Formal verification of Chi models using PHAVer

Formal Verification of Chi models using PHAVer

K.L. Man, R.R.H. Schiffelers
Eindhoven University of Technology, Eindhoven, The Netherlands
{k.l.man, r.r.h.schiffelers}@tue.nl

Abstract

The hybrid Chi (χ) formalism 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 translation of χ to hybrid automata enables verification of χ specifications using existing hybrid automata based verification tools. In this paper, we describe the translation from χ 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 χ specification to a linear hybrid I/O automaton, and use PHAVer to verify properties.

1 Introduction

Hybrid systems represent a domain where the dynamics and control (DC) and computer science (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 formalism [1] is such a formalism. On the one hand, it can deal with continuous-time systems, piecewise affine (PWA) systems, mixed logic dynamical (MLD) systems, linear complementarity (LC), 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 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. The history of the χ 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 [2], 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

[3]. Later, the hybrid language and simulator were developed [4, 5]. For the purpose of verification, the discrete-event part of the language was mapped onto the process algebra χ_σ by means of a syntactical translation. The semantics of χ_σ was defined using a structured operational semantics style (SOS), bisimulation relations were derived, and a model checker was built [6]. In this way, verification of discrete-event χ models was made possible [7]. The hybrid χ formalism defined in [1] 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. In the remainder of this paper, we usually refer to hybrid χ as χ.

One of the most successful formalisms for hybrid system verification is the theory of hybrid automata. In [1], formal translations between χ and hybrid automata (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 of) χ to hybrid automata enables verification of χ specifications using existing hybrid automata based verification tools.

This paper is organized as follows: Section 2 gives a short introduction of χ. Connections between χ and other formalisms from DC and CS can be found in Section 3. The translation from χ to hybrid automata is described in Section 4. The relation between hybrid automata and the linear hybrid I/O automata that are used for the verification tool PHAVer (Polyhedral Hybrid Automaton Verifyer) is discussed in Section 5. In Section 6, we translate a χ specification of a water-level monitor to a corresponding linear hybrid I/O automaton, and use PHAVer to verify properties. Finally, some concluding remarks are made in Section 7.

2 The hybrid χ language

In this section, we briefly introduce the χ language. The introduction presented here is adapted from [1].

A χ process is a triple $\langle p, \sigma, E\rangle$, where p denotes a process term, σ 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 σ and the environment E, together define the variables that exist in the χ process and the variable classes to which they belong. In χ, there is the predefined variable $\mathsf{time} \in \mathrm{dom}(\sigma)$, that denotes the current (model) time.

Process terms are the 'core' elements of the χ formalism. The set of process terms P is defined by the following grammar for the process terms $p \in P$:

$$
\begin{array}{rl}
p ::= & W : r \gg l_{\mathrm{a}} \mid u \mid \bot \mid \delta \mid [p] \mid u \curvearrowright p \mid p;\, p \\
\mid & b \rightarrow \mid p \,[\!]\, p \mid p \parallel p \mid h \,!!\, \mathbf{e}_n \mid h \,??\, \mathbf{x}_n \mid \partial_A(p) \\
\mid & \upsilon_H(p) \mid X \mid \iota_{J^+}(p) \mid [\![_{\mathrm{V}}\, \sigma_\bot, C, L \text{ `}\mid\text{'} p \,]\!] \mid [\![_{\mathrm{H}}\, H \text{ `}\mid\text{'} p \,]\!] \\
\mid & [\![_{\mathrm{R}}\, R \text{ `}\mid\text{'} p \,]\!]
\end{array}
$$

$p \in P$ consists of atomic process term: action predicate $W : r \gg l_{\mathrm{a}}$, delay predicate u, inconsistent process term $\bot$, deadlock process term δ, send process term $h \,!!\, \mathbf{e}_n$, receive process term $h \,??\, \mathbf{x}_n$; unary operators: the any delay operator [ ], signal emission operator $\curvearrowright$, guarded operator $\rightarrow$, encapsulation operator $\partial_A()$, urgent communication operator $\upsilon_H(\ )$, recursive definition X, jump enabling operator $\iota_{J^+}(\ )$, variable scope operator $[\![_{\mathrm{V}}\, \sigma_\bot, C, L \mid\ ]\!]$, channel scope operator $[\![_{\mathrm{H}}\, H \mid\ ]\!]$, recursion scope operator $[\![_{\mathrm{R}}\, R \mid\ ]\!]$, and binary operators: sequential composition ; , alternative composition operator $[\!]$, and parallel composition operator $\parallel$.

A more detailed explanation of χ and the semantics of χ can be found in [1].

3 Connections between other formalisms and χ

As mentioned previously, one of the most important concepts of χ is the integration between the DC and CS world views. Affine systems are powerful tools for describing or approximating hybrid systems (DC world views). General translation schemes from continuous-time piecewise affine systems and discrete-time piecewise affine systems to χ are defined, which show that these formalisms are closely related.

On the other hand, the study of hybrid systems in computer science (CS world views) is mainly focussed on the theory of hybrid automata. Formal translation from the theory of hybrid automata to χ has 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.

All above-mentioned formal translations can be found in [1]. For illustration purposes, in this section, we only give the translation of continuous-time piecewise affine systems to χ, and the translation from hybrid automata to χ by means of the thermostat example.

3.1 Continuous-time PWA to χ

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

$$
\begin{array}{rcll}
\dot{x}(t) & = & A_i x(t) + B_i u(t) + f_i & \text{for} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \in \Omega_i, \\
y(t) & = & C_i x(t) + D_i u(t) + g_i &
\end{array}
$$

where i $(i = 1, \ldots, N)$ is the number of the mode. Each mode i is defined in a region Ω_i, which is a convex polyhedron, given by a finite number of linear inequalities, in the input/state space. Here, $u(t) \in \mathbb{R}^m$, $x(t) \in \mathbb{R}^n$, and $y(t) \in \mathbb{R}^l$ denote the input, state and output, respectively, at time t. Furthermore, f_i,

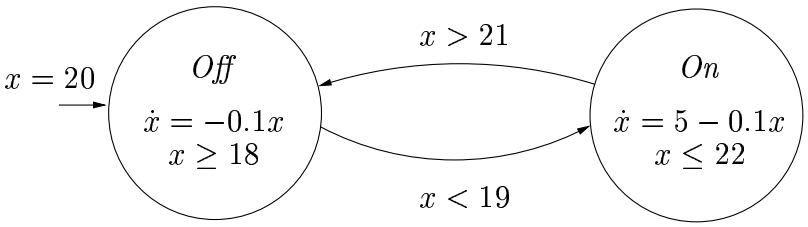

Figure 1: A hybrid automaton model of a thermostat

and g_i denote constants. 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.

Continuous-time PWA systems using the Caratheodory solution concept, can be translated to χ as follows:

$$
\begin{array}{l}
\langle\ \mathsf{cont}\ x,\ \mathsf{alg}\ y \\
,\ x = x_0 \\
|\ (\Omega_1 \Rightarrow \dot{x} = A_1 x + B_1 u + f_1,\ y = C_1 x + D_1 u + g_1) \\
\quad \wedge \\
\quad \vdots \\
\quad \wedge \\
\quad (\Omega_N \Rightarrow \dot{x} = A_N x + B_N u + f_N,\ y = C_N x + D_N u + g_N) \\
\rangle
\end{array}
$$

The state variables x are modeled 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 specification, 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 $(\Omega_i \Rightarrow \dot{x} = A_i x + B_i u + f_i,\ y = C_i x + D_i u + g_i)$.

3.2 Hybrid automata to χ

This section shows the translation of a hybrid automaton model of a thermostat to χ. The hybrid automaton (adapted from [8]) is shown in Figure 1. 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 22 degrees.

This model is translated to χ specification (with some simplification), which results as follows:

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

4 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. [8, 9]. Other definitions allow the more general case of piecewise differential functions, e.g. [10]. Most hybrid automata definitions do not define urgent transitions, or they define urgent transitions in a restrictive way, as in [11]. 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 [8] the variables can in principle perform arbitrary jumps unless restricted by the jump predicate, in [11], 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 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 [9], the solution concept that allows piecewise differentiable functions is based on [10], and the definition of urgent transitions was inspired by [12]. The syntax of the hybrid automata definition HA_{u} is given in Section 4.1. Section 4.2 defines the syntax of the subset χ_{sub} of the χ language that is translated.

4.1 HA_{u} automata definition

A hybrid automaton $HA_{\mathrm{u}} = (X, V, \mathrm{init}, \mathrm{inv}, \mathrm{flow}, E, \mathrm{source}, \mathrm{target}, \mathrm{urgent}, \mathrm{guard}, \mathrm{jump}, \Sigma, \mathrm{event})$ consists of the following components:

- 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 $\in E \to \{\text{true}, \text{false}\}$, that assigns to each edge a boolean: true for an urgent edge, and false for a none-urgent edge.
- A finite set Σ of events, and an edge labeling function event $\in E \to \Sigma$ 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 $\in E \to V$ returns the source location of a given edge, and function target $\in E \to V$ returns the target location of a given edge.

4.2 The χ_{sub} language

The subset χ_{sub} of the χ language that is translated consists of processes $\langle p,\sigma,(\text{dom}(\sigma)\setminus\{\mathsf{time}\},J,\emptyset,H,\emptyset)\rangle$, where $p \in P_{\text{sub}}$ consists of the guarded atomic process terms: guarded action predicate $b \to W : r \gg l_{\text{a}}$, guarded send $b \to h\,!!\,\mathbf{e}_n$, guarded receive $b \to h\,??\,\mathbf{x}_n$, delay predicate u, consistent deadlock process term δ, and guarded inconsistent process term $b \to \bot$, the unary operators the any delay $[\,]$, repetition $*$, encapsulation $\partial_A(\,)$, urgent communication $\upsilon_H(\,)$ and jump enabling ι_{J+}, and the binary operators sequential composition $;$, alternative composition $[\!]$, and parallel composition $\|$. Formally, P_{sub} is defined by:

$$
\begin{array}{rcl}
P_{\text{sub}} ::= & u \;\mid\; \delta \;\mid\; b \to \bot \;\mid\; b \to W : r \gg l_{\text{a}} \\
\mid & b \to h\,!!\,\mathbf{e}_n \;\mid\; b \to h\,??\,\mathbf{x}_n \;\mid\; [P_{\text{sub}}] \\
\mid & *P_{\text{sub}} \;\mid\; \iota_{J+}(P_{\text{sub}}) \;\mid\; P_{\text{sub}};\, P_{\text{sub}} \\
\mid & P_{\text{sub}} \,[\!]\, P_{\text{sub}} \;\mid\; P_{\text{sub}} \,\|\, P_{\text{sub}} \;\mid\; \partial_A(P_{\text{sub}}) \\
\mid & \upsilon_H(P_{\text{sub}})
\end{array}
$$

In χ_{sub} processes, there are no discrete variables ($\text{dom}(\sigma) = C \cup \{\mathsf{time}\}$), no algebraic variables ($L = \emptyset$), and no recursion variables ($R = \emptyset$).

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 consistent deadlock, the action predicate, undelayable send and undelayable receive process terms.

4.3 Translation

In [1], 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 [13].

5 Verification of χ_{sub} specifications using PHAVer

PHAVer (Polyhedral Hybrid Automaton Verifyer) [14] 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 in Section 4.1. 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 constraint is defined as a constraint over a set of variables X that is of the form $\sum_i a_i v_i + b \lozenge 0$, with $a_i, b \in \mathbb{Z}$, $v_i \in X$, and $\lozenge \in \{<, \leq, =\}$. 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, transitions cannot be urgent.

Note that in [1], the relation between linear hybrid I/O-automata and HA_{u} automata has been formalized.

6 Case study: the water-level monitor

The verification of a χ_{sub} specification using PHAVer is illustrated by means of an example: the water-level monitor, which is taken from [15]. 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 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 is denoted by the variable y, drops by 2 per second; when the pump is on, the water-level rises by 1 per second. There is a time delay of 2 seconds 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 χ_{sub} as follows:

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

This specification is translated into a hybrid automaton HA_{u}, which is shown in Figure 2.

The input language of PHAVer is a straightforward textual representation of linear hybrid I/O-automata [16]. Using a code-generator, the code which can be used as input for PHAVer is automatically generated from the linear hybrid I/O-automaton model.

The safety property that the water-level has to be kept between $1 < y < 12$ has been verified using PHAVer. PHAVer reported that this safety property holds in all locations. In [1], we have a theorem that states that this safety property 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.

7 Conclusions and future work

This paper presents the main aspects of the current status of the χ formalism. It illustrates that the χ formalism is closely related to other formalisms from DC and CS.

We describe the translation of a subset of χ to hybrid automata that enables verification of χ specifications using existing hybrid automata based verification tools. Moreover, we discuss the relation between hybrid automata and the linear hybrid I/O automata that are used for the verification tool PHAVer (Polyhedral Hybrid Automaton Verifyer). As a case study, we translate a χ specification of a water-level monitor to a corresponding linear hybrid I/O automaton, and use PHAVer to verify properties.

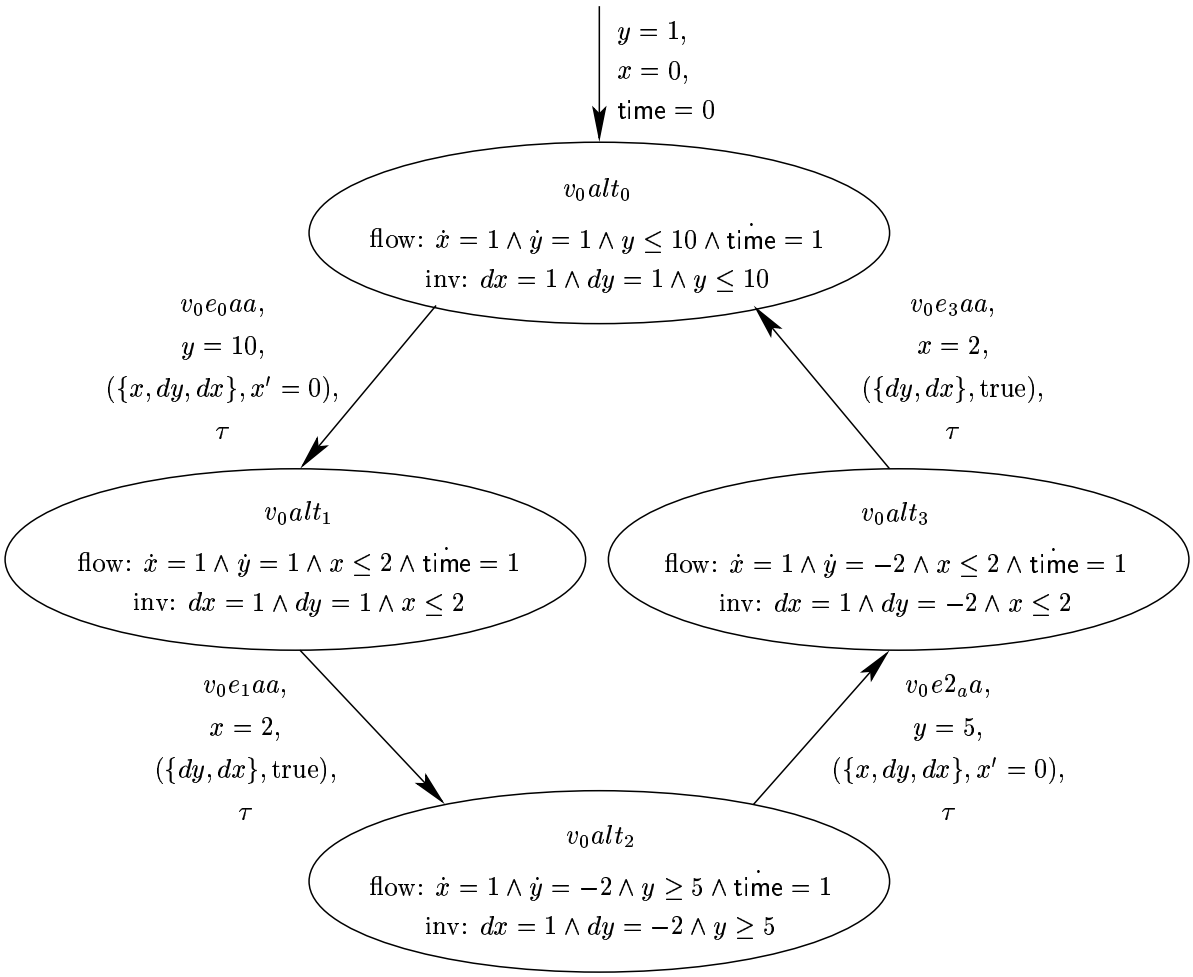

Figure 2: Generated water-level monitor automaton.

As an alternative to analyze χ specifications using hybrid automata based verification tools, χ simulators can be used to simulate χ specifications. Recently, a symbolic simulator has been developed for χ.

Like in ACP [17] and HyPA [18], a set of basic terms (in a subset of χ) has been defined into which all closed terms (of the subset of χ) can be rewritten using χ properties. This is so-called elimination, which is a useful step for algebraic analysis, because it reduces the complexity of specifications (without recursion variables) by transforming them into simpler forms. The elimination result allows to eliminate the parallel composition from many χ specifications, and it can be regarded as a preprocessing step for the linearization (transformation of a recursive specification into linear form) of χ processes.

Acknowledgments

The authors would like to thank J.C.M. Baeten, D.A. van Beek, M.A. Reniers and J.E. Rooda for the supervision of our Ph.D research work. Furthermore, we thank Rolf Theunissen for performing the case study.

References

[1] Man, K.L., Schiffelers, R.R.H.: Formal Specification and Analysis of Hybrid Systems. PhD thesis, Eindhoven University of Technology (To be accepted)

[2] Naumoski, G., Alberts, W.: A Discrete-Event Simulator for Systems Engineering. PhD thesis, Eindhoven University of Technology (1998)

[3] 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.

[4] Fábián, G.: A Language and Simulator for Hybrid Systems. PhD thesis, Eindhoven University of Technology (1999)

[5] 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

[6] Bos, V., Kleijn, J.J.T.: Formal Specification and Analysis of Industrial Systems. PhD thesis, Eindhoven University of Technology (2002)

[7] Bos, V., Kleijn, J.J.T.: Automatic verification of a manufacturing system. Robotics and Computer Integrated Manufacturing **17** (2000) 185–198

[8] Henzinger, T.A.: The theory of hybrid automata. In Inan, M., Kurshan, R., eds.: Verification of Digital and Hybrid Systems. Volume 170 of NATO