Relating hybrid Chi to other formalisms

D.A. van Beek, J.E. Rooda, R.R.H. Schiffelers [1,2]

Department of Mechanical Engineering
Eindhoven University of Technology (TU/e)
Eindhoven, The Netherlands

K.L. Man, M.A. Reniers[3]

Department of Mathematics and Computer Science
Eindhoven University of Technology (TU/e)
Eindhoven, The Netherlands

Abstract

The hybrid χ (Chi) formalism is suited to modeling, simulation and verification of hybrid systems. It integrates concepts from dynamics and control theory with concepts from computer science, in particular from process algebra and hybrid automata. In this paper, we first provide an overview of χ. Then, the χ formalism is related to other formalisms by means of translation schemes: a translation scheme from continuous-time PWA systems to χ, a translation scheme from discrete-time PWA systems to χ, and a translation scheme from hybrid automata to χ. In order to be able to use existing model checkers that use hybrid automata like input languages, we developed and implemented a translation scheme from a subset of χ to hybrid automata. To illustrate this approach, a case study has been performed: a water level monitor has been modeled using χ. Using the implemented translation scheme from χ to hybrid automata, we obtain a hybrid automata model for the water level monitor. From this model, code that can be used as input for the model checker PHAVer is generated.

Keywords: hybrid systems, hybrid automata, PWA systems, modelling, simulation, verification.

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.

The DC world view is that of a predominantly continuous-time system, which is modeled by means of differential (algebraic) equations, or by means of a set of trajectories. Hybrid phenomena are modeled by means of discontinuous functions and/or switched equation systems. The evolution of a hybrid system in the

[1] Work partially done in the framework of the HYCON Network of Excellence, contract number FP6-IST-511368

[2] Email: D.A.v.beek@tue.nl, J.E.Rooda@tue.nl, R.R.H.Schiffelers@tue.nl

[3] Email: kalok@email.it, M.A.Reniers@tue.nl

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

continuous-time domain is considered as a set of piecewise continuous functions of time (one for each variable).

Analysis and synthesis of hybrid systems in the DC domain are done, among others, by means of piecewise affine (PWA) systems, mixed logic dynamical (MLD) systems or linear complementarity (LC) systems, see [22] for an overview relating these different classes. A different framework to consider hybrid systems are differential (algebraic) equations with discontinuous right-hand sides, the semantics of which can be defined using differential inclusions. Such differential inclusions allow modeling of relays, valves or any kind of on/off switching elements at a high level of abstraction in control systems with so-called sliding modes [17,46].

The CS world view is that of a predominantly discrete-event system. A well-known model is a (hybrid) automaton, but modeling of discrete-event systems is also based on, among others, process algebras, Petri nets, and data flow languages. For modeling and analysis of hybrid phenomena, discrete-event formalisms are extended in different ways with some form of differential (algebraic) equations. The most influential hybrid system model is that of a hybrid automaton such as defined in [35,1,3,24,44,31,30]. An essential difference between such a hybrid automaton and a DC hybrid system model is that where in the DC hybrid system model there are no actions, in the hybrid automaton, discontinuities take place mainly by means of (labeled) actions. By means of actions, the hybrid automaton switches from one mode/location to another mode/location.

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 χ (Chi) formalism [6,32] [4] 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 χ 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, such as integrated circuit manufacturing plants, breweries, and process industry plants [4]. These plants consist of many independently operating entities such as machines, buffers, liquid storage tanks and reactors. The entities interact with each other in a discrete fashion, for example the exchange of products or information, or in a continuous fashion, for example sharing a liquid flow.

The χ formalism has been designed to model interacting parallel entities representing both discrete and continuous behavior in an easy and intuitive way. This is ensured by means of the following concepts: 1) different classes of variables: discrete and continuous, of subclass jumping or non-jumping, and algebraic (see Section 2.2 for more details on these classes); 2) a small number of atomic state-

[4] The χ language as defined in [32] has small corrections w.r.t. the version defined in [6]

ments and operators on them that can be freely combined, and that have been designed to support easy and intuitive modeling; 3) different interaction mechanisms: handshake synchronization and synchronous communication that allow interaction between processes without sharing variables, and shared variables that enable modular composition of continuous-time or hybrid processes; 4) its 'consistent equation semantics' that enforces state changes to be consistent with delay predicates, that combine the invariant and flow clauses of hybrid automata; 5) differential algebraic equations as a process term as in mathematics; 6) process terms for scoping that integrate abstraction, local variables, local channels and local recursion definitions; 7) process definition and instantiation that enable process re-use, encapsulation, hierarchical and/or modular composition of processes; and 8) several user-friendly syntactic extensions.

The semantics of χ is defined by means of deduction rules in the style of Plotkin's Structural Operational Semantics (SOS) [38,39] that associate a hybrid transition system with a χ process. A set of axioms is presented for a notion of bisimilarity [34]. The χ formalism integrates ease of modeling with a straightforward semantics.

Although the semantics is formally defined, the straightforward and elegant syntax and semantics are also highly suited to non-computer scientists. For example, the χ language is successfully used in the courses 'Analysis of manufacturing systems', 'Supervisory machine control' and 'Analysis of hybrid systems' for students of the Department of Mechanical Engineering at the Eindhoven University of Technology.

In this paper, we investigate the relations between other formalisms and χ. One of the formalisms to describe hybrid systems are piecewise affine systems [45]. General translation schemes from continuous-time piecewise affine systems and discrete-time piecewise affine systems to χ are defined. This shows that piecewise affine systems can be represented by equivalent χ specifications. Another formalism to describe hybrid systems is the theory of hybrid automata. Formal translations between the theory of hybrid automata and χ (in both directions) have been defined. The translation from hybrid automata to χ aims to show that the χ formalism is at least as expressive as the theory of hybrid automata. The translation from a subset (χ_{sub}) of χ to hybrid automata enables verification of χ_{sub} specifications using existing hybrid automata based verification tools. Furthermore, it is proved that any transition of a χ_{sub} specification can be mimicked by a transition in the corresponding hybrid automaton and vice versa, which indicates that the translation as defined in this paper is correct. As an additional benefit, the later translation enables the possibility to verify properties about the χ model using verification tools that are based on hybrid automata. This is illustrated by means of a case study using the model checker PHAVer (Polyhedral Hybrid Automaton Verifier) [19].

Related work

The χ formalism is a hybrid process algebra, and is thus related to the other hybrid process algebras: HyPA [15], the process algebra for hybrid systems $\text{ACP}_{\text{hs}}^{\text{srt}}$ [9], the ϕ-Calculus [43], the hybrid formalisms based on CSP [21,13], and the process algebra from [47]. A detailed comparison between these formalisms can be found in [6,32], which also discusses the relations between χ and the hybrid automaton

definitions [24,3,44,31,30,35,26], the formalisms based on hybrid automata such as Charon [2] and Masaccio [23], and the hybrid automaton based tools HyTech [25] and PHAVer [19]. Other related work based on hybrid action systems and the synchronous approach is discussed below.

Hybrid action systems [42] extend conventional (discrete) actions with differential actions, that are used to model continuous-time dynamics. The formalism is based on a much smaller set of primitives than the χ language. Unlike χ, which allows in principle orthogonal combination of its primitives, combination of the primitives in hybrid action systems is quite restricted. Another main difference is the semantics of parallel composition. Where conventional (discrete) actions interleave in parallel composition, as in χ, the continuous-time behavior of parallel composition of differential actions is defined as 'linear composition' (involving among others addition of trajectories). In χ, $u \parallel u'$ equals $u \wedge u'$, so that parallel composition of delay predicates u, used for the specification of differential algebraic equations, is defined as conjunction.

A nice overview of the synchronous approach, as adopted by the three synchronous languages Esterel, Lustre and Signal, is given in [8]. Essential to this approach is the division of time into discrete instants and the distinction of inputs and outputs of a system. Such a synchronous system is a discrete-time system if the time instants are equally spaced, and otherwise a discrete-event system. Execution of a synchronous model is, in principle, a deterministic transformation, at each time-instant, of the values for each of the inputs and internal state to the values of the outputs and the internal state. The main purpose of these formalisms is the development of safety-critical, embedded, discrete-time or discrete-event control systems. A characteristic difference with the hybrid automaton related formalisms, discussed above, is the semantics of parallel composition. The hybrid automaton related languages all have an interleaving (non-deterministic) semantics for the execution of actions. The synchronous languages, in principle, define the behavior of parallel composition of synchronous input/output systems as the (deterministic) conjunction of the behaviors. This reduces the state-explosion problem of parallel composition, but complicates implementations of distributed systems. Another difference is that continuous-time and hybrid systems (a combination of discrete-event or discrete-time systems with continuous-time systems) cannot be specified using the synchronous languages Esterel, Lustre and Signal. The synchronous approach is to some extent also present in the χ language, since the parallel composition of delay predicates is defined as conjunction ($u \parallel u'$ equals $u \wedge u'$), and each connection between an input and an output of synchronous models can be modeled by means of an algebraic variable in χ.

Several formalisms have defined translations to and from hybrid automata. An informal translation of a hybrid automaton to HyPa is defined in [14]. In [10], a formal transformation of a hybrid automaton and of the parallel composition of hybrid automata to $\mathrm{ACP}_{\mathrm{hs}}^{\mathrm{srt}}$, with proof of correctness, is defined. Finally, in [41], linear hybrid action systems have been shown to be a strict subclass of linear hybrid automata, and a translation of a linear hybrid action system, with proof of correctness, to a linear hybrid automaton has been defined.

In [12] it has been shown that different timed model checkers each have their own

strengths and weaknesses. Therefore, apart from the translations between hybrid χ and hybrid automata as defined in this paper, for verification of timed χ models, translations to several tools are defined: 1) a translation [48] to the action-based process algebra μCRL [20], used as input language for the verification tool CADP [16]; (2) a translation to PROMELA, a state-based, imperative language, used as input language for the verification tool SPIN [27]; and (3) a translation [11] to the timed automaton based input language of the UPPAAL [29] verification tool.

Outline

This paper is organized as follows: In Section 2, the χ formalism is presented, and its use is illustrated by means of examples. Translation schemes from PWA systems to χ are described in Section 3. Section 4 describes the translation of hybrid automata to χ and the translation of (a subset of) χ to hybrid automata, and the case study. Finally, some concluding remarks are given in Section 5.

2 The χ language

This section presents a concise definition of the syntax and informal semantics of χ. The syntax definition is incomplete in the sense that the syntax of predicates, expressions, etc. is omitted. This is done because different implementations of χ, such as tools for simulation, verification, or real-time control, may impose different syntactical restrictions. The intention of this section is to define the χ formalism that encompasses a variety of (future) tools without posing unnecessary syntactical restrictions. Futhermore, for the sake of brevity, the scope operators (variable scope, channel scope and recursion scope), the signal emission, jump enabling, encapsulation, and urgent communication operators, and several syntactic extentions are omitted.

2.1 Syntax

A χ process is of the following form:

$$
\begin{array}{l}
\langle\ \mathsf{disc}\ s_1, \ldots, s_k \\
,\ \mathsf{cont}\ x_1, \ldots, x_l \\
,\ \mathsf{alg}\ z_1, \ldots, z_m \\
,\ \mathsf{chan}\ h_1, \ldots, h_n \\
,\ i \\
,\ X_1 \mapsto p_1, \ldots, X_r \mapsto p_r \\
::\ p \\
\rangle,
\end{array}
$$

where $s_1, \ldots, s_k$ denote the discrete variables, $x_1, \ldots, x_l$ denote the non-jumping continuous variables, $z_1, \ldots, z_m$ denote the algebraic variables, $h_1, \ldots, h_n$ denote the urgent channels, i denotes an initialization predicate that restricts the allowed values of the variables initially, $X_1 \mapsto p_1, \ldots, X_r \mapsto p_r$ denote the recursion definitions, $X_1, \ldots, X_r$ denote recursion variables, and p, $p_1, \ldots, p_r$ are process terms.

In the notation defined above, it is required that the discrete, continuous, and algebraic variables are distinct. Besides the declared variables, the existence of the predefined reserved real-valued global variable time which denotes the current time, the value of which is initially zero, is assumed. This variable cannot be declared. It can only be used as a 'read-only' variable in expressions. When time passes, variable time increases with rate 1.

As a shorthand, the keyword preceding variables of a certain type is omitted when there are no variables of that type, and the keyword chan is omitted when there are no channel declarations. Also the initialization predicate i and the recursive definitions $X_1 \mapsto p_1, \ldots, X_r \mapsto p_r$ may be omitted, indicating a predicate that always holds and an empty list of recursive definitions, respectively.

The set of process terms (statements) P is defined by the following grammar for the process terms $p \in P$:

$p ::= W : r \gg l_{\mathrm{a}}$	action predicate	W	set of variables
		r	predicate
		l_{a}	action label
$\mid \mathbf{x}_n := \mathbf{e}_n$	multi-assignment	$\mathbf{x}_n$	variables
		$\mathbf{e}_n$	expressions
$\mid$ skip	internal action		
$\mid u$	delay predicate	u	predicate
$\mid [p]$	any delay		
$\mid p;\ p$	sequential composition		
$\mid b \rightarrow p$	guard	b	predicate
$\mid p \,[\!]\, p$	alternative composition		
$\mid p \parallel p$	parallel composition		
$\mid h \,!!\, \mathbf{e}_n$	send process term	h	channel
$\mid h \,!\, \mathbf{e}_n$	delayable send process term		
$\mid h \,??\, \mathbf{x}_n$	receive process term		
$\mid h \,?\, \mathbf{x}_n$	delayable receive process term		
$\mid \Delta d$	delay for d time units	d	numeric expr.
$\mid X$	recursion variable		
$\mid *p$	infinite repetition		
$\mid l_{\mathrm{p}}(\mathbf{x}_k, \mathbf{h}_m, \mathbf{e}_n)$	process instantiation	l_{p}	process label
		$\mathbf{x}_k$	actual variables
		$\mathbf{h}_m$	actual channels

Here, W is a set of variables such that $\mathsf{time} \notin W$, r is a predicate over variables (including the variable time), dotted continuous variables, and '$-$' superscripted variables (including the dotted variables, e.g. x^- and $\dot{x}^-$). The action label l_{a} is taken from a given set A_{label} which at least contains the special action label τ representing the internal or silent step [33]. Notation $\mathbf{x}_n$ $(n \geq 1)$ denotes the variables $x_1, \ldots, x_n$ such that $\mathsf{time} \notin \{\mathbf{x}_n\}$, $\mathbf{e}_n$ $(n \geq 1)$ denotes the expressions $e_1, \ldots, e_n$, and u and b are both predicates over variables (including the variable time) and dotted continuous variables. For $n = 0$, $h\,!!\,\mathbf{e}_n$ and $h\,??\,\mathbf{x}_n$ can be written $h\,!!$ and $h\,??$, respectively, where h is a channel. Finally, H is a set of channels, $\mathbf{h}_m$ denotes the actual channels $h_1, \ldots, h_m$.

The operators are listed in descending order of their binding strengths as follows $\rightarrow$, ; , $\{\|\,, [\!]\,\}$. 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 informal, concise explanation of this syntax is given below.

2.2 Informal semantics of processes

The behavior of χ processes is defined in terms of actions and delays [5]. Actions define instantaneous changes, where time does not change, to the values of variables. Delays involve the passing of time, where for all variables their trajectory as a function of time is defined.

The variables are grouped into different classes with respect to the delay behavior and action behavior. With respect to the delay behavior, the variables are divided into the following classes:

- The discrete variables, the values of which remain constant while delaying.
- The continuous variables, the values of which change according to an absolutely continuous function [6] of time while delaying. The values of continuous variables are further restricted by delay predicates, that are usually in the form of differential algebraic equations.
- The dotted continuous variables, the values of which change according to an integrable, possibly discontinuous function of time while delaying.
- The algebraic variables, that behave in a similar way as continuous variables. The differences are that algebraic variables may change according to a discontinuous function of time, and that algebraic variables are not allowed to occur as dotted variables.
- The predefined variable time, that denotes the current time.

With respect to the action behavior, the variables are divided into two classes:

[5] Formally, the behavior of χ processes is defined in terms of action transitions and time transitions (see Section 2.4). Informally, we use the term actions to refer to action transitions, and the term delays to refer to time transitions.

[6] A function $f(x)$ is *continuous* at $x \in X$ provided that for all $\varepsilon > 0$, there exists $\delta > 0$ so that $|x - y| \leq \delta$ implies $|f(x) - f(y)| \leq \varepsilon$. Roughly speaking, for single-valued functions this means that we can draw the graph of the function without taking the pencil of the paper. The class of absolutely continuous functions consists of continuous functions which are differentiable almost everywhere in Lebesgue sense. This class includes the differentiable functions.

- The non-jumping variables, the values of which by default do not change in actions. The changes of non-jumping variables need to be explicitly specified.
- The jumping variables, the values of which by default can jump to arbitrary values in actions. The values after jumping can be restricted by means of action predicates, send and receive process terms, or delay predicates (equations).

The discrete and continuous variable classes can be divided into jumping and non-jumping versions. For the other classes, such a division is not possible: the dotted continuous and algebraic variables are by definition jumping with respect to the action behavior, and the predefined variable time is by definition non-jumping.

2.3 Informal semantics of process terms

There are several means to change the value of a variable, depending on the class of the variable. The main means for changing the value of a variable are the action predicate and multi-assignment for instantaneous changes, and the delay predicate for the changes of variables over time.

An instantaneous change of the value of a discrete or continuous variable in χ is always connected to the execution of an action. In action predicates, the action is represented by a label. Other types of action are related to communication, which is treated below, in the paragraph on parallelism. *Action predicate* $W : r \gg l_{\mathrm{a}}$ denotes instantaneous changes to the variables from set W, by means of an action labeled l_{a}, such that predicate r is satisfied. The predefined global variable time cannot be assigned. The discrete and continuous variables that are not mentioned in W remain unchanged, and the variables from set W together with the dotted continuous variables and algebraic variables may obtain 'arbitrary' values, provided that the predicate r is satisfied and the process remains consistent. A '$-$' superscripted occurrence of a variable refers to the value of the variable prior to execution of the action predicate, and a normal (non-superscripted) occurrence of a variable refers to the value of that variable after the execution of the action predicate. Note that it can be the case that different instantaneous changes satisfy the predicate, this may result in non-determinism. Consider for example the following specification: $\langle\ \mathsf{disc}\ x :: \{x\} : x^2 = 1 \gg a\ \rangle$. After the discrete change, the value of x can be 1 or -1.

Multi-assignment $\mathbf{x}_n := \mathbf{e}_n$ for $n \geq 1$ is an abbreviation for an action predicate that simultaneously changes the values of variables $x_1, \ldots, x_n$ to the values of expressions $e_1, \ldots, e_n$, respectively. For $n = 1$, this gives an assignment $x := e$.

Process term skip is an abbreviation for an action predicate that can perform an internal action (τ), such that only the algebraic and dotted variables can change.

In principle, continuous and algebraic variables change arbitrarily over time when delaying, although, depending on the class of the variable, they may have to respect some continuity requirements, see [32,6] for more details. 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 u remains true over time. For example, invariants from hybrid automata can be modeled in χ using delay predicates.

Besides the specification of delay by means of delay predicates, arbitrary delay

can be described by means of the *any delay operator* $[p]$, where p denotes a process term. The resulting behavior is such that arbitrary delays are allowed. When $[p]$ delays, it remains unchanged and the delay behavior of p is ignored. The action behavior of p remains unchanged in $[p]$. When $[p]$ performs an action, the any delay operator disappears.

The *sequential composition* of process terms p and q $(p;\ q)$ behaves as process term p until p terminates, and then continues to behave as process term q.

The *guarded process term* $b \rightarrow p$ can do whatever actions p can do under the condition that the guard b evaluates to true. When $b \rightarrow p$ performs an action, the guard operator disappears. The guarded process term can delay according to p under the condition that during the delay the guard b holds. The guarded process term can perform arbitrary delays under the condition that during the delay, possibly excluding the first and last time point of the delay, the guard b does not hold. During delays, the guard operator remains.

The *alternative composition operator* $[]$ allows a non-deterministic choice between different actions of a process. With respect to time behavior, both sides of the composition have to synchronize. This means that the trajectories of the variables have to be agreed upon by both sides of the composition. This means that $[]$ is a strong time-deterministic [36] choice operator.

Parallelism can be specified by means of the *parallel composition operator* $\|$. Parallel processes interact by means of shared variables or by means of synchronous point-to-point communication/synchronization via a channel. Channels are denoted as labels (identifiers). The parallel composition $p \,\|\, q$ synchronizes the time behavior of p and q, interleaves the action behavior (including the instantaneous changes of variables) of p and q, and synchronizes matching send and receive actions. The synchronization of time behavior means that only the time behaviors that are allowed by both p and q are allowed by their parallel composition. The consistent equation semantics of χ enforces that actions by p (or q) are allowed only if the values of the variables before and after the actions are consistent with the other process term q (or p). This means, among others, that the delay predicates of q must hold before and after execution of an action by p.

By means of the *send process term* $h\,!!\,e_1, \ldots, e_n$, for $n \geq 1$, the values of expressions $e_1, \ldots, e_n$ are sent via channel h. By means of the *receive process term* $h\,??\,x_1, \ldots, x_n$, for $n \geq 1$, values for $x_1, \ldots, x_n$ are received from channel h. We assume that all variables in the sequence $\mathbf{x}_n$ are syntactically different. Communication in χ is the sending of values by one parallel process via a channel to another parallel process, where the received values are stored in variables. For communication, the acts of sending and receiving (values) have to take place in different parallel processes at the same moment in time. In case no values are sent and received, we refer to synchronization instead of communication.

Process terms $h\,!\,\mathbf{e}_n$, and $h\,?\,\mathbf{x}_n$ are the respective delayable counterparts of $h\,!!\,\mathbf{e}_n$ and $h\,??\,\mathbf{x}_n$. They are defined by means of the any delay operator as $[h\,!!\,\mathbf{e}_n]$ and $[h\,??\,\mathbf{x}_n]$, respectively.

The *delay process term* Δd denotes a process term that first delays for d time units, and then terminates by means of an internal action τ. The value of expression d is evaluated at the first delay or action by Δd. The delay process term

constrains the duration of a delay, while a delay predicate u constrains the trajectories of the model variables during the delay (and of course in case of invariants expressed in u, u also constrains the duration of the delay). The fact that process term Δd terminates by means of an action ensures that time-outs enforce a choice in alternative composition. Consider for example the following χ specification: $\langle\ \mathsf{cont}\ x, x = 0 :: \dot{x} = 1 \ [\!]\ \Delta 1\ \rangle$. By means of delay predicate $\dot{x} = 1$, the trajectories of variable x are limited. By means of $\Delta 1$, the duration of the delays is limited to 1. After a delay of 1 time unit, the process terminates.

Process term X denotes a *recursion variable* (identifier) that is defined after the variable declarations $(X_1 \mapsto p_1, \ldots, X_r \mapsto p_r)$. Recursion variable X can do whatever the process term of its definition can do.

Process term $*p$ represents the infinite *repetition* of process term p.

To support the hierarchical modeling of systems, it is convenient to allow local declarations of variables, channels and recursion definitions. For this purpose, the *process instantiation* process term $l_{\mathrm{p}}(\mathbf{x}_k, \mathbf{h}_m, \mathbf{e}_n)$ is introduced, which enables (re)-use of a process definition. A process definition is specified once, but it can be instantiated many times, possibly with different parameters: external variables $\mathbf{x}_k$, external channels $\mathbf{h}_m$, and expressions $\mathbf{e}_n$. Chi specifications in which process instantiations $l_{\mathrm{p}}(\mathbf{x}_k, \mathbf{h}_m, \mathbf{e}_n)$ are used have the following structure:

$$
\begin{array}{l}
pd_1 \\
\vdots \\
pd_j \\
\\
\langle\ \mathsf{disc}\ \ldots, \mathsf{cont}\ \ldots, \mathsf{alg}\ \ldots, \mathsf{chan}\ \ldots, i, L_R :: q\ \rangle,
\end{array}
$$

where for each process instantiation $l_{\mathrm{p}}(\mathbf{x}_k, \mathbf{h}_m, \mathbf{e}_n)$ occurring in process term q, a matching process definition pd_i of the form

$$l_{\mathrm{p}}(\mathsf{ext}\ \mathbf{x'}_k, \mathsf{chan}\ \mathbf{h'}_m, \mathsf{val}\ \mathbf{v}_n) = p$$

must be present among the process definitions $pd_1 \ldots pd_j$. Such a process instantiation behaves as its defining process term p, where the 'formal external' variables $x'_1, \ldots, x'_k$, the 'formal external' channels $h'_1, \ldots, h'_m$, and the 'value parameters' $v_1, \ldots, v_n$ are substituted by the 'actual external' variables $x_1, \ldots, x_k$, the 'actual external' channels $h_1, \ldots, h_m$, and the expressions $e_1, \ldots, e_n$, respectively.

2.4 Formal semantics

The semantics of χ is defined by means of deduction rules in the style of Plotkin's SOS that associate a hybrid transition system with a χ process as defined in [32,6]. Such a hybrid transition system has four different kinds of transition relations and predicates. They are called action transition, termination transition, time transition, and consistency predicate. They can be explained as follows:

- Action transition: The intuition of an action transition $M \xrightarrow{\xi, a, \xi'} M'$ is that the χ process M executes the discrete action $a \in \mathcal{A}$ and thereby transforms into the process M'. Here, ξ denotes the values of the variables before executing the discrete action, and ξ' denotes the values of the variables after the discrete action.

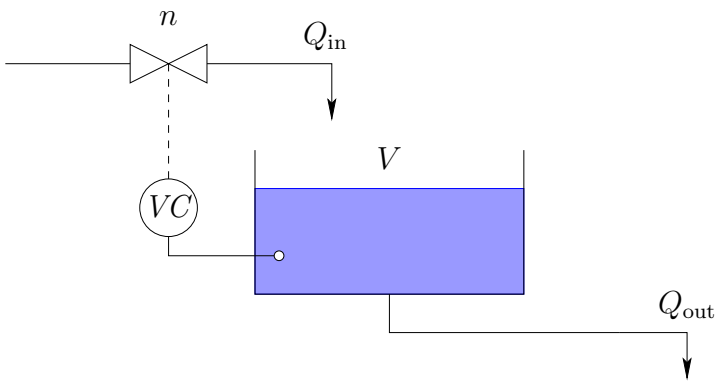


Fig. 1. A liquid storage tank with volume controller.

- Termination transition: The intuition of a (termination) transition $M \xrightarrow{\xi,a,\xi'} \checkmark$ is that the process M executes the discrete action a and thereby transforms into the terminated process $\checkmark$.
- Time transitions: The intuition of a time transition $M \overset{t,\rho}{\longmapsto} M'$ is that during the time transition, the values of the variables at each time-point $s \in [0,t]$ are given by $\rho(s)$. At the end-point t, the resulting process is M'.
- Consistency by means of a predicate: The intuition of a consistency predicate $M \overset{\xi}{\rightsquigarrow}$ is that the process M is consistent with the values of the variables given by ξ.

2.5 Examples

A liquid storage tank with volume controller

Figure 1 shows a liquid storage tank with a volume controller VC. The incoming flow Q_{in} is controlled by means of a valve n. The outgoing flow is given by equation $Q_{\text{out}} = \sqrt{V}$. The volume controller maintains the volume V of the liquid in the tank between 2 and 10.

The χ model of the system is as follows:

$$
\begin{array}{l}
\langle\ \mathsf{disc}\ n, \mathsf{cont}\ V, \mathsf{alg}\ Q_{\text{in}}, Q_{\text{out}} \\
,\ n = 0, V = 10 \\
::\ \dot{V} = Q_{\text{in}} - Q_{\text{out}} \\
,\ Q_{\text{in}} = n \cdot 5 \\
,\ Q_{\text{out}} = \sqrt{V} \\
\|\ *(\ V \leq 2 \rightarrow n := 1;\ V \geq 10 \rightarrow n := 0\ ) \\
\rangle\ .
\end{array}
$$

The volume controller is modeled by means of repetition $*(\dots)$. Initially, the volume in the tank equals 10 and the valve is closed ($n = 0, V = 10$) [7]. When the volume equals 2, the valve is opened ($V \leq 2 \rightarrow n := 1$). Note that the assignment $n := 1$ also changes the value of the algebraic variable Q_{in} to 5 since the equation $Q_{\text{in}} = n \cdot 5$

[7] As is common practice in mathematics, the comma in predicates denotes conjunction. E.g. i_1, i_2 denotes the predicate $i_1 \wedge i_2$.

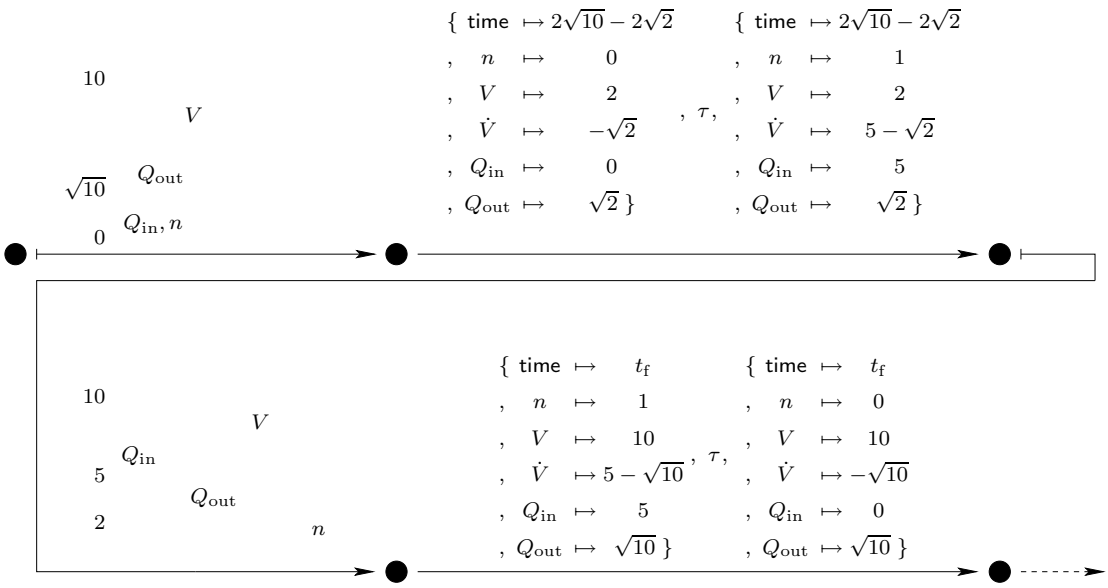


Fig. 2. Hybrid transition system of the Chi model.

should always hold. When the volume in the tank equals 10, the valve is closed again ($V \geq 10 \rightarrow n := 0$).

Figure 2 shows (a part of) the hybrid transition system of the χ model. The circles represent the states, arrows $\rightarrow$ represent action transitions that are labeled with the values of the variables prior to and after the transition as defined in Section 2.4, and arrows $\longmapsto$ represent time transitions. The labels (t, ρ) of the time transitions are represented graphically. Constant t_{f} denotes the value $-5\ln(15) - 5\ln(5 - \sqrt{10}) + 5\ln(5 + \sqrt{10}) + 5\ln(23) + 5\ln(5 - \sqrt{2}) - 5\ln(5 + \sqrt{2})$.

Assembly line

Figure 3 shows the iconic model of an assembly line, which is modeled as a discrete-event system. An assembly process A assembles three different parts that are supplied by three suppliers G. The order in which the parts are supplied is unknown, but each part should be received by the assembly process as soon as possible. When all three parts have been received, assembly may start. Assembly takes t_{A} units of time. When the products have been assembled, they are sent to an exit process E. The χ model consists of parallel instantiations of the three generator processes G, the assembly process A and the exit process E:

$\langle$ **chan** a, b, c, d

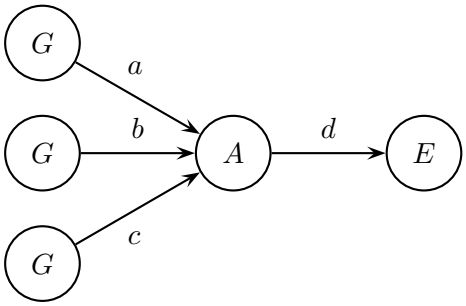


Fig. 3. Iconic model of an assembly line.

$$
\begin{aligned}
&:: G(a, 0, t_0) \parallel G(b, 1, t_1) \parallel G(c, 2, t_2) \\
&\parallel A(a, b, c, d, t_{\mathrm{A}}) \\
&\parallel E(d) \\
&\rangle,
\end{aligned}
$$

where $t_0, t_1, t_2, t_{\mathrm{A}}$ denote constants.

Each generator sends a part n every t time units:

$$\mathsf{proc}\ G(\mathsf{chan}\ a,\ \mathsf{val}\ n, t) = [\![*(\ a!n;\ \Delta t\ )]\!].$$

Receiving of the parts by the assembly process is modeled by means of the parallel composition $(a\,?\,x \parallel b\,?\,y \parallel c\,?\,z)$. This ensures that each part is received as soon as possible. The parallel composition terminates when all parts have been received.

$$
\begin{aligned}
&\mathsf{proc}\ A(\mathsf{chan}\ a, b, c, d,\ \mathsf{val}\ t) = \\
&[\![\ \mathsf{disc}\ x, y, z \\
&:: *(\ (\ a\,?\,x \parallel b\,?\,y \parallel c\,?\,z\ )\ ;\ \Delta t;\ d\,!(x, y, z)\ ) \\
&]\!].
\end{aligned}
$$

The exit process is simply:

$$\mathsf{proc}\ E(\mathsf{chan}\ a) = [\![\ \mathsf{disc}\ x :: *\ a\,?\,x\ ]\!].$$

A process algebra, such as χ, permits concise and elegant specifications. Compare, for example, the automaton specification in Figure 4 with the equivalent χ specification $*((a?x \parallel b?y \parallel c?z);\ \Delta 1;\ d\,!(x, y, z))$.

The automaton needs additional shared variables to mimic the communication behavior of χ. E.g. $h!1 \parallel h?x$ is translated to the transitions $h_1, x : h_1 = 1$ with label $h!$, and $h_1, x : h_1 = x$ with label $h?$, assuming that the labels $h!$ and $h?$ synchronize. Here, the variables before the colon are allowed to change, the other variables remain unchanged.

A restriction of an automaton specification is that it requires explicit specification of the locations. In process algebra, the 'locations' can be explicitly defined by means of recursion variables, but they can also be implicitly defined, as in the assembly model. Note that in hybrid automata, implicit locations can be specified using variables, i.e. state counters.

Another restriction of an automaton specification is that parallelism is generally allowed only at the top level, whereas in process algebra, parallelism is allowed at any level.

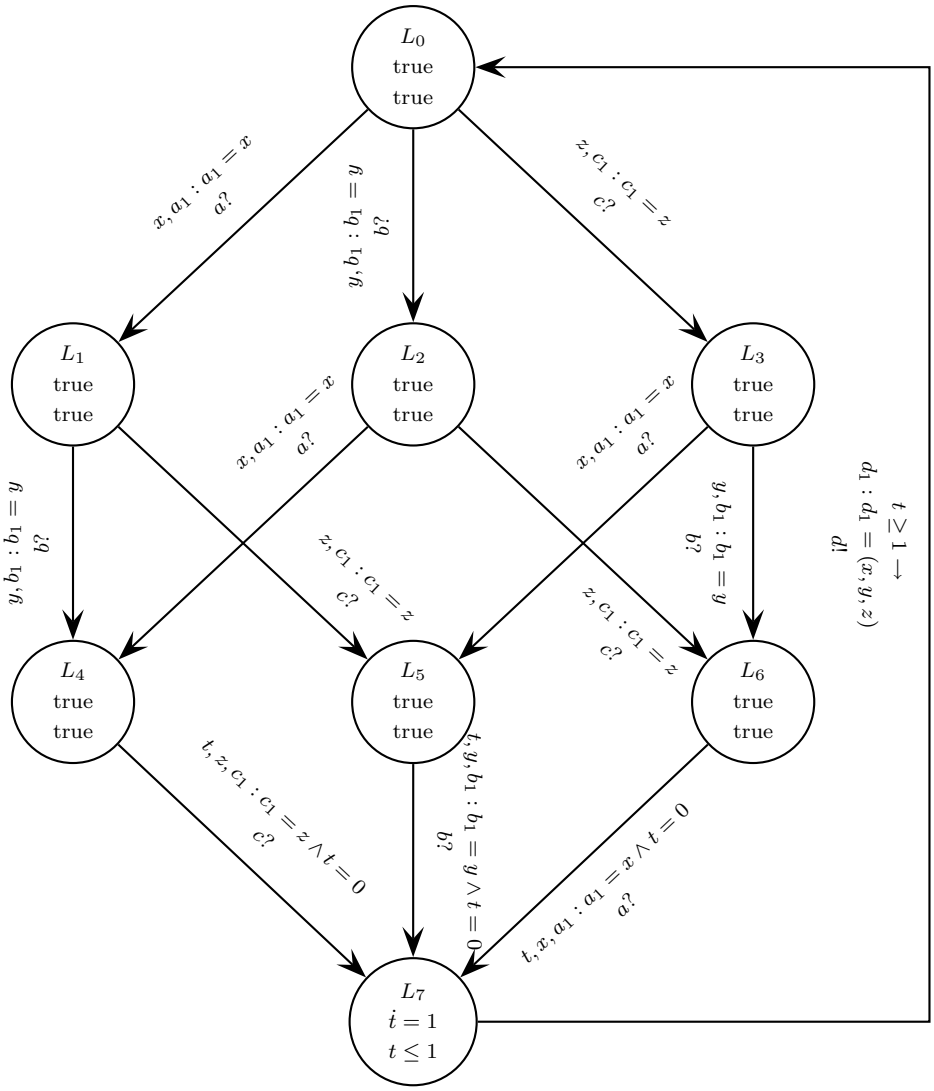


Fig. 4. Automaton specification of $*((a?x \parallel b?y \parallel c?z); \Delta 1; d!(x, y, z))$.

3 Translations of piecewise affine systems to Chi

In this section, two general translation schemes are given. One scheme defines the translation of continuous-time piecewise affine systems to a χ specification. The other scheme defines a translation of discrete-time piecewise affine systems to χ.

3.1 Continuous-time PWA

Continuous-time piecewise affine systems are described by N sets of affine differential equations:

$$\left.\begin{aligned} \dot{x}(t) &= A_i x(t) + B_i u(t) + f_i \\ y(t) &= C_i x(t) + D_i u(t) + g_i \end{aligned}\right\} \quad \text{if} \quad \Omega_i \ ,$$

where i $(i = 1, \ldots, N)$ denotes the number of the mode, $u(t) \in \mathbb{R}^m$, $x(t) \in \mathbb{R}^n$, and $y(t) \in \mathbb{R}^l$ denote the input, state and output, respectively, at time t. Furthermore,

f_i, and g_i denote constants. Each set of equations describes the dynamical behavior in a mode. In each mode, the trajectories of the state variables x are continuous functions of time. The trajectories of the input/output variables in a mode may be discontinuous functions of time. Each mode i is defined in a region represented by a predicate Ω_i given by a finite number of linear inequalities, in the input/state space.

A PWA system is well-posed if the regions do not overlap, and all regions together span $\mathbb{R}^{n+m}$. Given an initial state x_0 at t_0 and an input function u, the system evolves as follows. Let i_0 be the active mode at t_0, that is Ω_{i_0} holds for $x(t_0)$ and $u(t_0)$. For $t \in [t_0, t_1]$, where $t_1 > t_0$ and t_1 denotes the largest instance such that Ω_{i_0} still holds, the state and the output evolve according to the solution of the equations $\dot{x}(t) = A_{i_0}x(t) + B_{i_0}u(t) + f_{i_0}$ and $y(t) = C_{i_0}x(t) + D_{i_0}u(t) + g_{i_0}$, respectively. At time t_1, a (deterministic) mode switch occurs to a new active mode i_1 such that Ω_{i_1} holds. Note that the trajectory for the state variables x is continuous, since mode switches may only introduce discontinuities in the state derivatives and the output variables.

A well-posed continuous-time PWA system can be translated to the following χ specification:

$$
\begin{aligned}
&\langle\ \mathsf{cont}\ x,\ \mathsf{alg}\ y \\
&,\ x = x_0 \\
&::\quad (\Omega_1 \wedge \dot{x} = A_1x + B_1u + f_1 \wedge y = C_1x + D_1u + g_1) \\
&\quad \vee \\
&\quad \vdots \\
&\quad \vee\ (\Omega_N \wedge \dot{x} = A_Nx + B_Nu + f_N \wedge y = C_Nx + D_Nu + g_N) \\
&\rangle.
\end{aligned}
$$

The state variables x are modeled in χ by means of (non-jumping) continuous variables, with initial value x_0. The output variables y are modeled by means of algebraic variables. The behavior of u is not specified, as in the original PWA model. In the χ model, u could denote a function of time, or u could be defined as an algebraic variable, and additional equations specifying the behavior of u could be added. The behavior associated to a mode i is described by means of a delay predicate $(\Omega_i \wedge \dot{x} = A_ix + B_iu + f_i \wedge y = C_ix + D_iu + g_i)$. Since we consider well-posed PWA systems, exactly one of the $\Omega_1, \ldots, \Omega_N$ predicates evaluates true. The logical 'or' ($\vee$) composition of the behavior of the modes then allows only the behavior of one mode at a time, i.e. only that mode for which the corresponding conditions Ω_i hold. The system will evolve according to the Caratheodory solutions of the dynamics associated with the active mode. In case of a mode switch, the state variables x cannot jump (their value remains constant during the mode switch). At a mode switch, the (algebraic) output variables y are allowed to jump such that the dynamics of the resulting mode hold.

The behavior of continuous-time PWA systems is defined in terms of solutions (trajectories) $x(t)$ and $y(t)$ according to a solution concept. Commonly used solution concepts are the Caratheodory solution concept, which allows non-smooth, but

continuous solution functions for $x(t)$, and solution concepts based on differential inclusions, such as the Filippov [17] solution concept.

If the continuous-time PWA model as defined above has a solution (trajectory) $\rho(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$, according to the Caratheodory solution concept, then the χ model that results from translating this continuous-time PWA model using the translation as defined above can perform a time transition labeled with $\rho'(t)$ which has the same trajectories for x and y, but also includes trajectories for $\dot{x}$: $\rho'(t) = \begin{bmatrix} x(t) \\ y(t) \\ \dot{x}(t) \end{bmatrix}$, and vice versa. If we abstract from the trajectories for $\dot{x}$, then the delay behavior of the continuous-time PWA system and its χ model translation are the same. Since a continuous-time PWA system and its χ model translation cannot perform any actions, we know that the translation is correct.

Continuous-time PWA models with a solution concept based on differential inclusions, that allows sliding modes, can in principle also be translated to χ models using the convex equality operator as defined in [7].

3.2 Discrete-time PWA

Discrete-time PWA systems are described by

$$\left.\begin{aligned} x(k+1) &= A_i x(k) + B_i u(k) + f_i \\ y(k) &= C_i x(k) + D_i u(k) + g_i \end{aligned}\right\} \quad \text{if} \quad \Omega_i,$$

where i ($i = 1, \ldots, N$). Here, $u(k) \in \mathbb{R}^m$, $x(k) \in \mathbb{R}^n$, and $y(k) \in \mathbb{R}^l$ denote the input, state and output, respectively, at the k^{th} time-point.

A well-posed discrete-time PWA system can be translated to the following χ specification:

$\langle$ disc x, y, k
, $x = x_0, k = 0$
:: $*($ $\Omega_1 \rightarrow x, y, k := A_1 x + B_1 u.k + f_1, C_1 x + D_1 u.k + g_1, k + 1$
[]
$\vdots$
[] $\Omega_N \rightarrow x, y, k := A_N x + B_N u.k + f_N, C_N x + D_N u.k + g_N, k + 1$
$)$
$\rangle$.

The state variables x are modeled in χ by means of discrete variables, and are initialized to x_0. The output variables y and variable k are also modeled by means of discrete variables. We assume u to denote an array of points, such that $u.i$

denotes the value of u at the i^{th} time-point. In the repetition $*(\quad)$, the state and output variables are assigned new values according to exactly one of the modes, and k is increased by one. The behavior associated to a mode is described by means of a multi-assignment $x, y, k := A_i x + B_i u.k + f_i, C_i x + D_i u.k + g_i, k + 1$. The alternative composition of the behavior of the modes allows the state and output variables to be assigned new values according to the mode for which the corresponding guard (Ω_i) holds. For the translation of well-posed discrete-time PWA systems, always exactly one of the guards Ω_i holds. Since we have the properties true $\rightarrow p \underline{\leftrightarrow} p$, false $\rightarrow p \underline{\leftrightarrow}$ true, and true $[]\ p \underline{\leftrightarrow} p$, the behavior of the χ model is a sequence of (multi-) assignments, where in each mode the corresponding (multi-) assignment is executed.

The behavior of discrete-time PWA systems is defined in terms of a sequence of values $\begin{bmatrix} x(k+1) \\ y(k) \end{bmatrix}, \begin{bmatrix} x(k+2) \\ y(k+1) \end{bmatrix}, \ldots$. The action transitions of its χ model translation are directly related to the sequence of values defined by the discrete-time PWA model. In particular, the resulting values of the variables x and y on the k^{th} transition of the χ model correspond to the values $x(k+1)$ and $y(k)$, respectively, of the discrete-time PWA model. Here, the first transition of the χ model has index $k = 0$. Since a discrete-time PWA system does not have any delay behavior, nor does its χ model translation, we know that the translation is correct.

Example: Integrator

An integrator with upper saturation can be modeled as a discrete-time PWA model as follows:

$$x(k+1) = \begin{cases} x(k) + u(k) & \text{if } x(k) + u(k) \leq 1 \\ 1 & \text{if } x(k) + u(k) \geq 1 \end{cases}$$
$$y(k) = x(k).$$

This model can be translated to χ as follows:

$$\begin{aligned} &\langle\ \mathsf{disc}\ x, y, k \\ &,\ x = x_0, k = 0 \\ &::\ *(\ x + u.k \leq 1 \rightarrow x, y, k := x + u.k, x, k + 1 \\ &\quad []\ x + u.k \geq 1 \rightarrow x, y, k := 1, x, k + 1 \\ &\quad)\ \rangle\ . \end{aligned}$$

4 Relating hybrid automata and χ

One of the most influential formalisms for hybrid system specification and analysis is the theory of hybrid automata ([1,24]). In Section 4.1, the hybrid automaton model of [24] is translated to the χ formalism. This translation from hybrid automata to χ aims to show that the χ formalism is at least as expressive as the theory of hybrid automata. Note that in [5], a translation between χ and hybrid automata has been

defined. However, that version of χ differs considerably with the χ version used in this paper. For instance, the choice, reinitialization, and disrupt operators do not exist in the current χ language anymore, and the semantics of some operators, such as the guard operator has been changed. In Section 4.2, the translation from (a subset of) χ to hybrid automata is described. This translation enables verification of χ specifications using existing hybrid automata based verification tools, which is illustrated by means of a case study.

4.1 Translation of a hybrid automaton to Chi

A hybrid automaton [24] consists of the following components:

- A finite set of (real-valued) variables $X = \{x_1, \ldots, x_n\}$.
- A finite directed multi-graph (V, E), where V denotes a set of vertices (control modes) and E denotes a set of edges (control switches).
- Three vertex labeling functions init, inv, and flow that assign to each control mode $v \in V$ a predicate for initial, invariant 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 $X \cup \dot{X}$. The set $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ denotes the first derivatives of variables X.
- An edge labeling function jump, that assigns to each edge $e \in E$, a jump condition which is a predicate whose free variables are from $X \cup X'$. The set $X' = \{x'_1, \ldots, x'_n\}$ denotes the primed variables that represent values at the conclusion of a discrete change.
- A finite set Σ of events, and an edge labeling function event : $E \to \Sigma$ that assigns to each edge an event.

In order to translate a hybrid automaton to χ, two additional functions are defined on a hybrid automaton: function edges : $V \to \mathcal{P}(E)$ returns a set of outgoing edges for a location, and function target : $E \to V$ returns the target vertex of an edge. Furthermore, the function $\mathcal{T}$ translates a jump predicate to the predicate r of a χ action predicate $(W : r \gg l_a)$ by renaming variables occurring without a prime in a jump predicate to variables with superscript '$^-$', and renaming variables occurring with a prime '$'$' to variables without the prime. E.g. $\mathcal{T}(x' = 2x + y \wedge x \geq 0 \wedge y' = y)$ becomes $x = 2x^- + y^- \wedge x^- \geq 0 \wedge y = y^-$. In the latter expression, x^- and y^- refer to the values of x and y, respectively, before the discrete jump, and x and y refer to the value of variables x and y after the discrete jump. The class of hybrid automata to be translated to χ is restricted to the hybrid automata without initial time non-determinism. In this section, we consider hybrid automata where the initial condition of all but one control mode equals false. The one control mode with the initial condition not equal to false is called the *initial* control mode.

Furthermore, it should be possible to rewrite each flow predicate into one of the following forms: $\dot{\mathbf{x}} = f(\mathbf{x})$, $\dot{\mathbf{x}} \in f(\mathbf{x})$ or the predicate true. This means that we do not consider flow predicates such as false, or (equivalently) $\dot{x} = 0 \wedge \dot{x} = 1$. This ensures that the solutions (trajectories of the model variables) in hybrid automata and χ are the same.

Consider a hybrid automaton model which belongs to the class of automata as defined in the previous section, with n variables ($X = \{x_1, \ldots, x_n\}$), k control modes ($V = \{v_1, \ldots, v_k\}$), and one initial control mode v_1. The translation to a corresponding χ specification is defined as follows:

$$
\begin{aligned}
&\langle\ \mathsf{cont}\ x_1, \ldots, x_n \\
&,\ \mathrm{init}(v_1) \\
&,\ v_1 \mapsto \mathrm{flow}(v_1) \wedge \mathrm{inv}(v_1)\ [\!]\ \\
&\qquad\qquad ([\!]_{\,e:e\in\mathrm{edges}(v_1)}[X : \mathcal{T}(\mathrm{jump}(e)) \gg \mathrm{event}(e)];\ \mathrm{target}(e)) \\
&\ \ \vdots \\
&,\ v_k \mapsto \mathrm{flow}(v_k) \wedge \mathrm{inv}(v_k)\ [\!]\ \\
&\qquad\qquad ([\!]_{\,e:e\in\mathrm{edges}(v_k)}[X : \mathcal{T}(\mathrm{jump}(e)) \gg \mathrm{event}(e)];\ \mathrm{target}(e)) \\
&::\ v_1 \\
&\rangle\ .
\end{aligned}
$$

The variables $x_1, \ldots, x_n$ are declared as continuous variables. These variables are initialized by means of initialization predicate $\mathrm{init}(v_1)$.

A vertex v_i of the hybrid automaton model is translated using a corresponding recursion variable v_i in the χ model. The process term associated with this recursion variable consists of the alternative composition of the process term describing the continuous behavior of the vertex, and the alternative compositions of all individual process terms of the outgoing edges of this vertex. Below, these process terms are explained in more detail.

The continuous behavior of a vertex v_i is translated to a delay predicate in χ, consisting of the conjunction of the flow predicate and the invariant of the vertex. For each outgoing edge, the jump predicate of that edge is translated to an action predicate labeled with the event label of the edge ($X : \mathcal{T}(\mathrm{jump}(e)) \gg \mathrm{event}(e)$). Since all variables are allowed to jump, the set W of the action predicate equals the set X. The semantics of a hybrid automaton is such that when a guard of an edge is enabled, the transition via this edge can be taken, but it is not required to take this transition. Therefore, the action predicate associated with the edge is made delayable using the any delay operator $[\,]$. After the transition, the behavior is specified by the recursion variable associated with the target vertex ($\mathrm{target}(e)$).

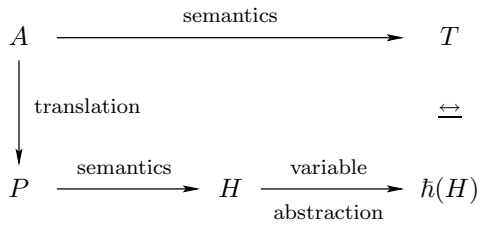
Note that for set $E = \{e_1, \ldots, e_k\}$, notation $[\!]_{\,e:e\in E}[X : \mathcal{T}(\mathrm{jump}(e)) \gg \mathrm{event}(e)];\ \mathrm{target}(e)$, denotes the process term $[X : \mathcal{T}(\mathrm{jump}(e_1)) \gg \mathrm{event}(e_1)];\ \mathrm{target}(e_1)[\!]\ \ldots\ [\!]\ [X : \mathcal{T}(\mathrm{jump}(e_k)) \gg \mathrm{event}(e_k)];\ \mathrm{target}(e_k)$.

This straightforward translation of a hybrid automaton to a χ model shows that χ is expressive enough to model phenomena that are usually studied by means of a hybrid automaton.

The semantics of a hybrid automaton [24] is a timed transition system with two types of transitions: action transitions (corresponding to control switches) and time transitions (corresponding to continuous behavior in a control mode). On the other hand, the semantics of a χ process is a hybrid transition system (see Section 2.4) which also has these two types of transitions.

The main difference between these semantics is in the labeling of the action and time transitions. In timed transition systems the labels of action transitions are simply the events of the hybrid automaton, whereas the labels of the action transitions of a hybrid transition system also contain the valuations of the model variables prior to and after the action. For time transitions, the labels in a timed transition system contain only the duration of the time transition whereas time transitions in hybrid transition systems also have the trajectory of the model variables as a label. Finally, a timed transition system can have many initial states whereas a hybrid transition system has only one initial state. This one initial state captures the behavior of all the initial states of the timed transition system.

Let $\hbar$ be a mapping that maps a hybrid transition system onto a timed transition system by removing valuations from action transitions and trajectories from time transitions. Furthermore, let HA be a hybrid automaton and let M_χ be the χ specification associated to it by the translation defined in this section. Furthermore, let TTS and HTS be the semantics of HA and M_χ, respectively.



Then, there exists a (strong-)bisimulation relation [33,37], denoted by $\underline{\leftrightarrow}$, between the states of TTS and the states of $\hbar(HTS)$ such that any transition from an initial state of TTS can be simulated by the initial state of $\hbar(HTS)$ and each transition from the initial state of $\hbar(HTS)$ is simulated by some initial state of TTS.

The following example shows the translation of a hybrid automaton model of a thermostat to χ. The hybrid automaton is shown in Figure 5. Variable x represents the temperature. The control modes are *On* and *Off*. Initially, the temperature equals 20 degrees, and the heater is off (control mode *Off*). The temperature falls according to the flow condition $\dot{x} = -0.1x$. According to the jump condition $x < 19$, the heater may go on as soon as the temperature falls below 19 degrees. The invariant condition $x \geq 18$ ensures that at the latest the heater will go on when the temperature equals 18 degrees. In the control mode *On*, the heater is on, and the temperature rises according to the flow condition $\dot{x} = 5 - 0.1x$. When the temperature rises above 21 degrees, the heater may turn off. Due to the invariant condition $x \leq 22$, at the latest the heater will turn off when the temperature equals

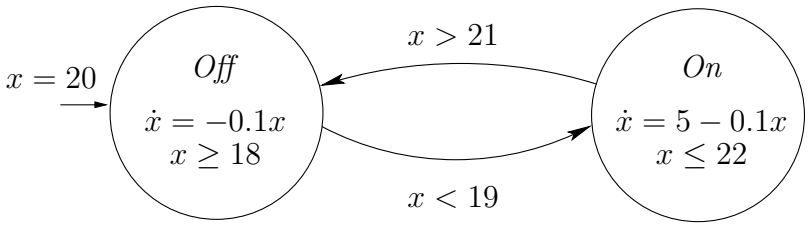


Fig. 5. A hybrid automaton model of a thermostat.

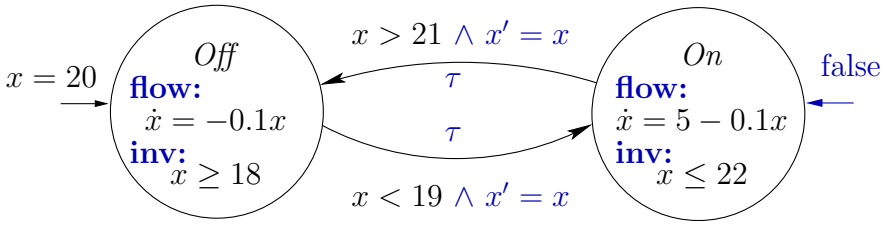


Fig. 6. Complete hybrid automaton model of a thermostat.

22 degrees.

Figure 5 is taken from [24], where the usual informal notation is used: the flow and invariant conditions are not labelled explicitly (any invariant condition can also be considered as a flow condition, possibly resulting in different behavior), events on the edges are ignored, and the initial and jump conditions are incomplete. In particular, in Figure 5 both edges should have an event label, the initial condition of mode *On* equals false, and the jump conditions of the edges should have been $x < 19 \wedge x' = x$ and $x > 21 \wedge x' = x$, respectively, expressing that the value of x (the temperature) is only tested and not adapted. The complete, formal hybrid automata model of the thermostat is shown in Figure 6. Using the translation scheme, this model is translated to χ, which results in the following χ specification:

$$
\begin{aligned}
&\langle\ \mathsf{cont}\ x \\
&,\ x = 20 \\
&,\ \mathit{Off}\ \mapsto\ \dot{x} = -0.1x \wedge x \geq 18\ [\!]\ [\{x\} : x < 19 \wedge x = x^- \gg \tau];\ \mathit{On} \\
&,\ \mathit{On}\ \mapsto\ \dot{x} = 5 - 0.1x \wedge x \leq 22\ [\!]\ [\{x\} : x > 21 \wedge x = x^- \gg \tau];\ \mathit{Off} \\
&::\ \mathit{Off} \\
&\rangle\ .
\end{aligned}
$$

4.2 Translation of χ 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. [24,3]. Other definitions allow the more general case of piecewise differential functions, e.g. [44]. Most hybrid automata definitions do not define urgent transitions, or they define urgent transitions in a restrictive way, as in [25]. In [35], 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 [24] the variables can in principle perform arbitrary jumps unless restricted by the jump predicate, in [25], 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 χ. Therefore, the translation uses a target hybrid automata definition, called HA_{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 [3], the solution concept that allows piecewise differentiable functions is based on [44], and the definition of urgent transitions was inspired by [35]. In hybrid statecharts [28], urgent transitions are defined in a similar way.

The subset χ_{sub} of the χ language that is translated consists of guarded action predicate $b \rightarrow W : r \gg l_{\text{a}}$, guarded send $b \rightarrow h\,!!\,\mathbf{e}_n$, and guarded receive $b \rightarrow h\,??\,\mathbf{x}_n$, the delay predicate u, the unary operators any delay $[\,]$, repetition $*$, and the binary operators sequential composition ;, alternative composition $[]$, and parallel composition $\|$. Process terms skip, and the multi-assignment $\mathbf{x}_n := \mathbf{e}_n$ can be translated by means of expressing them in terms of the more general action predicate. The delay process term Δd cannot be translated. In general, recursion variables cannot be translated, however, it is not difficult to translate a more restricted form of the use of recursion variables, such as guarded recursion. Since the HA_{u} definition has no hierarchy and no distinction between local and global variables, it is not possible to translate process instantiations. However, if the HA_{u} definition would be augmented with hierarchy, we do not expect fundamental problems. Furthermore, in χ_{sub} processes, there are no discrete variables, no algebraic variables, and no recursion variables.

In χ, 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 the action predicate, send and receive process terms.

In [32], we define a formal translation from χ_{sub} to HA_{u} automata. It is proved that any transition of a χ model can be mimicked by a transition in the corresponding hybrid automaton model and vice versa. This indicates that the translation is correct. Since a manual translation is very time consuming and error-prone, the translation has been automated by implementing it using the programming language Python [40].

PHAVer [19] is a tool for analyzing linear hybrid I/O-automata. If we restrict the linear hybrid I/O-automata 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_{u} automata as defined previously. As a consequence, the χ_{sub} specifications that can be verified using PHAVer are restricted to those specifications that result in HA_{u} automata with linear invariant and flow conditions, and linear jump conditions, where a linear condition is defined as a condition over a set of variables X that is of the form $\sum_i a_i v_i + b \bowtie 0$, with $a_i, b \in \mathbb{Z}$, $v_i \in X$, and $\bowtie \in \{<, \leq, =\}$. Furthermore, the χ_{sub} specifications are restricted to those specifications that result in HA_{u} automata which do not contain urgent transitions. This restriction is because in the semantics of the HA_{u} definition, transitions can be urgent, while in the linear hybrid I/O-automata, they cannot. Note that in [32], the relation between linear hybrid I/O-automata and HA_{u} automata has been formalized.

The verification of a χ_{sub} specification using PHAVer is illustrated by means of an example: the water level monitor, which is taken from [1]. First, the water level monitor is modeled using χ_{sub}. Then we translate the χ_{sub} specification to a hybrid automaton HA_{u}. Since the obtained hybrid automaton HA_{u} is a linear hybrid I/O-automaton, it is possible to verify properties of this automaton model

using PHAVer.

The water level in a tank, denoted by the variable y, 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 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 seconds between the time point that the monitor signals to change the status of the pump and time point that this 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 χ_{sub} as follows:

$$
\begin{aligned}
&\langle\ \mathsf{cont}\ x, y \\
&,\ x = 0, y = 1 \\
&::\ \dot{x} = 1 \\
&\|\ *(\ (\ \dot{y} = 1 \quad \wedge y \leq 10\ []\ [y \geq 10 \rightarrow x := 0]\ ) \\
&\quad ;\ (\ \dot{y} = 1 \quad \wedge x \leq 2\ \ []\ [x \geq 2 \rightarrow \text{skip}]\ ) \\
&\quad ;\ (\ \dot{y} = -2 \wedge y \geq 5\ \ []\ [y \leq 5 \rightarrow x := 0]\ ) \\
&\quad ;\ (\ \dot{y} = -2 \wedge x \leq 2\ \ []\ [x \geq 2 \rightarrow \text{skip}]\ ) \\
&\quad)\ \\
&\rangle.
\end{aligned}
$$

This specification is translated into a hybrid automaton HA_{u}, which is shown in Figure 7. This automaton is similar to the automaton from [1]. The automata model differs in the sense that the automaton obtained by translation contains additional variables (dx, and dy) and restrictions on them in the invariants of the locations, as a result of the translation. These additional variables are introduced because in χ, 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 χ_{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, the dotted variables $\dot{a}$ are replaced by variables da. However, these additional variables and restrictions on them do not affect the behavior w.r.t the original automaton from [1].

The input language of PHAVer is a straightforward textual representation of linear hybrid I/O-automata [18]. Using a code generator, the input code for PHAVer is automatically generated from the linear hybrid I/O-automaton model.

The safety property that the water level y has to be kept between 1 and 12 has been verified using PHAVer. PHAVer reported that this safety property holds in all locations. In [32], we have a theorem that states that this safety property then also holds in the hybrid automaton HA_{u}. Since we proved that any transition of a χ_{sub} specification can be mimicked by a transition in the corresponding hybrid automaton HA_{u} and vice versa, it can be concluded that this safety property also holds in the original χ_{sub} specification.

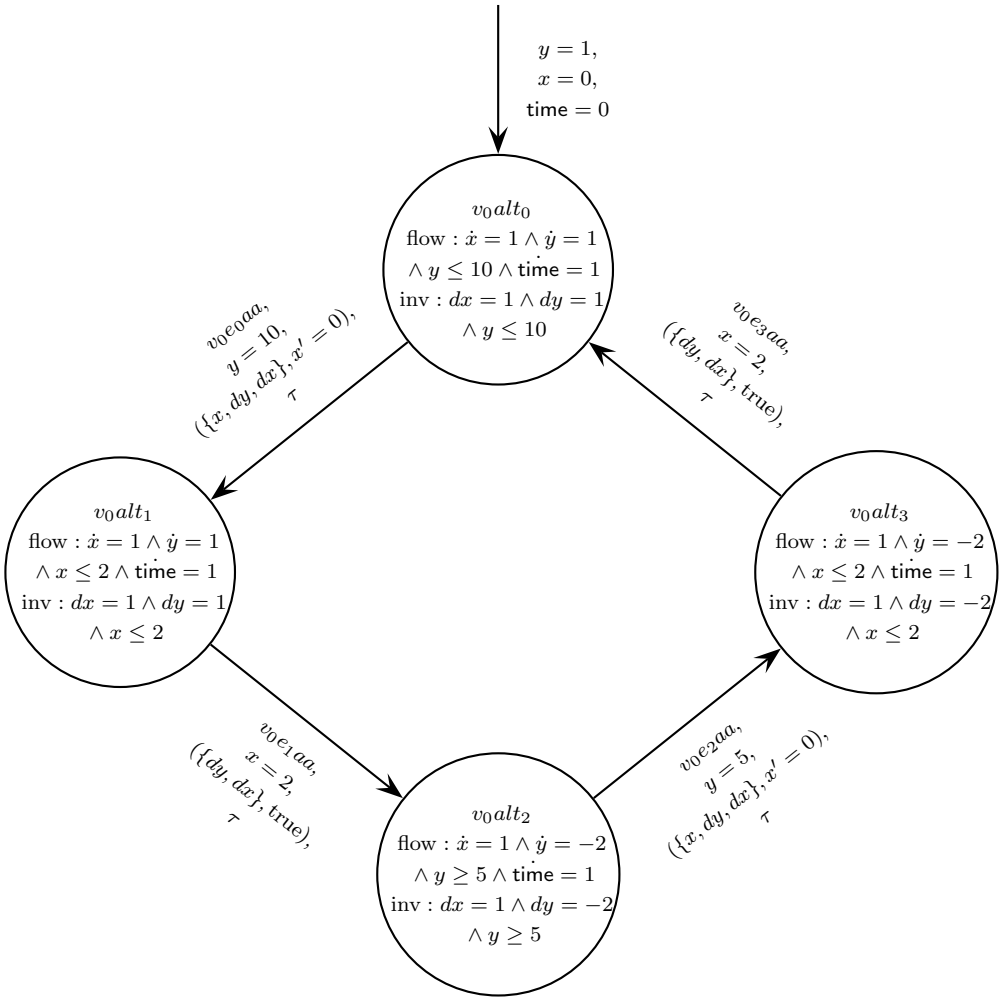


Fig. 7. Generated water level monitor automaton.

5 Concluding remarks

This paper relates the χ formalism to piecewise affine systems and to hybrid automata by means of (formal) translation schemes. The translations from piecewise affine systems and hybrid automata to χ have shown that both kind of models can be easily expressed in a straightforward manner, using a subset of χ. This shows that χ is at least as expressive as the combination of both formalisms. The χ formalism has several additional concepts that make life easier for the modeler:

- The sequential composition operator. Hybrid automata (and piecewise affine systems) do not have a sequential composition operator. To specify a simple sequential composition (the use of which is wide-spread in programs) in a hybrid automaton, for each element of the sequential composition a location needs to be defined, or other add-hoc encodings or transformations (attempting to merge the sequential statements or using a statement counter) are required.
- The possibility to freely combine the χ primitives and operators. This is different from hybrid automata (and piecewise affine systems), that have a quite

restricted syntax. Parallel composition is in general allowed only at the top level in hybrid automata. This restriction often leads to more complex hybrid automaton specifications than functionally equivalent χ specifications of the same system. Compare for example Figure 3 with the equivalent χ specification of the manufacturing system.

- Differential algebraic equations can be defined as in mathematics. There is no need to distinguish a flow clause and an invariant as in hybrid automata.
- Different interaction mechanisms: handshake synchronization and synchronous communication that allow interaction between processes without sharing variables (which is essential in distributed systems), and shared variables that enable modular composition of continuous-time or hybrid processes.
- Process terms for scoping that integrate abstraction, local variables, local channels and local recursion definitions; process definition and instantiation that enable process re-use, encapsulation, hierarchical and/or modular composition of processes; and three classes of variables: discrete, continuous and algebraic.

The translation of a subset of χ to hybrid automata enables verification of χ specifications using existing hybrid automata based verification tools. In this paper, we translate a χ specification to a hybrid automaton and use the verification tool PHAVer to verify properties.

Future work encompasses extending the subset of χ that can be translated to hybrid automata with the process term Δd and guarded recursion. For the translation of process instantiations, the HA_{u} definition should be augmented with scoping for the declaration of local variables. Currently, the parallel composition of processes is expanded by the translation. In order to avoid scaling up problems, the parallel composition operator could be kept, resulting in a network of parallel hybrid automata. This will complicate the translation, because of differences in synchronization behavior of the two languages. Where all hybrid automata that share the same event are forced to synchronize, in χ, synchronization between process terms that share communication channels is always on a point to point basis, between exactly two processes. Furthermore, as future work, translations to other model checkers such as HyTech, can be defined to verify properties of χ models.

References

[1] 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** (1995), pp. 3–34.

[2] Alur, R., T. Dang, J. Esposito, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. J. Pappas and O. Sokolsky, *Hierarchical modeling and analysis of embedded systems*, Proceeddings of the IEEE **91** (2003), pp. 11–28.

[3] Alur, R., T. A. Henzinger and P. H. Ho, *Automatic symbolic verification of embedded systems*, IEEE Transactions on Software Engineering **22** (1996), pp. 181–201.

[4] Beek, D. A. v., A. v. d. Ham and J. E. Rooda, *Modelling and control of process industry batch production systems*, in: E. Camacho, L. Basanez and J. de la Puente, editors, *15th Triennial World Congress of the International Federation of Automatic Control*, Barcelona, 2002, CD-ROM.

[5] Beek, D. A. v., N. G. Jansen, K. L. Man, M. A. Reniers, J. E. Rooda and R. R. H. Schiffelers, *Relating Chi to hybrid automata*, in: S. Chick, P. J. Sánchez, D. Ferrin and D. J. Morrice, editors, *Proceedings of the 2003 Winter Simulation Conference*, New Orleans, United States, 2003, pp. 632–640.

[6] Beek, D. A. v., K. L. Man, M. A. Reniers, J. E. Rooda and R. R. H. Schiffelers, *Syntax and consistent equation semantics of hybrid Chi*, Journal of Logic and Algebraic Programming **68** (2006), pp. 129–210.

[7] Beek, D. A. v., A. Pogromsky, H. Nijmeijer and J. E. Rooda, *Convex equations and differential inclusions in hybrid systems*, in: *43rd IEEE Conference on Decision and Control* (2004), pp. 1424–1429.

[8] Benveniste, A., P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic and R. de Simone, *The synchronous languages 12 years later*, Proceeddings of the IEEE **91** (2003), pp. 64–83.

[9] Bergstra, J. A. and C. A. Middelburg, *Process algebra for hybrid systems*, Theoretical Computer Science **335** (2005), pp. 215–280.

[10] Bergstra, J. A. and C. A. Middelburg, *Continuity controlled hybrid automata*, Journal of Logic and Algebraic Programming **68** (2006), pp. 5–53.

[11] Bortnik, E. M., D. A. v. Beek, J. M. v. d. Mortel-Fronczak and J. E. Rooda, *Verification of timed Chi models using Uppaal*, in: J. Filipe, J. Cetto and J.-L. Ferrier, editors, *ICINCO 2005, Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics, Barcelona, Spain, September 14-17, 2005, 4 Volumes / CD* (2005), pp. 486–492.

[12] Bortnik, E. M., N. Trčka, A. J. Wijs, B. Luttik, J. M. v. d. Mortel-Fronczak, J. C. M. Baeten, W. J. Fokkink and J. E. Rooda, *Analyzing a Chi model of a turntable system using Spin, CADP and Uppaal*, Journal of Logic and Algebraic Programming **65** (2005), pp. 51–104.

[13] Chaochen, Z., W. Ji and A. P. Ravn, *A formal description of hybrid systems*, in: R. Alur, T. A. Henzinger and E. D. Sonntag, editors, *Hybrid Systems III - Verification and Control*, Lecture Notes in Computer Science **1066** (1996), pp. 511–530.

[14] Cuijpers, P. J. L., "Hybrid Process Algebra," Ph.D. thesis, Eindhoven University of Technology (2004).

[15] Cuijpers, P. J. L. and M. A. Reniers, *Hybrid process algebra*, Journal of Logic and Algebraic Programming **62** (2005), pp. 191–245.

[16] Fernandez, J.-C., H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu and M. Sighireanu, *CADP: a protocol validation and verification toolbox*, in: Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, Lecture Notes in Computer Science **1102** (1996), pp. 437–440.

[17] Filippov, A. F., "Differential Equations with Discontinuous Right Hand Sides," Kluwer Academic Publishers, Dordrecht, 1988.

[18] Frehse, G., "Language Overview v.0.2.2.1 for PHAVer v.0.2.2," www.cs.ru.nl/~goranf (2004).

[19] Frehse, G., *PHAVer: Algorithmic verification of hybrid systems past HyTech*, in: M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop*, Lecture Notes in Computer Science **3414**, Springer-Verlag, 2005 pp. 258–273.

[20] Groote, J. F. and M. A. Reniers, *Algebraic process verification*, in: J. A. Bergstra, A. Ponse and S. A. Smolka, editors, *Handbook of Process Algebra*, Elsevier, 2001 pp. 1151–1208.

[21] He, J., *From CSP to hybrid systems*, in: A. W. Roscoe, editor, *A Classical Mind, Essays in Honour of C.A.R. Hoare*, Prentice Hall, 1994 pp. 171–189.

[22] Heemels, W. P. M. H., B. D. Schutter and A. Bemporad, *Equivalence of hybrid dynamical models*, Automatica **37** (2001), pp. 1085–1091.

[23] Henzinger, T. A., *Masaccio: A formal model for embedded components*, in: J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses and T. Ito, editors, *Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, International Conference IFIP TCS 2000, Sendai, Japan, August 17-19, 2000, Proceedings*, Lecture Notes in Computer Science **1872**, Springer-Verlag, 2000 pp. 549–563.

[24] Henzinger, T. A., *The theory of hybrid automata*, in: M. Inan and R. Kurshan, editors, *Verification of Digital and Hybrid Systems*, NATO ASI Series F: Computer and Systems Science **170**, Springer-Verlag, New York, 2000 pp. 265–292.

[25] Henzinger, T. A., P.-H. Ho and H. Wong-Toi, *A user guide to* HyTech, in: E. Brinksma, R. Cleaveland, K. G. Larsen, T. Margaria and B. Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, May 19-20, 1995, Proceedings*, Lecture Notes in Computer Science **1019** (1995), pp. 41–71.

[26] Henzinger, T. A., P.-H. Ho and H. Wong-Toi, *Algorithmic analysis of nonlinear hybrid systems*, IEEE Transactions on Automatic Control **43** (1998), pp. 540–554.

[27] Holzmann, G. J., "The SPIN Model Checker: Primer and Reference Manual," Addison Wesley Professional, Boston, 2003.

[28] Kesten, Y. and A. Pnueli, *Timed and hybrid statecharts and their textual representation*, in: J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems 2nd International Symposium*, Lecture Notes in Computer Science **571** (1992), pp. 591–620.

[29] Larsen, K. G., P. Pettersson and W. Yi, Uppaal *in a Nutshell*, International Journal on Software Tools for Technology Transfer **1** (1997), pp. 134–152.

[30] Lygeros, J., K. H. Johansson, S. Simic, J. Zhang and S. Sastry, *Dynamical properties of hybrid automata*, IEEE Transactions on Automatic Control **48** (2003), pp. 2–17.

[31] Lynch, N., R. Segala and F. Vaandrager, *Hybrid I/O automata*, Information and Computation **185** (2003), pp. 105–157.

[32] Man, K. L. and R. R. H. Schiffelers, "Formal Specification and Analysis of Hybrid Systems," Ph.D. thesis, Eindhoven University of Technology (2006).

[33] Milner, R., "A Calculus of Communicating Systems," Lecture Notes in Computer Science **92**, Springer-Verlag, 1980.

[34] Mousavi, M. R., M. A. Reniers and J. F. Groote, *Notions of bisimulation and congruence formats for SOS with data*, Information and Computation **200** (2005), pp. 107–147.

[35] Nicollin, X., A. Olivero, J. Sifakis and S. Yovine, *An approach to the description and analysis of hybrid systems*, in: R. L. Grossman, A. Nerode, A. P. Ravn and H. Rischel, editors, *Hybrid Systems*, LNCS **736** (1993), pp. 149–178.

[36] Nicollin, X. and J. Sifakis, *The algebra of timed processes, ATP: Theory and application*, Information and Computation **114** (1994), pp. 131–178.

[37] Park, D. M. R., *Concurrency and automata on infinite sequences*, in: P. Deussen, editor, *Proceedings 5^{th} GI Conference*, Lecture Notes in Computer Science **104** (1981), pp. 167–183.

[38] Plotkin, G. D., *A structural approach to operational semantics*, Technical Report DIAMI FN-19, Computer Science Department, Aarhus University (1981).

[39] Plotkin, G. D., *A structural approach to operational semantics*, Journal of Logic and Algebraic Programming **60-61** (2004), pp. 17–139.

[40] Python, http://www.python.org.

[41] Rönkkö, M. and X. Li, *Linear hybrid action systems*, Nordic Journal of Computing **8** (2001), pp. 159–177.

[42] Rönkkö, M., A. P. Ravn and K. Sere, *Hybrid action systems*, Theoretical Computer Science **290** (2003), pp. 937–973.

[43] Rounds, W. C. and H. Song, *The ϕ-Calculus: A language for distributed control of reconfigurable embedded systems*, in: O. Maler and A. Pnueli, editors, *Hybrid Systems : Computation and Control, 6th International Workshop*, Lecture Notes in Computer Science **2623**, Springer-Verlag, 2003 pp. 435–449.

[44] Schaft, A. J. v. d. and J. M. Schumacher, "An Introduction to Hybrid Dynamical Systems," Lecture Notes in Control and Information Sciences **251**, Springer-Verlag, 2000.

[45] Sontag, E., *Nonlinear regulation: The piecewise linear approach*, IEEE Transactions on Automatic Control **26** (1981), pp. 346–358.

[46] Utkin, V. I., "Sliding Modes in Control Optimization," Springer-Verlag, Berlin, 1992.

[47] Vereijken, J. J., *A process algebra for hybrid systems*, in: Bouajjani and Maler, editors, *The Second European Workshop on Real-Time and Hybrid Systems*, Grenoble, France, 1995.

[48] Wijs, A. J. and W. J. Fokkink, *From χ_t to μCRL: Combining performance and functional analysis*, in: *10th International Conference on Engineering of Complex Computer Systems (ICECCS 2005), 16-20 June 2005, Shanghai, China* (2005), pp. 184–193.