Formal Semantics of Hybrid Chi

R.R.H. Schiffelers, D.A. v. Beek, K.L. Man and M.A. Reniers

December 12, 2002

Abstract

Currently, there is a gap between simulation languages and verification formalisms. The χ language attempts to bridge this gap. The language was designed as a hybrid modeling and simulation language. It is based on communicating sequential processes (CSP) and differential algebraic equations. In this paper, the semantics of hybrid χ is formally specified using a structured operational semantics (SOS) and a number of associated functions. Unlike most other hybrid formalisms, the χ syntax and semantics can also deal with higher index systems of equations. Essential aspects of the semantics are illustrated using many small examples, and a somewhat larger example dealing with dry friction.

1 Introduction

Currently, there is a gap between simulation languages and verification formalisms. The χ language [1] attempts to bridge this gap. The language was designed as a hybrid modeling and simulation language. The χ 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 [2].

A formal semantics is a prerequisite for reasoning about models as it unambiguously defines the meaning of the models and hence increases the understanding of these. It also helps in the development of the language itself, the construction of tools (e.g. the simulator), and the development of verification techniques, by clearly separating the issues of meaning, and implementation of this meaning into tools. Using the discrete-event (DE) part of the χ language, an initial investigation into formal verification was performed [3]. Following that, a part of the χ language was formalized using a structured operational semantics resulting in χ_σ; bisimulation relations were derived and a model checker was built [4]. In this way, verification of DE χ models was made possible [5].

This paper describes the syntax and semantics of a subset of χ that also includes descriptions of continuous behavior: χ_{σ_h} (hybrid χ_σ). This semantics and the model checker for χ_σ are the basis of the χ_{σ_h} verification tool that will be developed. The difference between χ_{σ_h} and χ is that the structured data-types of χ—such as sets, arrays, and lists—and the high level operations

on these data-types—such as set quantification—are not formalized, and are thus not present in χ_{σ_h}. In all other respects, χ_{σ_h} retains the high expressivity of χ. This means that, unlike most other hybrid formalisms such as hybrid automata [6] or Petri nets [7], differential *algebraic* equations (DAEs) are fully supported, including higher index DAEs [8]. The χ_{σ_h} formalism can also deal with hybrid systems that can dynamically change from lower to higher index, and vice versa, such as treated in [9]. Due to the algebraic relations that can be present between variables, including between differential variables (higher index systems), the semantics of χ_{σ_h} is more complex than the semantics of, for example, hybrid automata.

2 The χ_{σ_h} Language

In this section, the syntax of the χ_{σ_h} language is introduced. The existence of the set of variables V, the set of channel labels C, the set of values Σ that contains the reals $\mathbb{R}$, the set of continuous variables Γ, the set of dependent differential variables $\mathcal{D}$, the Booleans $\mathbb{B} = \{true, false\}$ and the naturals $\mathbb{N}$ is assumed. The set T is used to represent points in time; usually $T = \mathbb{R}_{\geq 0}$. The basic building blocks of a χ_{σ_h} process description are introduced below:

- The skip process `skip` that performs an internal action. Its purpose is to enforce certain choices in an alternative composition.

- Assignment process terms of the form $x_0, \ldots, x_k := e_0, \ldots, e_k$ where $x_0, \ldots, x_k$ represent variables ($x_0, \ldots, x_k \in V$) and $e_0, \ldots, e_k$ are expressions which take values in Σ. Intuitively, the values of the expressions $e_0, \ldots, e_k$ are assigned in one step to the variables $x_0, \ldots, x_k$ respectively. The notation $\hat{x} := \hat{e}$ is used after wards as a shorthand for $x_0, \ldots, x_k := e_0, \ldots, e_k$. It is also assumed that there is at least one element in $\hat{x}$ and $\hat{e}$.

- Send process terms of the form $m!e$ where m is a channel label ($m \in C$) and e is a value $\in \Sigma$. The value of e is sent via channel label m.

- Receive process terms of the form $m?x$ where m is a channel label ($m \in C$) and x a variable ($x \in V$). x receives a value via channel label m.

- Function equation process terms of the form eq, where eq denotes a function equation. Therefore, $\dot{x} = 1$ is in fact a sloppy notation for $\dot{x} = 1(\cdot)$, where $x \in T \mapsto \mathbb{R}$, and $1(\cdot) \in T \mapsto \mathbb{R}$ is the constant function with value 1. The equations from eq can be solved by the variables in Γ. Function equation process terms are used to describe continuous behavior. A function equation has either one of the following forms:

  1. $rfe_1 \leq rfe_2$ or $rfe_1 \geq rfe_2$ where rfe_1 and rfe_2 represent arbitrary real-valued function expressions in which the derivative operator may not be used.

2. $rdfe_1 = rdfe_2$, where $rdfe_1$ and $rdfe_2$ represent arbitrary real-valued differential function expressions, i.e. real-valued function expressions in which the derivative operator may be used. The functions are piecewise differentiable functions of type $T \mapsto \mathbb{R}$.

- Instantaneous equation process terms of the form $us : eqs$, where us is a non-empty list of variables and eqs is a non-empty list of instantaneous equations separated by commas.
- Delay process terms of the form Δe_{num}, where e_{num} represents a numerical expression. It enables a process to delay n_e time units where n_e is the value of e_{num}.
- Nabla process terms of the form ∇b_n, where b_n represents a comparison of real-valued expressions using $=$, $\leq$, or $\geq$. Such a comparison is referred to as a nabla condition. The nabla process ∇b_n terminates by means of an internal action when either b_n is *false* or when no further delay is possible such that b_n remains *true*.

AP refers to the set of all atomic process terms, US refers to the set of all lists of unknowns for instantaneous equations, EQS refers to the set of all lists of instantaneous equations, EQ refers to the set of all function equations, B refers to the set of all boolean expressions, and BN refers to the set of all nabla conditions. Summarizing, in a BNF-like notation, the atomic process terms are the following:

$$\begin{array}{rcccccccc} AP & ::= & \texttt{skip} & | & \hat{x} := \hat{e} & | & m!e & | & m?x \\ & | & us : eqs & | & \Delta e_{num} & | & \nabla b_n, & & \end{array} \quad | \quad eq$$

where $x \in V$, $m \in C$, e denotes an arbitrary expression, $us \in US$, $eq \in EQ$, $eqs \in EQS$ and $b_n \in BN$. Process term are built from atomic process terms and boolean expressions (from a set B) using operators for alternative composition ($[\!]$), sequential composition ($;$), guarding ($\rightarrow$), repetition ($*$), parallel composition ($\|$), state ($[\![\ | \ ]\!]$), dependent differential variable ($::$), encapsulation (∂_A) , maximal progress (π) and abstraction (τ_A) as described by the following BNF-like notation:

$$\begin{array}{rcccccccccc} P & ::= & ap & | & b \rightarrow P & | & P;P & | & P \,[\!]\, P & | & P \,\|\, P \\ & | & P \,\|\, P & | & [\![\ \sigma, \Gamma \mid P \ ]\!] & | & x :: P & | & \partial_A(P) & | & \pi(P) \\ & | & \tau_A(P), & & & & & & & & \end{array} \quad | \quad *P$$

where $ap \in AP$ represents an arbitrary atomic process, b represents a boolean expression, σ represents a state, $x \in V$ represents a dependent differential variable and A represents the set of transitions to be encapsulated or to be abstracted from. The set of all process terms is denoted by P.

The operators have higher binding strength are listed (in descending order) as follows:

$$\{*, \quad ;, \quad \rightarrow, \quad ::\}, \quad \{[\!], \quad \|\}$$

where the operators inside the braces have the same binding strength. In addition, operators of equal binding strength associate to the left, and parentheses may be used to group expressions.

3 Structured Operational Semantics

In this section, the structured operational semantics (SOS) [10] of the subset of χ_{σ_h} that is used in this paper is presented. The main purpose of such an SOS is to define the behavior of χ_{σ_h} processes at a certain chosen level of abstraction. The meaning of a χ_{σ_h} process depends on the values of the discrete and continuous variables. A χ_{σ_h} process $\langle p, \sigma, \Gamma, \mathcal{D}\rangle$ is therefore a χ_{σ_h} process term $p \in P$, combined with a valuation $\sigma \in V \mapsto \Sigma$, a set of continuous variables $\Gamma \subseteq V$ and a set of dependent differential variables $\mathcal{D} \subset \Gamma$. We chose to represent the following in the SOS:

1. instantaneous execution of discrete transitions:

   (a) $\_ \xrightarrow{\_} \_ \subseteq (P \times (V \mapsto \Sigma) \times \mathcal{P}(V) \times \mathcal{P}(V)) \times A_\tau \times (P \times (V \mapsto \Sigma) \times \mathcal{P}(V) \times \mathcal{P}(V))$, where $A_\tau = A \cup \{\tau\}$, τ is the internal action and $A = \{\sigma_a, isa(m,c), ira(m,x,c), ca(m,x,c) \mid \sigma_a \in V \mapsto \Sigma, x \in V, m \in C, \mathrm{c} \in \Sigma\}$, where σ_a is an action representing assignment and instantaneous equations. The intuition of a transition $\langle p, \sigma, \Gamma, \mathcal{D}\rangle \xrightarrow{a} \langle p', \sigma', \Gamma, \mathcal{D}\rangle$ is that the process $\langle p, \sigma, \Gamma, \mathcal{D}\rangle$ described by the process term p with state σ executes the discrete action $a \in A_\tau$ and thereby transforms into the process $\langle p', \sigma', \Gamma, \mathcal{D}\rangle$.

   (b) $\_ \xrightarrow{\_} \langle \checkmark, \_, \_\rangle \subseteq (P \times (V \mapsto \Sigma) \times \mathcal{P}(V) \times \mathcal{P}(V)) \times A_\tau \times ((V \mapsto \Sigma) \times \mathcal{P}(V) \times \mathcal{P}(V))$. The intuition of a transition $\langle p, \sigma, \Gamma, \mathcal{D}\rangle \xrightarrow{a} \langle \checkmark, \sigma', \Gamma, \mathcal{D}\rangle$ is that the process $\langle p, \sigma, \Gamma, \mathcal{D}\rangle$ described by the process term p with state σ executes the discrete action a and thereby transforms into the terminated process $\langle \checkmark, \sigma', \Gamma, \mathcal{D}\rangle$.

2. continuous behavior: $\_ \xrightarrow{\_,\_} \_ \subseteq (P \times (V \mapsto \Sigma) \times \mathcal{P}(V) \times \mathcal{P}(V)) \times (V \mapsto (T \mapsto \Sigma)) \times T \times (P \times (V \mapsto \Sigma) \times \mathcal{P}(V) \times \mathcal{P}(V))$. The intuition of a transition $\langle p, \sigma, \Gamma, \mathcal{D}\rangle \xrightarrow{\varsigma, t} \langle p', \sigma', \Gamma, \mathcal{D}\rangle$ is that the variables in Γ of the process $\langle p, \sigma, \Gamma, \mathcal{D}\rangle$ behave (continuously) according to the solution functions in ς until (and including) time t and then result in the process $\langle p', \sigma', \Gamma, \mathcal{D}\rangle$.

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, we speak of a deduction axiom. As a shorthand, we sometimes write $\frac{H}{R}$ where R is a number of results separated by commas. It represents a deduction rule $\frac{H}{r}$ for each result $r \in R$.

Deduction Rules

In the deduction rules, the following functions are used: the solution function Ω, the solution function for instantaneous equations Ω_i, the function γ_E which makes the state consistent with the equations in the process term, the translation functions $\mathcal{T}_\mathrm{U}$ and $\mathcal{T}_\mathrm{E}$, and the clean-up nabla function $\mathrm{C_N}$. These functions are

explained in more detail after the explanation of the deduction rules. In order to simplify the deduction rules, the notation $\sigma\ldots$ is used as a shorthand for $\sigma, \Gamma, \mathcal{D}$.

$$\frac{}{\langle \texttt{skip}, \sigma\ldots\rangle \xrightarrow{\tau} \langle \checkmark, \sigma\ldots\rangle}\ 1 \qquad \frac{\forall_{0\leq i,j\leq k} : x_i = x_j \Rightarrow i = j}{\langle \hat{x} := \hat{e}, \sigma\ldots\rangle \xrightarrow{\sigma_{\hat{x},\hat{e}}} \langle \checkmark, \sigma_{\hat{x},\hat{e}}\ldots\rangle}\ 2$$

$$\frac{}{\langle m!e, \sigma\ldots\rangle \xrightarrow{isa(m,\bar{\sigma}(e))} \langle \checkmark, \sigma\ldots\rangle}\ 3 \qquad \frac{}{\langle m!e, \sigma\ldots\rangle \xrightarrow{\varsigma,t} \langle m!e, \sigma_{\varsigma t}\ldots\rangle}\ 4$$

$$\frac{}{\langle m?x, \sigma\ldots\rangle \xrightarrow{ira(m,x,c)} \langle \checkmark, \sigma[c/x]\ldots\rangle}\ 5 \qquad \frac{}{\langle m?x, \sigma\ldots\rangle \xrightarrow{\varsigma,t} \langle m?x, \sigma_{\varsigma t}\ldots\rangle}\ 6$$

$$\frac{\varsigma \in \Omega(\sigma, \Gamma, D, eq, true, t)}{\langle eq, \sigma\ldots\rangle \xrightarrow{\varsigma,t} \langle eq, \sigma_{\varsigma t}\ldots\rangle}\ 7 \qquad \frac{\sigma_i \in \Omega_{\mathrm{i}}(\sigma, \mathcal{T}_{\mathrm{U}}(us), \mathcal{T}_{\mathrm{E}}(eqs))}{\langle us : eqs, \sigma\ldots\rangle \xrightarrow{\sigma_i} \langle \checkmark, \sigma_i\ldots\rangle}\ 8$$

$$\frac{0 \leq t < \bar{\sigma}(e_{num})}{\langle \Delta e_{num}, \sigma\ldots\rangle \xrightarrow{\varsigma,t} \langle \Delta\bar{\sigma}(e_{num}) - t, \sigma\ldots\rangle}\ 9$$

$$\frac{}{\langle \Delta e_{num}, \sigma\ldots\rangle \xrightarrow{\varsigma,\bar{\sigma}(e_{num})} \langle \texttt{skip}, \sigma\ldots\rangle}\ 10$$

$$\frac{\sigma \models \neg b_n}{\langle \nabla b_n, \sigma\ldots\rangle \xrightarrow{\tau} \langle \checkmark, \sigma\ldots\rangle}\ 11 \qquad \frac{\varsigma \in \Omega(\sigma, \Gamma, \emptyset, \emptyset, b_n, t)}{\langle \nabla b_n, \sigma\ldots\rangle \xrightarrow{\varsigma,t} \langle \nabla b_n, \sigma_{\varsigma t}\ldots\rangle}\ 12$$

Rule 1 states that `skip` can perform the τ action. Rule 2 states that $\hat{x} := \hat{e}$ can perform the σ_a action if all elements in $\hat{x}$ are distinct, where $i, j, k \in \mathbb{N}$ and $\sigma_{\hat{x},\hat{e}}$ is defined as follows:

$$\sigma_{\hat{x},\hat{e}}(y) = (\sigma[[\bar{\sigma}(e_0)/x_0], \ldots, [\bar{\sigma}(e_k)/x_k]])(y) \quad = \quad \begin{cases} \bar{\sigma}(e_0) & \text{if} \quad y = x_0, \\ \vdots & \vdots \\ \bar{\sigma}(e_k) & y = x_k \\ \bar{\sigma}(y) & \text{otherwise} \end{cases}.$$

where $y \in V$, $\bar{\sigma}(e_0)$ denotes the evaluation of expression e_0 in state σ, and $\sigma[\bar{\sigma}(e_0)/x_0]$ denotes the substitution of the value $\bar{\sigma}(e_0)$ for the variable x_0 in state σ (likewise for all elements in $\hat{x}$ and $\hat{e}$). Rules 3 and 5 state that $m!e$ and $m?x$ can perform their corresponding actions. Note that the send process *sets* associates a channel label with a value and the receive process *gets* the value which is associated with the channel label. Rules 4 and 6 state that $m!e$ and $m?x$ can perform any time transition ς of duration t, where $\sigma_{\varsigma t} \in V \mapsto \Sigma$ is for all $x \in V$ defined as $\varsigma(t)(x)$ if $x \in \mathrm{dom}(\varsigma)$ and $\bar{\sigma}(x)$ otherwise. Rule 7 states that the function equation *eq* can perform a time transition of duration $t \geq 0$ for all solutions ς of the function equations *eq*. The solution ς is a function from time ($\mathrm{dom}(\varsigma) = [0\ldots t]$) to the value for the continuous variables in Γ. Rule 8 states that the instantaneous equation process can perform a σ_a action in case

there is at least a solution for the instantaneous equations. The functions $\mathcal{T}_{\mathrm{U}}$ and $\mathcal{T}_{\mathrm{E}}$ collect the unknowns for instantaneous equations and instantaneous equations respectively. Rule 9 states that Δe_{num} enables a process to delay for t time units if $0 \leq t < \bar{\sigma}(e_{num})$ holds. Rule 10 states that Δe_{num} transforms a process into `skip` at the delay time units of $\bar{\sigma}(e_{num})$. Rule 11 states that ∇b_n can terminate successfully if the condition b_n is not satisfied in state σ ($\sigma \models \neg b_n$). Rule 12 states that the nabla process ∇b_n can perform a time transition in case there is at least one solution ς which satisfies b_n during this time transition.

$$\frac{\sigma \models b,\ \langle p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle}{\langle b \to p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle}\ 13 \qquad \frac{\sigma \models b,\ \langle p, \sigma \ldots\rangle \xrightarrow{a} \langle p', \sigma' \ldots\rangle}{\langle b \to p, \sigma \ldots\rangle \xrightarrow{a} \langle p', \sigma' \ldots\rangle}\ 14$$

$$\frac{\sigma \models b,\ \mathrm{V}_\chi(b) \cap \Gamma = \emptyset,\ \langle p, \sigma \ldots\rangle \xrightarrow{\varsigma,t} \langle p', \sigma' \ldots\rangle}{\langle b \to p, \sigma \ldots\rangle \xrightarrow{\varsigma,t} \langle b \to p', \sigma' \ldots\rangle}\ 15$$

$$\frac{\sigma \models \neg b,\ \mathrm{V}_\chi(b) \cap \Gamma = \emptyset}{\langle b \to p, \sigma \ldots\rangle \xrightarrow{\varsigma,t} \langle b \to p, \sigma_{\varsigma t} \ldots\rangle}\ 16$$

Rules 13 and 14 state that a guarded process can perform an action transition if the guard evaluates to *true* and if the process argument can perform that transition. Rule 15 states that a guarded process can perform a time transition in case the guard evaluates to *true*, and there are no continuous variables used in the guard, and its process argument can perform that time transition. The function $\mathrm{V}_\chi \in B \to \mathcal{P}(V)$ extracts the variables from an expression; e.g. $\mathrm{V}_\chi(x > 2y^2) = \{x, y\}$. Rule 16 states that a guarded process can perform a time transition in case the guard evaluates to *false* and that there are no continuous variables used in the guard.

$$\frac{\langle p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle}{\langle p; q, \sigma \ldots\rangle \xrightarrow{a} \langle q, \sigma' \ldots\rangle}\ 17 \qquad \frac{\langle p, \sigma \ldots\rangle \xrightarrow{z} \langle p', \sigma' \ldots\rangle}{\langle p; q, \sigma \ldots\rangle \xrightarrow{z} \langle p'; q, \sigma' \ldots\rangle}\ 18$$

Rules 17 and 18 state that if process p can perform a transition, then the sequential composition $p; q$ can perform that transition also. The transition $\xrightarrow{z}$ is used as a shorthand for the transitions $\xrightarrow{a}$ and $\xrightarrow{\varsigma,t}$.

$$\frac{\langle p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle}{\langle p \parallel q, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle,\ \langle q \parallel p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle}\ 19$$

$$\frac{\langle p, \sigma \ldots\rangle \xrightarrow{a} \langle p', \sigma' \ldots\rangle}{\langle p \parallel q, \sigma \ldots\rangle \xrightarrow{a} \langle p', \sigma' \ldots\rangle,\ \langle q \parallel p, \sigma \ldots\rangle \xrightarrow{a} \langle p', \sigma' \ldots\rangle}\ 20$$

$$\frac{\langle p, \sigma \ldots\rangle \xrightarrow{\varsigma_p, t} \langle p', \sigma' \ldots\rangle,\ \langle q, \sigma \ldots\rangle \xrightarrow{\varsigma_q, t} \langle q', \sigma' \ldots\rangle,\ \forall_{x \in \Gamma} : \varsigma_p(x) = \varsigma_q(x)}{\langle p \parallel q, \sigma \ldots\rangle \xrightarrow{(\varsigma_p\ \cup\ \varsigma_q), t} \langle \mathrm{C_N}(p' \parallel q', \sigma' \ldots), \sigma' \ldots\rangle}\ 21$$

Rules 19 and 20 state that an alternative composition of two processes p and q can perform an action transition in case that either p or q can perform that action transition. Rule 21 states that if process p and q can perform a time transition with distinct solutions (i.e ς_p, t and ς_q, t are different) and if the solutions ς_p and ς_q result the same for all variables in Γ, then the alternative composition $p \parallel q$ can perform a time transition with the union solutions between solutions ς_p, t and ς_q, t (i.e. $(\varsigma_p\ \cup\ \varsigma_q), t$). After the time transition, the function $\mathrm{C_N}$ is applied to the process term.

Example

Consider the following example: $x := 1; (\nabla x \leq 5 \parallel \dot{x} = 1)$, for $\Gamma = \{x\}$. The assignment $x := 1$ can be executed using Rule 2. Using Rules 21, 12, and 7, the alternative composition $\nabla x \leq 5 \parallel \dot{x} = 1$ can perform a time transition for the union solutions between the solutions satisfy the nabla statement $\nabla x \leq 5$ and the solutions satisfy the equation $\dot{x} = 1$. In this example, the alternative composition $\nabla x \leq 5 \parallel \dot{x} = 1$ can perform time transition for different time steps ($t \in [0 \ldots 5]$) as long as the nabla condition is satisfied (i.e. $x \leq 5$ holds). In case that no subsequent time step $t > 0$ exists for which the equation and the nabla condition are both satisfied, the nabla process term is replaced by the `skip` process term. Thus, the process term becomes $\texttt{skip} \parallel \dot{x} = 1$. Using Rules 19 and 1, the process term successfully terminates.

$$\frac{\begin{array}{c}\sigma' \in \gamma_{\mathrm{E}}(p \parallel q, \sigma \ldots),\\ \langle p, \sigma' \ldots\rangle \xrightarrow{isa(m,\mathrm{c})} \langle \checkmark, \sigma' \ldots\rangle,\ \langle q, \sigma' \ldots\rangle \xrightarrow{ira(m,x)} \langle \checkmark, \sigma' \ldots\rangle\end{array}}{\begin{array}{c}\langle p \parallel q, \sigma \ldots\rangle \xrightarrow{ca(m,x,\mathrm{c})} \langle \checkmark, \sigma'[\mathrm{c}/x] \ldots\rangle,\\ \langle q \parallel p, \sigma \ldots\rangle \xrightarrow{ca(m,x,\mathrm{c})} \langle \checkmark, \sigma'[\mathrm{c}/x] \ldots\rangle\end{array}}\ 22$$

$$\frac{\begin{array}{c}\sigma' \in \gamma_{\mathrm{E}}(p \parallel q, \sigma \ldots),\\ \langle p, \sigma' \ldots\rangle \xrightarrow{isa(m,\mathrm{c})} \langle \checkmark, \sigma' \ldots\rangle,\ \langle q, \sigma' \ldots\rangle \xrightarrow{ira(m,x)} \langle q', \sigma' \ldots\rangle\end{array}}{\begin{array}{c}\langle p \parallel q, \sigma \ldots\rangle \xrightarrow{ca(m,x,\mathrm{c})} \langle q', \sigma'[\mathrm{c}/x] \ldots\rangle,\\ \langle q \parallel p, \sigma \ldots\rangle \xrightarrow{ca(m,x,\mathrm{c})} \langle q', \sigma'[\mathrm{c}/x] \ldots\rangle\end{array}}\ 23$$

$$\frac{\begin{array}{c}\sigma' \in \gamma_{\mathrm{E}}(p \parallel q, \sigma \ldots),\\ \langle p, \sigma' \ldots\rangle \xrightarrow{isa(m,\mathrm{c})} \langle p', \sigma' \ldots\rangle,\ \langle q, \sigma' \ldots\rangle \xrightarrow{ira(m,x)} \langle \checkmark, \sigma' \ldots\rangle\end{array}}{\begin{array}{c}\langle p \parallel q, \sigma \ldots\rangle \xrightarrow{ca(m,x,\mathrm{c})} \langle p', \sigma'[\mathrm{c}/x] \ldots\rangle,\\ \langle q \parallel p, \sigma \ldots\rangle \xrightarrow{ca(m,x,\mathrm{c})} \langle p', \sigma'[\mathrm{c}/x] \ldots\rangle\end{array}}\ 24$$

$$\frac{\begin{array}{c}\sigma' \in \gamma_{\mathrm{E}}(p \parallel q, \sigma \ldots),\\ \langle p, \sigma' \ldots\rangle \xrightarrow{isa(m,\mathrm{c})} \langle p', \sigma' \ldots\rangle,\ \langle q, \sigma' \ldots\rangle \xrightarrow{ira(m,x)} \langle q', \sigma' \ldots\rangle\end{array}}{\begin{array}{c}\langle p \parallel q, \sigma \ldots\rangle \xrightarrow{ca(m,x,\mathrm{c})} \langle p' \parallel q', \sigma'[\mathrm{c}/x] \ldots\rangle,\\ \langle q \parallel p, \sigma \ldots\rangle \xrightarrow{ca(m,x,\mathrm{c})} \langle q' \parallel p', \sigma'[\mathrm{c}/x] \ldots\rangle\end{array}}\ 25$$

$$\frac{\sigma' \in \gamma_{\mathrm{E}}(p \parallel q, \sigma \ldots),\ \langle p, \sigma', \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma'' \ldots\rangle}{\langle p \parallel q, \sigma \ldots\rangle \xrightarrow{a} \langle q, \sigma'' \ldots\rangle,\ \langle q \parallel p, \sigma \ldots\rangle \xrightarrow{a} \langle q, \sigma'' \ldots\rangle}\ 26$$

$$\frac{\sigma' \in \gamma_{\mathrm{E}}(p \parallel q, \sigma \ldots),\ \langle p, \sigma' \ldots\rangle \xrightarrow{a} \langle p', \sigma'' \ldots\rangle}{\langle p \parallel q, \sigma \ldots\rangle \xrightarrow{a} \langle p' \parallel q, \sigma'' \ldots\rangle,\ \langle q \parallel p, \sigma \ldots\rangle \xrightarrow{a} \langle q \parallel p', \sigma'' \ldots\rangle}\ 27$$

$$\frac{\begin{array}{c}\langle p, \sigma \ldots\rangle \xrightarrow{\varsigma_p, t} \langle p', \sigma' \ldots\rangle,\\ \langle q, \sigma \ldots\rangle \xrightarrow{\varsigma_q, t} \langle q', \sigma' \ldots\rangle,\ \forall_{x \in \Gamma} : \varsigma_p(x) = \varsigma_q(x)\end{array}}{\langle p \parallel q, \sigma \ldots\rangle \xrightarrow{(\varsigma_p\ \cup\ \varsigma_q), t} \langle \mathrm{C_N}(p' \parallel q', \sigma' \ldots), \sigma' \ldots\rangle}\ 28$$

Rule 22 states that if process p and q can perform an action transition from a consistent state and they can also perform matching send and receive actions, then the parallel compositions $p \parallel q$ and $q \parallel p$ can perform a communication action and perform that action transition as well, where $\mathrm{c} \in \Sigma$. Rule 23 states that if process p can perform an action transition from a consistent state which can terminate successfully, and process q can perform an action transition to another process term q' from the same consistent state, in addition, process p and q can perform matching send and receive actions, then the parallel compositions $p \parallel q$ and $q \parallel p$ can perform a communication action and both convey to process term q'. Rule 24 is similar to Rule 23. If process p and q can perform an action

transition from a consistent state and they can also perform matching send and receive actions, then the parallel compositions $p \parallel q$ and $q \parallel p$ can perform a communication action as defined by Rule 25. Rules 26 and 27 state that if process p can perform an action transition from a consistent state, then both the parallel compositions $p \parallel q$ and $q \parallel p$ can perform that action transition as well, such that the terminated process is removed. Rule 28 states that if process p and q can perform a time transition with distinct solutions (i.e ς_p, t and ς_q, t are different) and if the solutions ς_p and ς_q result the same for all variables in Γ, then the parallel composition $p \parallel q$ can perform a time transition with the union solutions between solutions ς_p, t and ς_q, t (i.e. $(\varsigma_p \cup \varsigma_q), t$). After the time transition, the function $\mathrm{C_N}$ is applied to the process term.

Example

Consider the following example: $y := 0; n := 1; (n := y \parallel y = 2n)$, for $\Gamma = \{y\}$. The assignment $y := 0$ can be executed using Rule 2. The same holds for the assignment $n := 1$. After that, the process term equals $n := y \parallel y = 2n$, and the state consists of the following valuations: $\{y \mapsto 0, n \mapsto 1\}$. The parallel composition $n := y \parallel y = 2n$ can perform an action in case that $n := y$ can perform that action from a consistent state using Rule 26. The consistent state is $\{y \mapsto 2, n \mapsto 1\}$; after the assignment $n := y$, the process term equals $y = 2n$, and the state is $\{y \mapsto 2, n \mapsto 2\}$. This process term can perform time transitions using Rule 7.

$$\frac{\langle p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle}{\langle *p, \sigma \ldots\rangle \xrightarrow{a} \langle *p, \sigma' \ldots\rangle}\ 29 \qquad \frac{\langle p, \sigma \ldots\rangle \xrightarrow{z} \langle p', \sigma' \ldots\rangle}{\langle *p, \sigma \ldots\rangle \xrightarrow{z} \langle p'; *p, \sigma' \ldots\rangle}\ 30$$

Rule 29 states that if process p can perform an action transition, then also the repetition $*p$ can perform that action transition. Rule 30 states that if process p can perform a transition to another process term, then also the repetition $*p$ can perform that transition.

$$\frac{\begin{array}{c}\langle p, \sigma \cup \sigma_s, \Gamma \cup \Gamma_s, \mathcal{D}\rangle \xrightarrow{a} \langle \checkmark, \sigma', \Gamma \cup \Gamma_s, \mathcal{D}\rangle \\ \hline \langle [\![\ \sigma_s, \Gamma_s |\ p\ ]\!], \sigma, \Gamma, \mathcal{D}\rangle \xrightarrow{a} \langle \checkmark, \sigma' \upharpoonright dom(\sigma), \Gamma, \mathcal{D}\rangle\end{array}}{\begin{array}{c}\langle p, \sigma \cup \sigma_s, \Gamma \cup \Gamma_s, \mathcal{D}\rangle \xrightarrow{z} \langle p', \sigma', \Gamma \cup \Gamma_s, \mathcal{D}\rangle \\ \hline \langle [\![\ \sigma_s, \Gamma_s |\ p\ ]\!], \sigma, \Gamma, \mathcal{D}\rangle \xrightarrow{z} \langle [\![\ \sigma' \upharpoonright dom(\sigma_s), \Gamma_s |\ p'\ ]\!], \sigma' \upharpoonright dom(\sigma), \Gamma, \mathcal{D}\rangle\end{array}}\ 31\ 32$$

Rules 31 and 32 state that the state process can perform a transition if its process argument can perform that transition, where $\upharpoonright$ denotes the restrict operator and is formally defined as follows:

$$\begin{aligned} f : X &\longrightarrow Y \\ (f \upharpoonright S) : X &\longrightarrow Y \\ (f \upharpoonright S)(v) &= \begin{cases} f(v) & \text{if} \quad v \in S, \\ \bot & \text{otherwise} \end{cases} \end{aligned}$$

where f denotes a function, X, Y and $S \subseteq X$ denote arbitrary sets. Furthermore, the syntax restrictions $dom(\sigma_s) \cap dom(\sigma) = \emptyset$ and $\Gamma_s \cap \Gamma = \emptyset$ are assumed.

$$\frac{\langle p, \sigma, \Gamma, \mathcal{D} \cup \{x\}\rangle \xrightarrow{a} \langle \checkmark, \sigma', \Gamma, \mathcal{D} \cup \{x\}\rangle}{\langle x :: p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle} \; 33$$

$$\frac{\langle p, \sigma, \Gamma, \mathcal{D} \cup \{x\}\rangle \xrightarrow{z} \langle p', \sigma', \Gamma, \mathcal{D} \cup \{x\}\rangle}{\langle x :: p, \sigma \ldots\rangle \xrightarrow{z} \langle x :: p', \sigma' \ldots\rangle} \; 34$$

Rules 33 and 34 state that a process with a dependent differential variable x can perform a transition if its process argument can perform that transition with x added to D.

$$\frac{\langle p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle,\ a \notin A}{\langle \partial_A(p), \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle} \; 35 \qquad \frac{\langle p, \sigma \ldots\rangle \xrightarrow{z} \langle p', \sigma' \ldots\rangle,\ z \notin A}{\langle \partial_A(p), \sigma \ldots\rangle \xrightarrow{z} \langle \partial_A(p'), \sigma' \ldots\rangle} \; 36$$

Rules 35 and 36 state that the encapsulation process can perform a transition if its process argument can perform such transition and if that transition is not in A, which denotes the set of transitions to be encapsulated.

$$\frac{\langle p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle}{\langle \pi(p), \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle} \; 37 \qquad \frac{\langle p, \sigma \ldots\rangle \xrightarrow{a} \langle p', \sigma' \ldots\rangle}{\langle \pi(p), \sigma \ldots\rangle \xrightarrow{a} \langle \pi(p'), \sigma' \ldots\rangle} \; 38$$

$$\frac{\forall a \in A,\ \langle p, \sigma \ldots\rangle \xnrightarrow{a},\ \langle p, \sigma \ldots\rangle \xrightarrow{\varsigma, t} \langle p', \sigma' \ldots\rangle}{\langle \pi(p), \sigma \ldots\rangle \xrightarrow{\varsigma, t} \langle \pi(p'), \sigma' \ldots\rangle} \; 39$$

Rules 37 and 38 state that if process p can perform an action transition, then also the maximal progress $\pi(p)$ can perform that action transition. A maximal progress process only performs a time transition if its process argument p can perform that time transition and if p cannot perform actions as defined by Rule 39.

$$\frac{\langle p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle,\ a \in A}{\langle \tau_A(p), \sigma \ldots\rangle \xrightarrow{\tau} \langle \checkmark, \sigma' \ldots\rangle} \; 40 \qquad \frac{\langle p, \sigma \ldots\rangle \xrightarrow{z} \langle p', \sigma' \ldots\rangle,\ z \in A}{\langle \tau_A(p), \sigma \ldots\rangle \xrightarrow{\tau} \langle \tau_A(p'), \sigma' \ldots\rangle} \; 41$$

$$\frac{\langle p, \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle,\ a \notin A}{\langle \tau_A(p), \sigma \ldots\rangle \xrightarrow{a} \langle \checkmark, \sigma' \ldots\rangle} \; 42 \qquad \frac{\langle p, \sigma \ldots\rangle \xrightarrow{z} \langle p', \sigma' \ldots\rangle,\ z \notin A}{\langle \tau_A(p), \sigma \ldots\rangle \xrightarrow{z} \langle \tau_A(p'), \sigma' \ldots\rangle} \; 43$$

Rules 40 and 41 state that if the process argument can perform the transition that is in A which is the set of transitions to be abstracted from, then the abstraction process can perform the τ action. Rules 42 and 43 state that if the process argument can perform a transition and this transition is not in A, then the abstraction process can also perform that transition.

Extract Function

The function $\xi \in P \times (V \mapsto \Sigma) \to \mathcal{P}(EQ)$ extracts the equations from a process term in a state. It is used in the functions $\mathrm{C_N}$ and γ_{E}.

$$
\begin{array}{lcl}
\xi(p \parallel q, \sigma) & = & \xi(p,\sigma) \cup \xi(q,\sigma) \\
\xi(p \,[\!]\, q, \sigma) & = & \xi(p,\sigma) \cup \xi(q,\sigma) \\
\xi(b \to p, \sigma) & = & \begin{cases} \xi(p,\sigma) & \sigma \models b \\ \emptyset & \sigma \models \neg b \end{cases} \\
\xi(p; q, \sigma) & = & \xi(p,\sigma) \\
\xi(*p, \sigma) & = & \xi(p,\sigma) \\
\xi(eq, \sigma) & = & eq \\
\xi(x :: p, \sigma) & = & \xi(p,\sigma) \\
otherwise & = & \emptyset
\end{array}
$$

Translation Function

The functions $\mathcal{T}_{\mathrm{U}} \in US \to \mathcal{P}(V)$ and $\mathcal{T}_{\mathrm{E}} \in EQS \to \mathcal{P}(EQS)$ extracts the unknowns for the instantaneous equations and instantaneous equations respectively, and are formally defined as follows:

$$
\begin{array}{lcllcl}
\mathcal{T}_{\mathrm{U}}(us_0, us_1) & = & \mathcal{T}_{\mathrm{U}}(us_0) \cup \mathcal{T}_{\mathrm{U}}(us_1) & \mathcal{T}_{\mathrm{E}}(eq_0, eq_1) & = & \mathcal{T}_{\mathrm{E}}(eqs_0) \cup \mathcal{T}_{\mathrm{E}}(eqs_1) \\
\mathcal{T}_{\mathrm{U}}(u) & = & \{u\} & \mathcal{T}_{\mathrm{E}}(eqs) & = & \{eqs\}
\end{array}
$$

where $us_0, us_1 \in US, \quad eqs_0, eqs_1, eqs \in EQS, \quad u \in V$

Clean-up Nabla Function

After a time transition of an alternative composition or a parallel composition of processes, the function $\mathrm{C_N} \in P \times (V \mapsto \Sigma) \times \mathcal{P}(V) \to P$ checks whether the nabla process terms in a process term can still perform a time transition of duration $t > 0$. If this transition is not possible for some nabla process terms, these nabla process terms are substituted by the `skip` process term, otherwise the nabla process term is returned. The function $\mathrm{C_N}$ is formally defined as $\mathrm{C_N}(p, \sigma, \Gamma, \mathcal{D}) = \mathrm{C'_N}(p, \sigma, \Gamma, \mathcal{D}, \mathcal{E})$, where $\mathcal{E} = \xi(p, \sigma)$ and the function $\mathrm{C'_N} \in P \times (V \mapsto \Sigma) \times \mathcal{P}(V) \times \mathcal{P}(V) \times \mathcal{P}(EQ) \to P$ is formally defined as

$$
\begin{array}{lcl}
\mathrm{C'_N}(p \parallel q, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & \mathrm{C'_N}(p, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) \parallel \mathrm{C'_N}(q, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) \\
\mathrm{C'_N}(p \,[\!]\, q, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & \mathrm{C'_N}(p, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) \,[\!]\, \mathrm{C'_N}(q, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) \\
\mathrm{C'_N}(b \to p, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & \begin{cases} b \to \mathrm{C'_N}(p, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & \sigma \models b \\ b \to p & \sigma \models \neg b \end{cases} \\
\mathrm{C'_N}(p; q, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & \mathrm{C'_N}(p, \sigma, \Gamma, \mathcal{D}, \mathcal{E}); q \\
\mathrm{C'_N}(x :: p, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & x :: p \\
\mathrm{C'_N}(\mathtt{skip}, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & \mathtt{skip} \\
\mathrm{C'_N}(\hat{x} := \hat{e}, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & \hat{x} := \hat{e} \\
\mathrm{C'_N}(m!e, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & m!e \\
\mathrm{C'_N}(m!x, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & m!x \\
\mathrm{C'_N}(eq, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & eq \\
\mathrm{C'_N}(us : eqs, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & us : eqs \\
\mathrm{C'_N}(\Delta e, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & \Delta e \\
\mathrm{C'_N}(\nabla b_n, \sigma, \Gamma, \mathcal{D}, \mathcal{E}) & = & \begin{cases} \nabla b_n & \exists t > 0 : \Omega(\sigma, \Gamma, \mathcal{D}, \mathcal{E}, b_n, t) \neq \emptyset \\ \mathtt{skip} & \text{otherwise} \end{cases}
\end{array}
$$

Consistency Function

The function $\gamma_{\mathrm{E}} \in P \times (V \mapsto \Sigma) \times \mathcal{P}(V) \times \mathcal{P}(V) \to \mathcal{P}(V \mapsto \Sigma)$ returns a set of states which are consistent with the equations $\mathcal{E}$ in a process term: $\gamma_{\mathrm{E}}(p, \sigma, \Gamma, \mathcal{D}) = \{\sigma_{\varsigma 0} \mid \varsigma \in \Omega(\sigma, \Gamma, \mathcal{D}, \mathcal{E}, \mathit{true}, 0)\}$, where $\mathcal{E} = \xi(p, \sigma)$, and $\sigma_{\varsigma 0} \in V \mapsto \Sigma$ is defined for all $x \in V$ as $\varsigma(0)(x)$ if $x \in \mathrm{dom}(\varsigma)$, and $\sigma(x)$ otherwise.

Solution Function

The function $\Omega \in (V \mapsto \Sigma) \times \mathcal{P}(V) \times \mathcal{P}(V) \times EQ \times BN \times T \to \mathcal{P}(V \mapsto (T \mapsto \Sigma))$ returns a set of solution functions for the continuous variables in Γ. Since the equations in $\chi_{\sigma_{\mathrm{h}}}$ are function equations, all variables are interpreted as functions. The discrete variables are interpreted as constant functions ($\forall i \in \mathrm{dom}(\varsigma') \setminus \Gamma, 0 \leq t \leq d : \varsigma'(i)(t) = \sigma(i)$). The initial conditions of the independent differential variables x ($x \in \mathrm{D}_\chi(eq) \setminus \mathcal{D}$) are specified by $\varsigma'(x)(0) = \sigma(x)$. The solution functions satisfy the equation ($\varsigma' \models eq$). Furthermore, the boolean expression of a nabla process must be satisfied by the solution functions at all times ($\forall 0 \leq t \leq d : \varsigma'_t \models b_n$) as well. Formally, the function Ω is defined as:
$\Omega(\sigma, \Gamma, D, eq, b_n, d) =$

$$
\begin{array}{lll}
\{ & \varsigma & \\
\mid & \exists \varsigma' \in V \mapsto (T \mapsto \Sigma) : & \\
 & \quad (\quad \mathrm{dom}(\varsigma') = \mathrm{dom}(\sigma) & \\
 & \quad , \quad \forall i \in \mathrm{dom}(\varsigma') : & \mathrm{dom}(\varsigma'(i)) = [0..d] \\
 & \quad , \quad \forall i \in \mathrm{dom}(\varsigma') \setminus \Gamma, 0 \leq t \leq d : & \varsigma'(i)(t) = \sigma(i) \\
 & \quad , \quad \forall x \in \mathrm{D}_\chi(eq) \setminus \mathcal{D} : & \varsigma'(x)(0) = \sigma(x) \\
 & \quad , \quad \varsigma' \models eq & \\
 & \quad , \quad \forall 0 \leq t \leq d : & \varsigma'_t \models b_n \\
 & \quad , \quad \mathrm{dom}(\varsigma) = \Gamma & \\
 & \quad , \quad \forall x \in \mathrm{dom}(\varsigma) : & \varsigma(x) \text{ piecewise differentiable,} \\
 & & \varsigma(x) = \varsigma'(x) \\
 & \quad) & \\
\}
\end{array}
$$

where $\mathrm{D}_\chi \in \mathcal{P}(EQ) \to \mathcal{P}(V)$, and $\mathrm{D}_\chi(eq)$ extracts the set of differential variables used in eq (e.g. $\mathrm{D}_\chi(\dot{x} = 1, y = \dot{z}) = \{x, z\}$), and where $\varsigma'_t : V \mapsto \Sigma$ is, for all $x \in \mathrm{dom}(\varsigma')$, defined as $\varsigma'_t(x) = \varsigma'(x)(t)$.

Solution Function for instantaneous equations

The function $\Omega_{\mathrm{i}} \in (V \mapsto \Sigma) \times \mathcal{P}(V) \times \mathcal{P}(EQ) \to \mathcal{P}(V \mapsto \Sigma)$ returns a set of state solutions of the instantaneous equations for the variables in $\mathcal{U}$. Formally, the function Ω_{i} is defined as:
$\Omega_{\mathrm{i}}(\sigma, \mathcal{U}, \mathcal{E}) =$

$$
\begin{array}{lll}
\{ \quad \sigma_i | \sigma' \in V \mapsto \Sigma : & & \\
\quad (\quad \mathrm{dom}(\sigma') = \mathrm{dom}(\sigma) & & \\
\quad , \quad \forall x \notin \mathcal{U} : & \sigma'(x) = \sigma(x) & \\
\quad , \quad \forall eq \in \mathcal{E} : & \sigma' \models eq & \\
\quad , \quad \mathrm{dom}(\sigma_i) = \mathcal{U} & & \\
\quad , \quad \forall x \in \mathrm{dom}(\sigma_i) : & \sigma_i(x) = \sigma'(x) & \\
\quad) & & \\
\}
\end{array}
$$

Example

Higher index systems of equations are equations where differential variables are related, and cannot be initialized independently. An example is $\dot{x} = z, \dot{y} = -z, y = x$, for $\Gamma = \{x, y, z\}$. There are two approaches of providing initial conditions for higher index systems. First, the modeler can explicitly provide consistent initial conditions; e.g. $x := (x + y)/2; y := x; (\dot{x} = z, \dot{y} = -z, y = x)$. If the initial state in this model is $\{x \mapsto 1, y \mapsto 3\}$, both variables will be initialized to 2: $\{x \mapsto 2, y \mapsto 2\}$. Second, the modeler can specify which of the variables can be independently initialized and which variables are dependent by using dependent differential variable operator; e.g. $x :: \dot{x} = z, \dot{y} = -z, y = x$ for $\Gamma = \{x, y, z\}$ and $\mathcal{D} = \{x\}$. If the initial state in this model is $\{x \mapsto 1, y \mapsto 3\}$, the state will be made consistent by using the initial value of y and adjusting the corresponding value of x in such a way that the equations become consistent. In this approach, the consistent state becomes $\{x \mapsto 3, y \mapsto 3\}$.

4 Dry Friction Example

A somewhat more complex example deals with modeling of dry friction as shown in Figure 1. A driving force F_d is applied to a body on a flat surface with

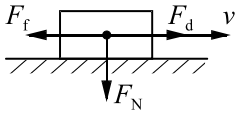

Figure 1: Dry friction.

frictional force F_f. In the model below, variable s represents the mode of the process; it can have the values "neg", "stop", and "pos". A corresponding hybrid automaton specification would have corresponding locations "neg", "stop", and "pos". These modes/locations correspond with velocities $v \leq 0$, $v = 0$, and $v \geq 0$, respectively of the body. In the mode "stop", the velocity v equals 0. The mode "stop", is maintained for as long as the driving force satisfies the conditions $F_\mathrm{d} \leq \mu_0 F_\mathrm{N}$ and $F_\mathrm{d} \geq -\mu_0 F_\mathrm{N}$. If either of these conditions can no longer be satisfied, the mode becomes "pos" or "neg", respectively. The mode

"pos", is maintained for as long as the condition $v \geq 0$ ($\nabla v \geq 0$) is satisfied. In this mode, frictional force F_f equals μF_N. When this condition can no longer be satisfied, the value of s becomes "stop". The condition $v \geq 0$ can no longer be satisfied when no time step can be taken such that v remains ≥ 0. This implies that at the time of the switch, v equals 0. The set of continuous variables Γ equals $\{x, v\}$; F_d represents some user defined function; F_N, m, and μ_0 are constants.

$$
\begin{array}{lll}
\dot{x} = v & \| & s = \text{"stop"} \quad \rightarrow v = 0 \\
& \| & s = \text{"pos"} \quad \rightarrow m\dot{v} = F_\mathrm{d} - \mu F_\mathrm{N} \\
& \| & s = \text{"neg"} \quad \rightarrow m\dot{v} = F_\mathrm{d} + \mu F_\mathrm{N} \\
& \| & *(\quad s = \text{"stop"} \quad \rightarrow \nabla F_\mathrm{d} \leq \mu_0 F_\mathrm{N};\ s := \text{"pos"} \\
& & [\!] \quad s = \text{"stop"} \quad \rightarrow \nabla F_\mathrm{d} \geq -\mu_0 F_\mathrm{N};\ s := \text{"neg"} \\
& & [\!] \quad s = \text{"pos"} \quad \rightarrow \nabla v \geq 0;\ s := \text{"stop"} \\
& & [\!] \quad s = \text{"neg"} \quad \rightarrow \nabla v \leq 0;\ s := \text{"stop"} \\
& &)
\end{array}
$$

A similar but shorter specification is

$$
\begin{array}{lll}
\dot{x} = v \| & *(& s = \text{"stop"} \quad \rightarrow (v = 0 \,[\!]\, \nabla F_\mathrm{d} \leq \mu_0 F_\mathrm{N};\ s := \text{"pos"}) \\
& [\!] & s = \text{"stop"} \quad \rightarrow (v = 0 \,[\!]\, \nabla F_\mathrm{d} \geq -\mu_0 F_\mathrm{N};\ s := \text{"neg"}) \\
& [\!] & s = \text{"pos"} \quad \rightarrow (m\dot{v} = F_\mathrm{d} - \mu F_\mathrm{N} \,[\!]\, \nabla v \geq 0;\ s := \text{"stop"}) \\
& [\!] & s = \text{"neg"} \quad \rightarrow (m\dot{v} = F_\mathrm{d} + \mu F_\mathrm{N} \,[\!]\, \nabla v \leq 0;\ s := \text{"stop"}) \\
&) &
\end{array}
$$

The only difference in the behavior of the two models is that in the last model no equations are valid directly after a new value has been assigned to s (e.g. $s :=$ "pos"). The only possible subsequent transition is a time transition, that causes a new equation ($m\dot{v} = F_\mathrm{d} - \mu F_\mathrm{N}$ if $s :=$ "pos") to be valid. The model can be simplified further as:

$$
\begin{array}{lll}
\dot{x} = v \| & *(& v = 0 \\
& [\!] & \nabla F_\mathrm{d} \leq \mu_0 F_\mathrm{N}; (m\dot{v} = F_\mathrm{d} - \mu F_\mathrm{N} \,[\!]\, \nabla v \geq 0) \\
& [\!] & \nabla F_\mathrm{d} \geq -\mu_0 F_\mathrm{N}; (m\dot{v} = F_\mathrm{d} + \mu F_\mathrm{N} \,[\!]\, \nabla v \leq 0) \\
&) &
\end{array}
$$

In this model, initially the process is in the mode "stop". The modes are no longer modeled explicitly.

5 Conclusions and Future Research

The semantics of the hybrid χ language has been formally specified using a relative small set of deduction rules and associated functions. The semantics is more complex than that of most other hybrid formalisms, because the χ language is primarily a modeling language and not a verification formalism; the language is highly expressive and can be used to specify a wide range of systems, including pure discrete-event systems, and higher index differential algebraic systems of equations. Future work entails the extension of the discrete-event χ verification tool to enable verification of hybrid models. Furthermore, the hybrid χ simulator will be redesigned to correspond to the new syntax and formal semantics, which implies a considerable improvement.

Acknowledgments

The authors would like to thank Pieter Cuijpers and Erjen Lefeber for stimulating and helpful discussions.

References

[1] D. A. van Beek and J. E. Rooda, "Languages and applications in hybrid modelling and simulation: Positioning of Chi," *Control Engineering Practice*, vol. 8, no. 1, pp. 81–91, 2000.

[2] D. A. van Beek, A. van den Ham, and J. E. Rooda, "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.

[3] J. Kleijn, M. Reniers, and J. Rooda, "A process algebra based verification of a production system," in *Second IEEE Conference on Formal Engineering Methods* (J. Staples, M. Hinchley, and S. Liu, eds.), (Brisbane, Australia), pp. 90–99, IEEE, Dec. 1998.

[4] V. Bos and J. Kleijn, *Formal Specification and Analysis of Industrial Systems*. PhD thesis, Eindhoven University of Technology, 2002.

[5] V. Bos and J. Kleijn, "Automatic verification of a manufacturing system," *Robotics and Computer Integrated Manufacturing*, vol. 17, no. 3, pp. 185–198, 2000.

[6] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," in *Theoretical Computer Science* 138, pp. 3–34, Springer, 1995.

[7] R. David and H. Alla, "On hybrid Petri nets," *Discrete Event Dynamic Systems: Theory & Applications*, vol. 11, no. 1-2, pp. 9–40, 2001.

[8] G. Fábián, D. A. van Beek, and J. E. Rooda, "Index reduction and discontinuity handling using substitute equations," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 7, no. 2, pp. 173–187, 2001.

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

[10] G. Plotkin, "A structural approach to operational semantics," Tech. Rep. DIAMI FN-19, Computer Science Department, Aarhus University, 1981.