \mathrm{inv}_p(v_{0_p}) \wedge \mathrm{inv}_q(v_{0_q})$, $\mathrm{flow}(v_0) = \mathrm{flow}_p(v_{0_p}) \wedge \mathrm{flow}_q(v_{0_q})$,
$\mathrm{done}(v_0) = \mathrm{done}_p(v_{0_p}) \wedge \mathrm{done}_q(v_{0_q})$,
$E = \{e' \mid e \in E_{pq}, \mathrm{source}_{pq}(e) \in \{v_{0_p}, v_{0_q}\}\}$, $E \cap E_{pq} = \emptyset$,
$\mathrm{dom}(\mathrm{source}) = \mathrm{dom}(\mathrm{target}) = \mathrm{dom}(\mathrm{urgent}) = \mathrm{dom}(\mathrm{guard}) =$
$\mathrm{dom}(\mathrm{jump}) = \mathrm{dom}(\mathrm{event}) = E$,

$$\forall_{e' \in E} : \begin{array}{ll} \mathrm{source}(e') = v_0, & \mathrm{target}(e') = \mathrm{target}_{pq}(e), \\ \mathrm{urgent}(e') = \mathrm{urgent}_{pq}(e), & \mathrm{guard}(e') = \mathrm{guard}_{pq}(e), \\ \mathrm{jump}(e') = \mathrm{jump}_{pq}(e), & \mathrm{event}(e') = \mathrm{event}_{pq}(e). \end{array}$$

The invariant, flow and done conditions of the initial location $v_0$ of $\mathcal{T}_J(p \,[]\, q)$ are the conjunction of the invariants, flow conditions and done conditions of $v_{0_p}$ and $v_{0_q}$, respectively. All outgoing edges from the original initial nodes, i.e., $v_{0_p}$ an $v_{0_q}$, are copied to the new initial node with their original target. It can be the case that the original initial nodes are not reachable anymore. In that case, they can of course be removed from the hybrid automaton fragment.

*Repetition operator* Process term $*p$ represents the infinite repetition of process term $p$. Let $L_*(\mathcal{T}_J(p)) = (V_p, v_{0_p}, \mathrm{inv}_p, \mathrm{flow}_p, \mathrm{done}_p, E_p, \mathrm{source}_p, \mathrm{target}_p, \mathrm{urgent}_p, \mathrm{guard}_p, \mathrm{jump}_p, \Sigma_p, \mathrm{event}_p)$, then

$$\begin{aligned} \mathcal{T}_J(*p) = (V, v_{0_p}, \mathrm{inv}_p \restriction V, \mathrm{flow}_p \restriction V, \mathrm{done}_p \restriction V, \\ E_p, \mathrm{source}_p, \mathrm{target}, \mathrm{urgent}_p, \mathrm{guard}_p, \mathrm{jump}_p, \Sigma_p, \mathrm{event}_p), \end{aligned}$$

where $V = \{v \mid v \in V_p, \neg\mathrm{done}_p(v)\}$, $\mathrm{dom}(\mathrm{target}) = E_p$

$$\forall_{e \in E_p} : \mathrm{target}(e) = \begin{cases} v_{0_p} & \text{if } \mathrm{done}_p(\mathrm{target}_p(e)), \\ \mathrm{target}_p(e) & \text{otherwise.} \end{cases}$$

The end-points of the edges that go to terminating locations are reconnected to the initial location $v_{0_p}$. The terminating locations are removed. As mentioned previously, this is safe, because we never create hybrid automaton fragments with outgoing edges in terminating locations.

### 3.3 Correctness of the translation

In [2], it is proved that any transition of a $\chi$ model can be mimicked by a transition in the corresponding hybrid automaton model and vice versa. This indicates that the translation is correct.

## 4 Analysis of $\chi_{\mathrm{sub}}$ specifications using PHAVer

Hybrid systems can be modeled as hybrid automata and verified using reachability analysis techniques. Over the past years, various hybrid systems techniques (hybrid automata based) and tools (e.g. HyTech [18], and PHAVer [13] have been successfully introduced into the verification of hybrid systems. PHAVer (Polyhedral Hybrid Automaton Verifyer) [13] is a tool for analyzing linear hybrid I/O-automata. If we restrict the linear hybrid I/O-automata from [19, 20] to the class of linear hybrid I/O-automata without input variables and without output variables, then this class of linear hybrid I/O-automata is a subclass of the $HA_{\mathrm{u}}$ automata as defined in Section 3.1. As a consequence, the $\chi_{\mathrm{sub}}$ specifications that can be verified using PHAVer are restricted to those specifications that result in $HA_{\mathrm{u}}$ automata with linear invariant and flow conditions, and linear jump conditions, where a linear constraint is defined as a constraint over a set of variables $X$ that is of the form $\sum_i a_i v_i + b \Diamond 0$, with $a_i, b \in \mathbb{Z}$, $v_i \in X$, and $\Diamond \in \{<, \leq, =\}$. Furthermore, the $\chi_{\mathrm{sub}}$ specifications are restricted to those specifications that result in $HA_{\mathrm{u}}$ automata which do not contain urgent transitions. This restriction is because in the semantics of the $HA_{\mathrm{u}}$ definition, transitions can be urgent, while in the linear hybrid I/O-automata, transitions cannot be urgent. Note that in [2], the relation between linear hybrid I/O-automata and $HA_{\mathrm{u}}$ automata has been formalized.

### 4.1 Case study

The verification of a $\chi_{\mathrm{sub}}$ specification using PHAVer is illustrated by means of an case study: the water level monitor, which is taken from [14].

The water level in a tank is controlled through a monitor, which continuously senses the water level and turns a pump on and off. When the pump is off, the water level, denoted by the variable $y$, drops by 2 units per second;

when the pump is on, the water level rises by 1 unit per second. There is a time delay of 2 time units between the time that the monitor signals to change the status of the pump and time that the change becomes effective (this is modeled by the variable $x$). Initially the water level is 1 and the pump is turned on. The water-level monitor is modeled in $\chi_{\text{sub}}$ as follows:

$$
\begin{array}{l}
\langle\ \mathsf{cont}\ x, y \\
,\ x = 0, y = 1 \\
\mid *(\ (\ \dot{x} = 1 \wedge \dot{y} = 1 \quad \wedge y \leq 10\ []\ [y \geq 10 \rightarrow \{x\} : x = 0 \gg \tau]\ ) \\
\quad ;\ (\ \dot{x} = 1 \wedge \dot{y} = 1 \quad \wedge x \leq 2 \quad []\ [x \geq 2 \rightarrow \{\emptyset\} : \text{true} \gg \tau]\ ) \\
\quad ;\ (\ \dot{x} = 1 \wedge \dot{y} = -2 \wedge y \geq 5 \quad []\ [y \leq 5 \rightarrow \{x\} : x = 0 \gg \tau]\ ) \\
\quad ;\ (\ \dot{x} = 1 \wedge \dot{y} = -2 \wedge x \leq 2 \quad []\ [x \geq 2 \rightarrow \{\emptyset\} : \text{true} \gg \tau]\ ) \\
\quad )\ \\
\rangle
\end{array}
$$

This specification is translated into a linear hybrid automaton $HA_{\text{u}}$, which is shown in Figure 3.

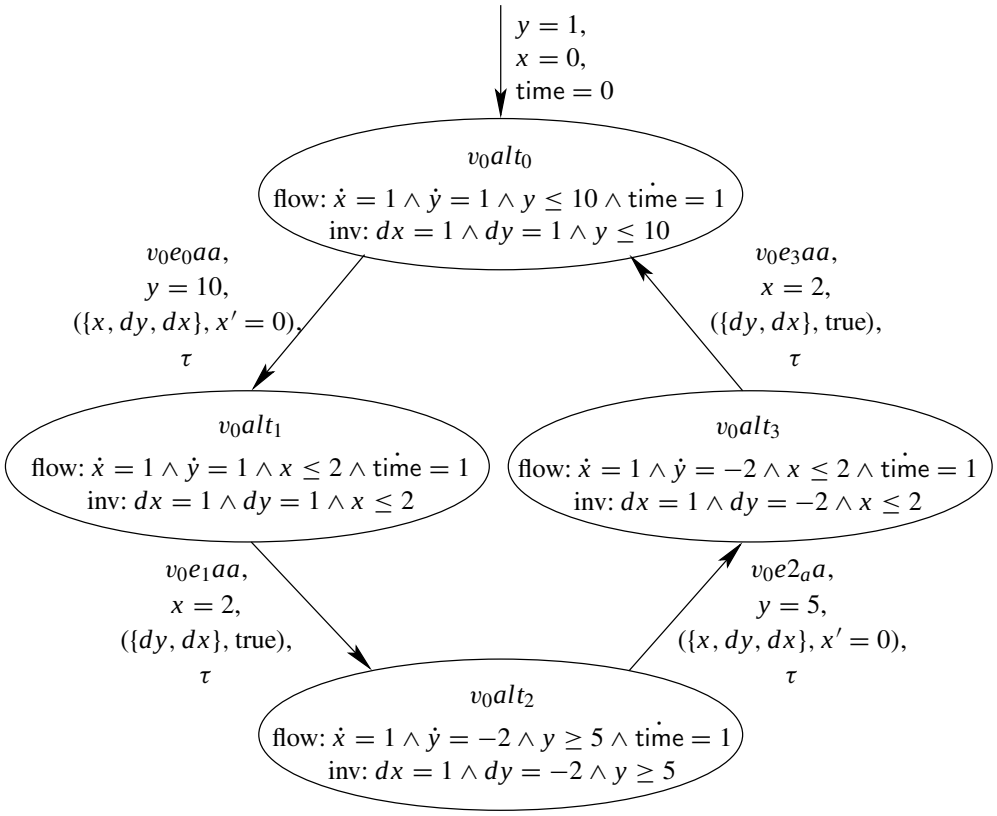


**Fig. 3.** Generated water-level monitor automaton.

This hybrid automaton $HA_{\text{u}}$ is linear, which means there exists a corresponding linear hybrid I/O-automaton that can be verified using PHAVer. The safety property that the water level has to satisfy $1 < y < 12$ has been verified using PHAVer. PHAVer reported that this safety property holds in all locations of the corresponding linear hybrid I/O-automaton, which also indicates that this safety property holds in the linear hybrid automaton $HA_{\text{u}}$. From Section 3.3, we know that any transition of a $\chi_{\text{sub}}$ specification can be mimicked by a transition in the corresponding hybrid automaton $HA_{\text{u}}$ and vice versa, which indicates this safety property also holds in the original $\chi_{\text{sub}}$ specification.

## 5 Concluding remarks

A formal translation of a subset of $\chi$ to hybrid automata is defined. Note that the subset of $\chi$ that is defined in this paper does not contain the parallel composition operator. The translation of this $\chi$ operator, and several additional operators, to hybrid automata is defined in [2]. To the best of our knowledge, none of the hybrid automaton definitions from literature is expressive enough to be used as the target for the translation of hybrid $\chi$. Therefore, the translation uses a target hybrid automata definition that uses features from different hybrid automata definitions. In [2] it is proved that any transition of a $\chi$ model can be mimicked by a transition in the corresponding hybrid automaton model and vice versa, which indicates that the translation is correct. The translation enables verification of $\chi$ specifications using existing hybrid automaton based verification tools. By means of a case study, it is shown that the translation can indeed be used to verify properties of a $\chi$ specification using the hybrid automaton based tool PHAVer.

## Acknowledgments

The authors would like to thank Rolf Theunissen for performing the case study using PHAVer.

## References

1. van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H.: Syntax and consistent equation semantics of hybrid Chi. Journal of Logic and Algebraic Programming (2006) to appear.
2. Man, K.L., Schiffelers, R.R.H.: Formal Specification and Analysis of Hybrid Systems. PhD thesis, Eindhoven University of Technology (2006)
3. Naumoski, G., Alberts, W.: A Discrete-Event Simulator for Systems Engineering. PhD thesis, Eindhoven University of Technology (1998)
4. van Beek, D.A., van den Ham, A., Rooda, J.E.: Modelling and control of process industry batch production systems. In: 15th Triennial World Congress of the International Federation of Automatic Control, Barcelona (2002) CD-ROM.
5. Arends, N.W.A.: A Systems Engineering Specification Formalism. PhD thesis, Eindhoven University of Technology (1996)
6. Fábián, G.: A Language and Simulator for Hybrid Systems. PhD thesis, Eindhoven University of Technology (1999)
7. van Beek, D.A., Rooda, J.E.: Languages and applications in hybrid modelling and simulation: Positioning of Chi. Control Engineering Practice **8** (2000) 81–91
8. Bos, V., Kleijn, J.J.T.: Formal Specification and Analysis of Industrial Systems. PhD thesis, Eindhoven University of Technology (2002)
9. Bos, V., Kleijn, J.J.T.: Automatic verification of a manufacturing system. Robotics and Computer Integrated Manufacturing **17** (2000) 185–198
10. Alur, R., Henzinger, T.A., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Transactions on Software Engineering **22** (1996) 181–201
11. van der Schaft, A.J., Schumacher, J.M.: An Introduction to Hybrid Dynamical Systems. Volume 251 of Springer Lecture Notes in Control and Information Sciences. Springer (2000)
12. Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: An approach to the description and analysis of hybrid systems. In: Workshop on Theory of Hybrid Systems. (1992) 149–178
13. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In Morari, M., Thiele, L., eds.: Hybrid Systems: Computation and Control, 8th International Workshop. Volume 3414 of Lecture Notes in Computer Science. Springer-Verlag (2005) 258–273
14. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science **138** (1995) 3–34
15. Plotkin, G.D.: A structural approach to operational semantics. Technical Report DIAMI FN-19, Computer Science Department, Aarhus University (1981)
16. Henzinger, T.A.: The theory of hybrid automata. In Inan, M., Kurshan, R., eds.: Verification of Digital and Hybrid Systems. Volume 170 of NATO ASI Series F: Computer and Systems Science. Springer-Verlag, New York (2000) 265–292
17. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: A user guide to HYTECH. In: First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS. Lecture Notes in Computer Science 1019, Springer Verlag (1995) 41–71
18. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HYTECH: A model checker for hybrid systems. International Journal on Software Tools for Technology Transfer **1** (1997) 110–122
19. Frehse, G., Han, Z., Krogh, B.: Assume-guarantee reasoning for Hybrid I/O-Automata by over-approximation of continuous interaction. In: 43rd IEEE Conference on Decision and Control, Nassau Bahamas (2004) 479–484
20. Frehse, G.: Compositional verification of hybrid systems using simulation relations. PhD thesis, Radbout University Nijmegen (2004)