In: *First International Workshop on Formal Modeling and Analysis of Timed Systems, LNCS, 2003. To be published.*

# Formal Semantics of Hybrid Chi

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

Department of Mechanical Engineering and Department of Mathematics and Computer Science
Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands

{R.R.H.Schiffelers,D.A.v.Beek,K.L.Man,M.A.Reniers,J.E.Rooda}@tue.nl

**Abstract.** The verification formalism / modeling and simulation language hybrid Chi is defined. The semantics of hybrid Chi is formally specified using Structured Operational Semantics (SOS) and a number of associated functions. The $\chi$ syntax and semantics can also deal with local scoping of variables and/or channels, implicit differential algebraic equations, such as higher index systems, and they are very well suited for specification of pure discrete event systems.

## 1 Introduction

The hybrid $\chi$ (Chi) language 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 [1]. For the purpose of verification, the discrete-event part of the language was mapped onto the $\chi_\sigma$ process algebra, for which a structured operational semantics was defined, bisimulation relations were derived, and a model checker was built [2]. In this way, verification of DE $\chi$ models was made possible [3].

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 $\chi$ language, including the continuous part, resulted in the $\chi_{\sigma_\mathrm{h}}$ process algebra, described in this paper, and in a more elegant $\chi$ modeling language. The $\chi$ language now has the same operators, with the same semantics, as the $\chi_{\sigma_\mathrm{h}}$ formal language. The $\chi$ modeling language extends $\chi_{\sigma_\mathrm{h}}$ with, among others, parameterized process and experiment definitions and instantiations. A straightforward syntactical translation of $\chi$ to $\chi_{\sigma_\mathrm{h}}$ is described in [4].

The $\chi_{\sigma_\mathrm{h}}$ language is a hybrid process algebra, and is thus related to other hybrid process algebras, such as HyPa [5], the $\phi$-Calculus [6], and hybrid formalisms based on CSP [7], [8]. It is also related to hybrid Petri nets [9], hybrid I/O automata [10], hybrid automata [11], and to work derived from hybrid automata, such as Charon [12] and Masaccio [13]. The main difference between the $\chi$ formalism and these other formalisms is that $\chi$ is overall a more expressive formalism. Higher expressivity means either that certain phenomena can be modeled in $\chi$ whereas they cannot be modeled in some other formalisms, or that certain phenomena can be modeled more concisely or more intuitively in $\chi$. The higher expressivity is a result of:

1. The relatively large number of operators dedicated to modeling of discrete-event behavior. This makes it easy to abstract from continuous behavior and specify pure discrete-event models, without any continuous variables. In this respect, $\chi$ has much in common with the $\phi$-Calculus [6], and the hybrid formalisms based on CSP [7], [8].
2. The division of continuous variables into three subclasses. This allows for specification of steady state initialization, initialization of algebraic variables, consistent initialization of higher index systems, mode switches accompanied by index changes [14], and variables changing dynamically from differential to algebraic. In HyPa [5], such phenomena can in principle also be specified. HyPa, however requires a categorization of variables attached to every equation, whereas in $\chi$ this can be specified once, by means of a scope operator.
3. The scope operator combined with parameterized process definition and instantiation that enable hierarchical composition of processes. In this respect, the $\chi$ language is related to Charon [12], that allows components to be defined and instantiated. Local variables and variable abstraction are present in many formalisms. In $\chi$, however, the concepts of variable abstraction and channel abstraction (comparable with action abstraction in other formalisms) are integrated in the scope operator, which also provides a local scope for the three classes of continuous variables and for recursive process definitions.

Section 2 describes the syntax of the $\chi_{\sigma_\mathrm{h}}$ language. In Section 3, the semantics of $\chi_{\sigma_\mathrm{h}}$ is formally specified using a structured operational semantics (SOS) and a number of associated functions. Examples in Section 4 are used to illustrate the language.

## 2 Syntax of the $\chi_{\sigma_\mathrm{h}}$ Language

A $\chi_{\sigma_\mathrm{h}}$ process is a triple $\langle p, \sigma, E\rangle$, where $p$ denotes a process term, $\sigma$ denotes a valuation, and $E$ denotes an environment. A valuation is a partial function from variables to values (constants). 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. An environment is a five-tuple $(E_\Gamma, E_\mathrm{J}, E_\mathrm{F}, E_\mathrm{C}, E_\mathrm{R})$, where $E_\Gamma, E_\mathrm{J}, E_\mathrm{F}$ denote sets of "normal" continuous variables, jumping continuous variables, and fixed continuous variables, respectively. In most models, the normal continuous variables are used. The behavior of these variables depends on the way they occur in equations: a normal continuous variable that occurs differentiated or algebraic (not differentiated) behaves as a fixed continuous variable or as a jumping continuous variable, respectively (see the semantics of function $\Omega$ in Section 3). All variables must be in the domain of $\sigma$. The variables that are not in any of the sets $E_\Gamma, E_\mathrm{J}, E_\mathrm{F}$ are discrete. In the environment, $E_\mathrm{C}$ denotes a set of channel labels, and $E_\mathrm{R}$ denotes a recursive process definition. A recursive process definition is a partial function from recursion variables to process terms. Syntactically, a recursive process definition is denoted by a set of pairs $\{X_0 \mapsto p_0, \ldots, X_m \mapsto p_m\}$, where $X_i$ denotes a recursion variable and $p_i$ the process term defining it. Process terms $P$ in $\chi_{\sigma_\mathrm{h}}$ are built from atomic process terms ($AP$) using operators for combining them:

$$\begin{array}{rl} AP ::= & \mathsf{skip} \mid x := e \mid m!e \mid m?x \mid u \mid \Delta e_\mathrm{n} \\ P ::= & AP \mid X \mid b \to P \mid P \rhd P \mid P ; P \mid P \oplus P \\ & \mid P \, [\!] \, P \mid P \parallel P \mid [\![ \, \sigma, E \mid P \, ]\!] \mid \partial(P) \mid \pi(P) \end{array}$$

An informal (concise) explanation of this syntax is given below. Section 3 gives a more detailed account of their meaning.

The process term $\mathsf{skip}$ 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 ($u$) are used to control these changes, i.e., a predicate restricts the allowed behavior of the continuous variables. In $\chi$ two types of predicates over continuous and discrete variables are allowed: (1) differential equations of the form $rde_1 = rde_2$ where $rde_1$ and $rde_2$ are real-valued expressions in which the derivative operator may be used (e.g., $\dot{x} = -x + y$), and (2) predicates in which the derivative operator may not be used (e.g., $x \geq 0$, $y = 2x + 2$, $\mathsf{true}$).

More complex process terms can be obtained by composing process terms by means of among others sequential composition (;), choice ($\oplus$), alternative composition ($[\!]$), parallel composition ( $\parallel$ ) and guarding a process term $p$ by a boolean expression $b$: $b \to p$. The process term $b \to p$ 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. The encapsulation operator $\partial$ is introduced to block internal send and receive events in order to assure that only their synchronous execution takes place.

Some of the atomic process terms in $\chi_{\sigma_\mathrm{h}}$ are delay-able (sending and receiving), others are not delay-able ($\mathsf{skip}$, assignments). By means of the delay process term $\Delta e_\mathrm{n}$ a process can be forced to delay for the amount of time units specified by the value of numerical expression $e_\mathrm{n}$. By means of the maximal progress operator $\pi$, 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 finished yet (this in contrast with sequential composition). This is useful for describing mode switches and interrupts/disrupts.

In $\chi$, 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 scope process term $[\![\ \sigma, E \mid p\ ]\!]$ is used to declare a local scope. Here $\sigma$ denotes a valuation of local variables, and $E$ denotes a local environment as defined in the beginning of this section.

The operators are listed in descending order of their binding strength as follows $\{;, \rightarrow, \rhd\}, \{\oplus, [\!], \parallel\}$. 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.

## 3 Semantics of the $\chi_{\sigma_\mathrm{h}}$ Language

In this section, the structured operational semantics (SOS) of $\chi_{\sigma_\mathrm{h}}$ is presented. It associates a hybrid transition system [15] with a $\chi_{\sigma_\mathrm{h}}$ process.

### 3.1 General Description of the SOS

The main purpose of such an SOS is to define the behavior of $\chi_{\sigma_\mathrm{h}}$ processes at a certain chosen level of abstraction. The meaning of a $\chi_{\sigma_\mathrm{h}}$ process depends on the values of the variables and on the environment. A set $V$ of variables, and a set $\mathbb{C}$ of channel labels that may be used in $\chi_{\sigma_\mathrm{h}}$ specifications are assumed. The values of the variables at a specific moment in time are captured by means of a valuation, i.e., a partial function from the variables to the union of the set of values $\Lambda$ (containing at least the booleans $\mathbb{B}$, and the reals $\mathbb{R}$) and a "value" $\perp$ (indicating undefinedness). The set of all valuations is denoted $\Sigma$: $\Sigma = V \mapsto (\Lambda \cup \{\perp\})$. The set $T$ is used to represent points in time; usually $T = \mathbb{R}_{\geq 0}$. The set of environments $ES$ is defined as $ES = \mathcal{P}(V) \times \mathcal{P}(V) \times \mathcal{P}(V) \times \mathcal{P}(\mathbb{C}) \times RS$, where $\mathcal{P}$ denotes the powerset function and $RS = XS \mapsto P$ denotes the set of all partial functions of recursion variables $XS$ to process terms $P$. The elements of an environment $E \in ES$ can be obtained by means of five functions: $\Gamma, \mathcal{J}, \mathcal{F} \in ES \rightarrow \mathcal{P}(V)$, $\mathcal{C} \in ES \rightarrow \mathcal{P}(\mathbb{C})$, and $\mathcal{R} \in ES \rightarrow RS$. The function $\Gamma$ is defined as $\Gamma(E_\Gamma, E_\mathrm{J}, E_\mathrm{F}, E_\mathrm{C}, E_\mathrm{R}) = E_\Gamma$. The functions $\mathcal{J}$, $\mathcal{F}$, $\mathcal{C}$ and $\mathcal{R}$ are defined in a similar way to the function $\Gamma$. The SOS is chosen to represent the following:

1. instantaneous execution of discrete transitions:

(a) $\_ \xrightarrow{-} \_ \subseteq (P \times \Sigma \times ES) \times (A_\tau \times \Sigma) \times (P \times \Sigma \times ES)$, where $A_\tau$ denotes the actions, and is defined as $A_\tau = \{\alpha(m, c) \mid \alpha \in \{isa, ira, ca\}, m \in \mathbb{C}, c \in \Lambda\} \cup \{\tau\}$. Here, $isa, ira, ca$ denote action labels for internal send action, internal receive action and communication action respectively, $m \in \mathbb{C}$ denotes a channel, $c \in \Lambda$ denotes a value, and $\tau$ is the internal action. The intuition of a transition $\langle p, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle p', \sigma', E\rangle$ is that the process $\langle p, \sigma, E\rangle$ executes the discrete action $a \in A_\tau$ and thereby transforms into the process $\langle p', \sigma', E\rangle$, where $\sigma'$ denotes the accompanying valuation of the process term $p'$ after the discrete action $a$ is executed.

(b) $\_ \xrightarrow{-} \langle \checkmark, \_, \_\rangle \subseteq (P \times \Sigma \times ES) \times (A_\tau \times \Sigma) \times (\Sigma \times ES)$. The intuition of a transition $\langle p, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle \checkmark, \sigma', E\rangle$ is that the process $\langle p, \sigma', E\rangle$ executes the discrete action $a$ and thereby transforms into the terminated process $\langle \checkmark, \sigma', E\rangle$.

2. continuous behavior: $\_ \overset{-}{\rightsquigarrow} \_ \subseteq (P \times \Sigma \times ES) \times ((T \mapsto \Sigma) \times T) \times (P \times \Sigma \times ES)$. The intuition of a transition $\langle p, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p', \varsigma(t), E\rangle$ is that the variables in $\mathrm{dom}(\sigma)$ behave (continuously) according to the trajectories in $\varsigma$ until (and including) time $t$ and then result in the process $\langle p', \varsigma(t), E\rangle$, where $\varsigma(t) \in \Sigma$ is the valuation at the end point $t$ of the trajectory $\varsigma$.

These relations and predicates are defined through so-called deduction rules. A deduction rule is of the form $\frac{H}{r}$, where $H$ is a number of hypotheses separated by commas and $r$ is the result of the rule. The result of a deduction rule can be derived if all of its hypotheses are derived. In case the set of hypotheses is empty, the deduction rule is called a deduction axiom. The notation $\frac{H}{R}$, where $R$ is a number of results separated by commas, is a shorthand for a deduction rule $\frac{H}{r}$ for each result $r \in R$. In order to increase the readability of the $\chi_{\sigma_\mathrm{h}}$ deduction rules, some abbreviations are used. The notation

$$\frac{\langle p_1, \sigma_1, E_1\rangle \xrightarrow{a_1,\sigma_1'} \left\langle \begin{matrix} {q_1}_0 \\ \vdots \\ {q_1}_n \end{matrix}, \sigma_1', E_1 \right\rangle, \;\cdots,\; \langle p_m, \sigma_m, E_m\rangle \xrightarrow{a_m,\sigma_m'} \left\langle \begin{matrix} {q_m}_0 \\ \vdots \\ {q_m}_n \end{matrix}, \sigma_m', E_m \right\rangle, \; C}{\langle r, \sigma, E\rangle \xrightarrow{b,\sigma'} \left\langle \begin{matrix} s_0 \\ \vdots \\ s_n \end{matrix}, \sigma', E \right\rangle}$$

where ${q_j}_i, s_i \in P \cup \{\checkmark\}$ and $C$ denotes an optional hypothesis that must be satisfied in the deduction rule, is an abbreviation for the following rules (one for each $i$):

$$\frac{\langle p_1, \sigma_1, E_1\rangle \xrightarrow{a_1,\sigma_1'} \langle {q_1}_i, \sigma_1', E_1\rangle, \;\cdots,\; \langle p_m, \sigma_m, E_m\rangle \xrightarrow{a_m,\sigma_m'} \langle {q_m}_i, \sigma_m', E_m\rangle, \; C}{\langle r, \sigma, E\rangle \xrightarrow{b,\sigma'} \langle s_i, \sigma', E\rangle}$$

Based on [10] we use the following definitions of operators $\cup$, $\restriction$, and $\downarrow$ applied on functions. If $f$ is a function, $\mathrm{dom}(f)$ and $\mathrm{range}(f)$ denote the domain and range of $f$, respectively. If $S$ is a set, $f \restriction S$ denotes the restriction of $f$ to $S$, that is, the function $g$ with $\mathrm{dom}(g) = \mathrm{dom}(f) \cap S$, such that $g(c) = f(c)$ for each $c \in \mathrm{dom}(g)$.

If $f$ and $g$ are functions with $\mathrm{dom}(f) \cap \mathrm{dom}(g) = \emptyset$, then $f \cup g$ denotes the unique function $h$ with $\mathrm{dom}(h) = \mathrm{dom}(f) \cup \mathrm{dom}(g)$ satisfying the condition: for each $c \in \mathrm{dom}(h)$, if $c \in \mathrm{dom}(f)$ then $h(c) = f(c)$, and $h(c) = g(c)$ otherwise.

If $f$ is a function whose range is a set of functions and $S$ is a set, then $f \downarrow S$ denotes the function $g$ with $\mathrm{dom}(g) = \mathrm{dom}(f)$ such that $g(c) = f(c) \upharpoonright S$ for each $c \in \mathrm{dom}(g)$. If $f$ is a function whose range is a set of functions, all of which have a particular element $d$ in their domain, then $f \downarrow d$ denotes the function $g$ with $\mathrm{dom}(g) = \mathrm{dom}(f)$ such that $g(c) = f(c)(d)$ for each $c \in \mathrm{dom}(g)$.

## 3.2 Deduction Rules

**Atomic process terms** For the deduction rules of the atomic process terms, it is assumed that $\Gamma(E), \mathcal{J}(E), \mathcal{F}(E) \subseteq \mathrm{dom}(\sigma)$, $m \in \mathcal{C}(E)$, $x \in \mathrm{dom}(\sigma)$, and $\bar{\sigma}(e), \bar{\sigma}(e_{\mathrm{n}}), c \in \Lambda$.

Rule 1 states that the skip process term performs the $\tau$ action to the terminated process ✓ and has no effect on the valuation or environment.

The execution of the assignment process term $x := e$ (see Rule 2) leads to a new valuation where all variables are unchanged except for variable $x$. $\sigma[\bar{\sigma}(e)/x]$ denotes the update of valuation $\sigma$ such that the new value of variable $x$ is $\bar{\sigma}(e)$, which denotes the value of $e$ with respect to $\sigma$. Internal send and receive process terms are intended to be used in parallel composition (see Rule 25). The value of expression $e$ which is sent via channel $m$ is evaluated in valuation $\sigma$ (see Rule 3). The receive process term $m?x$ can receive any value $c$ (see Rule 4).

$$\frac{}{\langle \mathsf{skip}, \sigma, E\rangle \xrightarrow{\tau,\sigma} \langle \checkmark, \sigma, E\rangle}\ 1 \qquad \frac{}{\langle x := e, \sigma, E\rangle \xrightarrow{\tau,\sigma[\bar{\sigma}(e)/x]} \langle \checkmark, \sigma[\bar{\sigma}(e)/x], E\rangle}\ 2$$

$$\frac{}{\langle m!e, \sigma, E\rangle \xrightarrow{isa(m,\bar{\sigma}(e)),\sigma} \langle \checkmark, \sigma, E\rangle}\ 3 \qquad \frac{}{\langle m?x, \sigma, E\rangle \xrightarrow{ira(m,c),\sigma[c/x]} \langle \checkmark, \sigma[c/x], E\rangle}\ 4$$

The predicate process term can perform a time transition for all trajectories $\varsigma$ for predicate $u$ as defined by Rule 5.

$$\frac{\varsigma \in \Omega(\sigma, \Gamma(E), \mathcal{J}(E), \mathcal{F}(E), u, t)}{\langle u, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle u, \varsigma(t), E\rangle}\ 5$$

Function $\Omega \in \Sigma \times \mathcal{P}(V) \times \mathcal{P}(V) \times \mathcal{P}(V) \times U \times T \to \mathcal{P}(T \mapsto \Sigma)$, where $U$ denotes the set of all predicates , returns a set of trajectories from time to a valuation for the variables, given a valuation representing the current values of the variables, a set of normal continuous variables, a set of jumping continuous variables, a set of fixed continuous variables, a predicate and a time point that denotes the duration of the trajectory. Formally, the function $\Omega$ is defined as:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\Omega(\sigma, \mathit{\Gamma}_E, J_E, F_E, u, d) =} \\
\multicolumn{2}{l}{\{\ \varsigma' \downarrow \mathrm{dom}(\sigma)} \\
\multicolumn{2}{l}{\mid\ \varsigma' \in T \mapsto ((V \cup V') \mapsto \mathit{\Lambda})} \\
\multicolumn{2}{l}{,\ \mathrm{dom}(\varsigma') = [0, d],\ d > 0} \\
,\ \forall_{\sigma' \in \mathrm{range}(\varsigma')} & \mathrm{dom}(\sigma') = \mathrm{dom}(\sigma) \cup \{x' \mid x \in \mathcal{D}(u)\} \\
,\ \forall_{0 \le t \le d,\ x \in \mathrm{dom}(\sigma) \setminus (\mathit{\Gamma}_E \cup J_E \cup F_E)} & (\varsigma' \downarrow x)(t) = \sigma(x) \\
,\ \forall_{x \in (\mathcal{D}(u) \setminus J_E) \cup F_E} & (\varsigma' \downarrow x)(0) = \sigma(x) \\
,\ \forall_{x \in (\mathit{\Gamma}_E \cup J_E \cup F_E) \setminus \mathcal{D}(u)} & \varsigma' \downarrow x \text{ is a bounded function that is continuous} \\
 & \text{almost everywhere.} \\
,\ \forall_{x \in \mathcal{D}(u)} & \varsigma' \downarrow x' \text{ is a bounded function that is continuous} \\
 & \text{almost everywhere.} \\
,\ \forall_{0 \le t \le d} & \varsigma'(t) \models \mathcal{T}_{\mathrm{u}}(u) \\
,\ \forall_{0 \le t \le d,\ x \in \mathcal{D}(u)} & (\varsigma' \downarrow x)(t) = (\varsigma' \downarrow x)(0) + \int_0^t (\varsigma' \downarrow x')(s) ds \\
\} &
\end{array}
$$

In lines 5 and 6 of the body of function $\Omega$, it is assumed that the value of $x$ is defined ($\sigma(x) \in \mathit{\Lambda}$). Function $\mathcal{D} \in U \to \mathcal{P}(V)$ extracts the differential variables from a predicate. E.g. $\mathcal{D}(x = \dot{y}) = \{y\}$. Function $\mathcal{T}_{\mathrm{u}} \in U \to U'$ replaces every occurrence of the derivative $\dot{x}$ of a variable with name $x$ in a predicate $u \in U$ by a fresh variable $x' \in V'$ that has the same name as $x$ postfixed with the prime character. The set $V'$ is defined as $V' = \{x' \mid x \in V\}$, and $U'$ denotes the set of predicates on variables $x \in V$ and $x' \in V'$. For example, the application of function $\mathcal{T}_{\mathrm{u}}$ to the equation $\dot{x} = -y + \dot{z}$ gives the equation $x' = -y + z'$.

The behavior of each variable $x$ is described by a function of time $\varsigma' \downarrow x$. The behavior of the discrete variables $x \in \mathrm{dom}(\sigma) \setminus (\mathit{\Gamma}_E \cup J_E \cup F_E)$ is specified by constant functions ($\forall_{0 \le t \le d}\ (\varsigma' \downarrow x)(t) = \sigma(x)$). The initial conditions of the non-jumping differential variables $x \in \mathcal{D}(u) \setminus J_E$ and the fixed continuous variables $x \in F_E$ are specified by $(\varsigma' \downarrow x)(0) = \sigma(x)$. The behavior $\varsigma' \downarrow x$ of the algebraic variables $x \in (\Gamma_E \cup J_E \cup F_E) \setminus \mathcal{D}(u)$ and the behavior $\varsigma' \downarrow x'$ of the derivatives ($x'$ such that $x \in \mathcal{D}(u)$) is a bounded function (not set-valued) that is continuous almost everywhere (except for a set of measure zero). The trajectory $\varsigma'$ satisfies the predicate for all time points of its domain ($\forall_{0 \le t \le d}\ \varsigma'(t) \models \mathcal{T}_{\mathrm{u}}(u)$). The function $\varsigma' \downarrow x$ of a differential variable $x \in \mathcal{D}(u)$ is the integral of the function $\varsigma' \downarrow x'$ of its derivative.

For a normal continuous variable $x \notin (J_E \cup F_E)$, its occurrence in $u$ as differential (occurring differentiated in $u$) or algebraic (not occurring differentiated in $u$), determines the behavior of the variable at the beginning of a time transition. I.e. in time transitions at $t = 0$, differential variables may not behave discontinuously (i.e. may not jump so that $(\varsigma' \downarrow x)(0) = \sigma(x)$). Algebraic variables, on the other hand, may show discontinuous behavior at $t = 0$, so that for these variables $(\varsigma' \downarrow x)(0)$ may be different from $\sigma(x)$. In some cases, differential variables may jump. This is, for example, the case in steady state initializations ($\dot{x} = 0$). E.g. in $\dot{x} = -x + 1 \parallel \dot{x} = 0$, where $x \in J_E$, $(\varsigma' \downarrow x)(0)$ jumps to 1, independently of $\sigma(x)$. The set of fixed continuous variables $F_E$ is needed in cases where algebraic variables need to be initialized. For example consider $\dot{x} = \mathrm{f}(x, y, z) \parallel \dot{y} = \mathrm{g}(x, y, z) \parallel \mathrm{h}(x, y, z) = 0$. Normally, $x$ and $y$ are initialized, and the value of $z$ is then determined by the equations. If, for example, the modeler would prefer to initialize variables $x$ and $z$, so that the value of $y$ is then determined by the equations, the sets $F_E$ and $J_E$ should be such that $z \in F_E$ and $y \in J_E$. Such initializations are common in, for instance, chemical systems.

$$\frac{\varsigma \in \Omega(\sigma, \Gamma(E), \mathcal{J}(E), \mathcal{F}(E), \mathsf{true}, t)}{\langle m!e, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle m!e, \varsigma(t), E\rangle}\ 6 \qquad \frac{\varsigma \in \Omega(\sigma, \Gamma(E), \mathcal{J}(E), \mathcal{F}(E), \mathsf{true}, t)}{\langle m?x, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle m?x, \varsigma(t), E\rangle}\ 7$$

$$\frac{\bar{\sigma}(e_{\mathrm{n}}) = 0}{\langle \Delta e_{\mathrm{n}}, \sigma, E\rangle \xrightarrow{\tau,\sigma} \langle \checkmark, \sigma, E\rangle}\ 8 \quad \frac{0 < t \leq \bar{\sigma}(e_{\mathrm{n}}),\ \varsigma \in \Omega(\sigma, \Gamma(E), \mathcal{J}(E), \mathcal{F}(E), \mathsf{true}, t)}{\langle \Delta e_{\mathrm{n}}, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle \Delta\bar{\sigma}(e_{\mathrm{n}}) - t, \varsigma(t), E\rangle}\ 9$$

Rules 6 and 7 state that $m!e$ and $m?x$ can perform any time transition $\varsigma, t$ that satisfies $\varsigma \in \Omega(\sigma, \Gamma(E), \mathcal{J}(E), \mathcal{F}(E), \mathsf{true}, t)$. The predicate $\mathsf{true}$ does not restrict the continuous behavior of the (continuous) variables.

The delay process term specifies a certain amount of delay. The full amount of delay does not have to be performed in one transition (see Rule 9). Note that $\bar{\sigma}(e_{\mathrm{n}})$ denotes the value of expression $e_{\mathrm{n}}$ with respect to valuation $\sigma$ before the delay. In case that the amount of delay is zero, the delay process term terminates with an internal action as defined by Rule 8. Since there are no rules for the case that the amount of delay is negative, such a delay leads to deadlock.

**Recursion variable** Recursion is used among others to model repetition. The recursion variable $X$ simply behaves as the process term given by $\mathcal{R}(E)(X)$. Here $\mathcal{R}(E)(X)$ is the process term that is defined for recursion variable $X$ in recursive process definition $\mathcal{R}(E)$. It is assumed that $X \in \mathrm{dom}(\mathcal{R}(E))$.

$$\frac{\langle \mathcal{R}(E)(X), \sigma, E\rangle \xrightarrow{a,\sigma'} \langle {\checkmark \atop p'}, \sigma', E\rangle}{\langle X, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle {\checkmark \atop p'}, \sigma', E\rangle}\ 10 \quad \frac{\langle \mathcal{R}(E)(X), \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p', \varsigma(t), E\rangle}{\langle X, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p', \varsigma(t), E\rangle}\ 11$$

**Guard operator** In case that the guard $b$ evaluates to $\mathsf{false}$ (i.e. $\sigma \models \neg b$), there are no transitions. In case that the guard evaluates to $\mathsf{true}$ (i.e. $\sigma \models b$), the guarded process term simply behaves as $p$.

$$\frac{\langle p, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle {\checkmark \atop p'}, \sigma', E\rangle,\ \sigma \models b}{\langle b \rightarrow p, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle {\checkmark \atop p'}, \sigma', E\rangle}\ 12 \quad \frac{\langle p, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p', \varsigma(t), E\rangle,\ \sigma \models b}{\langle b \rightarrow p, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p', \varsigma(t), E\rangle}\ 13$$

**Sequential composition operator** The sequential composition of the process terms $p$ and $q$ behaves as process term $p$ until $p$ terminates, and then continues to behave as process term $q$.

$$\frac{\langle p, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle {\checkmark \atop p'}, \sigma', E\rangle}{\langle p; q, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle {q \atop p'; q}, \sigma', E\rangle}\ 14 \quad \frac{\langle p, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p', \varsigma(t), E\rangle}{\langle p; q, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p'; q, \varsigma(t), E\rangle}\ 15$$

**Disrupt operator** The disrupt operator $p \rhd q$ is introduced to model a kind of sequential composition, where the process term $q$ may take over execution from process term $p$ at any moment, without waiting for its termination.

$$\frac{\langle p,\sigma,E\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p'},\sigma',E\rangle}{\langle p \rhd q,\sigma,E\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p' \rhd q},\sigma',E\rangle}\ 16 \qquad \frac{\langle p,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p',\varsigma(t),E\rangle}{\langle p \rhd q,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p' \rhd q,\varsigma(t),E\rangle}\ 17$$

$$\frac{\langle q,\sigma,E\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{q'},\sigma',E\rangle}{\langle p \rhd q,\sigma,E\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{q'},\sigma',E\rangle}\ 18 \qquad \frac{\langle q,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle q',\varsigma(t),E\rangle}{\langle p \rhd q,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle q',\varsigma(t),E\rangle}\ 19$$

**Choice operator** The effect of applying the choice operator to the process terms $p$ and $q$ is that the execution of a transition by either one of them results in a definite choice.

$$\frac{\langle p,\sigma,E\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p'},\sigma',E\rangle}{\langle p \oplus q,\sigma,E\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p'},\sigma',E\rangle,\ \langle q \oplus p,\sigma,E\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p'},\sigma',E\rangle}\ 20$$

$$\frac{\langle p,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p',\varsigma(t),E\rangle}{\langle p \oplus q,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p',\varsigma(t),E\rangle,\ \langle q \oplus p,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p',\varsigma(t),E\rangle}\ 21$$

**Alternative composition operator** The action behavior of the alternative composition operator is equal to that of the choice operator (see Rule 22). The weak time-determinism principle is adopted for the time transitions. This principle means that the passage of time by itself cannot result in making a choice between two alternatives that can perform that time transition with the same trajectory $\varsigma$ and the same time step $t$. This is captured in Rule 24. Rule 23 states that if one of the two process terms $p$ and $q$ can perform a time transition and the other cannot, then the alternative composition can also perform that time transition, but loses the alternative that could not perform a time transition.

$$\frac{\langle p,\sigma,E\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p'},\sigma',E\rangle}{\langle p \mathbin{[\!]} q,\sigma,E\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p'},\sigma',E\rangle,\ \langle q \mathbin{[\!]} p,\sigma,E\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p'},\sigma',E\rangle}\ 22$$

$$\frac{\langle p,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p',\varsigma(t),E\rangle,\ \langle q,\sigma,E\rangle \not\rightsquigarrow}{\langle p \mathbin{[\!]} q,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p',\varsigma(t),E\rangle,\ \langle q \mathbin{[\!]} p,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p',\varsigma(t),E\rangle}\ 23$$

$$\frac{\langle p,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p',\varsigma(t),E\rangle,\ \langle q,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle q',\varsigma(t),E\rangle}{\langle p \mathbin{[\!]} q,\sigma,E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p' \mathbin{[\!]} q',\varsigma(t),E\rangle}\ 24$$

**Parallel composition operator** The parallel composition of the processes $p$ and $q$ has as its behavior with respect to action transitions the interleaving of the behaviors of $p$ and $q$ (see Rule 26). The time transitions of the parallel composition of two process terms have to synchronize to obtain the time transition (with same trajectory $\varsigma$ and the same time step $t$) of their parallel composition as defined by Rule 27. The parallel composition allows the synchronization of matching send and receive actions. A send action $isa(m, c)$ and a receive action $ira(m', c')$ match iff $m = m'$ and $c = c'$ (i.e. the channels used for sending and receiving are same, and also the value sent and the value received are identical). The result of the synchronization is a communication action $ca(m, c)$ as defined by Rule 25.

$$\frac{\langle p, \sigma, E\rangle \xrightarrow{isa(m,c),\sigma'} \left\langle \begin{array}{c} \checkmark \\ p' \\ \checkmark \\ p' \end{array}, \sigma', E \right\rangle,\ \langle q, \sigma', E\rangle \xrightarrow{ira(m,c),\sigma''} \left\langle \begin{array}{c} \checkmark \\ \checkmark \\ q' \\ q' \end{array}, \sigma'', E \right\rangle}{\langle p \parallel q, \sigma, E\rangle \xrightarrow{ca(m,c),\sigma''} \left\langle \begin{array}{c} \checkmark \\ p' \\ q' \\ p' \parallel q' \end{array}, \sigma'', E \right\rangle,\ \langle q \parallel p, \sigma, E\rangle \xrightarrow{ca(m,c),\sigma''} \left\langle \begin{array}{c} \checkmark \\ p' \\ q' \\ q' \parallel p' \end{array}, \sigma'', E \right\rangle} \; 25$$

$$\frac{\langle p, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle \begin{array}{c} \checkmark \\ p' \end{array}, \sigma', E\rangle}{\langle p \parallel q, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle \begin{array}{c} q \\ p' \parallel q \end{array}, \sigma', E\rangle,\ \langle q \parallel p, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle \begin{array}{c} q \\ q \parallel p' \end{array}, \sigma', E\rangle} \; 26$$

$$\frac{\langle p, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p', \varsigma(t), E\rangle,\ \langle q, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle q', \varsigma(t), E\rangle}{\langle p \parallel q, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p' \parallel q', \varsigma(t), E\rangle} \; 27$$

**Scope operator** By means of the scope operator, local variables (optionally with an initial value) and a local environment can be introduced in a $\chi_{\sigma_\mathrm{h}}$ process. The application of the scope operator to a process $p$ results in the behavior of the process $p$ after the addition of the local variables (in fact the valuation for the local variables) to the global valuation ($\mu(\sigma, \sigma_\mathrm{s})$), and the addition of the local environment to the global environment ($\mu_\mathrm{E}(E, E_\mathrm{s})$). Function $\mu \in \Sigma \times \Sigma \to \Sigma$ merges two valuations. If $\sigma,\ \sigma' \in \Sigma$, $\mu(\sigma, \sigma')$ denotes the valuation $\sigma''$ with $\mathrm{dom}(\sigma'') = \mathrm{dom}(\sigma) \cup \mathrm{dom}(\sigma')$, such that $\forall_{x \in \mathrm{dom}(\sigma')}\ \sigma''(x) = \sigma'(x)$ and $\forall_{x \in \mathrm{dom}(\sigma) \setminus \mathrm{dom}(\sigma')}\ \sigma''(x) = \sigma(x)$. Function $\mu_\mathrm{E} \in ES \times ES \to ES$ merges two environments. It is defined as $\mu_\mathrm{E}(E, E_\mathrm{s}) = (\Gamma(E) \cup \Gamma(E_\mathrm{s}), \mathcal{J}(E) \cup \mathcal{J}(E_\mathrm{s}), \mathcal{F}(E) \cup \mathcal{F}(E_\mathrm{s}), \mathcal{C}(E) \cup \mathcal{C}(E_\mathrm{s}), \mu_\mathrm{R}(\mathcal{R}(E), \mathcal{R}(E_\mathrm{s})))$. Function $\mu_\mathrm{R} \in RS \times RS \to RS$ merges two recursive process definitions. If $R, R' \in RS$, $\mu_\mathrm{R}(R, R')$ denotes the recursive process definition $R''$, with $\mathrm{dom}(R'') = \mathrm{dom}(R) \cup \mathrm{dom}(R')$ such that $\forall_{x \in \mathrm{dom}(R')}\ R''(x) = R'(x)$ and $\forall_{x \in \mathrm{dom}(R) \setminus \mathrm{dom}(R')}\ R''(x) = R(x)$.

The scope operator is also used for abstraction: action abstraction and data abstraction. The skip and assignment actions are internal ($\tau$) actions already. The internal send and receive actions on a local channel are encapsulated (blocked). Therefore, they need not be abstracted. The only action that needs to be abstracted by substitution of a $\tau$ action (action abstraction) is the communication action $ca(m, c)$ via a local channel $m \in \mathcal{C}(E_{\mathrm{s}})$ (see Rule 28). Function $\mathrm{ch} \in A_\tau \to \mathbb{C}\ \cup \{\perp\}$ extracts the channel label from an action. It is defined as $\mathrm{ch}(\alpha(m, c)) = m$ and $\mathrm{ch}(\tau) = \perp$.

The changes of local variables are abstracted (made invisible) outside the scope operator, by removing them from the transition arrow. For action transitions, data abstraction is defined using $\sigma_{\mu\mathrm{s}}$, where $\sigma_{\mu\mathrm{s}}$ denotes $\mu(\sigma, \sigma' \restriction (\mathrm{dom}(\sigma) \setminus \mathrm{dom}(\sigma_{\mathrm{s}})))$, as shown in rules 28 and 29. The changed valuation of local variables is stored in the local valuation $(\sigma' \restriction \mathrm{dom}(\sigma_{\mathrm{s}}))$.

For time transitions, data abstraction is defined using $\varsigma_\sigma$, where $\varsigma_\sigma$ denotes $\varsigma \downarrow (\mathrm{dom}(\sigma) \setminus \mathrm{dom}(\sigma_{\mathrm{s}})) \cup \varsigma_{\mathrm{corr}}$. The correction function $\varsigma_{\mathrm{corr}}$ specifies the continuous behavior of the variables in the start valuation that were redefined in the local valuation $\sigma_{\mathrm{s}}$. It is defined as $\varsigma_{\mathrm{corr}} \in \Omega(\sigma \restriction \mathrm{dom}(\sigma_{\mathrm{s}}), \Gamma(E) \cap \mathrm{dom}(\sigma_{\mathrm{s}}), \mathcal{J}(E) \cap \mathrm{dom}(\sigma_{\mathrm{s}}), \mathcal{F}(E) \cap \mathrm{dom}(\sigma_{\mathrm{s}}), \mathsf{true}, t)$ (see Rule 30).

$$\frac{\langle p, \mu(\sigma, \sigma_{\mathrm{s}}), \mu_{\mathrm{E}}(E, E_{\mathrm{s}})\rangle \xrightarrow{ca(m,c),\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p'}, \sigma', \mu_{\mathrm{E}}(E, E_{\mathrm{s}})\rangle,\ m \in \mathcal{C}(E_{\mathrm{s}})}{\langle [\![\ \sigma_{\mathrm{s}}, E_{\mathrm{s}} \mid p\ ]\!], \sigma, E\rangle \xrightarrow{\tau,\sigma_{\mu\mathrm{s}}} \langle \genfrac{}{}{0pt}{}{\checkmark}{[\![\ \sigma' \restriction \mathrm{dom}(\sigma_{\mathrm{s}}), E_{\mathrm{s}} \mid p'\ ]\!]}, \sigma_{\mu\mathrm{s}}, E\rangle}\ 28$$

$$\frac{\langle p, \mu(\sigma, \sigma_{\mathrm{s}}), \mu_{\mathrm{E}}(E, E_{\mathrm{s}})\rangle \xrightarrow{a,\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p'}, \sigma', \mu_{\mathrm{E}}(E, E_{\mathrm{s}})\rangle,\ \mathrm{ch}(a) \notin \mathcal{C}(E_{\mathrm{s}})}{\langle [\![\ \sigma_{\mathrm{s}}, E_{\mathrm{s}} \mid p\ ]\!], \sigma, E\rangle \xrightarrow{a,\sigma_{\mu\mathrm{s}}} \langle \genfrac{}{}{0pt}{}{\checkmark}{[\![\ \sigma' \restriction \mathrm{dom}(\sigma_{\mathrm{s}}), E_{\mathrm{s}} \mid p'\ ]\!]}, \sigma_{\mu\mathrm{s}}, E\rangle}\ 29$$

$$\frac{\langle p, \mu(\sigma, \sigma_{\mathrm{s}}), \mu_{\mathrm{E}}(E, E_{\mathrm{s}})\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p', \varsigma(t), \mu_{\mathrm{E}}(E, E_{\mathrm{s}})\rangle}{\langle [\![\ \sigma_{\mathrm{s}}, E_{\mathrm{s}} \mid p\ ]\!], \sigma, E\rangle \overset{\varsigma_\sigma,t}{\rightsquigarrow} \langle [\![\ \varsigma(t) \restriction \mathrm{dom}(\sigma_{\mathrm{s}}), E_{\mathrm{s}} \mid p'\ ]\!], \varsigma_\sigma(t), E\rangle}\ 30$$

**Encapsulation operator** The behavior of the encapsulation of a process $\partial(p)$ is the same as the behavior of the process argument $p$ with the restriction that only actions from the set $A_{\mathrm{x}} = \{ca(m, c) \mid m \in \mathbb{C}, c \in \Lambda\} \cup \{\tau\}$ can be executed (see Rule 31). In this way, internal send actions $isa(m, c)$ and internal receive actions $ira(m, c)$ are blocked, and only communication actions $ca(m, c)$ and $\tau$ actions are allowed. Encapsulation has no effect on time transitions, as defined by Rule 32.

$$\frac{\langle p, \sigma, E\rangle \xrightarrow{a_{\mathrm{x}},\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{p'}, \sigma', E\rangle,\ a_{\mathrm{x}} \in A_{\mathrm{x}}}{\langle \partial(p), \sigma, E\rangle \xrightarrow{a_{\mathrm{x}},\sigma'} \langle \genfrac{}{}{0pt}{}{\checkmark}{\partial(p')}, \sigma', E\rangle}\ 31 \qquad \frac{\langle p, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p', \varsigma(t), E\rangle}{\langle \partial(p), \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle \partial(p'), \varsigma(t), E\rangle}\ 32$$

**Maximal progress operator** The maximal progress operator gives action transitions a higher priority than time transitions. Rule 33 states that action behavior is not affected by maximal progress. Time transitions are allowed only if it is not possible to perform any action transitions as defined by Rule 34.

$$\frac{\langle p, \sigma, E\rangle \xrightarrow{a,\sigma'} \langle \overset{\checkmark}{p'}, \sigma', E\rangle}{\langle \pi(p), \sigma, E\rangle \xrightarrow{a,\sigma'} \langle \overset{\checkmark}{\pi(p')}, \sigma', E\rangle}\ 33 \qquad \frac{\langle p, \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle p', \varsigma(t), E\rangle,\ \langle p, \sigma, E\rangle \not\rightarrow}{\langle \pi(p), \sigma, E\rangle \overset{\varsigma,t}{\rightsquigarrow} \langle \pi(p'), \varsigma(t), E\rangle}\ 34$$

For all $\chi_{\sigma_{\mathrm{h}}}$ operators, strong (state-based) bisimulation has been proven to be a congruence.

## 4 Examples

The two examples in this section are related to the kind of hybrid systems that can be modeled by means of hybrid automata and related formalisms. This makes it easier to become familiar with $\chi_{\sigma_{\mathrm{h}}}$ specifications. In practice, however, a modeler would specify models in the $\chi$ language, which has a more user-friendly syntax for the scope operator.

### 4.1 Dry Friction Phenomenon

A driving force $F_{\mathrm{d}}$ is applied to a body on a flat surface with frictional force $F_{\mathrm{f}}$ (Figure 1). 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 $|F_{\mathrm{d}}| < \mu_0 F_{\mathrm{N}}$, the frictional force neutralizes the applied driving force. Instead of locations (hybrid automaton), $\chi$ uses recursion variables to specify the

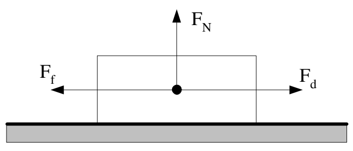

**Fig. 1.** Dry friction

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 ($v = 0 \rightarrow v = 0 \parallel -\mu_0 F_{\mathrm{N}} \leq F_{\mathrm{d}} \leq \mu_0 F_{\mathrm{N}}$) can delay. Otherwise, the process term ($F_{\mathrm{d}} \leq -\mu_0 F_{\mathrm{N}} \rightarrow \mathrm{neg} \oplus F_{\mathrm{d}} \geq \mu_0 F_{\mathrm{N}} \rightarrow \mathrm{pos}$) after the disrupt operator $\rhd$ takes over. The choice operator $\oplus$ specifies that either process term $F_{\mathrm{d}} \leq -\mu_0 F_{\mathrm{N}} \rightarrow \mathrm{neg}$ or $F_{\mathrm{d}} \geq \mu_0 F_{\mathrm{N}} \rightarrow \mathrm{pos}$ is executed. Therefore, depending on the value of $F_{\mathrm{d}}$, either the process term specified by recursion variable (mode) neg or pos is executed. The mode "pos" is maintained until condition $v \leq 0 \wedge F_{\mathrm{d}} < \mu_0 F_{\mathrm{N}}$ becomes true. In $\chi$, action transitions have priority over time transitions. Therefore, when $v \leq 0$

and $F_\mathrm{d} < \mu_0 F_\mathrm{N}$, the process term skip is enabled and is immediately executed. Subsequently the mode "stop" is executed again. Symbols $m$, $F_\mathrm{N}$, $\mu_0$, $\mu$, $x_0$ and $v_0$ are constants.

```
⟨ $\pi([\![ \emptyset,\ ( \emptyset,\emptyset,\emptyset,\emptyset$
            , { stop $\mapsto (v = 0 \to v = 0 \parallel -\mu_0 F_\mathrm{N} \le F_\mathrm{d} \le \mu_0 F_\mathrm{N})$
                      $\rhd( F_\mathrm{d} \le -\mu_0 F_\mathrm{N} \to \mathrm{neg} \oplus F_\mathrm{d} \ge \mu_0 F_\mathrm{N} \to \mathrm{pos}\ )$
              , pos  $\mapsto (v \ge 0 \parallel m\dot{v} = F_\mathrm{d} - \mu F_\mathrm{N})$
                      $\rhd\ (v \le 0 \wedge F_\mathrm{d} < \mu_0 F_\mathrm{N} \to \mathsf{skip};\ \mathrm{stop})$
              , neg  $\mapsto (v \le 0 \parallel m\dot{v} = F_\mathrm{d} + \mu F_\mathrm{N})$
                      $\rhd\ (v \ge 0 \wedge F_\mathrm{d} > -\mu_0 F_\mathrm{N} \to \mathsf{skip};\ \mathrm{stop})$
              }
            )
        | $F_\mathrm{d} = \sin(t) \parallel \dot{t} = 1 \parallel \dot{x} = v \parallel (\mathrm{neg} \oplus \mathrm{stop} \oplus \mathrm{pos})$
        $]\!])$
  , $\{t \mapsto 0, x \mapsto x_0, v \mapsto v_0, F_\mathrm{d} \mapsto \perp\},\ (\{t, x, v, F_\mathrm{d}\}, \emptyset, \emptyset, \emptyset, \emptyset)$
⟩
```

## 4.2 Railroad Gate Controller

In [16] a railroad gate controller is modeled using a hybrid automaton. When a train approaches the gate the controller must close the gate. The controller has a reaction delay of $\alpha$ time units. After the train has passed the gate the controller must open the gate. The purpose of the model is to determine the value of $\alpha$, to ensure that the gate is always fully closed when the train is at a certain distance from the gate.

A formal specification of the railroad gate controller using $\chi_{\sigma_\mathrm{h}}$ is given below. Channels *approach*, *exit*, *open* and *close* are used for pure synchronization, no data is communicated. The train, gate and controller are modeled using different scopes. The scope process term modeling the train consists of a parallel composition of an infinite loop $(*(\ldots))$ and an equation $(\dot{x} = v)$. The velocity of the train can be any function of time between 40 and 50. The process waits until the train has reached position $x = 1000$ and then synchronizes with the controller (*approach* !). The train is now approaching the gate. If the train has reached the exit position $x = 2100$, the train synchronizes with the controller, the position $x$ of the train is reset to zero $(x := 0)$, and the loop is re-executed. The train is now past the gate. The scope process term modeling the gate consists of a parallel composition of an infinite loop and an equation $(\dot{\phi} = n)$. The infinite loop is an alternative composition of four process terms. The first process term waits until the gate is closed $(\phi = 0)$ and then turns off the gate. The second process term waits until the gate is open $(\phi = 90)$. The third and fourth process term wait for synchronization with the controller in order to open or close the gate (*open* ? and *close* ? respectively). The four process terms delay in parallel until one of the four events $(\nabla\, \phi \le 0,\ \nabla\, \phi \ge 90,$ *open* ?, *close* ?) takes place. The controller consists of an infinite loop. It tries to synchronize with the train, in order to open or close the gate (*approach* ? and *exit* ? respectively). The constant $\alpha$ is used to model the reaction delay in the controller. After $\alpha$ time units $(\varDelta\alpha)$

the controller synchronizes with the gate, and the loop is re-executed. In the specification, some abbreviations are used which are listed in the table below.

| **Abbreviation** | **Meaning** |
|---|---|
| $* p$ | $[\![\ \emptyset, (\emptyset, \emptyset, \emptyset, \emptyset, \{X \mapsto p; X\}) \mid X\ ]\!]$ |
| $\nabla x \geq e$ | $x \leq e \rhd (x \geq e \to \mathsf{skip})$ |
| $\nabla x \leq e$ | $x \geq e \rhd (x \leq e \to \mathsf{skip})$ |
| $m!$ | $m!\mathsf{true}$ |
| $m?$ | $[\![\ \{x \mapsto \perp\}, (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \mid m?x\ ]\!]$ |

$$
\begin{array}{l}
\langle\ \pi(\partial(\ [\![\ \{v \mapsto \perp\}, (\{v\}, \emptyset, \emptyset, \emptyset, \emptyset) \\
\qquad\quad \mid \dot{x} = v \parallel *(\ (40 \leq v \leq 50 \ [\!]\ \nabla x \geq 1000);\ \mathit{approach}\,! \\
\qquad\qquad\qquad\qquad ;\ (30 \leq v \leq 50 \ [\!]\ \nabla x \geq 2100);\ \mathit{exit}\,!;\ x := 0 \\
\qquad\qquad\qquad\qquad ) \\
\qquad\quad ]\!] \\
\qquad \parallel [\![\ \{n \mapsto 0\}, (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \\
\qquad\quad \mid \dot{\phi} = n \parallel *(\ n < 0 \to (\nabla \phi \leq 0;\ n := 0) \\
\qquad\qquad\qquad\qquad [\!]\ n > 0 \to (\nabla \phi \geq 90;\ n := 0) \\
\qquad\qquad\qquad\qquad [\!]\ \mathit{open}\,?;\ n := 9 \\
\qquad\qquad\qquad\qquad [\!]\ \mathit{close}\,?;\ n := -9 \\
\qquad\qquad\qquad\qquad ) \\
\qquad\quad ]\!] \\
\qquad \parallel *(\ \mathit{approach}\,?;\ \mathit{\Delta}\alpha;\ \mathit{close}\,! \ [\!]\ \mathit{exit}\,?;\ \mathit{\Delta}\alpha;\ \mathit{open}\,!\ ) \\
\qquad )) \\
,\ \{x \mapsto 0, \phi \mapsto 90\}, (\{x, \phi\}, \emptyset, \emptyset, \{\mathit{approach}, \mathit{exit}, \mathit{open}, \mathit{close}\}, \emptyset) \\
\rangle
\end{array}
$$

## 5 Conclusions and Future Research

The semantics of the hybrid $\chi$ language has been formally specified using a relatively small set of deduction rules and associated functions. The language is highly expressive and can be used to specify a wide range of systems, including pure discrete-event systems, systems with local scoping of variables and/or channels, and systems of implicit algebraic differential equations. Future work entails the extension of the discrete-event $\chi$ verification tool to enable verification of hybrid models. Furthermore, the hybrid $\chi$ simulator will be redesigned to correspond to the new syntax and formal semantics.

### Acknowledgments

The authors would like to thank Pieter Cuijpers, Albert Hofkamp, Niek Jansen, Erjen Lefeber, Sasha Pogromsky, Frits Vaandrager, and the anonymous referees for their helpful comments on drafts of this paper.

## References

1. 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.
2. Bos, V., Kleijn, J.J.T.: Formal Specification and Analysis of Industrial Systems. PhD thesis, Eindhoven University of Technology (2002)
3. Bos, V., Kleijn, J.J.T.: Automatic verification of a manufacturing system. Robotics and Computer Integrated Manufacturing **17** (2000) 185–198
4. Schiffelers, R.R.H., van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E.: A hybrid language for modeling, simulation and verification. In Engell, S., Guéguen, H., Zaytoon, J., eds.: IFAC Conference on Analysis and Design of Hybrid Systems, Saint-Malo, Brittany, France (2003) 235–240
5. Cuijpers, P.J.L., Reniers, M.A.: Hybrid process algebra. Technical Report CS-Report 03-07, Eindhoven University of Technology, Department of Computer Science, The Netherlands (2003)
6. Rounds, W.C., Song, H.: The $\phi$-calculus - a hybrid extension of the $\pi$-calculus to embedded systems. Technical Report CSE 458-02, University of Michigan, USA (2002)
7. Jifeng, H.: From CSP to hybrid systems. In Roscoe, A.W., ed.: A Classical Mind, Essays in Honour of C.A.R. Hoare. Prentice Hall (1994) 171–189
8. Chaochen, Z., Ji, W., Ravn, A.P.: A formal description of hybrid systems. In Alur, R., Henzinger, T.A., Sonntag, E.D., eds.: Hybrid Systems III - Verification and Control. Lecture Notes in Computer Science 1066. Springer-Verlag (1996) 511–530
9. David, R., Alla, H.: On hybrid Petri nets. Discrete Event Dynamic Systems: Theory & Applications **11** (2001) 9–40
10. Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid I/O automata. Technical Report MIT-LCS-TR-827d, MIT Laboratory for Computer Science, Cambridge, MA 02139 (2003) to appear in Information and Computation.
11. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. In: Theoretical Computer Science. Volume 138. Springer-Verlag (1995) 3–34
12. Alur, R., Dang, T., Esposito, J., Hur, Y., Ivancic, F., Kumar, V., Lee, I., Mishra, P., Pappas, G.J., Sokolsky, O.: Hierarchical modeling and analysis of embedded systems. In: First Workshop on Embedded Software EMSOFT'01. (2001)
13. Henzinger, T.A.: Masaccio: A formal model for embedded components. In: First IFIP International Conference on Theoretical Computer Science (TCS). Lecture Notes in Computer Science 1872. Springer-Verlag (2000) 549–563
14. Mosterman, P.J., Ciolfi, J.E.: Embedded code generation for efficient reinitialization. In: 15th Triennial World Congress of the International Federation of Automatic Control. (2002) CD-ROM.
15. Cuijpers, P.J.L., Reniers, M.A., Heemels, W.P.M.H.: Hybrid transition systems. Technical Report CS-Report 02-12, Eindhoven University of Technology, Department of Computer Science, The Netherlands (2002)
16. Alur, R., Henzinger, T.A., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Transactions on Software Engineering **22** (1996) 102–119