Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

RELATING CHI TO HYBRID AUTOMATA

Bert van Beek
Niek G. Jansen
Koos E. Rooda
Ramon R.H. Schiffelers

Department of Mechanical Engineering
Eindhoven University of Technology
P.O.Box 513, 5600 MB, Eindhoven,
THE NETHERLANDS

Ka L. Man
Michel A. Reniers

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O.Box 513, 5600 MB, Eindhoven,
THE NETHERLANDS

ABSTRACT

A hybrid automaton is one of the most popular formal models for hybrid system specification. The Chi language is a hybrid formalism for modeling, simulation and verification. It consists of a number of operators that operate on all process terms, including differential algebraic equations. This paper relates the two formalisms by means of a formal translation from a hybrid automaton model to a Chi model, and a comparison of the semantics of the two models in terms of their respective transition systems. The comparison is illustrated by means of three examples: a thermostat, a railroad gate controller, and dry friction.

1 INTRODUCTION

A hybrid automaton (Alur et al. 1995; Henzinger 2000b; Gueguen and Lefebvre 2001; Johansson et al. 1999) is one of the most popular formal models for hybrid system specification. This paper relates the hybrid automaton of Henzinger (2000b) to the hybrid χ (Chi) formalism.

The χ formalism was originally designed as a modeling and simulation language for specification of discrete-event (DE), continuous time (CT) or combined DE/CT models (so-called hybrid models). The language and simulator have been successfully applied to a large number of industrial cases, such as an integrated circuit manufacturing plant, a brewery, and process industry plants (van Beek, van den Ham, and Rooda 2002).

One of the goals of our research is the development of a hybrid verification formalism / modeling and simulation language with associated verification and simulation tools. The recent formalization of the χ language, including the continuous part, resulted in the χ_{σ_h} process algebra (Schiffelers et al. 2003a) and in a more elegant χ modeling language. The χ language now has the same operators, with the same semantics, as the χ_{σ_h} formal language. The χ modeling language extends χ_{σ_h} with, among others, parameterized process and experiment definitions, and with instantiations of the defined processes and experiments. A straightforward syntactical translation of χ to χ_{σ_h} is described in (Schiffelers et al. 2003b). The relation between χ and χ_{σ_h} is illustrated in Figure 1. A χ_{σ_h} process can be simulated and properties can be verified. A χ experiment can also be compiled directly to obtain a simulator. Reasons for this can be to gain speed, or to make it easier to provide user friendly error information related to the χ specification.

The χ language is related both to simulation languages, see van Beek and Rooda (2000) and to formal languages such as HyPa (Cuijpers and Reniers 2003), hybrid formalisms based on CSP (Jifeng 1994), (Chaochen, Ji, and Ravn 1996), hybrid I/O automata (Lynch, Segala, and Vaandrager 2003), hybrid automata (Alur et al. 1995), and to work derived from hybrid automata, such as Charon (Alur et al. 2001) and Masaccio (Henzinger 2000a). In particular, the χ disrupt, choice, recursion, and reinitialization operators are inspired by HyPA (Cuijpers and Reniers 2003). Overall, χ is more expressive than other formal languages, see Schiffelers et al. (2003a). It is suited to 1) modeling of a wide range of control systems, such as regulatory control, sequence control, scheduling algorithms, hierarchical control, agent based control; 2) modeling of complex concurrent interacting physical systems; and 3) modeling of DAE-based hybrid phenomena, consistent initialization of higher index sys-

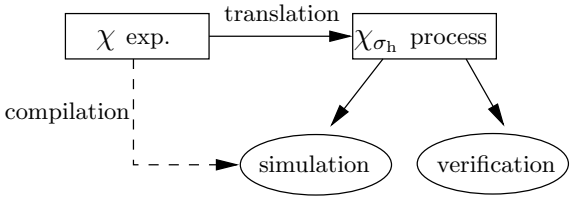


Figure 1: From χ to χ_{σ_h}

tems (Fábián, van Beek, and Rooda 2001), and mode switches accompanied by index changes such as treated in Mosterman and Ciolfi (2002).

In Section 2, a general translation scheme from a general hybrid automaton to a corresponding χ specification is presented. Examples of modeling a rail gate controller and the dry friction phenomenon using both hybrid automata and χ are presented in Section 3 and 4.

2 TRANSLATION

In this section, a general translation scheme from a hybrid automaton to a corresponding χ specification is presented. The syntax of the χ language is given in the Appendix.

2.1 Description Hybrid Automaton

A hybrid automaton (Henzinger 2000b) 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, 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 (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 $X \cup \dot{X}$.

- 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 $X \cup X'$.

- 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 $\in V \to \mathcal{P}(E)$ returns a set of outgoing edges for a location, and function target $\in E \to V$ returns the target vertex of an edge.

2.2 Translation Scheme

Consider a hybrid automaton model with n variables ($X = \{x_1, \ldots, x_n\}$) and k control modes ($V = \{v_1, \ldots, v_k\}$) to be translated to a corresponding χ specification. The translation is defined as follows:

$$
\begin{array}{l}
\mathsf{proc}\ \mathit{HybridAutomatonIn}\chi\ () = \\
[\![\ \mathsf{cont}\ \ x_1, \ldots, x_n : \mathrm{real} \\
,\ \{v_1 \mapsto (\mathrm{inv}(v_1) \parallel \mathrm{flow}(v_1)) \triangleright \\
\qquad (\oplus_{e:e \in \mathrm{edges}(v_1)}\ \mathcal{T}(\mathrm{jump}(e)) \gg \\
\qquad \mathrm{inv}(\mathrm{target}(e)) \to \mathsf{skip}; \mathrm{target}(e)) \\
\quad \vdots \\
\ ,\ v_k \mapsto (\mathrm{inv}(v_k) \parallel \mathrm{flow}(v_k)) \triangleright \\
\qquad (\oplus_{e:e \in \mathrm{edges}(v_k)}\ \mathcal{T}(\mathrm{jump}(e)) \gg \\
\qquad \mathrm{inv}(\mathrm{target}(e)) \to \mathsf{skip}; \mathrm{target}(e)) \\
\ \}\\
|\ \mathrm{init}(v_1) \gg v_1 \oplus \ldots \oplus \mathrm{init}(v_k) \gg v_k \\
]\!]
\end{array}
$$

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 process term describing the continuous behavior of the vertex, disrupted by the choice composition 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 the parallel composition of its invariant and flow predicates ($\mathrm{inv}(v_i) \parallel \mathrm{flow}(v_i)$).

For each outgoing edge, the jump predicate of that edge is translated to a reinitialization predicate ($\mathcal{T}(\mathrm{jump}(e))$), where function $\mathcal{T}$ renames variables occurring with a prime "′" superscript in a jump predicate to variables with superscript "+". E.g. $\mathcal{T}(x' - y' = z)$ becomes $x^+ - y^+ = z$. Here, x^+ and y^+ refer to the values of x and y after the discrete jump that is equivalent to the notation x' used in Henzinger (2000b).

The event label of the edge is translated to the skip process term. This translation implies an event abstraction which is explained in more detail in Section 2.3.

The semantics of hybrid automata contain a kind of look-ahead such that after a control switch the invariant of the target vertex of an edge must hold (using the values of the variables as defined after the reinitialization according to the jump conditions), otherwise the transition cannot occur. Using χ, this look-ahead is modeled by means of guarding the skip process term with the invariant predicate of its target vertex ($\mathrm{inv}(\mathrm{target}(e))$).

After the transition, the behavior is specified by the recursion variable associated with the target vertex ($\mathrm{target}(e)$).

The choice operator ($\oplus$) is used to combine the individual process terms of the outgoing edges.

Note that for $\text{edges}(v_i) = \{e_1, \ldots, e_k\}$, notation $(\oplus_{e:e\in\text{edges}(v_i)}(\mathcal{T}(\text{jump}(e)) \gg \text{inv}(\text{target}(e)) \rightarrow \mathsf{skip};\ \text{target}(e))$, denotes the process term $\mathcal{T}(\text{jump}(e_1)) \gg \text{inv}(\text{target}(e_1)) \rightarrow \mathsf{skip}; \text{target}(e_1) \oplus \ldots \oplus \mathcal{T}(\text{jump}(e_k)) \gg \text{inv}(\text{target}(e_k)) \rightarrow \mathsf{skip}; \text{target}(e_k))$.

The 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 translation from a χ model to a hybrid automaton is more difficult since χ has a richer set of operators, especially for specification of discrete-event systems. Also, χ has more support for DAE-based modeling of hybrid phenomena, such as mentioned in the introduction. The translation from parallel composition of hybrid automata to parallel composition of χ process terms and vice versa is further complicated 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.

2.3 Semantical Comparison

Previously, a translation of a hybrid automaton to a χ process has been described at the syntactical level. In this section, the relation between a hybrid automaton and its associated χ process is discussed on the semantical level.

The semantics of a hybrid automaton (Henzinger 2000b) 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 (Cuijpers, Reniers, and Heemels 2002; Schiffelers et al. 2003a) 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 valuation of the model variables. 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.

A second difference is the existence of time transitions with a duration of zero in the timed transition systems. Each state in the semantics of a hybrid automaton has a zero-duration time transition to itself. Between different states there never is a zero-duration time transition. Such transitions are not present in the hybrid transition systems associated to χ processes.

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.

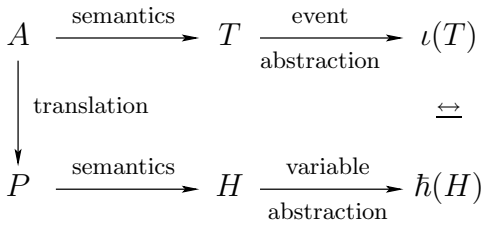
Finally, the translation presented before shows that the events of the control switches are all translated into the process term skip. So, in the timed transition system associated with a hybrid automaton a wide range of action labels representing events may occur, whereas in the hybrid transition system of the corresponding χ process only internal transitions (denoted τ) occur. Thus, the translation abstracts from the names of the events.

In order to describe the relation between a hybrid automaton and its χ-'equivalent' more precisely, first some abstraction mechanisms are introduced to overcome the above-mentioned differences.

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.

Let ι be a mapping that maps a timed transition system onto a timed transition system where all labels of action transitions are replaced by the label τ and where all zero-duration transitions are removed.

Let A be a hybrid automaton and let P be the χ process associated to it by the translation defined in this paper. Furthermore, let T and H be the semantics of A and P, respectively.



Then, there exists a (strong-)bisimulation relation ($\underline{\leftrightarrow}$) between the states of $\iota(T)$ and the states of $\hbar(T)$ such that any transition from an initial state of T can be simulated by the initial state of $\hbar(H)$ and each transition from the initial state of $\hbar(H)$ is simulated by some initial state of T.

2.4 A Thermostat

This example shows the translation of a hybrid automaton to χ. The hybrid automaton of Figure 2 models a thermostat. Variable x represents the temperature. The control modes are *On* and *Off*. Figure 2 is taken

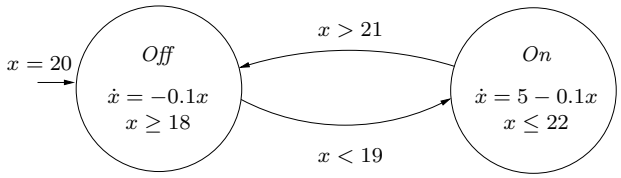


Figure 2: A Hybrid Automaton Model of a Thermostat

from Henzinger (2000b), where the usual informal notation is used: events on the edges are ignored, and the jump conditions are incomplete. In particular, in Figure 2 both edges should have an event label, the jump conditions of the edges should have been $x < 19 \wedge x' = x$ and $x > 21 \wedge x' = x$, respectively. In this paper we use the same informal notation for the hybrid automata in the examples, to improve readability. The χ specifications, however are formal.

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. Due to the invariant condition $x \geq 18$, 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.

Using χ, process *Thermostat* consists of a continuous variable x which represents the temperature, and two recursion variables *On* and *Off* which refer to the control modes *On* and *Off* of the hybrid automaton model, respectively. Initially the heater is off and the temperature is 20 degrees. The mode *Off* can delay as long as $x \geq 18$. The temperature decreases at a rate of $\dot{x} = -0.1x$. When the guard $x < 19$ on the right hand side of the disrupt operator $\triangleright$ becomes true, and the guard $(x \leq 22)$ of the skip process term evaluates to true, the skip process term can take over the delay by means of performing a τ action. After that, the recursion variable *On* takes over. The mode *On* can delay as long as $x \leq 22$. When the guard $x > 21$ becomes true, skip can perform a τ action and the recursion variable *Off* takes over.

```
proc Thermostat () =
[[ cont x : real
, { Off  ↦(x ≥ 18 ‖ ẋ = −0.1x)▷
          (x < 19 ≫ x ≤ 22 → skip; On)
  , On   ↦(x ≤ 22 ‖ ẋ = 5 − 0.1x)▷
          (x > 21 ≫ x ≥ 18 → skip; Off)
  }
| x+ = 20 ≫ Off ⊕ false ≫ On
]]
```

Modeling the thermostat directly in χ leads to the following specification:

```
proc Thermostat () =
[[ cont x : real = 20
, { Off  ↦ (x ≥ 18 ‖ ẋ = −0.1x)   ▷(x < 19≫ On)
  , On   ↦ (x ≤ 22 ‖ ẋ = 5 − 0.1x)▷(x > 21≫ Off)
  }
| Off
]]
```

The hybrid transition system of this specification contains only time transitions, i.e. there are no action transitions due to switching from one control mode into another.

3 RAILROAD GATE CONTROL

In Section 2, a general translation from a hybrid automaton to χ is defined. This section shows that modeling directly in χ, and using χ's expressivity, leads to more elegant models.

Consider a train on a circular track, a gate and a controller. When the train approaches the gate, the controller must lower the gate. The controller has a reaction delay of u time units. After the train has passed the gate the controller must raise the gate. The purpose of the model is to determine the value of u, such that the gate is always fully lowered when the train is at a certain distance from the gate.

Figure 3 shows the hybrid automaton model of the railroad gate controller as defined in Henzinger (2000b).

The χ model takes into account that there is only one train on the circular track, which implies that the transitions of the self loops of the controller automaton can never occur. Figure 4 shows the iconic model of the railroad gate controller. The dashed lines with arrow heads represent synchronization channels (*approach*, *exit*, *raise*, *lower*).

Process *Rail* is a formal specification of the informal iconic model from Figure 4. Channels *approach*, *exit*, *raise* and *lower* are of type void, which means that they are used for pure synchronization, no data is communicated.

```
proc Rail(u : real) =
[[ chan approach, exit, raise, lower : void
, cont x : real, cont y : real = 90
```

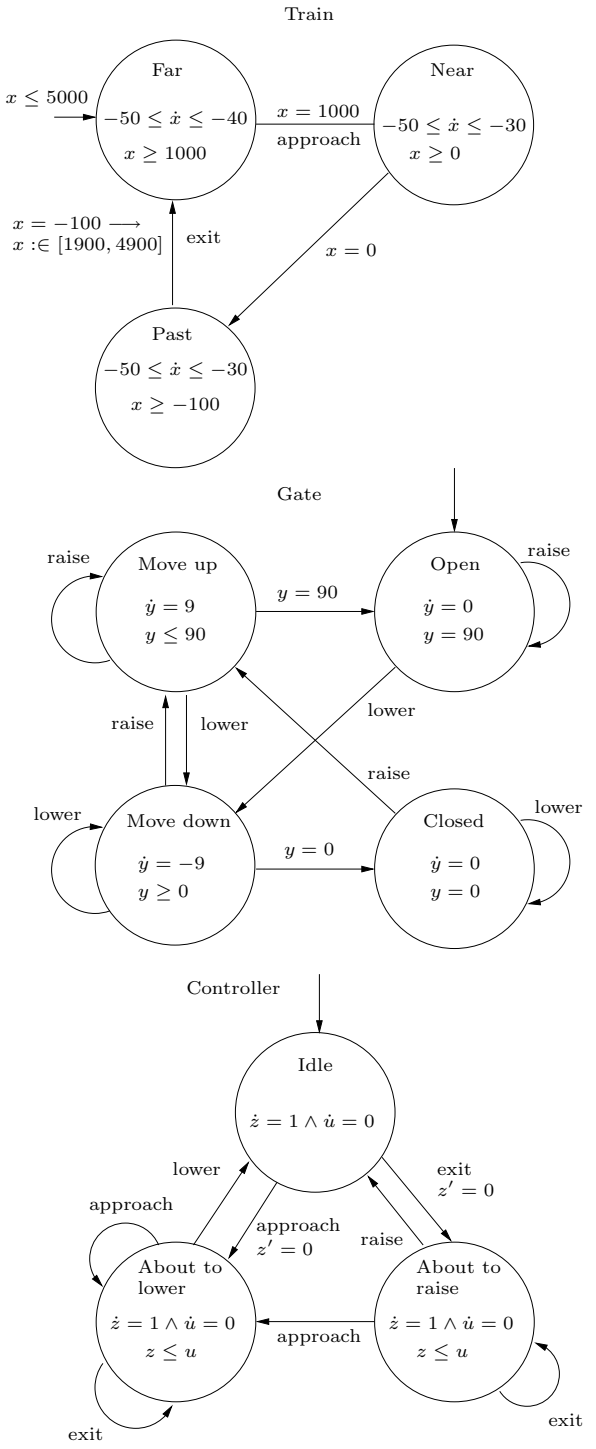


Figure 3: Railroad Gate Control Automaton

$$
\begin{array}{l}
\mid\ \mathit{Train}(x, \mathit{approach}, \mathit{exit}) \parallel \mathit{Gate}(y, \mathit{raise}, \mathit{lower}) \\
\parallel\ C(\mathit{approach}, \mathit{exit}, \mathit{raise}, \mathit{lower}, u) \\
]\!]
\end{array}
$$

A train is modeled by process definition *Train*, which consists of a reinitialization $x^+ \le 5000 \gg$ followed by an infinite loop $(*(\ldots))$. The velocity of the train can be any function of time between -50 and -40. The process waits until the train has reached position $x = 1000$ and then synchronizes with the controller $\pi(\mathit{approach}\,!)$. Maximal progress operator π enforces the synchronization $\mathit{approach}\,!$ to take place immediately, without delay. The train is now approaching the gate. If the train has reached the exit position, such that $x = -100$, the process synchronizes with the controller, the position x of the train is reset to a value between 1900 and 4900, and the loop is re-executed.

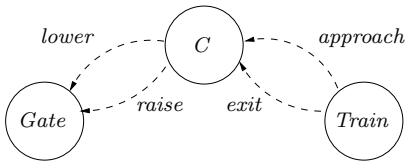


Figure 4: Iconic Model of the Railroad Gate Controller

$$
\begin{array}{l}
\mathsf{proc}\ \mathit{Train}(\mathsf{ext}\ x : \mathrm{real},\ \mathit{approach}, \mathit{exit} : !\,\mathrm{void}) = \\
[\![\ x^+ \le 5000 \gg \\
\quad *(\ (-50 \le \dot{x} \le -40 \ [\!]\ \nabla x \ge 1000);\ \pi(\mathit{approach}\,!) \\
\quad\ ;\ (-50 \le \dot{x} \le -30 \ [\!]\ \nabla x \le -100) \\
\quad\ ;\ 1900 \le x^+ \le 4900 \gg \pi(\mathit{exit}\,!) \\
\quad\ ) \\
]\!]
\end{array}
$$

A gate is modeled by process definition *Gate*, which consists of a parallel composition of an equation $(\dot{y} = n)$ and an infinite loop. The infinite loop is an alternative composition of four process terms. The first process term waits until the gate is lowered $(y = 0)$ and then stops the gate. The second process term waits until the gate is raised $(y = 90)$. The third and fourth process term wait for synchronization with the controller in order to raise or lower the gate ($\mathit{raise}\,?$ and $\mathit{lower}\,?$ respectively). The four process terms delay in parallel until one of the four events $(\nabla y \le 0,\ \nabla y \ge 90,\ \mathit{raise}\,?,\ \mathit{lower}\,?)$ takes place.

$$
\begin{array}{l}
\mathsf{proc}\ \mathit{Gate}(\mathsf{ext}\ y : \mathrm{real},\ \mathit{raise}, \mathit{lower} : ?\,\mathrm{void}) = \\
[\![\ \mathsf{var}\ n : \mathrm{nat} = 0 \\
\mid\ \dot{y} = n \parallel *(\ n < 0 \to \nabla y \le 0;\ n := 0 \\
\qquad\qquad [\!]\ n > 0 \to \nabla y \ge 90;\ n := 0 \\
\qquad\qquad [\!]\ \mathit{raise}\,?;\ n := 9 \\
\qquad\qquad [\!]\ \mathit{lower}\,?;\ n := -9 \\
\qquad\qquad) \\
]\!]
\end{array}
$$

A controller is modeled by process definition C which consists of an infinite loop of three alternatives. It waits in parallel for one of the following events to occur: an approaching train ($\mathit{approach}\,?$), a leaving train ($\mathit{exit}\,?$),

or if atr is true, the end of the reaction delay ($\Delta\, u \rhd \mathsf{skip}$) that precedes raising of the gate. Parameter u is used to model the reaction delay in the controller. Process term $\Delta\, u$ terminates after u units of time, and process term $\Delta u \rhd \mathsf{skip}$ terminates after any interval between 0 and u units of time, because the skip internal action can take over the delay at any point in time. Boolean variable atr is true if and only if the hybrid automaton that models the controller is in control mode (vertex) 'About to raise'.

```
proc C( approach, exit : ?void
      , raise, lower : !void, u : real
      )=
[[ var atr : bool = false
 | *( approach?; atr := false; (Δ u ▷ skip); π(lower!)
    [] exit?; atr := true
    [] atr → (Δ u ▷ skip); atr := false; π(raise!)
    )
]]
```

4 DRY FRICTION PHENOMENON

Figure 5 shows a driving force F_{d} applied to a body on a flat surface with frictional force F_{f}. When the body is moving with positive velocity v, the frictional force is given by $F_{\mathrm{f}} = \mu F_{\mathrm{N}}$, where $F_{\mathrm{N}} = mg$. When the velocity of the body is zero and $-\mu_0 F_{\mathrm{N}} \le F_{\mathrm{d}} \le \mu_0 F_{\mathrm{N}}$ ($\mu_0 > \mu$), the frictional force neutralizes the applied driving force.

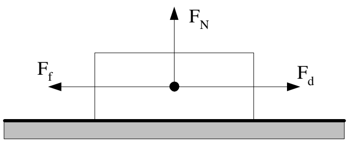


Figure 5: Dry Friction

In this section, the dry friction phenomenon is modeled using χ. Furthermore, it is shown that an attempt to model this phenomenon using hybrid automata as defined in Section 2.1 failed.

4.1 χ Specification of Dry Friction

In the χ specification of the dry friction phenomenon, recursion variables are used to specify the modes "neg", "stop", and "pos". The mode "stop" requires that v is initially 0. The mode "stop" is maintained for as long as the parallel composition ($v = 0 \to v = 0 \parallel -\mu_0 F_{\mathrm{N}} \le F_{\mathrm{d}} \le \mu_0 F_{\mathrm{N}}$) can delay. Otherwise the process term ($F_{\mathrm{d}} \le -\mu_0 F_{\mathrm{N}} \to \mathrm{neg} \oplus F_{\mathrm{d}} \ge \mu_0 F_{\mathrm{N}} \to \mathrm{pos}$) after the disrupt operator $\rhd$ takes over. The choice operator $\oplus$ specifies that either process term $F_{\mathrm{d}} \le -\mu_0 F_{\mathrm{N}} \to \mathrm{neg}$ or $F_{\mathrm{d}} \ge \mu_0 F_{\mathrm{N}} \to \mathrm{pos}$ is executed. Therefore, depending on the value of F_{d}, either the process term specified by recursion variable (mode) "neg" or "pos" is executed. The mode "pos" ("neg") is maintained until condition $v \le 0 \wedge F_{\mathrm{d}} < \mu_0 F_{\mathrm{N}}$ ($v \ge 0 \wedge F_{\mathrm{d}} > -\mu_0 F_{\mathrm{N}}$) becomes true. Using the maximal progress operator π, action transitions have priority over time transitions. Therefore, when $v \le 0$ and $F_{\mathrm{d}} < \mu_0 F_{\mathrm{N}}$, the empty action skip is enabled and immediately executed. Subsequently the mode "stop" is executed again. Initially either the mode "neg", "stop" or "pos" is chosen (neg $\oplus$ stop $\oplus$ pos), based on the initial values of v and F_{d}. F_{d} equals the sine function of t; m, F_{N}, μ_0, μ are constants.

```
proc Dryfriction( m, F_N, μ_0, μ : real )=
[[ cont x, v, F_d, t : real
 , { stop ↦ ( v = 0 → v = 0 || −μ_0 F_N ≤ F_d ≤ μ_0 F_N )
            ▷( F_d ≤ −μ_0 F_N → neg
             ⊕ F_d ≥ μ_0 F_N   → pos
             )
   , pos ↦  ( m v̇ = F_d − μ F_N || v ≥ 0 )
            ▷( v ≤ 0 ∧ F_d < μ_0 F_N → skip; stop )
   , neg ↦  ( m v̇ = F_d + μ F_N || v ≤ 0 )
            ▷( v ≥ 0 ∧ F_d > −μ_0 F_N → skip; stop )
   }
 | ṫ = 1 || F_d = sin(t) || ẋ = v || π(neg ⊕ stop ⊕ pos)
]]
```

4.2 No Hybrid Automaton for Dry Friction?

The hybrid automaton specification in Figure 6 has three locations "neg", "stop", and "pos". These locations/modes correspond with the invariants $v \le 0$, $v = 0$, and $v \ge 0$, respectively. In the mode "stop", the friction force F_{f} neutralizes the applied driving force F_{d} and the velocity v equals 0. The mode "stop" is maintained for as long as the driving force satisfies the condition $-\mu_0 F_{\mathrm{N}} \le F_{\mathrm{d}} \le \mu_0 F_{\mathrm{N}}$. If this condition can no longer be satisfied, the mode becomes "pos" or "neg", respectively. The condition $F_{\mathrm{d}} < \mu_0 F_{\mathrm{N}}$ ($F_{\mathrm{d}} > -\mu_0 F_{\mathrm{N}}$) prevents the automaton to go back to location "stop" immediately after a transition from mode "stop" to "pos" ("neg").

The mode "pos", is maintained for as long as the condition $v \ge 0$ is satisfied. In this mode, frictional force F_{f} equals μF_{N}. When this condition can no longer be satisfied, the mode becomes "stop".

However, this hybrid automaton does not model the dry friction phenomenon correctly. Suppose, the automaton is in the mode "pos", $F_{\mathrm{d}} > \mu F_{\mathrm{N}}$, $v > 0$, and the driving force F_{d} decreases. When $F_{\mathrm{d}} < \mu F_{\mathrm{N}}$, the velocity decreases, and eventually $v = 0$. Now, the transition to mode "stop" has to be taken, although the invariant $v \ge 0$ still holds. If this transition would not be taken, and $F_{\mathrm{d}} = \mu F_{\mathrm{N}}$, the automaton would remain in mode "pos". When the driving force would increase, the body would start moving when $F_{\mathrm{d}} \ge \mu F_{\mathrm{N}}$ instead

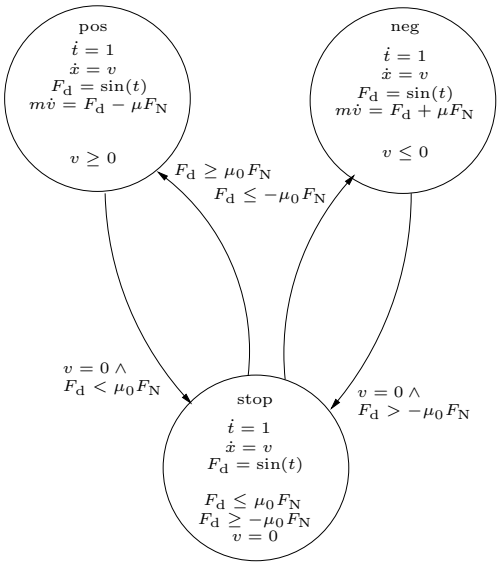


Figure 6: (Incorrect) Dry Friction Automaton

of $F_\mathrm{d} \geq \mu_0 F_\mathrm{N}$. It is not possible to enforce the transition by removing the equal sign of the invariant $v \geq 0$ in mode "pos". This would disable the transition, because zero would then become an accumulation point: v would approach 0 infinitely close, but v would not become equal to zero. Furthermore, it would disable the transition from mode "stop" to mode "pos".

Due to the maximal progress operator in χ, the transition to mode "stop" is taken, even though the equations in mode "pos" still can perform delay transitions.

The dry friction phenomenon could be modeled using a "hybrid automaton" if the transition with guard $v = 0$ (and $F_\mathrm{d} < \mu_0 F_\mathrm{N}$, or $F_\mathrm{d} > -\mu_0 F_\mathrm{N}$) would be taken as soon as its guard became true.

5 CONCLUSIONS

The translation of a single hybrid automaton to χ has been shown to be straightforward. In general, hybrid automata should not be translated directly to χ. Modeling directly in χ, and thus using χ's expressive power, leads to more elegant models. The formal semantics, the χ simulator (Fábián 1999) and the model checker for the discrete-event part of χ (Bos and Kleijn 2002) are the basis of the new χ simulation and χ_{σ_h} verification tools that will be developed.

APPENDIX: SYNTAX DEFINITION OF χ

In this section, a subset of the syntax of the χ language is introduced in an extended BNF-like notation, where brackets '[' and ']' enclose optional items. A process definition has syntax

$$PD ::= \mathsf{proc}\ p_\mathrm{i}\ \text{`('}\ [D_\mathrm{f}]\ \text{`)'}\ =\ \text{`}[\![\ \text{'}\ [D\ [\text{`,'}\ \text{`}\{\text{'}\ R\ \text{`}\}\text{'}]\ \text{`}|\text{'}]\ P\ \text{`}]\!]\text{'}$$

where p_i denotes a process identifier, D_f and D denote declarations, R denotes a parameterless recursive process definition, and P denotes a process term.

The declaration of the formal parameters D_f of a process definition has the following syntax, where *vis* is a comma separated list of variable identifiers, *cis* is a comma separated list of channel identifiers, and t is a type.

$$\begin{array}{rcl} D_\mathrm{f} & ::= & \mathit{vis}\ \text{`:'}\ t \quad | \quad \mathit{cis}\ \text{`:}\ !\text{'}\ t \quad | \quad \mathit{cis}\ \text{`:}\ ?\text{'}\ t \\ & | & \mathsf{ext}\ \ \mathit{vis}\ \text{`:'}\ t \quad | \quad D_\mathrm{f}\ \text{`,'}\ D_\mathrm{f} \end{array}$$

The declaration $\mathit{vis} : t$ denotes the declaration of variables of type t; $\mathit{cis} : !t$ and $\mathit{cis} : ?t$ denote the declaration of channels *cis* of type t used for sending or receiving, respectively; and $\mathsf{ext}\ \ \mathit{vis} : t$ declares *vis* as external (shared) variables of type t.

The declaration of variables and channels in a process definition D has the following syntax. These variables and channels are by definition local, and cannot be used outside the process in which they are declared:

$$\begin{array}{l} D ::= \mathsf{var}\ \ \mathit{vis}\ \text{`:'}\ t\ [\text{`='}\ c] \mid \mathsf{cont}\ \ \mathit{vis}\ \text{`:'}\ t\ [\text{`='}\ c] \\ \quad \mid \mathsf{chan}\ \ \mathit{cis}\ \text{`:'}\ t \mid D\ \text{`,'}\ D \end{array}$$

$$R ::= \mathit{ri}\ \text{`}\mapsto\text{'}\ P \mid R\ \text{`,'}\ R$$

where c is a constant expression, *ri* a recursion variable, and R a recursive process definition, i.e. a partial function from recursion variables to process terms. The declarations $\mathsf{var}\ \ \mathit{vis} : t$, $\mathsf{cont}\ \ \mathit{vis} : t$, and $\mathsf{chan}\ \ \mathit{cis} : t$ denote the declaration of discrete variables, continuous variables, and channels of type t respectively. Optionally, variables can be initialized at their declaration ([`=' c]).

Process term P is built from atomic process terms (statements) AP, using operators for combining them:

$$\begin{array}{rcl} AP & ::= & \mathsf{skip} \quad | \quad x\ \text{`:='}\ e \quad | \quad m\ \text{`!'}\ e \quad | \quad m\ \text{`?'}\ x \\ & | & u \quad | \quad \text{`}\Delta\text{'}\ e_\mathrm{n} \quad | \quad \text{`}\nabla\text{'}\ b_\mathrm{n} \end{array}$$

$$\begin{array}{rcl} P & ::= & AP \mid X \quad | \quad i\ \text{`}\gg\text{'}\ P \quad | \quad b\ \text{`}\rightarrow\text{'}\ P \\ & | & P\ \text{`;'}\ P \quad | \quad P\ \text{`}\triangleright\text{'}\ P \quad | \quad P\ \text{`}\oplus\text{'}\ P \\ & | & P\ \text{`}[]\text{'}\ P \quad | \quad P\ \text{`}\parallel\text{'}\ P \quad | \quad \text{`}*\text{'}\ P \\ & | & p_\mathrm{i}\text{`('}d_\mathrm{a}\text{`)'} \quad | \quad \pi(P) \end{array}$$

An informal (concise) explanation of this syntax is given below.

The process term skip represents an internal action. The value of variables can be changed instantaneously

through assignments. An assignment is a process term of the form $x := e$ with x a variable and e an expression. In principle, the continuous variables change arbitrarily over time. Predicates u over discrete variables, continuous variables, and the derivatives of continuous variables are used to control these changes. I.e., a predicate restricts the allowed behavior of the continuous variables. More complex process terms can be obtained by combining process terms by means of among others sequential composition (;), choice ($\oplus$), alternative composition ($[\!]$), parallel composition ($\|$), prefixing a process p term by a reinitialization predicate i: $i \gg p$, and guarding a process term p by a boolean expression b: $b \rightarrow p$. The process term $i \gg p$ denotes the process term that behaves as p starting from the reinitialized state if the reinitialization predicate i can be satisfied and deadlocks otherwise. The process term $b \rightarrow p$ denotes the process term that behaves as process term p in case the boolean expression b evaluates to true and deadlocks otherwise.

Processes interact either through the use of shared variables or by synchronous point-to-point communication over a channel. By means of $m!e$, the value of expression e is sent over channel m. By means of $m?x$ a value is received from channel m in variable x. The acts of sending and receiving a value have to take place at the same moment in time.

Some of the atomic process terms in χ are delay-able (sending and receiving), others are not delay-able (skip, assignments). By means of the delay process term Δe_{n} a process can be forced to delay for the amount of time units specified by the value of numerical expression e_{n}. A nabla process term of the form ∇b_{n}, where b_{n} represents a boolean variable or a comparison of real-valued expressions using $\leq$, or $\geq$, terminates by means of an internal action if b_{n} is true, and blocks otherwise. By means of the maximal progress operator π, execution of actions can be given priority over passage of time.

The disrupt operator ($\rhd$) is used for describing that a process is allowed to take over execution from another process even if that process is not terminated yet (this in contrast with sequential composition). This is useful for describing mode switches and interrupts/disrupts.

In χ, two operators can be used for the purpose of describing alternative behaviors; the choice operator ($\oplus$) and the alternative composition operator ($[\!]$). The choice operator allows choice between different kinds of continuous behavior of a process, where the choice depends on the initial state of the continuous-time or hybrid process. The alternative composition operator allows choice between different actions/events of a process, usually between time-events, state-events or communication events of a discrete-event controller. In such a case, time-passing should not make a choice. The choice is delayed until the first action is possible.

A process instantiation, $p_{\mathrm{i}}(d_{\mathrm{a}})$, where p_i refers to a process definition, and d_{a} denotes a comma separated list of expressions (actual parameters), is used to instantiate process definitions. It is assumed that instantiation is finite and therefore recursive process instantiations are not allowed.

The operators are listed in descending order of their binding strength as follows $\{\gg, \rightarrow, ;, \rhd, *\}, \{\oplus, [\!], \|\}$. The operators inside the braces have equal binding strength. In addition, operators of equal binding strength associate to the left, and parentheses may be used to group expressions.

REFERENCES

Alur, R., C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. 1995. The algorithmic analysis of hybrid systems. In *Theoretical Computer Science*, Volume 138, 3–34. Springer-Verlag.

Alur, R., T. Dang, J. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. J. Pappas, and O. Sokolsky. 2001, Oct. Hierarchical modeling and analysis of embedded systems. In *First Workshop on Embedded Software EMSOFT'01*.

Bos, V., and J. J. T. Kleijn. 2002. *Formal specification and analysis of industrial systems*. Ph. D. thesis, Eindhoven University of Technology.

Chaochen, Z., W. Ji, and A. P. Ravn. 1996. A formal description of hybrid systems. In *Hybrid Systems III - Verification and Control*, ed. R. Alur, T. A. Henzinger, and E. D. Sonntag, Lecture Notes in Computer Science 1066, 511–530. Springer-Verlag.

Cuijpers, P. J. L., and M. A. Reniers. 2003. Hybrid process algebra. Technical Report CS-Report 03-07, Eindhoven University of Technology, Department of Computer Science, The Netherlands.

Cuijpers, P. J. L., M. A. Reniers, and W. P. M. H. Heemels. 2002. Hybrid transition systems. Technical Report CS-Report 02-12, Eindhoven University of Technology, Department of Computer Science, The Netherlands.

Fábián, G. 1999. *A language and simulator for hybrid systems*. Ph. D. thesis, Eindhoven University of Technology.

Fábián, G., D. A. van Beek, and J. E. Rooda. 2001. Index reduction and discontinuity handling using substitute equations. *Mathematical and Computer Modelling of Dynamical Systems* 7 (2): 173–187.

Gueguen, H., and M. A. Lefebvre. 2001. A comparison of mixed specification formalisms. *APII JESA Journal Europeen des Systemes Automatises* 35 (4): 381–394.

Henzinger, T. A. 2000a. Masaccio: A formal model for embedded components. In *First IFIP International Conference on Theoretical Computer Science (TCS)*, Lecture Notes in Computer Science 1872, 549–563. Springer-Verlag.

Henzinger, T. A. 2000b. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, ed. M. Inan and R. Kurshan, Volume 170 of *NATO ASI Series F: Computer and Systems Science*. New York: Springer-Verlag.

Jifeng, H. 1994. From CSP to hybrid systems. In *A Classical Mind, Essays in Honour of C.A.R. Hoare*, ed. A. W. Roscoe, 171–189. Prentice Hall.

Johansson, K. H., M. Egerstedt, J. Lygeros, and S. Sastry. 1999. On the regularization of zeno hybrid automata. *Systems-&-Control-Letters* 38:141–150.

Lynch, N. A., R. Segala, and F. W. Vaandrager. 2003, Jan. Hybrid I/O automata. Technical Report MIT-LCS-TR-827d, MIT Laboratory for Computer Science, Cambridge, MA 02139. to appear in Information and Computation.

Mosterman, P. J., and J. E. Ciolfi. 2002. Embedded code generation for efficient reinitialization. In *15th Triennial World Congress of the International Federation of Automatic Control*. CD-ROM.

Schiffelers, R. R. H., D. A. van Beek, K. L. Man, M. A. Reniers, and J. E. Rooda. 2003a. Formal semantics of hybrid Chi. In *First International Workshop on Formal Modeling and Analysis of Timed Systems*. To be published.

Schiffelers, R. R. H., D. A. van Beek, K. L. Man, M. A. Reniers, and J. E. Rooda. 2003b, Jun. A hybrid language for modeling, simulation and verification. In *IFAC Conference on Analysis and Design of Hybrid Systems*, ed. S. Engell, H. Guéguen, and J. Zaytoon, 235–240. Saint-Malo, Brittany, France.

van Beek, D. A., and J. E. Rooda. 2000. Languages and applications in hybrid modelling and simulation: Positioning of Chi. *Control Engineering Practice* 8 (1): 81–91.

van Beek, D. A., A. van den Ham, and J. E. Rooda. 2002. Modelling and control of process industry batch production systems. In *15th Triennial World Congress of the International Federation of Automatic Control*. Barcelona. CD-ROM.

AUTHOR BIOGRAPHIES

D.A. (BERT) VAN BEEK is lecturer at the Systems Engineering Group at the Eindhoven University of Technology since 1986. His research interests include modeling, simulation and verification of hybrid systems. He is associate editor of Simulation: Transactions of The Society for Modeling and Simulation International. He can be reached at `<d.a.v.beek@tue.nl>` and at `http://se.wtb.tue.nl/~vanbeek`.

NIEK G. JANSEN graduated in 2003 in Mechanical Engineering with respect to the design and formalization of the χ language.

KA L. MAN is a Ph.D. research assistant in the Department of Mathematics and Computer Science at the Eindhoven University of Technology. His research interests focus on formal analysis of hybrid Systems. He can be reached at `<k.l.man@tue.nl>`.

MICHEL A. RENIERS is lecturer at the Design and Analysis of Systems Group from the Department of Mathematics and Computer Science at the Eindhoven University of Technology since 1999. His research interests include formal methods for the specification, analysis and verification of timed and hybrid systems. He can be reached at `<M.A.Reniers@tue.nl>` and at `http://www.win.tue.nl/~michelr`

J.E. (KOOS) ROODA received his Ph.D. degree from Twente University of Technology, The Netherlands. Since 1985 he is full professor of (Manufacturing) Systems Engineering at the Department of Mechanical Engineering of Eindhoven University of Technology. His research fields of interest are modeling and analysis of manufacturing systems. His interest is especially in control of manufacturing lines and in supervisory control of manufacturing machines. He can be reached at `<j.e.rooda@tue.nl>` and at `http://se.wtb.tue.nl/`.

RAMON R.H. SCHIFFELERS is a Ph.D. research assistant in the Department of Mechanical Engineering at the Eindhoven University of Technology. His research interests include modeling, simulation and verification of hybrid systems. He can be reached at `<r.r.h.schiffelers@tue.nl>`.