Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2008-01

Syntax and Formal Semantics of Chi 2.0

D.A. van Beek A.T. Hofkamp
M.A. Reniers J.E. Rooda
R.R.H. Schiffelers

ISSN: 1872-1567

SE Report: Nr. 2008-01
Eindhoven, September 2008
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

This report defines the syntax and formal semantics of the Chi 2.0 formalism. The Chi formalism integrates concepts from dynamics and control theory with concepts from computer science, in particular from process algebra and hybrid automata. It combines a high expressivity and ease of modeling with a formal semantics.

The Chi language is defined by means of an abstract and a concrete syntax. The purpose of the abstract syntax is to allow a straightforward definition of the structured operational semantics (SOS), which associates a hybrid transition system with a Chi process. The Chi semantics is compositional, and bisimulation is a congruence for all operators. The concrete syntax offers modeling equivalents for the elements of the abstract syntax, and it introduces new syntax to ensure better readability and easier modeling. The meaning of the concrete syntax is defined by means of a mapping to the abstract syntax.

The Chi language provides among others discrete, continuous, and algebraic variables, and equation process terms for modeling differential algebraic equations (DAEs), including fully implicit or switched DAEs. Steady state initialization can be specified, and higher index DAEs in Chi are equivalent to the corresponding index 1 DAEs, obtained after differentiation of the hidden constraints. The invariant process term in Chi corresponds to invariants in hybrid automata.

The following operators are provided (among others): the parallel composition, alternative composition (choice), and sequential composition operators; and the recursion scope operator for modeling automata. The parallel composition operator allows shared variables, shared synchronizing and non-synchronizing action labels, and shared CSP channels for synchronous communication.

Two main ways of expressing urgency are provided: First, action labels and channels can be declared as urgent. Delaying is possible only if, and for as long as no urgent actions are enabled. Synchronizing actions are enabled only when the guards of all participating actions in a parallel composition are enabled. Second, urgency can be defined locally by means of the time can progress (tcp) process term, which allows delays for as long as the tcp predicate is true.

Scope operators are available for hierarchical modeling. They are used to declare local variables, local action labels, and local channels. Process definition and instantiation provide process re-use and encapsulation. Hybrid automata and networks of hybrid automata can easily be expressed in Chi. Since Chi is a process algebra, its operators can be arbitrarily combined, resulting in a high modeling flexibility. [1]

[1] Work partially done in the framework of the HYCON Network of Excellence, contract number FP6-IST-511368; as part of the ITEA project Twins 05004; and as a part of the DARWIN project at Philips Healthcare under the responsibility of the Embedded Systems Institute, partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.

Contents

1 Introduction

This report defines the abstract and concrete syntax of the Chi 2.0 formalism, and its formal semantics. The abstract syntax is used to define the semantics, and to enable formal reasoning about properties of Chi specifications. The abstract syntax is defined in Section 2, along with an informal description of the semantics. The formal semantics is defined in Section 3. The concrete syntax offers modeling equivalents for the elements of the abstract syntax, and it extends the syntax to ensure better readability and easier modeling. The meaning of the concrete syntax is defined by means of a mapping to the abstract syntax. Section 4 defines the subset of the concrete syntax for which the semantics is formally defined. For the complete concrete syntax of Chi, including among others many additional data types, we refer to the Chi 2.0 reference manual [11]. The syntax and the semantics are illustrated by means of small example models. For additional examples, we refer to the Lecture Notes 'Analysis of Hybrid Systems using Chi' [4].

1.1 What is new

The are many changes in Chi 2.0 with respect to the previous version of Chi as defined in [3] and [12]. This previous version of Chi is referred to as Chi 1.0. The major changes are:

Guards and conditional expressions In Chi 1.0, there was a guard operator $b \rightarrow p$, and there were no conditional expressions. The semantics of this guard operator was quite complex. Also, use of strict inequalities in the guard, such as in $x < 1 \rightarrow \dot{x} = 2 \parallel x \geq 1 \rightarrow \dot{x} = 3$, led to non-intuitive semantics.

In Chi 2.0, the guard is no longer an operator. The guard is now part of atomic action process terms. This considerably simplifies use and semantics of the guard. It is also more in line with the use of the guard in hybrid automata.

In Chi 2.0 conditional expressions have been introduced. Conditional expressions simplify modeling of switched linear systems, such as piecewise affine systems [22]. E.g. $\dot{x} = (x < 1 \rightarrow 2 \mid x \geq 1 \rightarrow 3)$.

Synchronizing action labels In Chi 1.0, all basic action labels were non-synchronizing. As a result, the transformation from networks of hybrid automata to Chi 1.0 was limited: either the parallel composition needed to be eliminated before transformation, or action synchronization was limited to 2-party synchronization (as in CSP channel synchronization [10]).

In Chi 2.0, basic action labels are non-synchronizing by default. They can be defined as synchronizing by means of the new synchronization operator. This means that a network of hybrid automata can be translated in a straightforward manner to a 'network' of parallel Chi 2.0 processes; thus preserving parallel composition.

Urgency In Chi 1.0, urgency was defined by means of the guard operator, the delay enabling operator, and the urgent communication operator. Atomic actions could not delay. Delay behavior could be added by means of the delay enabling operator. Delay behavior could be restricted by means of the urgent communication operator. Even though the definition of urgency in Chi 1.0 was already a step forward when compared to other languages, the semantics of the urgent communication operator was not intuitive when combined with the variable scope operator. Also, the urgent communication operator could not be eliminated, using 'process algebraic linearization' [1, 2], in all cases.

In Chi 2.0, atomic actions can perform arbitrary delays. Delay behavior cannot be added (the delay enabling operator has been removed); it can only be restricted. Local urgency is

defined by means of the new 'time can progress' process term, and global urgency is defined by declaring action labels and channels as either urgent or non-urgent. The semantics is intuitive in all cases, and process algebraic linearization is no longer restricted due to urgency.

Initialization In Chi 1.0, the signal emission operator was used, leading in some cases to unintuitive semantics. The behavior of this operator was different from the initial edges as used in hybrid automata, leading to difficulties in the transformation between hybrid automata and Chi models. Also, process algebraic linearization of Chi models that combine signal emission operators with variable scope operators was not always possible.

In Chi 2.0, the initialization operator is used instead of the signal emission operator. The behavior of the initialization operator is analogous to the behavior of initial edges in hybrid automata, and process algebraic linearization is no longer restricted due to modeling of initialization.

Hybrid automata Chi 1.0 extended hybrid automata in several aspects, but it lacked synchronizing actions and its primitives for modeling initialization were incompatible with hybrid automata.

Chi 2.0 extends hybrid automata in many ways, without introducing incompatibilities. Transformations from (networks of) hybrid automata to Chi 2.0 are trivial. Transformations from Chi 2.0 to hybrid automata are more difficult, due to the greater expressivity of Chi. By means of process algebraic linearization, it should in principle be possible to reduce Chi 2.0 models to a simplified form, allowing straightforward transformations to hybrid automata.

Combining send and receives with updates In Chi 1.0, send and receive actions on channels could not be atomically combined with assignments or other update expressions.

In Chi 2.0, send and receive actions on channels can be atomically combined with multi-assignments or arbitrary update expressions, in a way similar to the combination of action labels with assignments or update expressions.

Hidden constraints In Chi 1.0, systems of differential algebraic equations containing 'hidden constraints' (see Section 2.6.1), were not bisimilar (see Section 3.8.2), to systems of equations where the hidden constraint are made explicit (obtained after differentiation).

In Chi 2.0, these systems of equations are indeed bisimilar. This means, among others, that higher index systems (see Appendix A) can be transformed to lower index systems by means of index reduction algorithms without changing the meaning of the models.

Concrete and abstract format In Chi 1.0, a model in the concrete syntax was defined in the abstract syntax as a collection of abstract Chi processes: one process for each valuation. For many abstract models, there was no transformation to a corresponding concrete model. Note that in Chi 1.0, the concepts of a concrete and abstract syntax were not used. Instead, Chi 1.0 was defined in terms of a syntax that combined the abstract format with the so called 'syntactic extensions'.

In Chi 2.0, apart from a very small number of contrived models, there is now a one to one transformation from each Chi concrete model to a corresponding Chi abstract model, and vice versa.

Variable types The declaration of the three types of variable (discrete, continuous and algebraic) has been changed. In Chi 1.0 in the abstract format, three sets were needed to define the three types. In the concrete format, there were three variable declaration keywords for the three types.

In Chi 2.0 in the abstract format, the 'dynamic' type of a variable (discrete, continuous, algebraic) is defined by means of one 'dynamic type mapping'. In the concrete format, the dynamic type is combined with the 'static' type (e.g. boolean, integer).

Deduction rules In Chi 1.0, the negative premisses in some of the deduction rules in the semantics lead to the need for some manual proofs to show that the Chi semantics is compositional (stateless bisimilarity is a congruence with respect to all Chi operators, see Section 3.8.2).

In Chi 2.0, there are no longer any negative premisses in the deduction rules (see Section 3). Manual congruence proofs are no longer necessary, since all deduction rules satisfy the *process-tyft* format (see Section 3.8.2). Also, although Chi 2.0 is more expressive than Chi 1.0, the total number of deduction rules has been decreased from 53 in Chi 1.0 to 44 in Chi 2.0.

1.2 Acknowledgements

The authors would like to thank Jos Baeten and Pieter Cuijpers for helpful comments and stimulating discussions. Of the other people that have in one way or another helped in developing Chi 2.0, we would like to mention Pieter Collins, Dennis Hendriks, Asia van de Mortel-Fronczak, Uzma Khadim, Erjen Lefeber, Jasen Markovski, Mihaly Petreczky, and Sasha Pogromsky.

2 Abstract syntax and informal semantics

This section presents a concise definition of the abstract syntax and informal semantics of the Chi formalism. The syntax definition is incomplete in the sense that the syntax of predicates, expressions, etc. is defined at a high level of abstraction. This is done because different implementations of Chi, such as tools for simulation, verification, or real-time control, may impose different syntactical restrictions. The intention of this section is to define the Chi formalism that encompasses a variety of (future) tools without posing unnecessary syntactical restrictions.

2.1 Notations and mathematical definitions

2.1.1 Functions

Notation $f : M \to G$ defines a total function with $\mathrm{dom}(f) = M$ and codomain G. Notation $g : M \rightharpoonup G$ defines a partial function g, with definition domain M and codomain G. The domain $\mathrm{dom}(g) \subseteq M$ of the partial function g is the set of all elements $x \in M$ such that $g(x)$ is defined. The range of f, denoted by $\mathrm{range}(f)$, is the set of all values that the function takes when x takes values in the domain.

We use the term mapping for those functions that are syntactically defined using the literal notation $\{x_1 \mapsto f_1(x_1), \ldots, x_n \mapsto f_n(x_n)\}$, as for example used in Section 2.2.

The following definitions of the operators $\restriction$ and $\cup$ applied on functions are used:

- 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.

2.1.2 Sets and types

Values and variables

- $\mathbb{B} = \{\text{true}, \text{false}\}$ denotes the set of booleans.
- $\mathbb{R}$ denotes the set of all real values.
- Λ denotes the set of all values. It contains at least the booleans $\mathbb{B}$ and the reals $\mathbb{R}$.
- $\mathcal{V}$ denotes the set of all variables, including the predefined variable time.
- $\dot{\mathcal{V}} = \mathcal{V} \cup \{\dot{x} \mid x \in \mathcal{V}\}$ denotes the union of the set of all variables with the set of the dotted versions of the variables.
- $\dot{\mathcal{V}}^- = \dot{\mathcal{V}} \cup \{x^- \mid x \in \dot{\mathcal{V}}\}$ denotes the union of the set of all variables (including their dotted versions) with the set of the minus superscripted versions of those variables.
- $\Sigma = \mathcal{V} \rightharpoonup \Lambda$ denotes the set of all variable valuations. A variable valuation is a partial function from variables to values. It captures the values of the variables (at some moment in time).
- $\Sigma_\perp = \mathcal{V} \rightharpoonup \Lambda_\perp$ denotes the set of all variable valuations, where variables may have the undefined 'value' $\perp$ ($\perp \notin \Lambda$). Here $\Lambda_\perp = \Lambda \cup \{\perp\}$.

  In this report, we use the convention $\sigma \in \Sigma$ and $\sigma_\perp \in \Sigma_\perp$.
- $\dot{\Sigma} = \dot{\mathcal{V}} \rightharpoonup \Lambda$ denotes the set of valuations for all variables and dotted variables. These valuations are referred to as 'extended valuations'.
- $\mathcal{D} = \mathcal{V} \rightharpoonup \{\mathsf{disc}, \mathsf{cont}, \mathsf{alg}\}$ denotes the set of all dynamic type mappings: partial functions from variables to dynamic types.

  Here, $\{\mathsf{disc}, \mathsf{cont}, \mathsf{alg}\}$ denotes the set of dynamic types of the Chi variables.

  Functions that derive the set of variables of a certain type from a dynamic type mapping are defined in Section 2.1.4.
- Let $V \subseteq \dot{\mathcal{V}}^-$ denote a set of (possibly dotted or minus superscripted) variables. Then:
  - $\text{Expr}(V)$ denotes the set of all expressions over the variables from V.
  - $\text{Pred}(V)$ denotes the set of all predicates over the variables from V.

Action labels, channels, recursion, process terms and environments

- $\mathcal{L}_{\text{basic}}$ denotes the set of all basic action labels.
- $\mathcal{H}$ denotes the set of all channels.
- $\mathcal{U}_{\text{ah}} = (\mathcal{L}_{\text{basic}} \cup \{\tau\} \cup \mathcal{H}) \rightharpoonup \mathbb{B}$ denotes the set of all urgency mappings in the form of a partial function from basic action labels, the internal action label τ (see Section 2.5), and channels, to boolean values, indicating whether an action label or channel is urgent or not.
- $\mathcal{P}_{\text{abstract}}$ denotes the set of all process terms of the abstract Chi language as defined in Section 2.5.
- $\mathcal{M}$ denotes the set of all recursion variables, also referred to as modes.

- $\mathcal{R} = \mathcal{M} \rightharpoonup \mathcal{P}_{\text{abstract}}$ denotes the set of all recursion mappings: partial functions from recursion variables to (abstract) process terms.
- $\mathcal{E} = \mathcal{D} \times \mathcal{U}_{\text{ah}} \times 2^{\mathcal{V}} \times \mathcal{R}$ denotes the set of all environments of a Chi process (see Section 2.2).

2.1.3 Functions on sets

We use among others, the following operations on sets. Let $S \subseteq \mathcal{S}$ denote a set of items of type $\mathcal{S}$. Then:

- $\overline{S} = \mathcal{S} \setminus S$ denotes the complement of S.
- 2^S denotes the powerset of S.

2.1.4 Functions on dynamic type mappings

To derive the set of variables of a certain type from a dynamic type mapping, the following notations are defined. Let $D : \mathcal{D}$ be a dynamic type mapping. Then:

- D_{disc}, D_{cont}, D_{alg} denote the discrete, continuous, and algebraic variables, respectively; they are defined as $D_t = \{x \in \text{dom}(D) \mid D(x) = t\}$ for $t \in \{\text{disc}, \text{cont}, \text{alg}\}$.
- $D_{\text{state}} = D_{\text{disc}} \cup D_{\text{cont}}$ denotes the set of *state* variables.

E.g. Let D be syntactically defined as $D = \{k \mapsto \mathsf{disc}, n \mapsto \mathsf{disc}, x \mapsto \mathsf{cont}, y \mapsto \mathsf{alg}\}$. Then $D_{\text{disc}} = \{k, n\}$, $D_{\text{cont}} = \{x\}$, $D_{\text{alg}} = \{y\}$, and $D_{\text{state}} = \{k, n, x\}$.

In this report, we use the convention that the variables used in valuations from Σ and $\Sigma_\perp$ are the state variables: the declared discrete and continuous variables. Extended valuations $\dot{\Sigma}$ include all declared variables and the dotted versions of the declared continuous variables in their domain, but they do not include undefined variables.

2.2 Syntax of processes

A Chi process is a triple $\langle p, \sigma_\perp, E \rangle$, where $p \in \mathcal{P}_{\text{abstract}}$ denotes a process term of the abstract Chi language, $\sigma_\perp : \Sigma_\perp$ denotes a variable valuation, and $E \in \mathcal{E}$ denotes an environment. Syntactically, a variable valuation is denoted by a set of pairs $\{x_0 \mapsto c_0, \ldots, x_n \mapsto c_n\}$, where each x_i denotes a different variable and c_i its corresponding value (possibly undefined: $\perp$). A variable valuation is also referred to simply as a valuation.

An environment E is a tuple (D, U, J, R). Here, $D : \mathcal{D}$ is a dynamic type mapping that defines variables as either discrete, continuous or algebraic. Syntactically, a dynamic type mapping is denoted by a set of pairs $\{x_0 \mapsto t_0, \ldots, x_n \mapsto t_n\}$, where each x_i denotes a different variable and t_i its associated dynamic type. Furthermore, $U : \mathcal{U}_{\text{ah}}$ is an urgency mapping, $J \subseteq \mathcal{V}$ denotes the set of jumping variables, and $R : \mathcal{R}$ denotes a recursion mapping. Syntactically, an urgency mapping for actions and channels is denoted by a set of pairs $\{ah_0 \mapsto b_0, \ldots, ah_m \mapsto b_m\}$, where each ah_i denotes a different action label or channel, respectively, and b_i its associated boolean

value which is true if the action label or channel is urgent and false otherwise. Syntactically, a recursion mapping is denoted by a set of pairs $\{X_0 \mapsto p_0, \ldots, X_r \mapsto p_r\}$, where each X_i denotes a different recursion variable (see Section 2.5) and p_i its associated process term.

To ensure that the variables, channels and recursion variables occurring in Chi processes are consistently defined, each Chi process $\langle p, \sigma_\perp, (D, U, J, R)\rangle$ is assumed to satisfy the following requirements:

- All variables (possibly carrying an additional minus superscript, as in x^-) occurring free in p or in the range of R, meaning that they are not declared (in a variable scope process term, see Section 2.5) in p or in the range of R, respectively, are in the domain of D, or in case of dotted variables $\dot{x}$ or $\dot{x}^-$, their non-dotted counterparts x are in D_{cont}.
- All basic action labels and channels occurring free in p or in the range of R are in the domain of U.
- All recursion variables occurring free in p or in the range of R are in the domain of R.
- Basic action labels (and the internal action), channels, and recursion variables are all different. Thus, the sets $\mathcal{L}_{\text{basic}} \cup \{\tau\}$, $\mathcal{H}$, and $\mathcal{M}$, are piecewise disjoint.
- Jumping variables are defined: $J \subseteq \text{dom}(D)$.
- The predefined variable 'time' is included in the dynamic type mapping as a continuous variable: $\mathsf{time} \in \text{dom}(D)$ and $D(\mathsf{time}) = \mathsf{cont}$.
- The valuation $\sigma_\perp$ is defined precisely for the state variables defined by the dynamic type mapping D: $\text{dom}(\sigma_\perp) = D_{\text{state}}$.

2.3 Informal semantics of processes

The valuation σ captures the values of those variables that are relevant for determining the future behaviors of a process. The domain of the valuation σ in a Chi process $\langle p, \sigma, E\rangle$ consists of the discrete variables and the continuous variables, including the predefined continuous variable time. The dotted continuous variables and the algebraic variables are not included in the domain of σ, because their values depend only on the process term p, possibly together with the values of the other variables. The values of the dotted continuous variables and algebraic variables are included in the trajectories that show the time dependent behavior of the values of the variables. The trajectories represent externally visible (observable) information of a process. They are also needed to ensure consistency of Chi processes, as further explained in Section 2.3.2.

The behavior of Chi processes is defined in terms of actions, delays, and consistency[2]. Actions define instantaneous changes to the values of variables. Delays involve passage of time, where for all variables their trajectory as a function of time is defined.

2.3.1 Variables

There are four classes of variables: discrete, continuous, dotted continuous, and algebraic variables; and there is the predefined continuous variable time, that denotes the current time.

[2]Formally, the behavior of Chi processes is defined in terms of action transitions, time transitions, and consistency transitions, see Section 3. Informally, we use the term actions to refer to action transitions, the term delays to refer to time transitions, and we use the terminology that a process is consistent to refer to the fact that the process can execute a consistency transition.

Note that the predefined variable time only behaves as the physical time if it does not change in actions. Sufficient conditions for this are that time does not occur in the set of jumping variables J, that it does not occur in a set W associated to an update predicate r, as in $W : r$, and that it does not occur as a variable in a receive process term (such as 'true $\rightarrow h\,?\,\mathsf{time} : \emptyset$: true', see Section 2.5).

The differences with respect to the time dependent behavior of the four classes of variables (as defined by the solution function for each variable, see Section 3.1.2) are as follows:

- The discrete variables remain constant while delaying.
- The continuous variables change according to an absolutely continuous function of time while delaying (see Section 3.6.1).
- The dotted continuous variables change according to an integrable, possibly discontinuous, function of time while delaying. This solution function is the derivative function (piecewise) of the solution function for the associated continuous variable, under the condition that the latter function is (piecewise) differentiable. The relation between the dotted continuous variables and the continuous variables is explained in more detail in Section 3.6.1.
- The algebraic variables may change according to a discontinuous function of time while delaying.

The differences with respect to the action behavior of the four classes of variables are as follows:

- The discrete and continuous variables are allowed to change only if they are in the set of jumping variables J of the environment, if they are in the set W associated to an update predicate r, as in $W : r$, see Section 2.5, or if they occur as a variable in a receive process term, as in true $\rightarrow h\,?\,x : \emptyset$: true, see Section 2.5.
- The algebraic variables are in principle always allowed to change in action transitions, as long as the update predicates r are satisfied, and the process remains consistent.

The differences with respect to the behavior in all transitions of the four classes of variables are as follows:

- The resulting value of a discrete or continuous variable, including the predefined continuous variable time, in a transition always equals its starting value in the next transition.
- For algebraic and dotted continuous variables there is no such relation. The reason for this is that the discrete and continuous variables are the state variables, whereas the algebraic and dotted continuous variables are not part of the state of a process.

Further explanation on the semantics of the behavior of the different classes of variables is found in Section 3.6.1 on the equation process term, in Section 3.6.2 on the invariant, Section 3.6.4 on the action update process term, in Section 3.6.5 on the send and receive update process terms, and in Section 3.7.4 on parallel composition.

2.3.2 Consistency

In Chi, only consistent processes have behavior. This is comparable to hybrid automata, where the invariants of active locations must hold. Informally, in Chi, the 'active' equations and invariants must hold.

Consider, for example, the process $\langle\ \mathsf{eqn}\ y = n \parallel n := 1, \{n \mapsto 0, \mathsf{time} \mapsto 0\}, (\{n \mapsto \mathsf{disc}, \mathsf{time} \mapsto \mathsf{cont}, y \mapsto \mathsf{alg}\}, \emptyset, \emptyset, \emptyset)\ \rangle$ consisting of the discrete variable n, the predefined continuous variable time, and the algebraic variable y. For the process terms used in the examples, and their informal semantics, see Sections 2.5 and 2.6, respectively. Initially, the value of n equals 0, and thus the value of y equals 0. After the assignment of 1 to n, the equation $y = n$ should still hold, and thus the value of y changes to 1.

Consistency also ensures that inconsistent processes cannot be reached. E.g, in $\mathsf{inv}\ x \leq 0 \parallel x := 1$, the assignment to x cannot be executed. In fact, in Chi, only consistent processes can perform action or delay transitions, and the result of an action or delay transition is always a consistent process.

2.4 Syntax and informal semantics of expressions

An expression e in Chi is a combination of values, variables, operators, and functions that can in principle be evaluated. Apart from the 'normal' mathematical expressions, the Chi language also allows *conditional* expressions. Let $e \in \mathrm{Expr}(\dot{V})$ denote an arbitrary expression. Then the syntax of conditional expressions is:

$$
\begin{array}{lcl}
e_{\mathrm{cond}} & ::= & (e_{\mathrm{c}}) \\
e_{\mathrm{c}} & ::= & u \rightarrow e \\
& \mid & e_{\mathrm{c}}\ \text{`|'}\ e_{\mathrm{c}}
\end{array}
$$

The value of a conditional expression $(u_1 \rightarrow e_1 \mid \cdots \mid u_n \rightarrow e_n)$ is the value of an expression e_i for which the associated guard u_i is true, assuming that such a guard exists. An example of the use of a conditional expression is the equation process term $\mathsf{eqn}\ x = (y < 0 \rightarrow 0 \mid 0 \leq y \leq 1 \rightarrow y \mid y > 1 \rightarrow 1)$ that defines the value of variable x to be equal to the value of variable y, unless the value of y is smaller than 0 or bigger than 1, in which case the value of x is limited to 0 or 1, respectively. The meaning of conditional expressions is more precisely defined in Section 3.5.

2.5 Syntax of process terms

The complete set of process terms $\mathcal{P}_{\mathrm{abstract}}$ of the abstract syntax is defined below by the grammar for the process terms $p \in \mathcal{P}_{\mathrm{abstract}}$. The process terms can be divided in two classes: the atomic process terms p_{atom}, that represent the smallest process terms; and the compound process terms p, that are constructed from one or more (atomic) process terms by means of operators.

	Process term	**Operator name**	**Id**	**Id name**
$p ::=$	p_{atom}			
$\mid$	$u \gg p$	initialization	u	predicate
$\mid$	$p; p$	sequential composition		
$\mid$	$p \,[\!]\, p$	alternative composition		

|	$p \parallel p$	parallel composition		
|	$\gamma_A(p)$	synchronizing action	A	set of basic action labels
|	$\partial_H(p)$	channel encapsulation	H	set of channels
|	$[\![_{\mathrm{R}}\ R\ ::\ p\ ]\!]$	recursion scope	R	recursion mapping
|	$[\![_{\mathrm{V}}\ D, \sigma_\bot ::\ p\ ]\!]$	variable scope	D	dynamic type mapping
			$\sigma_\bot$	valuation
|	$[\![_{\mathrm{A}}\ U_A ::\ p\ ]\!]$	action scope	U_A	urgency mapping for action labels
|	$[\![_{\mathrm{H}}\ U_H ::\ p\ ]\!]$	channel scope	U_H	urgency mapping for channels

Here, $u \in \mathrm{Pred}(\dot{\mathcal{V}})$ is a predicate over variables and dotted continuous variables. In $\gamma_A(p)$, the set $A \subseteq \mathcal{L}_{\mathrm{basic}}$ is a set of synchronizing basic action labels (see Section 2.6.5), and in $\partial_H(p)$, the set H is a set of channels for which synchronous communication is enforced.

The syntax of the recursion mapping R, and dynamic type mapping D with valuation $\sigma_\bot$, occurring in the recursion and variable scope operators $[\![_{\mathrm{R}}\ R\ ::\ p\ ]\!]$ and $[\![_{\mathrm{V}}\ D, \sigma_\bot ::\ p\ ]\!]$, respectively, is the same as the syntax for R, D, and $\sigma_\bot$ occurring in a process $\langle p, \sigma_\bot, (D, U, J, R)\rangle$ (see Section 2.2). The relation that is assumed to hold between the valuation σ and the dynamic type mapping D of a process $\langle p, \sigma, (D, U, J, R)\rangle$ as defined in Section 2.2, is also assumed to hold between the valuation $\sigma_\bot$ and the dynamic type mapping D of a variable scope operator $[\![_{\mathrm{V}}\ D, \sigma_\bot ::\ p\ ]\!]$: the valuation $\sigma_\bot$ is assumed to be defined precisely for the state variables defined by the associated dynamic type mapping D: $\mathrm{dom}(\sigma_\bot) = D_{\mathrm{state}}$.

In the action scope operator, $U_A : \mathcal{L}_{\mathrm{basic}} \rightharpoonup \mathbb{B}$ is an urgency mapping for basic action labels. The domain of U_A may not include the internal action label τ, since this action label may not be redefined. In the channel scope operator, $U_H : \mathcal{H} \rightharpoonup \mathbb{B}$ is an urgency mapping for channels.

The initialization operator and the binary operators are listed in descending order of their binding strength as follows $\gg$, ; , $[\!]$, $\|$.

The set of atomic process terms $\mathcal{P}_{\mathrm{atom}}$ is defined by the following grammar for the process terms $p_{\mathrm{atom}} \in \mathcal{P}_{\mathrm{atom}}$:

	Process term	**Process term name**	**Id**	**Description**
$p_{\mathrm{atom}} ::=$	$\mathsf{eqn}\ u$	equation	u	predicate
|	$\mathsf{inv}\ u$	invariant		
|	$\mathsf{tcp}\ u$	time can progress		
|	$u \to a : W : r$	guarded action update	a	action label
			W	set of variables
			r	update predicate
|	$u \to h\,!\,\mathbf{e}_n : W : r$	guarded send update	h	channel
			$\mathbf{e}_n$	expressions
|	$u \to h\,?\,\mathbf{x}_n : W : r$	guarded receive update	$\mathbf{x}_n$	variables
|	$u \to h\,!?\ \mathbf{x}_n := \mathbf{e}_n : W : r$	guarded communication update		
|	X	recursion variable		
|	p_{abbr}			

Here, $u \in \mathrm{Pred}(\dot{\mathcal{V}})$ is a predicate over variables and dotted variables; the action label a is taken from the set of basic action labels $\mathcal{L}_{\mathrm{basic}}$, or it can be the special action label τ representing the internal or silent step: $a \in \mathcal{L}_{\mathrm{basic}} \cup \{\tau\}$; $W \subseteq \dot{\mathcal{V}}$ is a set of variables (algebraic and dotted variables may be in W even though this has no effect); update predicate $r \in \mathrm{Pred}(\dot{\mathcal{V}}^-)$ is a predicate

over variables, dotted continuous variables, and '$^-$' superscripted variables (including the dotted variables, e.g. $\dot{x}$ and $\dot{x}^-$); and $h \in \mathcal{H}$ is a channel. For $n \geq 1$, $\mathbf{e}_n$ denotes the sequence of expressions $e_1, \ldots, e_n$, where $e_1, \ldots, e_n \in \text{Expr}(\dot{\mathcal{V}})$, and $\mathbf{x}_n$ denotes the sequence of variables $x_1, \ldots, x_n$, including the dotted continuous variables ($x_1, \ldots, x_n \in \dot{\mathcal{V}}$). For $n = 0$, $h\,!\,\mathbf{e}_n$, $h\,?\,\mathbf{x}_n$, and $h\,!?\,\mathbf{x}_n := \mathbf{e}_n$ denote $h\,!$, $h\,?$, and $h\,!?$, respectively, where h is a channel. Finally, $X \in \mathcal{M}$ is a recursion variable, and p_{abbr} represents abbreviations that are defined in the section below. As is common practice in mathematics, the comma in predicates u and r denotes conjunction. E.g. eqn u_1, u_2 denotes the equation process term eqn $u_1 \wedge u_2$.

2.5.1 Abbreviations

Process term p_{abbr} represents the following abbreviations:

	Process term	**Process term name**
$p_{\text{abbr}} ::=$	δ	deadlock
$\mid$	$\bot$	inconsistent
$\mid$	$\mathbf{x}_n := \mathbf{e}_n$	multi-assignment
$\mid$	p_{sync}	
$p_{\text{sync}} ::=$	a	action label
$\mid$	$h\,!\,\mathbf{e}_n$	send
$\mid$	$h\,?\,\mathbf{x}_n$	receive
$\mid$	$h\,!?\ \mathbf{x}_n := \mathbf{e}_n$	communicate

The deadlock process term δ cannot perform actions or delays. It is however consistent. It is defined as the tcp predicate false:

$$\delta \triangleq \text{tcp false}$$

In Chi, only consistent processes can do action or delay transitions, and the result of an action or delay transition is always a consistent process. Some process terms are consistent for certain valuations and inconsistent for other valuations. E.g. the invariant inv $x \geq 0$ is consistent for all values of x greater or equal to zero, and inconsistent otherwise. The inconsistent process term $\bot$ is inconsistent for all valuations and it cannot perform any transition. It is defined as an equation that never holds:

$$\bot \triangleq \text{eqn false}$$

The multi-assignment process term $\mathbf{x}_n := \mathbf{e}_n$ is defined as a guarded internal action update process term that changes the values of the variables $x_1, \ldots, x_n$ to the values of the expressions $e_1, \ldots, e_n$, respectively.

$$\mathbf{x}_n := \mathbf{e}_n \triangleq \text{true} \rightarrow \tau : \{\mathbf{x}_n\} : \mathbf{x}_n = \mathbf{e}_n^-$$

Here e^- denotes the result of replacing all variables (and dotted variables) in e by their '$^-$'

superscripted version, and $\mathbf{x}_n = \mathbf{e}_n^-$ denotes $x_1 = e_1^- \wedge \cdots \wedge x_n := e_n^-$. For example, process term $x := 2x + yz$ is defined as true $\rightarrow \tau : \{x\} : x = 2x^- + y^- z^-$ (see also Section 4.3.2).

The action label, send, receive, and communicate process terms (represented by p_{sync}) are defined as:

$$p_{\text{sync}} \triangleq \text{true} \rightarrow p_{\text{sync}} : \emptyset : \text{true}$$

2.6 Informal semantics of process terms

Strictly speaking, a Chi process term p cannot perform actions nor delays. Only the Chi process $\langle p, \sigma, E \rangle$, that is obtained by adding a valuation and an environment to p, can, in principle, perform actions and delays. Therefore, when we informally refer to a process term that performs actions or delays, we refer to the process term together with a valuation and environment.

2.6.1 Manipulating the values of variables

In Chi, there are several classes of variables, and there are several means to change the value of a variable, depending on the class of the variable. The main means for changing the value of a variable are the equation process term, for changes over time, and the action update process term, for instantaneous changes.

Equation process term In principle, continuous and algebraic variables change arbitrarily over time when delaying, although, depending on the class of the variable, they may have to respect some continuity requirements (see Section 3.6.1 for more details). An *equation process term* eqn u, usually in the form of a differential algebraic equation, restricts the allowed behavior of the continuous, dotted continuous, and algebraic variables in such a way that the value of the predicate remains true over time.

Bisimilarity is an important concept to indicate equivalence of process terms. Process terms p and q are bisimilar (see Section 3.8.2 for a formal definition), denoted as $p \underline{\leftrightarrow} q$, when their external behavior is the same.

The semantics of equation process terms is defined in such a way that:

- Conjunction and parallel composition are equivalent for equations.
- Systems of equations that have hidden constraints[3] are bisimilar to systems of equations in which the hidden constraints are made explicit (obtained after differentiation).

Therefore, that for any continuous variable y:

$$\text{eqn } y = 1 \;\; \underline{\leftrightarrow} \;\; \text{eqn } y = 1, \dot{y} = 0 \;\; \underline{\leftrightarrow} \;\; \text{eqn } y = 1 \parallel \text{eqn } \dot{y} = 0 \tag{1}$$

[3] For background information on hidden constraints and the associated higher index systems see Appendix A.

This property obviously holds not only for $y = 1$, but for any equation $y = c$, where c is a value (number) of type real. As a result of this property, the following property also holds

$$\mathsf{eqn}\ y = 1 \parallel \mathsf{eqn}\ z = \dot{y} \ \underline{\leftrightarrow}\ \mathsf{eqn}\ y = 1 \parallel \mathsf{eqn}\ \dot{y} = 0 \parallel \mathsf{eqn}\ z = 0$$

Although the process term $y = 1 \parallel \mathsf{eqn}\ z = \dot{y}$ appears to be consistent with initial conditions $\{y \mapsto 1, \dot{y} \mapsto 1, z \mapsto 1\}$, this is not the case. The process term $\mathsf{eqn}\ y = 1 \parallel \mathsf{eqn}\ \dot{y} = 0 \parallel \mathsf{eqn}\ z = 0$ is obviously not consistent with these initial conditions. This means that there is no behavior (no solutions) for this initial valuation. Both process terms are consistent with initial conditions $\{y \mapsto 1, \dot{y} \mapsto 0, z \mapsto 0\}$.

An essential (although not yet proven) property of the semantics is the substitution property (see Property 3.12 in Section 3.8.3):

$$\mathsf{eqn}\ y = e \parallel p \ \underline{\leftrightarrow}\ \mathsf{eqn}\ y = e \parallel p[e/y], \tag{2}$$

where y is a variable, e an expression, and p a process term. The substitution property is the basis of the so called 'consistent equation semantics' of Chi. The meaning of the substitution property is that a variable which is defined to be equal to an expression can be replaced by its defining expression in all parallel contexts. An example of this property is

$$\mathsf{eqn}\ y = 1 \parallel x := y \ \underline{\leftrightarrow}\ \mathsf{eqn}\ y = 1 \parallel x := 1$$

As a result of the property defined in (1) in combination with the substitution property (2), the following property holds:

$$\mathsf{eqn}\ y = 1 \parallel z := \dot{y} \ \underline{\leftrightarrow}\ \mathsf{eqn}\ y = 1 \parallel z := 0$$

Invariant process term The invariant process term $\mathsf{inv}\ u$, usually in the form of $\mathsf{inv}\ x \geq e$ or $\mathsf{inv}\ x \leq e$, is comparable to an invariant in a hybrid automaton. An invariant process term $\mathsf{inv}\ u$ differs from an equation process term $\mathsf{eqn}\ u$ with respect to the consistency behavior. This is explained in more detail below.

The initial conditions of an equation process term $\mathsf{eqn}\ u$ should satisfy the equation process term *and* its hidden constraints (if any). The initial conditions of an invariant process term $\mathsf{inv}\ u$ need to satisfy only the invariant itself. Therefore, the system of equations

$$\mathsf{inv}\ y = 1,\ z = \dot{y}$$

is consistent with valuation $\{y \mapsto 1, \dot{y} \mapsto 1, z \mapsto 1\}$, even though there is no subsequent delay behavior. The difference between the equation and invariant process terms becomes more clear when both systems are prefixed with a multi-assignment in a sequential composition. For the process term

$$y, z := 1, 1;\ \mathsf{eqn}\ y = 1, z = \dot{y},$$

the multi-assignment cannot be executed, because $\{y \mapsto 1, \dot{y} \mapsto 1, z \mapsto 1\}$ are inconsistent initial conditions for $\mathsf{eqn}\ y = 1, z = \dot{y}$. The multi-assignment in the process term

$$y, z := 1, 0;\ \mathsf{eqn}\ y = 1, z = \dot{y}$$

can be executed, because $\{y \mapsto 1, \dot{y} \mapsto 0, z \mapsto 0\}$ are consistent initial conditions for the equations.

For the process term

$$y, z := 1, 1;\ \mathsf{inv}\ y = 1, z = \dot{y},$$

the multi-assignment can be executed, because $\{y \mapsto 1, \dot{y} \mapsto 1, z \mapsto 1\}$ are consistent initial conditions for $\mathsf{inv}\ y = 1, z = \dot{y}$.

A more representative combination of equations and invariants is as follows:

$$\mathsf{eqn}\ \dot{x} = 1 \parallel \mathsf{inv}\ x \geq 1$$

Here the trajectory of variable x is determined by means of the equation process term, and the invariant process term restricts the initial value of x and the length of the trajectory (delay interval).

Guarded action update process term An instantaneous change of the value of a discrete or continuous variable in Chi is always connected to the execution of an action. In action update process terms, the action is represented by a label. Other types of action are related to communication, which is explained in the paragraph on parallelism in Section 2.6.5. *Guarded action update process term* $u \rightarrow a : W : r$ denotes instantaneous changes to the variables from set W (and to the variables from the set J of jumping variables from the environment), by means of an action labeled a, such that predicate r and guard u are both satisfied. For non-urgent actions a, the instantaneous changes can be preceded by arbitrary delays, and can thus take place at any point in time. For urgent actions a, the instantaneous changes can be preceded by delays for as long as the guard u remains false (assuming that action label a does not synchronize with other labels in a parallel composition). The discrete and continuous variables that are not mentioned in W and that are not in the set of jumping variables of the environment remain unchanged. The other variables, including the algebraic and dotted continuous variables may obtain arbitrary values, provided that the predicate r is satisfied and the process remains consistent.

A '$^-$' superscripted occurrence of a variable in r refers to the value of the variable in the extended valuation prior to execution of the action update process term, and a normal (non-superscripted) occurrence of a variable in r refers to the value of that variable in the extended valuation that results from the execution of the action update process term. E.g. incrementing the value of variable x by 1 is represented by the guarded update process term 'true $\rightarrow \tau : \{x\} : x = x^- + 1$'. The guard u is evaluated in the extended valuation prior to execution of the action update process term. A predicate r is satisfied if evaluating the '$^-$' superscripted variables in the original extended valuation and evaluating the normal occurrences of the variables in the resulting extended valuation means that the predicate is true. The reason to use an extended valuation for evaluating update predicate r and guard u is that in such predicates also algebraic and dotted continuous variables may be used. Note that it can be the case that different instantaneous changes satisfy the predicate. This may result in non-determinism, as in 'true $\rightarrow \tau : \{x\} : x^2 = 1$'.

Note that the (multi-)assignment is not a primitive in Chi, as for example in [5]. This is because action update process terms are more expressive than assignments. An assignment can be expressed as an action update process term (see Section 4.3), but not the other way around. Consider for example the action update process term 'true $\rightarrow \tau : \{x\} : x \in [0, 1]$', that changes the value of x to a value in the interval $[0, 1]$. Also, the predicate of an action update process term may consist of a conjunction of implicit equations, e.g. 'true $\rightarrow \tau : \{\mathbf{x}\} : f_1(\mathbf{x}, \mathbf{x}^-) = 0 \wedge \ldots \wedge f_n(\mathbf{x}, \mathbf{x}^-) = 0$'. The solution of such a system of equations, if present, is not always expressible in an explicit form. The system may also have multiple solutions.

2.6.2 Initialization

The initialization operator process term $u \gg p$ restricts the initial conditions of a process term p to those initial conditions that satisfy predicate u.

2.6.3 Sequential composition

The *sequential composition* of process terms p and q behaves as process term p until p terminates (which can only happen if the state after termination of p represents consistent initial conditions for q), and then continues to behave as process term q.

2.6.4 Choice

The *alternative composition operator* $[]$ allows a non-deterministic choice between the actions of its operands. The participants in the alternative composition have to synchronize to obtain the time behavior. This means that the passage of time cannot make a choice between the operands of an alternative composition operator. Also, the trajectories of the variables have to be agreed upon by both participants.

2.6.5 Parallelism

Parallelism can be specified by means of the *parallel composition operator* $\|$. Parallel processes interact by means of shared variables, by means of synchronous point-to-point communication / synchronization via a channel, or by means of synchronizing action labels. The parallel composition $p \parallel q$ synchronizes the time behavior of p and q, interleaves the non-synchronizing action behavior (including the instantaneous changes of variables) of p and q, synchronizes matching send and receive actions, and synchronizes synchronizing action labels. The synchronization of time behavior means that only the time behaviors that are allowed by both p and q are allowed by their parallel composition. The consistent equation semantics of Chi enforces that actions by p (or q) are allowed only if the values of the variables before and after the actions represent consistent initial conditions for the other process term q (or p). This means, among others, that the active equations and invariants of q must hold before and after execution of an action by p. The active part of a process term mainly changes as a result of execution of sequential process terms: in a sequential composition $p_1; p_2; \ldots p_n$, where the active process term is p_1, termination of p_1 results in p_2 becoming the active process term.

By means of the *guarded send update process term* $u \rightarrow h\,!\,e_1, \ldots, e_n : W : r$, for $n \geq 1$, the values of expressions $e_1, \ldots, e_n$ (evaluated w.r.t. the extended valuation) are sent via channel h, and the update predicate r and the guard u need to be satisfied in the same way as for the action update process term. For $n = 0$, this reduces to $u \rightarrow h\,! : W : r$ and nothing is sent via the channel. By means of the *guarded receive update process term* $u \rightarrow h\,?\,x_1, \ldots, x_n : W : r$, for $n \geq 1$, values for $x_1, \ldots, x_n$ are received from channel h, and the update predicate and the guard need to be satisfied in the same way as for the action update process term. For $n = 0$, this reduces to $u \rightarrow h\,? : W : r$, and nothing is received via the channel. Communication in Chi is the sending of values by one parallel process via a channel to another parallel process, where the received values (if any) are stored in variables. For communication, the acts of sending and receiving (values) have to take place simultaneously in different parallel processes. In case no values are sent and received, we refer to synchronization instead of communication.

Where actual synchronization and communication by means of channels always takes place between exactly two partners (a sending and a receiving process), synchronization by means of the *guarded action update process term* $u \rightarrow a : W : r$ can involve any number of partners. In fact, all partners of a parallel composition that share an action label $a \in \mathcal{L}_{\text{basic}}$ in their set of synchronizing actions must synchronize whenever any of these partners executes an action update $a : W : r$. By means of the synchronizing action process term $\gamma_A(p)$, the set of synchronizing

action labels $A \subseteq \mathcal{L}_{\text{basic}}$ is added to the set of synchronizing action labels of process term p, and synchronization takes place on the basis of common synchronizing action labels. Note that most automata synchronize on *all* action labels that are present on their edges. In such cases, the set of synchronizing action labels is called the *alphabet* of the automaton. Their is no such requirement in Chi: when the synchronizing action operator γ_A is not used in a process term p (or when the operator is used only with empty sets A), the set of synchronizing action labels of p is empty.

In order to be able to model open systems (i.e. systems that interface with the environment), it is necessary not to enforce communication via the external channels of the model (e.g. the channels that send or receive from the environment). For communication via internal channels, however, the communication of matching send and receive actions, in general is not only an option, but an obligation. In such models, the separate occurrence of the send action and the receive action via an internal channel is undesired. The *encapsulation operator* $\partial_H(p)$ is introduced to block the send and receive actions via channels from the set H. This operator thus ensures that for all channels from set H, only the synchronous execution of matching send and receive actions takes place.

Action labels and channels can be declared as urgent or non-urgent in processes (see Section 2.2), or in action scopes and channel scopes, respectively. Urgent action labels and urgent channels may prevent delays from taking place, and/or they may restrict the length of a delay interval. Delaying is possible only if, and for as long as no urgent actions are enabled (see Section 2.6.6).

An urgent action via an *action label* is enabled in a parallel composition if:

- the urgent action label is enabled (guard is true) and non-synchronizing in an operand of the parallel composition, or
- the action label is enabled and synchronizing in both operands of the parallel composition.

An urgent (communication) action via an urgent *channel* is enabled in a parallel composition if:

- a send process term and a receive process term on the same urgent channel in the two respective operands of the parallel composition are both enabled (guards are true).

2.6.6 Urgency

There are two main ways of expressing urgency:

- global action label urgency and global channel urgency by means of urgent action labels and urgent channels, respectively. This kind of urgency is expressed in the environment of a process by means of an urgency mapping.
- local urgency by means of the time can progress process term.

These concepts are explained in detail in the following sections. For more information on the use of urgency, see Appendix B.

Non-synchronizing urgent action labels Consider the following example process:

$$
\begin{array}{l}
\langle\ \mathsf{time} \geq 1 \rightarrow a : \emptyset : \mathrm{true} \\
,\ \{\mathsf{time} \mapsto 0\} \\
,\ (\{\mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{true}\}, \emptyset, \emptyset) \\
\rangle
\end{array}
$$

This process delays until time point 1 is reached. At that time point, the guard $\mathsf{time} \geq 1$ becomes true, so that the action label a becomes enabled. This action label is declared as urgent by means of the urgency mapping $\{a \mapsto \mathrm{true}\}$ in the environment. Therefore, at time point 1, no further delaying is possible, and the guarded action update statement '$\mathsf{time} \geq 1 \rightarrow a : \emptyset : \mathrm{true}$' *must* execute the action, after which the process terminates. When the action label is declared as non-urgent, as in the following process:

$$
\begin{array}{l}
\langle\ \mathsf{time} \geq 1 \rightarrow a : \emptyset : \mathrm{true} \\
,\ \{\mathsf{time} \mapsto 0\} \\
,\ (\{\mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{false}\}, \emptyset, \emptyset) \\
\rangle,
\end{array}
$$

initially, *arbitrary* delays are possible. When a time point has been reached for which $\mathsf{time} \geq 1$ holds (after a delay), the guard is enabled. From then on, the action update statement can execute the action and terminate. The choice between delaying and executing the action is nondeterministic.

Urgency restricts the length of delays, independently of consistency. In the example below, the action update statement '$\mathsf{time} \geq 1 \rightarrow a : \emptyset : \mathrm{true}$' can never do the action, because that would mean enabling the invariant process term '$\mathsf{inv}\ \mathrm{false}$', which can never be satisfied. Therefore, when time point 1 is reached after delaying, no further delaying is possible due to the enabled urgent action. Since the action cannot be executed, the process deadlocks at time point 1.

$$
\begin{array}{l}
\langle\ \mathsf{time} \geq 1 \rightarrow a : \emptyset : \mathrm{true};\ \mathsf{inv}\ \mathrm{false} \\
,\ \{\mathsf{time} \mapsto 0\} \\
,\ (\{\mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{true}\}, \emptyset, \emptyset) \\
\rangle
\end{array}
$$

In a parallel composition of non-synchronizing action labels, such as:

$$
\begin{array}{l}
\langle\ \mathsf{time} \geq c_1 \rightarrow a : \emptyset : \mathrm{true} \parallel \mathsf{time} \geq c_2 \rightarrow a : \emptyset : \mathrm{true} \\
,\ \{\mathsf{time} \mapsto 0\} \\
,\ (\{\mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{true}\}, \emptyset, \emptyset) \\
\rangle,
\end{array}
$$

where c_1 and c_2 represent constants ($c_1, c_2 \geq 0$), the action label a is enabled when the guard $\mathsf{time} \geq c_1$ or the guard $\mathsf{time} \geq c_2$ is true. Since a is an urgent action label, time can progress until the first of the two time points c_1 or c_2 is reached. This is at time point $\min(c_1, c_2)$. At that time point, the action a the guard of which is true may be executed. If $c_1 = c_2$, both actions are executed at the same time point, one after the other (interleaving semantics).

Synchronizing urgent action labels In a parallel composition of synchronizing action labels, such as:

$$\begin{array}{l}\langle\ \gamma_{\{a\}}(\mathsf{time} \geq c_1 \rightarrow a : \emptyset : \mathrm{true}) \parallel \gamma_{\{a\}}(\mathsf{time} \geq c_2 \rightarrow a : \emptyset : \mathrm{true}) \\ ,\ \{\mathsf{time} \mapsto 0\} \\ ,\ (\{\mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{true}\}, \emptyset, \emptyset) \\ \rangle,\end{array}$$

where $\gamma_{\{a\}}(p)$ is the synchronizing action process term, the action label a is enabled only if both partners of the parallel composition have an enabled guard for action label a. Thus both guards $\mathsf{time} \geq c_1$ and $\mathsf{time} \geq c_2$ must be true. Therefore, time can progress until the time point $\max(c_1, c_2)$ is reached (assuming $c_1, c_2 \geq 0$). Then both guarded action update process terms simultaneously execute the action.

Urgent channels Urgent channels allow progress of time until a send and a receive process term on matching channels are enabled in different operands of a parallel composition. Therefore, the following process can perform arbitrary delays, even though the channel is declared as urgent:

$$\begin{array}{l}\langle\ \mathsf{time} \geq 1 \rightarrow h! : \emptyset : \mathrm{true} \\ ,\ \{\mathsf{time} \mapsto 0\} \\ ,\ (\{\mathsf{time} \mapsto \mathsf{cont}\}, \{h \mapsto \mathrm{true}\}, \emptyset, \emptyset) \\ \rangle\end{array}$$

In the process below, time can progress until the time point $\max(c_1, c_2)$ is reached (assuming $c_1, c_2 \geq 0$), and both guards become true.

$$\begin{array}{l}\langle\ \partial_{\{h\}}(\mathsf{time} \geq c_1 \rightarrow h! : \emptyset : \mathrm{true} \parallel \mathsf{time} \geq c_2 \rightarrow h? : \emptyset : \mathrm{true}) \\ ,\ \{\mathsf{time} \mapsto 0\} \\ ,\ (\{\mathsf{time} \mapsto \mathsf{cont}\}, \{h \mapsto \mathrm{true}\}, \emptyset, \emptyset) \\ \rangle\end{array}$$

The delay behavior of the process without the encapsulation operator is the same. The only difference is that where in the process above, the only action that can be done is the simultaneous execution of the send and receive actions, in the process below, the send and receive update process terms can also execute their actions independently of each other.

$$\begin{array}{l}\langle\ \mathsf{time} \geq c_1 \rightarrow h! : \emptyset : \mathrm{true} \parallel \mathsf{time} \geq c_2 \rightarrow h? : \emptyset : \mathrm{true} \\ ,\ \{\mathsf{time} \mapsto 0\} \\ ,\ (\{\mathsf{time} \mapsto \mathsf{cont}\}, \{h \mapsto \mathrm{true}\}, \emptyset, \emptyset) \\ \rangle\end{array}$$

Time can progress process term Where the guards of urgent action labels and urgent channels are used to simultaneously *enable* action execution *and* to *restrict* progress of time, the *time can progress predicate process term* '$\mathsf{tcp}\ u$' *only restricts* the progress of time. Another difference is that urgency by means of urgent action labels and urgent channels is a *global* concept that may depend on the synchronization behavior of multiple partners of a parallel composition, whereas

the time can progress process term is used to specify *local* urgency, independently of synchronization behavior. The process term $\mathsf{tcp}\ u$ allows delays for as long as the predicate u is true. Only at the end point of the delay, the predicate is allowed to be false.

A difference between the time can progress process term $\mathsf{tcp}\ u$ and the invariant process term $\mathsf{inv}\ u$ is that invariants are required to hold for all points of the delay interval, including the final point. Another difference is that the value of the time can progress predicate u in $\mathsf{tcp}\ u$ is irrelevant for the action behavior of processes, whereas invariants *are* required to hold in actions.

To restrict the length of the delays of a process term p, a time can progress process term $\mathsf{tcp}\ u$ is usually composed in an alternative composition, as in $p \,[]\, \mathsf{tcp}\ u$. In this way, the tcp process term is removed after the first action of p. Consider for example the process:

$$\begin{array}{l}\langle\ \mathsf{eqn}\ \dot{x} = 1 \parallel (x \geq 1 \rightarrow a : \emptyset : \mathrm{true} \,[]\, \mathsf{tcp}\ x < 2) \\ ,\ \{x \mapsto 0, \mathsf{time} \mapsto 0\} \\ ,\ (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{false}\}, \emptyset, \emptyset) \\ \rangle\end{array}$$

Here, action label a is non-urgent. Initially, only delays are possible. When the value of the guard becomes true at time point 1, the action can be executed, but delays are also possible until time point 2. At that time point, the tcp process term $\mathsf{tcp}\ x < 2$ prevents further delaying. Therefore, the action can be executed at any time point t between 1 and 2 ($1 \leq t \leq 2$). The same process can also be executed starting from an initial valuation $\{x \mapsto 10, \mathsf{time} \mapsto 0\}$:

$$\begin{array}{l}\langle\ \mathsf{eqn}\ \dot{x} = 1 \parallel (x \geq 1 \rightarrow a : \emptyset : \mathrm{true} \,[]\, \mathsf{tcp}\ x < 2) \\ ,\ \{x \mapsto 10, \mathsf{time} \mapsto 0\} \\ ,\ (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{false}\}, \emptyset, \emptyset) \\ \rangle\end{array}$$

In this case, initially no delays are possible. Therefore, the action must be executed immediately.

When an invariant is used, as in:

$$\begin{array}{l}\langle\ \mathsf{eqn}\ \dot{x} = 1 \parallel (x \geq 1 \rightarrow a : \emptyset : \mathrm{true} \,[]\, \mathsf{inv}\ x \leq 2) \\ ,\ \{x \mapsto 10, \mathsf{time} \mapsto 0\} \\ ,\ (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{false}\}, \emptyset, \emptyset) \\ \rangle,\end{array}$$

then initially the process is inconsistent: no consistency transition is possible, and therefore no behavior is possible.

When the tcp predicate is the negation of the guard, as in

$$\begin{array}{l}\langle\ \mathsf{eqn}\ \dot{x} = 1 \parallel (x \geq 1 \rightarrow a : \emptyset : \mathrm{true} \,[]\, \mathsf{tcp}\ x < 1) \\ ,\ \{x \mapsto 0, \mathsf{time} \mapsto 0\} \\ ,\ (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{false}\}, \emptyset, \emptyset) \\ \rangle,\end{array}$$

progress of time is possible until the exact time point that the guard becomes true. In this case, changing the definition of the action label to urgent ($\{a \mapsto \text{true}\}$) does not change the meaning of the process.

2.6.7 Recursive definitions

Process term X denotes a recursion variable (identifier) that is defined either in the environment of the process, or in a recursion scope operator process term $[\![_{\mathrm{R}}\ R :: p\ ]\!]$, as discussed in Section 2.6.8. Among others, recursion is used to model repetition and to model automata. Recursion variable X can do whatever the process term of its definition can do.

2.6.8 Hierarchical modeling

Thus far, it has been assumed that all variables that are allowed to occur in a Chi process term are declared in the environment. To support hierarchical modeling of systems, it is convenient to also allow local declarations of variables. For this purpose, the *variable scope operator* process term $[\![_{\mathrm{V}}\ D, \sigma_\perp :: p\ ]\!]$ is introduced, where D denotes a mapping defining the local variables and their dynamic types, and $\sigma_\perp$ denotes a (possibly partially defined) valuation of the local state variables. It is allowed that variables with the same name as local variables have been declared on a more global level already. Any occurrence of a variable from $\mathrm{dom}(D)$ in process term p refers to the local variable and not to any more global declaration of the same variable name.

For similar purposes, local recursive definitions are declared by means of a *recursion scope* process term $[\![_{\mathrm{R}}\ R :: p\ ]\!]$, local action labels are declared by means of an *action scope* process term $[\![_{\mathrm{A}}\ U_A :: p\ ]\!]$, and local channels are declared by means of a *channel scope* process term $[\![_{\mathrm{H}}\ U_H :: p\ ]\!]$. The domains of recursion mapping R, urgent action mapping U_A, and urgent channel mapping U_H are used to declare the recursion variables, the action labels and the channels, respectively. The mapping R defines for each recursion variable its associated process term, and the mappings U_A and U_H define for each action and channel, respectively whether it is urgent or non-urgent. The action and channel scope operators abstract (replace by the internal action τ) the actions via the local action labels and channels, respectively. Furthermore, the channel scope operator blocks the separate send and receive actions via local channels.

3 Formal semantics

3.1 Notations and mathematical definitions

3.1.1 Sets and types

- $\mathcal{L}_{\mathrm{com}} = \{h!cs, h?cs, h!?cs \mid h \in \mathcal{H}, cs \in \Lambda^*\}$ denotes the set of all communication action labels. Here $cs \in \Lambda^*$ denotes a list $[c_1, \ldots, c_n]$ of values ($c_i \in \Lambda$, $1 \leq i \leq n$).

- $\mathcal{L} = \mathcal{L}_{\mathrm{basic}} \cup \mathcal{L}_{\mathrm{com}}$ denotes the set of all action labels apart from the internal action label τ. To ensure that basic action labels do not synchronize with channels, the set of basic action labels and the set of communication action labels $\mathcal{L}_{\mathrm{com}}$ are required to be disjoint: $\mathcal{L}_{\mathrm{basic}} \cap \mathcal{L}_{\mathrm{com}} = \emptyset$.

- $\mathcal{L}_\tau = \mathcal{L} \cup \{\tau\}$ denotes the set of all action labels.
- $\mathcal{T} = \mathbb{R}$ denotes the set of all time points.
- $\mathcal{T}_{\geq 0} = \{t \in \mathcal{T} \mid t \geq 0\}$ denotes the set of all non-negative time points.
- $val_{\text{ah}} = (\mathcal{L}_{\text{basic}} \cup \{\tau\} \cup \mathcal{H}) \rightharpoonup \mathbb{B}$ denotes the set of all action/channel valuations. These are mappings from a basic action label, a τ action label, or a channel, to a boolean value, where the boolean value represents the value of a guard associated to the action label or channel.

3.1.2 Trajectories and solution functions

Two kinds of trajectory exist:

- A variable trajectory, which is usually referred to simply as trajectory and denoted by ρ, where $\rho : \mathcal{T} \rightharpoonup \dot{\Sigma}$ and $\dot{\Sigma} = \dot{\mathcal{V}} \rightharpoonup \Lambda$. A variable trajectory is a function from time to a variable valuation. It can be defined more precisely as $\rho : \mathcal{T} \rightharpoonup (V \to \Lambda)$, with $V \subseteq \dot{\mathcal{V}}$, indicating that the domains of the valuations $V \to \Lambda$ are the same for all time points of the trajectory.
- An action/channel guard trajectory, usually denoted by means of θ_x, $x \in \{\text{y}, \text{n}, \text{s}, \text{r}\}$ and usually referred to simply as guard trajectory, where $\theta_x : \mathcal{T}_{\geq 0} \rightharpoonup val_{\text{ah}}$, is a function from time to an action/channel valuation. It can be defined more precisely as $\theta_x : \mathcal{T}_{\geq 0} \rightharpoonup (L \to \Lambda)$, with $L \subseteq (\mathcal{L}_{\text{basic}} \cup \{\tau\} \cup \mathcal{H})$, indicating that the domains of the valuations $L \to \Lambda$ are the same for all time points of the guard trajectory.

  The identifier θ is often used to denote a quadruple of guard trajectories $(\theta_\text{y}, \theta_\text{n}, \theta_\text{s}, \theta_\text{r})$. Here, $\theta_\text{y} : \mathcal{T}_{\geq 0} \rightharpoonup (\mathcal{L}_{\text{basic}} \rightharpoonup \mathbb{B})$ represents the guard trajectory for the synchronizing action labels, $\theta_\text{n} : \mathcal{T}_{\geq 0} \rightharpoonup ((\mathcal{L}_{\text{basic}} \cup \{\tau\} \cup \mathcal{H}) \rightharpoonup \mathbb{B})$ represents the guard trajectory for the non-synchronizing action labels and for the 'communication channels' (combination of send and receive action on matching channel), and $\theta_\text{s}, \theta_\text{r} : \mathcal{T}_{\geq 0} \rightharpoonup (\mathcal{H} \rightharpoonup \mathbb{B})$ represent the guard trajectories for the 'send action' and 'receive action' channels (no communication yet), respectively.

The $\downarrow$ operator can be applied to any trajectory; the $\vee$ and $\wedge$ operators can be applied to action/channel guard trajectories only. The operators are defined below.

Let $f : \mathcal{T} \rightharpoonup (\mathcal{Y} \rightharpoonup \Lambda)$ denote a trajectory (with $\mathcal{Y} = \mathcal{V} \cup \mathcal{L}_{\text{basic}} \cup \{\tau\} \cup \mathcal{H}$), $t \in \mathcal{T}$ denote a time point, $S \subseteq \mathcal{Y}$ denote a set, and $x \in \mathcal{Y}$ denote a variable, action label or channel. Then:

- $f \downarrow S$ denotes the trajectory h with $\text{dom}(h) = \text{dom}(f)$ such that $h(t) = f(t) \restriction S$ for each time point $t \in \text{dom}(h)$.
- $f \downarrow x$ denotes the *solution function* $v : \mathcal{T} \rightharpoonup \Lambda$ with $\text{dom}(v) = \text{dom}(f)$ such that $v(t) = f(t)(x)$ for each $t \in \text{dom}(v)$.

Let $f : T \to (Y \to \mathbb{B})$ and $g : T \to (Z \to \mathbb{B})$ denote guard trajectories with arbitrary $T \subseteq \mathcal{T}$, and arbitrary $Y, Z \subseteq \mathcal{L}_{\text{basic}} \cup \{\tau\} \cup \mathcal{H}$, and let $\diamond \in \{\vee, \wedge\}$ denote either the 'logical and' operator, or the 'logical or' operator. Then:

- $f \diamond g$ denotes the trajectory $h : T \to ((Y \cup Z) \to \mathbb{B})$ such that

$$\forall_{t \in T, x \in Y \cup Z}\, h(t)(x) = \begin{cases} f(t)(x) & \text{if } x \in Y \setminus Z \\ g(t)(x) & \text{if } x \in Z \setminus Y \\ f(t)(x) \diamond g(t)(x) & \text{if } x \in Y \cap Z \end{cases}$$

Finally, some abbreviations for guard trajectories are defined. Let $U : (\mathcal{L}_{\text{basic}} \cup \{\tau\} \cup \mathcal{H}) \rightharpoonup \mathbb{B}$ denote an urgency mapping, $t \in \mathcal{T}_{\geq 0}$ denote a time point, $x \in \mathcal{L}_{\text{basic}} \cup \{\tau\} \cup \mathcal{H}$ denote a basic action label, τ label, or a channel, $\rho : \mathcal{T} \rightharpoonup \dot{\Sigma}$ denote a variable trajectory, and $u \in \text{Pred}(\dot{\mathcal{V}})$ denote a predicate. Then the following notations denote trajectories:

- $\text{fls}_{tU} : [0, t] \to (\text{dom}(U) \to \{\text{false}\})$ denotes a guard trajectory that is false for all basic action labels and channels from the domain of U.
- $\text{fls}_{t\emptyset} : [0, t] \to (\emptyset \to \mathbb{B})$ denotes a guard trajectory for $[0, t]$ to a action valuation on an empty domain.
- $\text{fls}_{t\emptyset(tU)^3} = (\text{fls}_{t\emptyset}, \text{fls}_{tU}, \text{fls}_{tU}, \text{fls}_{tU})$ denotes a quadruple of guard trajectories, used in the SOS rules that define the delay behavior for the equation, invariant, and tcp process terms.
- $\theta_{tx\rho u} : [0, t] \to (\{x\} \to \mathbb{B})$ denotes a guard trajectory for a single basic action label or channel x. The value of the guard trajectory at each time point $s \in [0, t]$ is the value of the guard u when evaluated in the valuation $\rho(s)$: $\forall_{s \in [0,t]}\, \theta_{tx\rho u}(s)(x) = \rho(s)(u)$. The notation $\rho(s)(u)$ is defined in Section 3.4.

3.2 General description of the SOS

This section presents the structured operational semantics (SOS [18]) of Chi. It associates an extended version of the hybrid transition system as defined in [6] with a Chi process. The main purpose of the SOS is to define the behavior of Chi processes at a certain chosen level of abstraction. The SOS is chosen to represent the following:

1. Discrete behavior by means of action transitions:
   (a) $\_ \xrightarrow{} \_ \subseteq (\mathcal{P}_{\text{abstract}} \times \Sigma \times \mathcal{E}) \times ((\mathcal{T} \rightharpoonup \dot{\Sigma}) \times \mathcal{L}_\tau \times \mathbb{B} \times 2^{\mathcal{V}} \times (\mathcal{T} \rightharpoonup \dot{\Sigma})) \times (\mathcal{P}_{\text{abstract}} \times \Sigma \times \mathcal{E})$. The intuition of an action transition $\langle p, \sigma, E\rangle \xrightarrow{\rho, \ell, b, W, \rho'} \langle p', \sigma', E'\rangle$ is that the process $\langle p, \sigma, E\rangle$ executes the discrete synchronizing (b is true) or non-synchronizing (b is false) action $\ell \in \mathcal{L}_\tau$ with trajectories ρ and ρ' and thereby transforms into the process $\langle p', \sigma', E'\rangle$, where σ' and E' denote the accompanying valuation and environment of the process term p', respectively, after the discrete action ℓ is executed. The initial valuations $\rho(0)$ and $\rho'(0)$ represent consistent initial conditions for the processes $\langle p, \sigma, E\rangle$ and $\langle p', \sigma', E'\rangle$, respectively. The relation between the valuations $\rho(0)$ and $\rho'(0)$ defines the externally visible changes in the values of the variables in the action transition. The set W represents the externally visible continuous and discrete variables that are allowed to change (jump) in this action transition. They need to be visible to obtain the proper semantics for synchronization in parallel composition.
   (b) $\_ \xrightarrow{} \langle \checkmark, \_, \_\rangle \subseteq (\mathcal{P}_{\text{abstract}} \times \Sigma \times \mathcal{E}) \times ((\mathcal{T} \rightharpoonup \dot{\Sigma}) \times \mathcal{L}_\tau \times \mathbb{B} \times 2^{\mathcal{V}} \times (\mathcal{T} \rightharpoonup \dot{\Sigma})) \times (\Sigma \times \mathcal{E})$. The intuition of a (termination) transition $\langle p, \sigma, E\rangle \xrightarrow{\rho, \ell, b, W, \rho'} \langle \checkmark, \sigma', E'\rangle$ is that the process $\langle p, \sigma, E\rangle$ executes the discrete synchronizing/non-synchronizing

(b is true/false) action ℓ with trajectories ρ and ρ', set of jumping variables W, and thereby transforms into the terminated process $\langle \surd, \sigma', E' \rangle$.

2. Continuous behavior by means of time transitions: $\_ \overset{-}{\longmapsto} \_ \subseteq (\mathcal{P}_{\text{abstract}} \times \Sigma \times \mathcal{E}) \times (\mathcal{T}_{\geq 0} \times (\mathcal{T} \rightharpoonup \dot{\Sigma}) \times (\mathcal{T} \rightharpoonup \mathit{val}_{\text{ah}})^4) \times (\mathcal{P}_{\text{abstract}} \times \Sigma \times \mathcal{E})$. The intuition of a time transition $\langle p, \sigma, E \rangle \overset{t,\rho,(\theta_{\text{y}},\theta_{\text{n}},\theta_{\text{s}},\theta_{\text{r}})}{\longmapsto} \langle p', \sigma', E' \rangle$ is that during the time transition, the extended valuation defining the values of the visible variables at each time-point $s \in [0, t]$ is given by $\rho(s)$. At the end-point t, the resulting process is $\langle p', \sigma', E' \rangle$. Furthermore, the quadruple $(\theta_{\text{y}}(s), \theta_{\text{n}}(s), \theta_{\text{s}}(s), \theta_{\text{r}}(s))$ represents four valuations, mapping action labels and/or channels to the values of the associated guards during the delay ($s \in [0, t]$). The first trajectory, θ_{y}, is defined for synchronizing action labels. It defines for each synchronizing action label the conjunction of the guards of all partners involved in the synchronization. The second trajectory, θ_{n} is defined for non-synchronizing action labels and for matching send and receive actions on channels. It defines for each non-synchronizing action label the trajectory of the guard, and for each matching send and receive action on a channel the conjunction of the send and receive guards. The third and forth components, θ_{s} and θ_{r}, define the values of the guards of send and receive actions on channels, respectively.

3. Consistency by means of consistency transitions: $\_ \overset{\sim}{\rightsquigarrow} \_ \subseteq (\mathcal{P}_{\text{abstract}} \times \Sigma_\perp \times \mathcal{E}) \times ((\mathcal{T} \rightharpoonup \dot{\Sigma}) \times 2^{\mathcal{L}_{\text{basic}}}) \times (\mathcal{P}_{\text{abstract}} \times \Sigma \times \mathcal{E})$. The intuition of a consistency transition $\langle p, \sigma_\perp, E \rangle \overset{\rho,A}{\rightsquigarrow} \langle p', \sigma, E' \rangle$, is that the initial valuation $\rho(0)$ of trajectory ρ represents consistent initial conditions for process term p in environment E; the elements from valuation $\sigma_\perp$ that have defined values represent the state part (valuation for the discrete and continuous variables) of the consistent initial conditions; and the process has synchronizing actions A. The initial conditions satisfy the active (see explanation below) equations, active invariants, and active initialization predicates (see Sections 3.6.1, 3.6.2 and 3.7.1).

   Informally, the active part of a sequential process term $p_1; \ldots; p_n$ is p_1, the active part of recursion scope process term $[\![_{\text{R}} \{X_1 \mapsto p_1, \ldots, X_r \mapsto p_r\} :: p_{r+1}]\!]$ is p_{r+1}, the active part distributes over the other operators, and for atomic process terms, the active part is the atomic process term itself.

   The reason for specifying the set of synchronizing actions on a consistency transition is that in this way, the inconsistent process term is a zero element for the synchronizing action operator (see Section 3.7.5 and Property 3.15).

The reason for specifying the trajectories ρ and ρ' on action and consistency transitions instead of (extended) valuations is that values of dotted variables $\dot{x}$ and consistent initial conditions are relevant only in the context of trajectories (on non-empty and non-singleton time intervals). The differential or integral relation between a variable x and its dotted version $\dot{x}$ can be defined only by means of trajectories on a time interval. The trajectories on the action and consistency transitions are also required for properties such as '$\mathsf{eqn}\ \ x = c \ \underline{\leftrightarrow}\ \mathsf{eqn}\ \ x = c \parallel \mathsf{eqn}\ \dot{x} = 0$' and '$\mathsf{eqn}\ y = e \parallel p \ \underline{\leftrightarrow}\ \mathsf{eqn}\ y = e \parallel p[e/y]$', as discussed in Section 2.6.1 and as defined in Section 3.8.3, to hold.

The signatures of the action and time transitions are such that the original valuations are completely defined ($\sigma \in \Sigma$). This means that a process $\langle p, \sigma_\perp, E \rangle$ cannot do action or time transitions if $\sigma_\perp$ contains one or more variables the value of which is undefined.

Several properties of the semantics are presented in Section 3.8.1. Note that for consistency transitions $\langle p, \sigma_\perp, E \rangle \overset{\rho,A}{\rightsquigarrow} \langle p', \sigma, E \rangle$, the new process term p' is different from the original process term p only in case that the active part of process term p contains an initialization operator $u \gg p$ (Section 3.7.1), a recursion variable X (Section 3.6.6), or a scope operator (Sections 3.7.7, 3.7.8,

3.7.9, 3.7.10). The resulting valuation σ is different from $\sigma_\perp$ only when the original valuation $\sigma_\perp$ contains variables the value of which is undefined.

The relations and predicates mentioned above 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 (transition) of the rule. The result of a deduction rule can be derived if all of its hypotheses can be derived. In case the set of hypotheses is empty, the deduction rule is called an axiom.

3.3 Abbreviations for deduction rules

To increase the readability of the Chi deduction rules, some additional abbreviations are used:

- $E \Vdash \langle p, \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle q, \sigma'\rangle$, where $q \in \mathcal{P}_{\text{abstract}} \cup \{\checkmark\}$, is an abbreviation for $\langle p, \sigma, E\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle q, \sigma', E\rangle$.
- $E \Vdash \langle p, \sigma\rangle \overset{t,\rho,\theta}{\longmapsto} \langle q, \sigma'\rangle$ is an abbreviation for $\langle p, \sigma, E\rangle \overset{t,\rho,\theta}{\longmapsto} \langle q, \sigma', E\rangle$.
- $E \Vdash \langle p, \sigma\rangle \overset{\rho,A}{\rightsquigarrow} \langle q, \sigma'\rangle$ is an abbreviation for $\langle p, \sigma, E\rangle \overset{\rho,A}{\rightsquigarrow} \langle q, \sigma', E\rangle$.
- $E \Vdash f_1, \ldots, f_n$, where f_i represents one of the previously defined transition relations (of the forms $\langle p, \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle q, \sigma'\rangle$ or $\langle p, \sigma\rangle \overset{t,\rho,\theta}{\longmapsto} \langle q, \sigma'\rangle$ or $\langle p, \sigma\rangle \overset{\rho,A}{\rightsquigarrow} \langle q, \sigma'\rangle$), is an abbreviation for $E \Vdash f_1, \ldots, E \Vdash f_n$.

Notation

$$\frac{E' \Vdash \langle p_1, \sigma_1\rangle \xrightarrow{\rho_1,\ell_1,b_1,W_1,\rho_1'} \left\langle \begin{matrix} q_{11} \\ \vdots \\ q_{1n} \end{matrix}, \sigma_1' \right\rangle, \ldots, \langle p_m, \sigma_m\rangle \xrightarrow{\rho_m,\ell_m,b_m,W_m,\rho_m'} \left\langle \begin{matrix} q_{m1} \\ \vdots \\ q_{mn} \end{matrix}, \sigma_m' \right\rangle,\ C}{E \Vdash \langle r, \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \left\langle \begin{matrix} s_1 \\ \vdots \\ s_n \end{matrix}, \sigma' \right\rangle} n$$

where $q_{ji}, s_i \in \mathcal{P}_{\text{abstract}} \cup \{\checkmark\}$, $p_i, r \in \mathcal{P}_{\text{abstract}}$, 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):

$$\frac{E' \Vdash \langle p_1, \sigma_1\rangle \xrightarrow{\rho_1,\ell_1,b_1,W_1,\rho_1'} \langle q_{1i}, \sigma_1'\rangle, \ldots, \langle p_m, \sigma_m\rangle \xrightarrow{\rho_m,\ell_m,b_m,W_m,\rho_m'} \langle q_{mi}, \sigma_m'\rangle,\ C}{E \Vdash \langle r, \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle s_i, \sigma'\rangle} n.i$$

Notation

$$\frac{E' \Vdash \langle p_1, \sigma_1\rangle \underset{\rho_1,A_1}{\overset{t,\rho_1,\theta_1}{\longmapsto}} \langle q_1, \sigma_1'\rangle, \ldots, \langle p_i, \sigma_i\rangle \underset{\rho_i,A_i}{\overset{t,\rho_i,\theta_i}{\longmapsto}} \langle q_i, \sigma_i'\rangle}{E \Vdash \langle r, \sigma\rangle \underset{\rho,A}{\overset{t,\rho,\theta}{\longmapsto}} \langle s, \sigma'\rangle} n$$

where n denotes the number of the rule, is an abbreviation for the two rules presented below. A rule for the delay part of Rule n:

$$\frac{E' \Vdash \langle p_1, \sigma_1\rangle \overset{t,\rho_1}{\longmapsto} \langle q_1, \sigma_1'\rangle, \ldots, \langle p_i, \sigma_i\rangle \overset{t,\rho_i,\theta}{\longmapsto} \langle q_i, \sigma_i'\rangle}{E \Vdash \langle r, \sigma\rangle \overset{t,\rho,\theta}{\longmapsto} \langle s, \sigma'\rangle}\, n.\text{a}$$

and a rule for the consistency part of Rule n:

$$\frac{E' \Vdash \langle p_1, \sigma_{\perp 1}\rangle \overset{\rho_1,A_1}{\rightsquigarrow} \langle q_1, \sigma_1'\rangle, \ldots, \langle p_i, \sigma_{\perp i}\rangle \overset{\rho_i,A_i}{\rightsquigarrow} \langle q_i, \sigma_i'\rangle}{E \Vdash \langle r, \sigma_\perp\rangle \overset{\rho,A}{\rightsquigarrow} \langle s, \sigma'\rangle}\, n.\text{b},$$

where $\sigma_\perp, \sigma_{\perp 1} \ldots \sigma_{\perp i}$ are syntactically equal to $\sigma, \sigma_1 \ldots \sigma_i$, respectively. The only difference is that the valuations $\sigma, \sigma_1 \ldots \sigma_i$ of the action transitions are element of Σ, whereas the valuations $\sigma_\perp, \sigma_{\perp 1} \ldots \sigma_{\perp i}$ of the consistency transitions are element of $\Sigma_\perp$.

Furthermore, the following abbreviations are used:

- $\frac{H}{R}$, where R is a number of results separated by commas, is an abbreviation for a set of deduction rules of the form $\frac{H}{r}$; one for each $r \in R$, and notation $E \Vdash \frac{H}{r}$ is an abbreviation for $\frac{E \Vdash H}{E \Vdash r}$.
- $\langle p, \sigma, E\rangle \overset{\lambda}{\rightarrow} \langle p', \sigma', E'\rangle$ is an abbreviation for $\langle p, \sigma, E\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle p', \sigma', E'\rangle$.
- $\langle p, \sigma_\perp, E\rangle \overset{\rho}{\rightsquigarrow}$ is an abbreviation for $\exists_{A,p',\sigma,E'} \langle p, \sigma_\perp, E\rangle \overset{\rho,A}{\rightsquigarrow} \langle p', \sigma, E'\rangle$.

3.4 Additional notations

Let $x \in \dot{\mathcal{V}}$ be a, possibly dotted, variable, $D : \mathcal{D}$ be a dynamic type mapping, $\sigma : \Sigma$ be a valuation, $\xi : \dot{\Sigma}$ be an extended valuation, $e \in \text{Expr}(\dot{\mathcal{V}})$ be an expression, $t \in \mathcal{T}$ be a time-point, $\rho : \mathcal{T} \rightharpoonup \dot{\Sigma}$ be a trajectory, $u \in \text{Pred}(\dot{\mathcal{V}})$ be a predicate, and $U : (\mathcal{L}_{\text{basic}} \cup \{\tau\} \cup \mathcal{H}) \rightharpoonup \mathbb{B}$ be partial function from basic action labels, the τ label, or channels to boolean values. Then the following notations are defined:

- $\sigma(x)$ and $\xi(x)$ denote the value of variable x in valuation σ and in extended valuation ξ, respectively, which corresponds to the usual syntax for function application.
- $\sigma(e)$, $\xi(e)$, and $\rho(t)(e)$ denote the value of expression e evaluated in valuation σ, and in extended valuations ξ and $\rho(t)$, respectively. In these notations, the domains of σ, ξ, and $\rho(t)$ are assumed to include at least the variables occurring in e.
- Notation $\xi \models u$, where the domain of ξ includes at least the variables occurring in predicate u, denotes the truth value of predicate u evaluated in valuation ξ. Thus $\xi \models u$ if and only if $\xi(u)$, where the meaning of $\xi(u)$ is as defined in the previous item.
- $\dot{D}_{\text{cont}} = \{\dot{x} \mid x \in D_{\text{cont}}\}$ denotes the set of dotted versions of the continuous variables.
- ξ_σ is an abbreviation for $\xi \restriction \text{dom}(\sigma)$.
- ρ_σ is an abbreviation for $\rho \downarrow \text{dom}(\sigma)$.
- ξ^- denotes the extended valuation defined by $\text{dom}(\xi^-) = \{x^- \mid x \in \text{dom}(\xi)\}$, and $\xi^-(x^-) = \xi(x)$.

- $\mathrm{def} : \Sigma \to 2^{\Sigma}$ maps a valuation to set of valuations with the same domain. For each valuation in the set, the defined variable-value pairs in σ remain unchanged, and the undefined pairs get arbitrary values. Formally: $\mathrm{def}(\sigma) = \{\sigma' \mid \sigma' : \mathrm{dom}(\sigma) \to \Lambda,\ \forall_{x \in \mathrm{dom}(\sigma)}\, \sigma(x) \neq \bot \Rightarrow \sigma'(x) = \sigma(x)\}$. The condition $\sigma \in \mathrm{def}(\sigma_\bot)$ in the hypothesis of the consistency transition for each atomic process term ensures that all variables that are initially undefined in $\sigma_\bot$ are defined after the first consistency transition.

- $\Omega(D, \sigma, t)$ denotes the set of allowed trajectories for a time transition of duration t (or a consistency transition if $t = 0$) as allowed by the solution concept function Ω_{FG} (which is defined in Section 3.6.1), such that the initial valuation $\rho(0)$ of each trajectory restricted to the variables in the current valuation σ equals σ. Dynamic type mapping D defines the dynamic types (continuous, discrete or algebraic) of the variables. Formally:

  $$\Omega : (\mathcal{D} \times \Sigma \times T) \to 2^{\mathcal{T} \rightharpoonup \dot{\Sigma}}$$

  $$\Omega(D, \sigma, t) = \{\rho \mid \rho \in \Omega_{FG}(D, t),\ \rho_\sigma(0) = \sigma\}$$

  The parameters F and G of the solution concept function are in fact also parameters of the function Ω. We use the sloppy notation Ω, without the parameters, to avoid cluttering a large number of SOS rules with F and G.

- $\Omega_4(D, \sigma)$ denotes the set of all tuples (ρ, ρ', ξ, ξ') such that ρ and ρ' are trajectories for duration zero as allowed by the solution concept function Ω_{FG}. The initial valuation $\rho(0)$ restricted to the variables in σ equals σ, and the valuations ξ and ξ' are the initial valuations of the trajectories ρ and ρ', respectively:

  $$\Omega_4 : (\mathcal{D} \times \Sigma) \to 2^{(\mathcal{T} \rightharpoonup \dot{\Sigma}) \times (\mathcal{T} \rightharpoonup \dot{\Sigma}) \times \dot{\Sigma} \times \dot{\Sigma}}$$

  $$\begin{aligned}\Omega_4(D, \sigma) = \{\ & (\rho, \rho', \xi, \xi') \\ & \mid \rho \in \Omega_{FG}(D, 0),\ \rho' \in \Omega_{FG}(D, 0), \\ & \quad \xi = \rho(0),\ \xi' = \rho'(0),\ \rho_\sigma(0) = \sigma \\ & \}\end{aligned}$$

  In the same way as in the definition of the previous function Ω, the parameters F and G have been omitted in the definition of function Ω_4.

- $\mathrm{urg} : ((\mathcal{L}_{\mathrm{basic}} \cup \{\tau\} \cup \mathcal{H}) \rightharpoonup \mathbb{B}) \to 2^{(\mathcal{L}_{\mathrm{basic}} \cup \{\tau\} \cup \mathcal{H})}$ is a function on an urgency mapping. The function application $\mathrm{urg}(U)$ returns the set of urgent actions and urgent channels from U. Formally: $\mathrm{urg}(U) = \{x \mid x \in \mathrm{dom}(U) \wedge U(x)\}$.

3.5 Conditional expressions

Let ξ denote an extended valuation, e denote an expression, and $\xi(e)$ denote the value of expression e when evaluated for extended valuation ξ. Then:

$$\xi((u_1 \to e_1 \mid \cdots \mid u_n \to e_n)) = \begin{cases} \xi(e_1) & \text{if } \xi(u_1) \\ \vdots & \\ \xi(e_n) & \text{if } \xi(u_n) \end{cases}$$

It is assumed that at least one of the conditions evaluates to true: $\xi \models (u_1 \vee \cdots \vee u_n)$. In case more than one of the conditions are true, it is assumed that the values of the associated expressions are the same. Formally, for all conditional expressions $(u_1 \to e_1 \mid \cdots \mid u_n \to e_n)$ we assume that $\forall_{\xi \in \dot{\Sigma},\ i,j \in \{1,\ldots,n\}}\ \xi \models (u_i \wedge u_j \Rightarrow e_i = e_j)$. Here, $\dot{\Sigma}$ denotes the set of extended valuations the domain of which contains at least the (dotted) variables used in u_i, u_j, e_i, and e_j.

3.6 Deduction rules for atomic process terms

The discussion of the atomic process terms starts with the equation process term, the semantics of which is defined in terms of the solution function Ω, which in turn is defined in terms of the solution concept function Ω_{FG}. This solution concept function is essential for the delay behavior of the other atomic process terms.

3.6.1 Equation process term

An *equation* process terms $\mathsf{eqn}\ u$, where u is a predicate over variables and dotted continuous variables, restricts the allowed behavior of the (dotted) continuous and algebraic variables in such a way that the value of the predicate remains true over time.

$$\frac{\rho \in \Omega(D, \sigma, t),\ \forall_{s \in \mathrm{dom}(\rho)}\ \rho(s) \models u}{(D, U, J, R) \Vdash \langle \mathsf{eqn}\ u, \sigma \rangle \overset{t, \rho, \mathrm{fls}_{t\emptyset(tU)^3}}{\longmapsto} \langle \mathsf{eqn}\ u, \rho_\sigma(t) \rangle}\ 1$$

$$\frac{\rho \in \Omega(D, \sigma, 0),\ \forall_{s \in \mathrm{dom}(\rho)}\ \rho(s) \models u,\ \sigma \in \mathrm{def}(\sigma_\perp)}{(D, U, J, R) \Vdash \langle \mathsf{eqn}\ u, \sigma_\perp \rangle \overset{\rho, \emptyset}{\rightsquigarrow} \langle \mathsf{eqn}\ u, \sigma \rangle}\ 2$$

The function application $\Omega(D, \sigma, t)$ is defined as $\{\rho \mid \rho \in \Omega_{FG}(D, t),\ \rho_\sigma(0) = \sigma\}$ in Section 3.4. This function ensures that the initial valuation $\rho_\sigma(0)$ for the state variables of the trajectory equals the starting valuation σ in Rule 1, and the final valuation σ in Rule 2. This final valuation σ equals the starting valuation $\sigma_\perp$ for all defined variables in $\sigma_\perp$.

The parameter $F \subseteq \mathcal{T} \rightharpoonup \Lambda$ of the solution concept function Ω_{FG} defines the type of solution functions that are allowed for the algebraic variables. The parameter $G \subseteq (\mathcal{T} \rightharpoonup \Lambda) \times (\mathcal{T} \rightharpoonup \Lambda)$ defines the type of solution functions allowed for the continuous variables and for the dotted continuous variables, and also the relation between the solution function of a continuous variable and the solution function of its associated dotted continuous variable (the 'derivative'). For a given F and G, the solution concept function $\Omega_{FG} : (2^{\mathcal{T} \rightharpoonup \Lambda} \times 2^{(\mathcal{T} \rightharpoonup \Lambda) \times (\mathcal{T} \rightharpoonup \Lambda)} \times \mathcal{D} \times \mathcal{T}) \to 2^{\mathcal{T} \rightharpoonup \dot{\Sigma}}$ is formally defined as:

$$\begin{array}{l}
\Omega_{FG}(D, t) = \\
\{\ \rho \\
\mid\ t \geq 0 \\
,\ \exists_{\varepsilon_1, \varepsilon_2 > 0}\ (\ \rho : [-\varepsilon_1, t + \varepsilon_2] \to ((\mathrm{dom}(D) \cup \dot{D}_{\mathrm{cont}}) \to \Lambda) \\
\qquad ,\ \forall_{x \in D_{\mathrm{disc}}} \quad \rho \downarrow x \text{ is a constant function} \\
\qquad ,\ \forall_{x \in D_{\mathrm{alg}}} \quad \rho \downarrow x \in F \\
\qquad ,\ \forall_{x \in D_{\mathrm{cont}}} \quad (\rho \downarrow x, \rho \downarrow \dot{x}) \in G \cap G_0 \\
\qquad ,\ \forall_{s \in \mathrm{dom}(\rho)}\ \rho(s)(\mathsf{time}) = \sigma(\mathsf{time}) + s \\
\qquad)\\
\}
\end{array}$$

The duration t of the time transition t is nonnegative ($t \geq 0$). The reason for introducing ε is discussed below. The trajectory ρ is a function from the time interval $[-\varepsilon_1, t + \varepsilon_2]$ to valuations, where the domain of each valuation consists of the variables and dotted continuous variables defined by dynamic type mapping $\mathcal{D}$, and the codomain is Λ.

The solution function for each discrete variable is restricted to a constant function. The solution function for each algebraic variable ($\rho \downarrow x$ for $x \in D_{\text{alg}}$) is required to be a function of type F. The definition of the trajectory as $\rho : [-\varepsilon_1, t + \varepsilon_2] \rightarrow ((\text{dom}(D) \cup \dot{D}_{\text{cont}}) \rightarrow \Lambda)$ ensures that for all algebraic variables $x \in D_{\text{alg}}$, the type of the solution function for x is defined by $(\rho \downarrow x) : [-\varepsilon_1, t + \varepsilon_2] \rightarrow \Lambda$, which are arbitrary functions of $[-\varepsilon_1, t + \varepsilon_2]$ to values (from the set of all values Λ). Having the set F as a parameter of the solution concept function allows us to restrict the trajectories of the algebraic variables to, for instance, the set of piecewise continuous functions, if this would be required for certain properties to hold.

The set G in the requirement $\forall_{x \in D_{\text{cont}}} (\rho \downarrow x, \rho \downarrow \dot{x}) \in G \cap G_0$ defines the additional requirements for the solution functions for the continuous variables and their associated dotted versions (the 'derivatives'). The set G_0 represents the predefined (minimal) requirements. It is defined as:

$$
\begin{aligned}
G_0 = \{ \ & (f, \dot{f}) \\
\mid \ & \exists_{t_0, t_1 \in \mathcal{T}, t_0 < 0 < t_1} f, \dot{f} : [t_0, t_1] \rightarrow \mathbb{R}, \ \dot{f} \text{ is } L^1 \text{ on } [t_0, t_1] \\
, \ & \forall_{t \in \text{dom}(f)} \ f \text{ is differentiable in } t \Rightarrow \dot{f}(t) = \tfrac{d}{dt} f(t) \\
, \ & \forall_{t \in \text{dom}(f)} \ f(t) = f(0) + \textstyle\int_0^t \dot{f}(s) ds \\
\} &
\end{aligned}
$$

This definition of G_0 requires the solution function $\rho \downarrow \dot{x}$ for the dotted variable $\dot{x}$ to be indeed the derivative function of the solution function $\rho \downarrow x$ for the continuous variable x, for time points where $\rho \downarrow x$ is differentiable ($\dot{f}(t) = \frac{d}{dt} f(t)$). Furthermore, the relation between f and $\dot{f}$ satisfies the Caratheodory solution concept [8] ($f(t) = f(0) + \int_0^s \dot{f}(s) ds$). This ensures that the relation between f and $\dot{f}$ is also defined for the points (if any) where function f is not differentiable. The function $\dot{f}$ is required to be of class L^1, so that the integral relation (in the Lebesgue sense) is defined [19].

The existence of the integral relation implies that $\rho \downarrow x$ is an absolutely continuous function (for all continuous variables), and that $\rho \downarrow x$ is differentiable almost everywhere (for all continuous variables) [19]. Thus the solution concept function Ω_{FG} restricts the trajectory $\rho \downarrow x$ of every continuous variable x to an absolutely continuous function, but it does allow a non-smooth trajectory for a continuous variable in the case that the trajectory of its 'derivative' $\rho \downarrow \dot{x}$ is discontinuous.

The use of the values ε_1, ε_2, and the set G_0 in the solution concept function Ω_{FG}, together wit the use of trajectories on action transitions, delay transitions, and consistency transitions are necessary to ensure four main properties of the semantics:

- The relation between the solution function $\rho \downarrow x$ for a continuous variable x and the solution function $\rho \downarrow \dot{x}$ for its dotted version $\dot{x}$ is a derivative relation where possible, and an integral relation otherwise. In this way, unnecessary spurious discontinuities in the solution function for a dotted continuous variable are prevented. The ε_1, ε_2 extensions of the delay interval $[0, t]$ ensure that differentiation at the beginning and end of the time interval is defined in the same way as differentiation at other points in the interval.
- Conjunction and parallel composition are equivalent for equations (see Property 3.12 in Section 3.8.3)
- A variable which is defined to be equal to an expression, can be replaced by its defining expression in all parallel contexts (substitution property, see Property 3.12 in Section 3.8.3).
- Systems of equations that have hidden constraints are bisimilar to systems of equations in

which the hidden constraints are made explicit (obtained after differentiation), as explained in Section 2.6.1.

For instance, $\mathsf{eqn}\ x = c$, with $c \in \mathbb{R}$ denoting an arbitrary real valued constant, is bisimilar to $\mathsf{eqn}\ x = c \parallel \mathsf{eqn}\ \dot{x} = 0$. Therefore, when the solution function for a continuous variable is constant, as is enforced by $\mathsf{eqn}\ x = c$, the solution function for the associated dotted continuous variable is zero. The integral relation $f(t) = f(0) + \int_0^s f'(s)ds$ is not enough to enforce this, because it allows the solution function for $\dot{x}$ to be a function that is zero almost everywhere, thus allowing spurious discontinuities in the solution function for $\dot{x}$.

Consider, for example, the process

$$\langle x = \mathsf{time}, \{x \mapsto \bot, \mathsf{time} \mapsto 0\}, E\rangle,$$

that defines the value of x to be equal to the value of predefined variable time.

The environment E defines the continuous variable x and the predefined continuous variable time: $E = (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \emptyset, \emptyset, \emptyset)$. The same environment E is used in the two processes below.

The solution function for the derivative $\dot{x}$ is the constant function 1, also at time point 0, without any spurious discontinuities.

Consider on the other hand, the process

$$\langle x = (\mathsf{time} \leq 2 \rightarrow 0 \mid \mathsf{time} > 2 \rightarrow (\mathsf{time} - 2)), \{x \mapsto \bot, \mathsf{time} \mapsto 0\}, E\rangle,$$

where $(\mathsf{time} \leq 2 \rightarrow 0 \mid \mathsf{time} > 2 \rightarrow (\mathsf{time} - 2))$ is a conditional expression as defined in Sections 2.4 and 3.5. The process defines the value of x to be equal to 0 until time point 2. The value of x increases with a derivative of 1 after time point 2. In this case, the solution function for the derivative $\dot{x}$ is the constant function 0, until time point 2, where it changes discontinuously to 1. At time point 2, $\dot{x}$ can have any value (non-determinism).

Consider, as a third example, the process

$$\langle \dot{x} = (\mathsf{time} \leq 2 \rightarrow 0 \mid \mathsf{time} > 2 \rightarrow 1), \{x \mapsto 0, \mathsf{time} \mapsto 0\}, E\rangle.$$

This process defines the value of the dotted variable $\dot{x}$ to be changing discontinuously from 0 to 1 at time point 2. The solution function for the continuous variable x is the continuous function

$$(\rho \downarrow x)(t) = \begin{cases} 0 & \text{if } t \leq 2 \\ t - 2 & \text{if } t \geq 2. \end{cases}$$

Other examples illustrating the properties of the semantics can be found in Section 2.6.1. The properties derived in Section 3.8 are valid for all parameters F and G. For the translation of hybrid automata to Chi, differentiable functions would be required for the solution functions of the continuous variables: $G = \{(f, \dot{f}) \mid f \text{ is differentiable, and } \dot{f} \text{ is the derivative function of } f\}$. In this way, the semantics of the Chi translation corresponds to the semantics of the hybrid

automaton. In other cases, differentiability can be too strong a restriction, so that piecewise continuous functions for the trajectories of the algebraic and dotted variables are assumed: $F = \{f \mid f \text{ is a piecewise continuous function}\}$, $G = \{(f, \dot{f}) \mid \dot{f} \text{ is a piecewise continuous function}\}$. It is also possible not to define additional restrictions: $F = \{f \mid \text{true}\}$, $G = \{(f, \dot{f}) \mid \text{true}\}$. For a process with for each variable just one solution function such as: $\langle\ \mathsf{eqn}\ \dot{x} = y,\ y = (\mathsf{time} < 1 \to 0 \mid \mathsf{time} \geq 1 \to 1),\ \{x \mapsto 0, \mathsf{time} \mapsto 0\},\ (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}, y \mapsto \mathsf{alg}\}, \emptyset, \emptyset)\ \rangle$, the different choices for F and G as proposed above have no effect. For a process that allows infinitely many solutions, such as $\langle\ \mathsf{eqn}\ \text{true},\ \{x \mapsto 0, \mathsf{time} \mapsto 0\},\ (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}, y \mapsto \mathsf{alg}\}, \emptyset, \emptyset)\ \rangle$, the different choices for F and G would obviously result in differences in the solution functions.

3.6.2 Invariant process term

An *invariant* process term $\mathsf{inv}\ u$, where u is a predicate over variables and dotted continuous variables, restricts the allowed behavior of the (dotted) continuous and algebraic variables in such a way that the value of the predicate remains true over the interval $[0, t]$ in the case of a delay transition, or at time point 0 in the case of a consistency transition. The only difference between the equation process term $\mathsf{eqn}\ u$ (see Section 3.6.1) and the invariant process term $\mathsf{inv}\ u$ is that the predicate u of the equation process term should be satisfied for all time points of the domain of the trajectory ρ ($[-\varepsilon, t + \varepsilon]$ for some $\varepsilon > 0$), whereas the predicate u of the invariant process term needs to be satisfied only for the time points in the interval $[0, t]$. More information relating invariant process terms to equation process terms can be found in Section 2.6.1.

$$\frac{\rho \in \Omega(D, \sigma, t),\ \forall_{s \in [0,t]}\, \rho(s) \models u}{(D, U, J, R) \Vdash \langle \mathsf{inv}\ u, \sigma \rangle \overset{t, \rho, \mathrm{fls}_{t\emptyset(tU)^3}}{\longmapsto} \langle \mathsf{inv}\ u, \rho_\sigma(t) \rangle}\ 3$$

$$\frac{\rho \in \Omega(D, \sigma, 0),\ \rho(0) \models u,\ \sigma \in \mathrm{def}(\sigma_\perp)}{(D, U, J, R) \Vdash \langle \mathsf{inv}\ u, \sigma_\perp \rangle \overset{\rho, \emptyset}{\rightsquigarrow} \langle \mathsf{inv}\ u, \sigma \rangle}\ 4$$

3.6.3 Time can progress process term

The predicate u in the time can progress process term $\mathsf{tcp}\ u$ is a predicate over variables and dotted continuous variables. It allows delays for as long as the predicate u is true, or in other words, until the time-point when the tcp predicate is false. Unlike the equation and invariant process terms $\mathsf{eqn}\ u$ and $\mathsf{inv}\ u$, which are also required to hold in actions to ensure consistency, the time can progress process terms are irrelevant for the action behavior of processes (see the end of Section 2.6.6 for more information). The set consisting of the time point zero $\{0\}$ is included in condition $s \in [0, t) \cup \{0\}$ in Rule 5, because the interval $[0, t)$ equals the empty set for $t = 0$.

$$\frac{\rho \in \Omega(D, \sigma, t),\ \forall_{s \in [0,t) \cup \{0\}}\, \rho(s) \models u}{(D, U, J, R) \Vdash \langle \mathsf{tcp}\ u, \sigma \rangle \overset{t, \rho, \mathrm{fls}_{t\emptyset(tU)^3}}{\longmapsto} \langle \mathsf{tcp}\ u, \rho_\sigma(t) \rangle}\ 5$$

$$\frac{\rho \in \Omega(D, \sigma, 0),\ \sigma \in \mathrm{def}(\sigma_\perp)}{(D, U, J, R) \Vdash \langle \mathsf{tcp}\ u, \sigma_\perp \rangle \overset{\rho, \emptyset}{\rightsquigarrow} \langle \mathsf{tcp}\ u, \sigma \rangle}\ 6$$

3.6.4 Guarded action update process term

Guarded action update process term $u \to a : W : r$ denotes delayable (Rule 8) instantaneous changes to the variables from set W and to the globally defined jumping variables J, by means of an action labeled $a \in \mathcal{L}_{\text{basic}} \cup \{\tau\}$, such that the guard u and predicate r are satisfied, see Rule 7.

The values of the variables from $\text{dom}(\sigma)$ in ξ are given by σ, and the dotted variables $\dot{D}_{\text{cont}}$ and the algebraic variables D_{alg} in ξ can in principle take any value ($(\rho, \rho', \xi, \xi') \in \Omega_4(D, \sigma)$ in Rule 7) as long as the guard and update predicate r are satisfied ($\xi \models u,\ \xi^- \cup \xi' \models r$). Variables occurring with a '$^-$' superscript in r are evaluated in ξ^-, which denotes the extended valuation with the values of variables before the discrete change. The guard u is also evaluated using the values for variables before the discrete change (in extended valuation ξ). The other ('normal') variables in r are evaluated in ξ', using the values for the variables *after* the discrete change. The valuation σ_{s} represents the values for the non-jumping state variables, that is the variables in σ other than the jumping variables $J \cup W$ ($\sigma_{\text{s}} = \sigma \upharpoonright \overline{J \cup W}$). The values of these variables remain unchanged in the action transition.

The guarded action update process term $u \to a : W : r$ allows arbitrary time transitions for non-urgent action labels a ($U(a)$ is false). These time transitions need to satisfy only the general solution function requirements ($\rho \in \Omega(D, \sigma, t)$, see Rule 8). These requirements ensure, among others, that trajectories of discrete variables are constant and that the trajectory of each continuous variable is an absolutely continuous function that starts with the value of the continuous variable in σ.

For urgent action labels a ($U(a)$ holds), the guard trajectory $\theta_{ta\rho u}$ (defined in Section 3.1.2) should be false at all time points (possibly excluding the last time point) of the delay, (see Rule 8). This behavior is analogous to the delay behavior of the tcp process term as defined in Section 3.6.3. The three guard trajectories fls_{tU} are trajectories over the interval $[0, t]$ to valuations from the set of all basic action labels and channels (the domain U from the environment) to false. The guard trajectory $\text{fls}_{t\emptyset}$ is a trajectory over $[0, t]$ to a valuation on an empty domain.

$$\frac{(\rho, \rho', \xi, \xi') \in \Omega_4(D, \sigma),\ \sigma_{\text{s}} = \sigma \upharpoonright \overline{J \cup W},\ \xi'_{\sigma_{\text{s}}} = \sigma_{\text{s}},\ \xi \models u,\ \xi^- \cup \xi' \models r}{(D, U, J, R) \Vdash \langle u \to a : W : r,\ \sigma \rangle \xrightarrow{\rho, a, \text{false}, W \cap \text{dom}(\sigma), \rho'} \langle \checkmark, \xi'_\sigma \rangle} \ 7$$

$$\frac{\rho \in \Omega(D, \sigma, t),\ U(a) \Rightarrow \forall_{s \in [0,t) \cup \{0\}} \neg \theta_{ta\rho u}(s)(a)}{(D, U, J, R) \Vdash \langle u \to a : W : r,\ \sigma \rangle \overset{t, \rho, (\text{fls}_{t\emptyset}, \text{fls}_{tU} \vee \theta_{ta\rho u}, \text{fls}_{tU}, \text{fls}_{tU})}{\longmapsto} \langle u \to a : W : r,\ \rho_\sigma(t) \rangle} \ 8$$

$$\frac{\rho \in \Omega(D, \sigma, 0),\ \sigma \in \text{def}(\sigma_\perp)}{(D, U, J, R) \Vdash \langle u \to a : W : r,\ \sigma_\perp \rangle \overset{\rho, \emptyset}{\rightsquigarrow} \langle u \to a : W : r, \sigma \rangle} \ 9$$

3.6.5 Guarded send and receive update process term

Guarded send and receive update process terms ($u \to h\,!\,\mathbf{e}_n : W : r$) and ($u \to h\,?\,\mathbf{x}_n : W : r$) denote delayable sending (Rule 13.a) of expression(s) $\mathbf{e}_n$ via channel h, and delayable receiving (Rule 13.b) of information via channel h into variable(s) $\mathbf{x}_n$, respectively, in such a way that guard u and predicate r are satisfied, allowing the variables from the sets W and J (and in case of the send process term the variables $\mathbf{x}_n$) to change. Evaluation of the guard u and update part $W : r$ occurs in the same way as in Rule 7.

The values of expressions $e_1, \ldots, e_n$ which are sent via channel h are evaluated in extended valuation ξ, see Rule 10, where $\mathbf{e}_n$ denotes $e_1, \ldots, e_n$, $[\xi(\mathbf{e}_n)]$ denotes the list of values $[\xi(e_1), \ldots, \xi(e_n)]$ for $n \geq 1$, and $\xi(e)$ denotes the value of expression e for extended valuation ξ. The case that n equals 0, represents the case where nothing is sent via the channel, and $\mathbf{e}_0$ and $[\xi(\mathbf{e}_0)]$ denote an empty expression and an empty list, respectively. For $n \geq 1$, the receive process term $h\,?\,\mathbf{x}_n$ can receive the list of values $[\mathbf{c}_n]$, see Rule 11, where $\mathbf{x}_n$ denotes $x_1, \ldots, x_n$, $\{\mathbf{x}_n\}$ denotes the set $\{x_1, \ldots, x_n\}$, $[\mathbf{c}_n]$ denotes the list of values $[c_1, \ldots, c_n]$, and $\xi'(\mathbf{x}_n) = \mathbf{c}_n$ is an abbreviation for $\xi'(x_1) = c_1, \ldots, \xi'(x_n) = c_n$. For $n = 0$, nothing is received, so that $\mathbf{x}_0$ and $\mathbf{c}_0$ are empty, and $\xi'(\mathbf{x}_0) = \mathbf{c}_0$ always holds.

The only use of the guarded communication update process term $u \rightarrow h\,!?\,\mathbf{x}_n := \mathbf{e}_n : W : r$ is to enable the elimination of parallel composition. For example, it allows rewriting of the parallel composition $h\,!\,\mathbf{e}_n \parallel h\,?\,\mathbf{x}_n$ as the alternative composition $h\,!\,\mathbf{e}_n;\ h\,?\,\mathbf{x}_n \ [\!]\ h\,?\,\mathbf{x}_n;\ h\,!\,\mathbf{e}_n \ [\!]\ h\,!?\,\mathbf{x}_n := \mathbf{e}_n$.

The meaning of the update part $W : r$ in the rules 10, 11, and 12 is analogous to the meaning of the update part $W : r$ in Rule 7. The delay behavior of the communication update process term as specified in Rule 14 is analogous to the delay behavior of the action update process term as specified in Rule 8. The delay behavior of the send and receive update process terms, as specified in Rule 13, is not restricted in the case of urgent channels, because separate send and receive actions cannot be urgent. Only the communication action, that is a result of the simultaneous execution of a send and receive action, can be urgent (see Rule 14). To understand the difference between the four guard trajectories in the delay transitions, see the explanation of Rule 27 in Section 3.7.4 on the semantics of the parallel composition operator.

$$\frac{(\rho, \rho', \xi, \xi') \in \Omega_4(D, \sigma),\ \sigma_{\mathrm{s}} = \sigma \upharpoonright \overline{J \cup W},\ \xi'_{\sigma_{\mathrm{s}}} = \sigma_{\mathrm{s}},\ \xi \models u,\ \xi^- \cup \xi' \models r}{(D, U, J, R) \Vdash \langle u \rightarrow h\,!\,\mathbf{e}_n : W : r,\ \sigma\rangle \xrightarrow{\rho,\ h![\xi(\mathbf{e}_n)],\ \mathrm{false},\ W \cap \mathrm{dom}(\sigma),\ \rho'} \langle \surd, \xi'_\sigma\rangle} \quad 10$$

$$\frac{\begin{array}{c}(\rho, \rho', \xi, \xi') \in \Omega_4(D, \sigma),\ \sigma_{\mathrm{s}} = \sigma \upharpoonright \overline{J \cup \{\mathbf{x}_n\} \cup W},\ \xi'_{\sigma_{\mathrm{s}}} = \sigma_{\mathrm{s}},\\ \xi'(\mathbf{x}_n) = \mathbf{c}_n,\ \xi \models u,\ \xi^- \cup \xi' \models r\end{array}}{(D, U, J, R) \Vdash \langle u \rightarrow h\,?\,\mathbf{x}_n : W : r,\ \sigma\rangle \xrightarrow{\rho,\ h?[\mathbf{c}_n],\ \mathrm{false},\ (\{\mathbf{x}_n\} \cup W) \cap \mathrm{dom}(\sigma),\ \rho'} \langle \surd, \xi'_\sigma\rangle} \quad 11$$

$$\frac{\begin{array}{c}(\rho, \rho', \xi, \xi') \in \Omega_4(D, \sigma),\ \sigma_{\mathrm{s}} = \sigma \upharpoonright \overline{J \cup \{\mathbf{x}_n\} \cup W},\ \xi'_{\sigma_{\mathrm{s}}} = \sigma_{\mathrm{s}},\\ \xi'(\mathbf{x}_n) = \xi(\mathbf{e}_n) = \mathbf{c}_n,\ \xi \models u,\ \xi^- \cup \xi' \models r\end{array}}{(D, U, J, R) \Vdash \langle u \rightarrow h\,!?\,\mathbf{x}_n := \mathbf{e}_n : W : r,\ \sigma\rangle \xrightarrow{\rho,\ h!?[\mathbf{c}_n],\ \mathrm{false},\ (\{\mathbf{x}_n\} \cup W) \cap \mathrm{dom}(\sigma),\ \rho'} \langle \surd, \xi'_\sigma\rangle} \quad 12$$

$$\frac{\rho \in \Omega(D, \sigma, t)}{\begin{array}{c}E \Vdash \langle u \rightarrow h\,!\,\mathbf{e}_n : W : r,\ \sigma\rangle \overset{t, \rho, (\mathrm{fls}_{t\emptyset}, \mathrm{fls}_{tU}, \mathrm{fls}_{tU} \vee \theta_{th\rho u}, \mathrm{fls}_{tU})}{\longmapsto} \langle u \rightarrow h\,!\,\mathbf{e}_n : W : r,\ \rho_\sigma(t)\rangle,\\ \langle u \rightarrow h\,?\,\mathbf{x}_n : W : r,\ \sigma\rangle \overset{t, \rho, (\mathrm{fls}_{t\emptyset}, \mathrm{fls}_{tU}, \mathrm{fls}_{tU}, \mathrm{fls}_{tU} \vee \theta_{th\rho u})}{\longmapsto} \langle u \rightarrow h\,?\,\mathbf{x}_n : W : r,\ \rho_\sigma(t)\rangle\end{array}} \quad 13$$

$$\frac{\rho \in \Omega(D, \sigma, t),\ U(h) \Rightarrow \forall_{s \in [0,t) \cup \{0\}}\, \neg\theta_{th\rho u}(s)(h)}{\begin{array}{c}(D, U, J, R) \Vdash \langle u \rightarrow h\,!?\,\mathbf{x}_n := \mathbf{e}_n : W : r,\ \sigma\rangle \overset{t, \rho, (\mathrm{fls}_{t\emptyset}, \mathrm{fls}_{tU} \vee \theta_{th\rho u}, \mathrm{fls}_{tU}, \mathrm{fls}_{tU})}{\longmapsto}\\ \langle u \rightarrow h\,!?\,\mathbf{x}_n := \mathbf{e}_n : W : r,\ \rho_\sigma(t)\rangle\end{array}} \quad 14$$

$$\frac{\rho \in \Omega(D,\sigma,0),\ \sigma \in \mathrm{def}(\sigma_\perp)}{\begin{array}{ll}(D,U,J,R) \Vdash & \langle u \to h\,!\,\mathbf{e}_n : W : r,\ \sigma_\perp\rangle \overset{\rho,\emptyset}{\rightsquigarrow} \langle u \to h\,!\,\mathbf{e}_n : W : r, \sigma\rangle,\\ & \langle u \to h\,?\,\mathbf{x}_n : W : r,\ \sigma_\perp\rangle \overset{\rho,\emptyset}{\rightsquigarrow} \langle u \to h\,?\,\mathbf{x}_n : W : r, \sigma\rangle,\\ & \langle u \to h\,!?\,\mathbf{x}_n := \mathbf{e}_n : W : r,\ \sigma_\perp\rangle \overset{\rho,\emptyset}{\rightsquigarrow} \langle u \to h\,!?\,\mathbf{x}_n := \mathbf{e}_n : W : r, \sigma\rangle\end{array}}\ 15$$

3.6.6 Recursion variable process term

A recursion variable process term X behaves as the process term given by $R(X)$. Here $R(X)$ is the process term that is defined for recursion variable X in recursion mapping R. This is equivalent to syntactically replacing recursion variable X by its defining process term $R(X)$. Recursion mapping R can be defined in the environment of the Chi process directly, or by means of the recursion scope operator, see Section 3.7.7.

$$(D,U,J,R) \Vdash \frac{\langle R(X),\sigma\rangle \xrightarrow{\lambda} \langle \begin{smallmatrix}\checkmark\\ p'\end{smallmatrix}, \sigma'\rangle}{\langle X,\sigma\rangle \xrightarrow{\lambda} \langle \begin{smallmatrix}\checkmark\\ p'\end{smallmatrix}, \sigma'\rangle}\ 16$$

$$(D,U,J,R) \Vdash \frac{\langle R(X),\sigma\rangle \overset{t,\rho,\theta}{\underset{\rho,A}{\longmapsto}} \langle p',\sigma'\rangle}{\langle X,\sigma\rangle \overset{t,\rho,\theta}{\underset{\rho,A}{\longmapsto}} \langle p',\sigma'\rangle}\ 17$$

3.7 Deduction rules for operators

3.7.1 Initialization operator

The initialization process term $u \gg p$ restricts the consistent initial conditions of process term p to those initial conditions that satisfy predicate u. The initialization predicate u in $u \gg p$ is similar to the initial condition of a hybrid automaton: actions or delays in a hybrid automaton can take place only from an initial state that satisfies the initial condition and invariant of the initial location. In case multiple automata are composed in a parallel composition, the initial conditions of the parallel automata are taken into consideration simultaneously. This behavior is reflected in Chi by the property $(u \gg p \parallel u' \gg q \underline{\leftrightarrow} u \wedge u' \gg (p \parallel q)$, see Property 3.9 in Section 3.8.3). The initial state specified by the initialization predicate u is obtained by means of a consistency transition. There is no other behavior for the initialization process term.

$$E \Vdash \frac{\langle p,\sigma_\perp\rangle \overset{\rho,A}{\rightsquigarrow} \langle p',\sigma\rangle,\ \rho(0) \models u}{\langle u \gg p,\sigma_\perp\rangle \overset{\rho,A}{\rightsquigarrow} \langle p',\sigma\rangle}\ 18$$

Consider for example the process

$$\langle\ \dot{x} = 0 \gg \mathsf{eqn}\ \dot{x} = -x + 1$$

$$
\begin{array}{l}
\langle\ \{x \mapsto \bot, \mathsf{time} \mapsto 0\} \\
,\ \{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \emptyset, \emptyset, \emptyset\}) \\
,\ (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \emptyset, \emptyset, \emptyset\}) \\
\rangle
\end{array}
$$

This process can do a consistency transition to the process

$$
\begin{array}{l}
\langle\ \mathsf{eqn}\ \dot{x} = -x + 1 \\
,\ \{x \mapsto 1, \mathsf{time} \mapsto 0\} \\
,\ (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \emptyset, \emptyset, \emptyset\}) \\
\rangle
\end{array}
$$

The initialization predicate $\dot{x} = 0$ defines steady state initialization (derivatives are zero). Note that restrictions on algebraic or on dotted continuous variables specified in the initialization predicate u of process term $u \gg p$ have effect only if the algebraic or dotted continuous variable is related to a state variable (continuous or discrete variable) by means of an equation, invariant or initialization predicate. Consider as a second example the process

$$
\begin{array}{l}
\langle\ y = 2 \gg \mathsf{eqn}\ \dot{x} = 3,\ y = 2x \\
,\ \{x \mapsto \bot, \mathsf{time} \mapsto 0\} \\
,\ (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}, y \mapsto \mathsf{alg}\}, \emptyset, \emptyset, \emptyset\}) \\
\rangle,
\end{array}
$$

Here, the algebraic variable y is related to the state variable x by means of the equation $y = 2x$. The process can do a consistency transition to the process

$$
\begin{array}{l}
\langle\ \mathsf{eqn}\ \dot{x} = 3,\ y = 2x \\
,\ \{x \mapsto 1, \mathsf{time} \mapsto 0\} \\
,\ (\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}, y \mapsto \mathsf{alg}\}, \emptyset, \emptyset, \emptyset\}) \\
\rangle,
\end{array}
$$

In the example process below, however, initialization of the algebraic variable ($y = 2 \gg \ldots$) has no effect.

$$
\begin{array}{l}
\langle\ y = 2 \gg n := y \\
,\ \{n \mapsto \bot, \mathsf{time} \mapsto 0\} \\
,\ (\{n \mapsto \mathsf{disc}, \mathsf{time} \mapsto \mathsf{cont}, y \mapsto \mathsf{alg}\}, \emptyset, \emptyset, \emptyset\}) \\
\rangle
\end{array}
$$

The process can de a consistency transition to the process

$$
\begin{array}{l}
\langle\ n := y \\
,\ \{n \mapsto c, \mathsf{time} \mapsto 0\} \\
,\ (\{n \mapsto \mathsf{disc}, \mathsf{time} \mapsto \mathsf{cont}, y \mapsto \mathsf{alg}\}, \emptyset, \emptyset, \emptyset\}) \\
\rangle,
\end{array}
$$

where c can be any value. The value of the discrete variable n after execution of the assignment $n := y$ can be any value, since the value of algebraic variable y is not restricted by any predicate. For an example of the initialization operator in combination with the variable scope operator, see Section 4.3.9 on the definition of the delay process term.

3.7.2 Sequential composition operator

The sequential composition of process terms p and q behaves as process term p until p terminates, and then continues to behave as process term q. When p terminates, the extended valuation $\rho'(0)$ represents consistent initial conditions for q (see Rule 19).

$$E \Vdash \frac{\langle p, \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \checkmark, \sigma'\rangle,\ \langle q, \sigma'\rangle \overset{\rho',A}{\rightsquigarrow} \langle q', \sigma''\rangle}{\langle p;\ q, \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle q', \sigma'\rangle}\ 19 \qquad E \Vdash \frac{\langle p, \sigma\rangle \xrightarrow{\lambda} \langle p', \sigma'\rangle}{\langle p;\ q, \sigma\rangle \xrightarrow{\lambda} \langle p';\ q, \sigma'\rangle}\ 20$$

$$E \Vdash \frac{\langle p, \sigma\rangle \overset{t,\rho,\theta}{\underset{\rho,A}{\longmapsto}} \langle p', \sigma'\rangle}{\langle p;\ q, \sigma\rangle \overset{t,\rho,\theta}{\underset{\rho,A}{\longmapsto}} \langle p';\ q, \sigma'\rangle}\ 21$$

3.7.3 Alternative composition operator

Applying the alternative composition operator to process terms p and q models a nondeterministic choice between p and q for action transitions. Process term p can perform action transitions only if the initial extended valuation $\rho(0)$ represents consistent initial conditions for process term q, as specified in Rule 22.

Consider for example the following process term: $\mathsf{inv}\ y = 1 \ [\!]\ x := y$. This corresponds to a hybrid automaton with one location, invariant $y = 1$, and an outgoing edge with assignment $x := y$. The invariant $y = 1$ ensures that the value of y equals 1 when the outgoing edge is taken.

The passage of time cannot result in making a choice between p and q, since the time transitions of the process terms p and q have to synchronize to obtain the time transition (with the same time step t and trajectory ρ) of their alternative composition as defined by the delay part of Rule 23. The resulting expression $\theta_p \vee \theta_q$ denotes the disjunction of the individual guard trajectories for p and q. Here θ_p denotes the quadruple $(\theta_{\mathrm{y}p}, \theta_{\mathrm{n}p}, \theta_{\mathrm{s}p}, \theta_{\mathrm{r}p})$, θ_q denotes the quadruple $(\theta_{\mathrm{y}q}, \theta_{\mathrm{n}q}, \theta_{\mathrm{s}q}, \theta_{\mathrm{r}q})$, and $\theta_p \vee \theta_q$ denotes the quadruple $(\theta_{\mathrm{y}p} \vee \theta_{\mathrm{y}q}, \theta_{\mathrm{n}p} \vee \theta_{\mathrm{n}q}, \theta_{\mathrm{s}p} \vee \theta_{\mathrm{s}q}, \theta_{\mathrm{r}p} \vee \theta_{\mathrm{r}q})$. The disjunction of the guard trajectories is taken because the alternative composition $p \ [\!]\ q$ can delay until p or q has an enabled urgent action. The consistency part of this rule defines that the set of synchronizing actions of an alternative composition of process terms is the union of the set of synchronizing actions of the elements.

$$E \Vdash \frac{\langle p, \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma'\rangle,\ \langle q, \sigma\rangle \overset{\rho}{\rightsquigarrow}}{\langle p \ [\!]\ q, \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma'\rangle,\ \langle q \ [\!]\ p, \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma'\rangle}\ 22$$

$$E \Vdash \frac{\langle p,\sigma\rangle \overset{t,\rho,\theta_p}{\underset{\rho,A}{\longmapsto}} \langle p',\sigma'\rangle,\ \langle q,\sigma\rangle \overset{t,\rho,\theta_q}{\underset{\rho,B}{\longmapsto}} \langle q',\sigma'\rangle}{\langle p \mathbin{[\![} q,\sigma\rangle \overset{t,\rho,\theta_p\vee\theta_q}{\underset{\rho,A\cup B}{\longmapsto}} \langle p' \mathbin{[\![} q',\sigma'\rangle} \ 23$$

3.7.4 Parallel composition operator

The parallel composition allows the synchronization of matching send and receive actions as defined by Rule 24. A send action $h!cs$ and a receive action $h'?cs'$ match iff $h = h'$ and $cs = cs'$; i.e. the channels used for sending and receiving are the same, and also the values sent and the values received are identical. Furthermore, the resulting trajectories ρ' of the send action and the receive action have to be the same. To allow the values of shared variables to change in the sending process term due to changes in their values caused by the receiving process term, and vice versa, the process term that causes the changes adds the set of changed variables to the set of jumping variables of the environment of the parallel process term. This is achieved by means of the sets W_1 and W_2 (see Rule 24 and also Rule 25). The result of the synchronization is a communication action that is represented by $h!?cs$.

The parallel composition requires synchronization of equal basic action labels in the case that the action label is synchronizing in both operands of the parallel composition (third parameter of the action transition label is true, see Rule 25).

The parallel composition of process terms p and q has as its behavior with respect to non-synchronizing action transitions the interleaving of the behaviors of p and q (see Rule 26). Action transitions are non-synchronizing if the action label ℓ of the action transition is non-synchronizing (accompanying parameter b is false), or if the action label ℓ is non-synchronizing in the other operand of the parallel composition ($\ell \notin A$). The purpose of the consistency transition $\langle q,\sigma\rangle \overset{\rho,A}{\rightsquigarrow} \langle q',\sigma''\rangle$ is to ensure consistency and to obtain the set of synchronizing action labels A. The only purpose of the consistency transition $\langle q',\sigma'\rangle \overset{\rho'}{\rightsquigarrow}$ is to ensure consistency.

$$\frac{\begin{array}{c}(D,U,J\cup W_2,R) \Vdash \langle p,\sigma\rangle \xrightarrow{\rho,h!cs,b_1,W_1,\rho'} \left\langle \begin{array}{c}\checkmark\\ p'\\ \checkmark\\ p'\end{array},\sigma'\right\rangle,\\ (D,U,J\cup W_1,R) \Vdash \langle q,\sigma\rangle \xrightarrow{\rho,h?cs,b_2,W_2,\rho'} \left\langle \begin{array}{c}\checkmark\\ \checkmark\\ q'\\ q'\end{array},\sigma'\right\rangle\end{array}}{\begin{array}{c}(D,U,J,R) \Vdash \langle p \parallel q,\sigma\rangle \xrightarrow{\rho,h!?cs,\mathrm{false},W_1\cup W_2,\rho'} \left\langle \begin{array}{c}\checkmark\\ p'\\ q'\\ p' \parallel q'\end{array},\sigma'\right\rangle,\\ \langle q \parallel p,\sigma\rangle \xrightarrow{\rho,h!?cs,\mathrm{false},W_1\cup W_2,\rho'} \left\langle \begin{array}{c}\checkmark\\ p'\\ q'\\ q' \parallel p'\end{array},\sigma'\right\rangle\end{array}} \ 24$$

$$\frac{(D, U, J \cup W_2, R) \Vdash \langle p, \sigma \rangle \xrightarrow{\rho, \ell, \text{true}, W_1, \rho'} \left\langle \begin{smallmatrix} \checkmark \\ p' \\ \checkmark \\ p' \end{smallmatrix}, \sigma' \right\rangle \quad (D, U, J \cup W_1, R) \Vdash \langle q, \sigma \rangle \xrightarrow{\rho, \ell, \text{true}, W_2, \rho'} \left\langle \begin{smallmatrix} \checkmark \\ \checkmark \\ q' \\ q' \end{smallmatrix}, \sigma' \right\rangle}{(D, U, J, R) \Vdash \langle p \parallel q, \sigma \rangle \xrightarrow{\rho, \ell, \text{true}, W_1 \cup W_2, \rho'} \left\langle \begin{smallmatrix} \checkmark \\ p' \\ q' \\ p' \parallel q' \end{smallmatrix} \quad , \sigma' \right\rangle} \; 25$$

$$E \Vdash \frac{\langle p, \sigma \rangle \xrightarrow{\rho, \ell, b, W, \rho'} \langle \begin{smallmatrix} \checkmark \\ p' \end{smallmatrix}, \sigma' \rangle, \; \langle q, \sigma \rangle \overset{\rho, A}{\rightsquigarrow} \langle q', \sigma'' \rangle, \; \neg b \vee \ell \notin A, \; \langle q', \sigma' \rangle \overset{\rho'}{\rightsquigarrow}}{\langle p \parallel q, \sigma \rangle \xrightarrow{\rho, \ell, b, W, \rho'} \langle \begin{smallmatrix} q' \\ p' \parallel q' \end{smallmatrix}, \sigma' \rangle, \; \langle q \parallel p, \sigma \rangle \xrightarrow{\rho, \ell, b, W, \rho'} \langle \begin{smallmatrix} q' \\ q' \parallel p' \end{smallmatrix}, \sigma' \rangle} \; 26$$

Rule 27 specifies that the time transitions of the process terms that are put in parallel synchronize on the time, and on the variable trajectory. The resulting quadruple of guard trajectories

$$(\theta_{\mathrm{y}pq}, \theta_{\mathrm{n}pq}, \theta_{\mathrm{s}p} \vee \theta_{\mathrm{s}q}, \theta_{\mathrm{r}p} \vee \theta_{\mathrm{r}q})$$

consists of the following elements:

- $\theta_{\mathrm{y}pq} = \theta_{\mathrm{y}p} \wedge \theta_{\mathrm{y}q}$: the conjunction of the two guard trajectories for the synchronizing action labels.
- $\theta_{\mathrm{n}pq} = (\theta_{\mathrm{n}p} \vee \theta_{\mathrm{n}q}) \vee (\theta_{\mathrm{s}p} \wedge \theta_{\mathrm{r}q}) \vee (\theta_{\mathrm{r}p} \wedge \theta_{\mathrm{s}q})$: the disjunction of
  - $\theta_{\mathrm{n}p} \vee \theta_{\mathrm{n}q}$: the disjunction of the guard trajectories for the non-synchronizing action labels from p and q,
  - $\theta_{\mathrm{s}p} \wedge \theta_{\mathrm{r}q}$: the conjunction of the guard trajectories for the send actions from p and matching receive actions from q, and
  - $\theta_{\mathrm{r}p} \wedge \theta_{\mathrm{s}q}$: the conjunction of the guard trajectories for the receive actions from p and matching send actions from q.
- $\theta_{\mathrm{s}p} \vee \theta_{\mathrm{s}q}$: the disjunction of the send actions from p and q.
- $\theta_{\mathrm{r}p} \vee \theta_{\mathrm{r}q}$: the disjunction of the receive actions from p and q.

The resulting time transition is possible only for as long as the disjunction of the resulting guard trajectories $\theta_{\mathrm{y}pq}$ and $\theta_{\mathrm{n}pq}$ is false. Note that $\theta_{\mathrm{n}p}(s)$, $\theta_{\mathrm{n}q}(s)$, and $\theta_{\mathrm{n}pq}(s)$ are complete functions on $\mathrm{dom}(U)$ for all $s \in [0, t]$ (whereas $\theta_{\mathrm{y}pq}(s)$ is a partial function on $\mathrm{dom}(U)$). Therefore, the disjunction $(\theta_{\mathrm{y}pq} \vee \theta_{\mathrm{n}pq})(s)$ is also a complete function on $\mathrm{dom}(U)$ for all $s \in [0, t]$. The resulting guard trajectories for the send and receive actions $\theta_{\mathrm{s}p} \vee \theta_{\mathrm{s}q}$ and $\theta_{\mathrm{r}p} \vee \theta_{\mathrm{r}q}$, respectively, do not affect the duration of time transitions, since channels are assumed to allow synchronous communication only.

The notation U_0 denotes an urgency mapping with the same domain as U, where all action labels and channels are defined as non-urgent. Formally, $U_0 : (\mathcal{L}_{\text{basic}} \cup \{\tau\} \cup \mathcal{H}) \rightharpoonup \mathbb{B}$, such that

$\mathrm{dom}(U_0) = \mathrm{dom}(U)$, and $\forall_{x \in \mathrm{dom}(U_0)}\, \neg U_0(x)$. By replacing the urgency mapping U of the result by the non-urgent urgency mapping U_0 in the hypothesis, the time transitions for the operands of the parallel composition are constructed based on *non-urgent* actions and channels.

$$\frac{(D, U_0, J, R) \Vdash \langle p, \sigma \rangle \overset{t,\rho,(\theta_{\mathrm{y}_p},\theta_{\mathrm{n}p},\theta_{\mathrm{s}p},\theta_{\mathrm{r}p})}{\longmapsto} \langle p', \sigma' \rangle,\ \langle q, \sigma \rangle \overset{t,\rho,(\theta_{\mathrm{y}_q},\theta_{\mathrm{n}q},\theta_{\mathrm{s}q},\theta_{\mathrm{r}q})}{\longmapsto} \langle q', \sigma' \rangle,\ \forall_{s \in [0,t) \cup \{0\}, x \in \mathrm{urg}(U)}\, \neg(\theta_{\mathrm{y}_{pq}} \vee \theta_{\mathrm{n}pq})(s)(x)}{(D, U, J, R) \Vdash \langle p \parallel q, \sigma \rangle \overset{t,\rho,(\theta_{\mathrm{y}_{pq}},\theta_{\mathrm{n}pq},\theta_{\mathrm{s}p} \vee \theta_{\mathrm{s}q},\theta_{\mathrm{r}p} \vee \theta_{\mathrm{r}q})}{\longmapsto} \langle p' \parallel q', \sigma' \rangle} \quad 27$$

The consistency transitions of the process terms that are put in parallel synchronize, and take the union of the sets of synchronizing actions, in the same way as for alternative composition, see Rules 23.b and 28.

$$E \Vdash \frac{\langle p, \sigma_\perp \rangle \overset{\rho,A}{\rightsquigarrow} \langle p', \sigma \rangle,\ \langle q, \sigma_\perp \rangle \overset{\rho,B}{\rightsquigarrow} \langle q', \sigma \rangle}{\langle p \parallel q, \sigma_\perp \rangle \overset{\rho,A \cup B}{\rightsquigarrow} \langle p' \parallel q', \sigma \rangle} \quad 28$$

3.7.5 Synchronizing action operator

The synchronizing action operator process term $\gamma_A(p)$ defines the action labels of set A to be synchronizing for process term p. It achieves this by setting the boolean value b on action transitions to true if the action ℓ is defined as synchronizing ($\ell \in A$, see Rule 29). The operator has no effect if the action is not defined as synchronizing ($\ell \notin A$, see Rule 30).

Time transitions are defined by Rule 31. The action labels from set A, that are defined as synchronizing, are moved from the guard trajectory for the non-synchronizing action labels θ_{n}, to the guard trajectory for the synchronizing action labels θ_{y}. The disjunction of the new guard trajectory $\theta_{\mathrm{n}} \downarrow \overline{A}$ with the guard trajectory $\mathrm{fls}_{tA} : [0, t] \to (A \to \{\mathrm{false}\})$ ensures that the non-synchronizing guard trajectories for the action labels in A are reset to false.

$$E \Vdash \frac{\langle p, \sigma \rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma' \rangle,\ \ell \in A}{\langle \gamma_A(p), \sigma \rangle \xrightarrow{\rho,\ell,\mathrm{true},W,\rho'} \langle \overset{\checkmark}{\gamma_A(p')}, \sigma' \rangle} \quad 29 \qquad E \Vdash \frac{\langle p, \sigma \rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma' \rangle,\ \ell \notin A}{\langle \gamma_A(p), \sigma \rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{\gamma_A(p')}, \sigma' \rangle} \quad 30$$

$$E \Vdash \frac{\langle p, \sigma \rangle \overset{t,\rho,(\theta_{\mathrm{y}},\theta_{\mathrm{n}},\theta_{\mathrm{s}},\theta_{\mathrm{r}})}{\longmapsto} \langle p', \sigma' \rangle}{\langle \gamma_A(p), \sigma \rangle \overset{t,\rho,(\theta_{\mathrm{y}} \vee (\theta_{\mathrm{n}} \downarrow A),\theta_{\mathrm{n}} \downarrow \overline{A} \vee \mathrm{fls}_{tA},\theta_{\mathrm{s}},\theta_{\mathrm{r}})}{\longmapsto} \langle \gamma_A(p'), \sigma' \rangle} \quad 31$$

The synchronizing action operator adds its set of synchronizing action labels to the consistency transition (see Rule 32), making them visible to process terms operating in a parallel context (see Section 3.7.4).

$$E \Vdash \frac{\langle p, \sigma_\perp \rangle \overset{\rho,B}{\rightsquigarrow} \langle p', \sigma \rangle}{\langle \gamma_A(p), \sigma_\perp \rangle \overset{\rho,A \cup B}{\rightsquigarrow} \langle \gamma_A(p'), \sigma \rangle} \quad 32$$

Consider, for example, the process term

$$\gamma_{\{a,b,c\}}(a \between b \between c) \parallel \gamma_{\{b,c\}}(a \between b \between c) \parallel \gamma_{\{c\}}(c \between d),$$

where $a, b, c, d \in \mathcal{L}_{\text{basic}}$ are basic action labels. This process term is equivalent to:

$$\begin{array}{l}
(\ \gamma_{\{a\}}(a);\ (\gamma_{\{b,c\}}(a \between b \between c) \parallel \gamma_{\{c\}}(c \between d)) \\
\between\ \gamma_{\{b\}}(b);\ \gamma_{\{c\}}(c \between d) \\
\between\ \gamma_{\{c\}}(c) \\
\between\ a;\ (\gamma_{\{a,b,c\}}(a \between b \between c) \parallel \gamma_{\{c\}}(c \between d)) \\
\between\ d;\ (\gamma_{\{a,b,c\}}(a \between b \between c) \parallel \gamma_{\{b,c\}}(a \between b \between c)) \\
)
\end{array}$$

The example illustrates that process terms in a parallel composition participate in basic action label based synchronization only via shared, synchronizing action labels. In the example, the three parallel process terms synchronize via action label c, and the first two process terms synchronize via action label b. None of the process terms synchronize via action label a or d.

The previous example illustrates that the set of synchronizing action labels can be smaller than the set of action labels that is actually used, as in $\gamma_{\{c\}}(c \between d)$. The set of synchronizing action labels can also be bigger than the set of action labels that is actually used. The process term:

$$\gamma_{\{a,b\}}(a \between b) \parallel \gamma_{\{a,b\}}(a),$$

is equivalent to:

$$\gamma_{\{a,b\}}(a),$$

because process term $\gamma_{\{a,b\}}(a)$ blocks the execution of the action label process term b in any parallel context that defines action label b as synchronizing.

Note that there are many different equivalent notations of synchronizing action operators in an alternative composition. For example, $\gamma_{\{a,b,c\}}(a) \between \gamma_{\{a,b,c\}}(b)$ is equivalent to $\gamma_{\{a,c\}}(a) \between \gamma_{\{b\}}(b)$, which is equivalent to $\gamma_{\{a\}}(a) \between \gamma_{\{b\}}(b) \between \gamma_{\{c\}}(\text{false} \to \tau : \emptyset : \text{true})$.

3.7.6 Channel encapsulation operator

The behavior of the channel encapsulation operator applied to a process term, $\partial_H(p)$, is the same as the behavior of its argument with the restriction that send and receive actions via channels from the set H ($H \subseteq \mathcal{H}$) cannot propagate beyond the channel encapsulation operator, as specified in Rule 33. In this rule, sr_H is an abbreviation of the set $\{h!cs, h?cs \mid h \in H, cs \in \Lambda^*\}$. Channel encapsulation has no effect on time transitions and consistency transitions, as defined by Rule 34. In this way, the channel encapsulation operator enforces the synchronous execution of matching send and receive actions.

$$E \Vdash \frac{\langle p, \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma'\rangle,\ \ell \notin \mathrm{sr}_H}{\langle \partial_H(p), \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{\partial_H(p')}, \sigma'\rangle} \ 33$$

$$E \Vdash \frac{\langle p, \sigma\rangle \overset{t,\rho,\theta}{\underset{\rho,A}{\longmapsto}} \langle p', \sigma'\rangle}{\langle \partial_H(p), \sigma\rangle \overset{t,\rho,\theta}{\underset{\rho,A}{\longmapsto}} \langle \partial_H(p'), \sigma'\rangle} \ 34$$

3.7.7 Recursion scope operator

By means of the recursion scope operator, local recursion definitions are introduced in a Chi process. The application of the recursion scope operator to a process term p with a 'global' valuation σ and a 'global' environment (D, U, J, R) behaves as p after taking the union of the local and global recursion mappings. In the rules below, $\{\mathbf{X} \mapsto \mathbf{q}\}$ denotes the recursion mapping $\{X_1 \mapsto q_1, \ldots, X_r \mapsto q_r\}$. To prevent conflicts with recursion variables already existing in the environment, the local recursion variables $\mathbf{X}$ are renamed to fresh variables $\mathbf{X}'$ with respect to the variables from the domain of R.

Notation $p[\mathbf{X}'/\mathbf{X}]$ denotes the process term that is obtained by substitution of the (free) variables $\mathbf{X}$ (an abbreviation of $X_1, \ldots, X_r$) in p by the fresh variables $\mathbf{X}'$ ($X'_1, \ldots, X'_r$), respectively; choosing the fresh variables $\mathbf{X}'$ in such a way that they remain free in p. Note that if the local recursion variables are all different from the global recursion variables ($\mathrm{dom}(R) \cap \{\mathbf{X}\} = \emptyset$), no renaming is necessary. Notation $\{\mathbf{X}' \mapsto \mathbf{q}[\mathbf{X}'/\mathbf{X}]\}$ denotes the recursion mapping $\{X'_1 \mapsto q_1[\mathbf{X}'/\mathbf{X}] \ldots X'_r \mapsto q_r[\mathbf{X}'/\mathbf{X}]\}$.

$$\frac{(D, U, J, R \cup \{\mathbf{X}' \mapsto \mathbf{q}[\mathbf{X}'/\mathbf{X}]\}) \Vdash \langle p[\mathbf{X}'/\mathbf{X}], \sigma\rangle \xrightarrow{\lambda} \langle \overset{\checkmark}{p'}, \sigma'\rangle}{(D, U, J, R) \Vdash \langle [\![_{\mathrm{R}}\ \{\mathbf{X} \mapsto \mathbf{q}\} :: p\ ]\!], \sigma\rangle \xrightarrow{\lambda} \langle \overset{\checkmark}{[\![_{\mathrm{R}}\ \{\mathbf{X}' \mapsto \mathbf{q}[\mathbf{X}'/\mathbf{X}]\} :: p'\ ]\!]}, \sigma'\rangle} \ 35$$

$$\frac{(D, U, J, R \cup \{\mathbf{X}' \mapsto \mathbf{q}[\mathbf{X}'/\mathbf{X}]\}) \Vdash \langle p[\mathbf{X}'/\mathbf{X}], \sigma\rangle \overset{t,\rho,\theta}{\underset{\rho,A}{\longmapsto}} \langle p', \sigma'\rangle}{(D, U, J, R) \Vdash \langle [\![_{\mathrm{R}}\ \{\mathbf{X} \mapsto \mathbf{q}\} :: p\ ]\!], \sigma\rangle \overset{t,\rho,\theta}{\underset{\rho,A}{\longmapsto}} \langle [\![_{\mathrm{R}}\ \{\mathbf{X}' \mapsto \mathbf{q}[\mathbf{X}'/\mathbf{X}]\} :: p'\ ]\!], \sigma'\rangle} \ 36$$

Consider, for example, the process term $[\![_{\mathrm{R}}\ \{X \mapsto Y, Y \mapsto x := 0\} :: [\![_{\mathrm{R}}\ \{Y \mapsto x := 1\} :: X\ ]\!]]\!]$. Local recursion variable Y with definition $Y \mapsto x := 1$ has the same name as the recursion variable Y in the recursion definition $Y \mapsto x := 0$ from the outer scope. The renaming of the local variable in the rules of the recursion scope operator ensures that the process term behaves as $[\![_{\mathrm{R}}\ \{X \mapsto Y, Y \mapsto x := 0\} :: [\![_{\mathrm{R}}\ \{Z \mapsto x := 1\} :: X\ ]\!]]\!]$. Thus, the value of variable x becomes 0.

3.7.8 Variable scope operator

By means of the variable scope operator, local variables are introduced in a Chi process. A variable scope operator process term

$$[\![_{\mathrm{V}}\, d_{\mathbf{x}}, \sigma_{\perp_{\mathbf{x}_{\mathrm{st}}}} :: p\,]\!],$$

that is used in an environment (D, U, J, R), with valuation σ, and where $d_{\mathbf{x}}$ denotes a dynamic type mapping with domain $\mathrm{dom}(d_{\mathbf{x}}) = \{\mathbf{x}\} = \{x_1, \ldots, x_n\}$, and $\sigma_{\perp_{\mathbf{x}_{\mathrm{st}}}}$ denotes a local valuation that has as domain the state variables of the domain of $d_{\mathbf{x}}$ ($\mathrm{dom}(\sigma_{\perp_{\mathbf{x}_{\mathrm{st}}}}) = d_{\mathbf{x}\mathrm{state}}$, see Section 2.1.4), behaves as p after taking the union of the global and local dynamic type mappings, and taking the union of the local and global valuations.

To ensure that all local variables are fresh with respect to the global variables, the local variables are first renamed. Thus $\mathbf{x}'$, in the rules below, denotes fresh variables $x'_1, \ldots, x'_n$ with respect to the variables from the domain of dynamic type mapping D. Notation $p[\mathbf{x}'/\mathbf{x}]$ denotes the process term that is obtained by substitution of the (free) variables $\mathbf{x}$ (an abbreviation of $x_1, \ldots, x_n$) in p by the fresh variables $\mathbf{x}'$, respectively; choosing the fresh variables $\mathbf{x}'$ in such a way that they remain free in p. Note that if the local variables are all different from the global variables ($\mathrm{dom}(d_{\mathbf{x}}) \cap \mathrm{dom}(D) = \emptyset$), no renaming is necessary.

Also note that the variables used in the recursion mapping R are not renamed to ensure that the bindings of these variables remain unchanged. In this way, the variables occurring in recursion mappings are bound statically, as is illustrated by the following example Chi process:

$$
\begin{array}{l}
\langle\ [\![_{\mathrm{R}}\, \{X \mapsto n := 1;\ y := n\} \\
\quad :: [\![_{\mathrm{V}}\, \{n \mapsto \mathsf{disc}\}, \{n \mapsto 2\} :: X;\ z := n\ ]\!] \\
\quad]\!] \\
,\ \{n \mapsto 0, y \mapsto 0, z \mapsto 0, \mathsf{time} \mapsto 0\} \\
,\ (\{n \mapsto \mathsf{disc}, y \mapsto \mathsf{disc}, z \mapsto \mathsf{disc}, \mathsf{time} \mapsto \mathsf{cont}\},\ \emptyset,\ \emptyset,\ \emptyset) \\
\rangle
\end{array}
$$

The process defines the discrete variables n, y, z that are initially 0, a recursion definition $X \mapsto n := 1;\ y := n$ in a recursion scope, and a variable scope operator that (re)defines n as a local variable that is initialized to 2. After performing the first transition, which is execution of the assignment $n := 1$, the process transforms into:

$$
\begin{array}{l}
\langle\ [\![_{\mathrm{R}}\, \{X \mapsto n := 1;\ y := n\} \\
\quad :: [\![_{\mathrm{V}}\, \{n' \mapsto \mathsf{disc}\}, \{n' \mapsto 2\} :: y := n;\ z := n'\ ]\!] \\
\quad]\!] \\
,\ \{n \mapsto 1, y \mapsto 0, z \mapsto 0, \mathsf{time} \mapsto 0\} \\
,\ (\{n \mapsto \mathsf{disc}, y \mapsto \mathsf{disc}, z \mapsto \mathsf{disc}, \mathsf{time} \mapsto \mathsf{cont}\},\ \emptyset,\ \emptyset,\ \emptyset) \\
\rangle
\end{array}
$$

The local variable n has been renamed to a fresh variable, according to Rule 37. In this way, the assignment $y := n$, as defined in the recursion scope, refers to the global variable n, and not to the local variable n of the variable scope. When the process term terminates (after execution of $y := n;\ z := n'$), the value of y equals 1, and the value of z equals 2. Thus, the variables used in the recursion definition are bound statically. The example also shows that the local variables that have been renamed to fresh variables cannot be renamed back to their original names after an action transition, since then a conflict with the globally defined variables of the same name could occur.

The variable scope operator is the only operator that affects the dynamic type mapping from the environment. However, it does not change any existing definitions of D, it only adds new definitions. In this way, it is ensured that the discrete, continuous, or algebraic variables in any Chi process $\langle p, \sigma, E \rangle$ remain discrete, continuous, or algebraic, respectively.

The local variables are invisible outside of the scope operator. This is done by means of data abstraction. For action transitions, data abstraction takes place by restricting the trajectories ρ and ρ', and the valuation σ', to the global variables, and by keeping only the global state variables in the set W. For time transitions, data abstraction takes place by restricting the trajectory to the global variables. In this way, all changes to local variables are removed.

In the rules below, the following abbreviations are used: trajectories ρ_D and ρ'_D denote $\rho \restriction \dot{D}_{\text{var}}$ and $\rho' \restriction \dot{D}_{\text{var}}$, respectively, where $\dot{D}_{\text{var}}$ is an abbreviation for $\text{dom}(D) \cup \{\dot{x} \mid x \in \text{dom}(D)\}$; and valuation σ'_σ denotes $\sigma' \restriction \text{dom}(\sigma)$.

$$\frac{(D \cup d_{\mathbf{x}}[\mathbf{x}'/\mathbf{x}], U, J, R) \Vdash \langle p[\mathbf{x}'/\mathbf{x}], \sigma \cup \sigma_{\perp_{\mathbf{x}_{\text{st}}}}[\mathbf{x}'/\mathbf{x}] \rangle \xrightarrow{\rho, \ell, b, W, \rho'} \langle \overset{\checkmark}{p'}, \sigma' \rangle}{(D, U, J, R) \Vdash \langle [\![\text{V}\, d_{\mathbf{x}}, \sigma_{\perp_{\mathbf{x}_{\text{st}}}} :: p]\!], \sigma \rangle \xrightarrow{\rho_D, \ell, b, W \cap D_{\text{state}}, \rho'_D} \langle [\![\text{V}\, d_{\mathbf{x}}[\mathbf{x}'/\mathbf{x}], \overset{\checkmark}{\sigma'} \restriction \{\mathbf{x}'\} :: p']\!], \sigma'_\sigma \rangle} \quad 37$$

$$\frac{(D \cup d_{\mathbf{x}}[\mathbf{x}'/\mathbf{x}], U, J, R) \Vdash \langle p[\mathbf{x}'/\mathbf{x}], \sigma \cup \sigma_{\perp_{\mathbf{x}_{\text{st}}}}[\mathbf{x}'/\mathbf{x}] \rangle \overset{t, \rho, \theta}{\underset{\rho, A}{\longmapsto}} \langle p', \sigma' \rangle}{(D, U, J, R) \Vdash \langle [\![\text{V}\, d_{\mathbf{x}}, \sigma_{\perp_{\mathbf{x}_{\text{st}}}} :: p]\!], \sigma \rangle \overset{t, \rho_D, \theta}{\underset{\rho_D, A}{\longmapsto}} \langle [\![\text{V}\, d_{\mathbf{x}}[\mathbf{x}'/\mathbf{x}], \sigma' \restriction \{\mathbf{x}'\} :: p']\!], \sigma'_\sigma \rangle} \quad 38$$

3.7.9 Action scope operator

By means of the action scope operator, local basic action labels are introduced in a Chi process by means of a local urgency mapping for action labels. The local urgency mapping is added to the global urgency mapping, after renaming the local action labels to fresh action labels with respect to the action labels and channels of the domain of urgency mapping U. Note that if the local action labels are all different from the global action labels and channels ($\{\mathbf{a}\} \cap \text{dom}(U) = \emptyset$), no renaming is necessary. As in the variable scope operator of Section 3.7.8, renaming does not take place in the recursion mapping R to ensure that the bindings of action labels in R remain unchanged. In the rules below, $\{\mathbf{a} \mapsto \mathbf{b}\}$ denotes the urgency mapping $\{a_1 \mapsto b_1, \ldots, a_m \mapsto b_m\}$.

The local basic actions are made invisible outside of the scope operator by replacing them by the internal τ action (see Rule 39). Note that no renaming is applied to action ℓ in Rule 40, because this action cannot refer to local action labels.

The action scope operator removes the local actions from the time transitions and consistency transitions, as defined by Rule 41. In the time transitions, the action scope operator restricts the guard trajectories to the set of globally defined action labels (and channels), as defined by Rule 41.a, where θ is used as an abbreviation of $(\theta_{\text{y}}, \theta_{\text{n}}, \theta_{\text{s}}, \theta_{\text{r}})$, and $\theta \restriction \text{dom}(U)$ is used as an abbreviation for $(\theta_{\text{y}} \restriction \text{dom}(U), \theta_{\text{n}} \restriction \text{dom}(U), \theta_{\text{s}} \restriction \text{dom}(U), \theta_{\text{r}} \restriction \text{dom}(U))$.

$$\frac{(D, U \cup \{\mathbf{a}' \mapsto \mathbf{b}\}, J, R) \Vdash \langle p[\mathbf{a}'/\mathbf{a}], \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma'\rangle,\ \ell \in \{\mathbf{a}'\}}{(D, U, J, R) \Vdash \langle [\![_{\mathrm{A}}\ \{\mathbf{a} \mapsto \mathbf{b}\} ::\ p\ ]\!], \sigma\rangle \xrightarrow{\rho,\tau,\mathrm{false},W,\rho'} \langle \overset{\checkmark}{[\![_{\mathrm{A}}\ \{\mathbf{a}' \mapsto \mathbf{b}\} ::\ p'\ ]\!]}, \sigma'\rangle} \quad 39$$

$$\frac{(D, U \cup \{\mathbf{a}' \mapsto \mathbf{b}\}, J, R) \Vdash \langle p[\mathbf{a}'/\mathbf{a}], \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma'\rangle,\ \ell \notin \{\mathbf{a}'\}}{(D, U, J, R) \Vdash \langle [\![_{\mathrm{A}}\ \{\mathbf{a} \mapsto \mathbf{b}\} ::\ p\ ]\!], \sigma\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{[\![_{\mathrm{A}}\ \{\mathbf{a}' \mapsto \mathbf{b}\} ::\ p'\ ]\!]}, \sigma'\rangle} \quad 40$$

$$\frac{(D, U \cup \{\mathbf{a}' \mapsto \mathbf{b}\}, J, R) \Vdash \langle p[\mathbf{a}'/\mathbf{a}], \sigma\rangle \overset{t,\rho,\theta}{\underset{\rho,A}{\longmapsto\!\!\!\!\!\rightsquigarrow}} \langle p', \sigma'\rangle}{(D, U, J, R) \Vdash \langle [\![_{\mathrm{A}}\ \{\mathbf{a} \mapsto \mathbf{b}\} ::\ p\ ]\!], \sigma\rangle \overset{t,\rho,\theta\restriction\mathrm{dom}(U)}{\underset{\rho,A\cap\mathrm{dom}(U)\}}{\longmapsto\!\!\!\!\!\rightsquigarrow}} \langle [\![_{\mathrm{A}}\ \{\mathbf{a}' \mapsto \mathbf{b}\} ::\ p'\ ]\!], \sigma'\rangle} \quad 41$$

3.7.10 Channel scope operator

By means of the channel scope operator, local channels are introduced in a Chi process by means of a local urgency mapping for channels. The local urgency mapping is added to the global urgency mapping, after renaming the local channels to fresh channels with respect to the action labels and channels of the domain of urgency mapping U. Note that if the local channels are all different from the global action labels and channels ($\{\mathbf{h}\} \cap \mathrm{dom}(U) = \emptyset$), no renaming is necessary. Renaming does not take place in the recursion mapping R to ensure that the bindings of action labels in R remain unchanged. In the rules below, $\{\mathbf{h} \mapsto \mathbf{b}\}$ denotes the urgency mapping $\{h_1 \mapsto b_1, \ldots, h_m \mapsto b_m\}$.

By means of action abstraction, the channel scope operator makes communication actions on local channels invisible outside of the scope operator. Action abstraction takes place by substituting communication actions ($h!?cs$) via a local channel by the internal τ action (see Rule 42). The internal send and receive actions ($h!cs$ and $h?cs$), see Rules 10 and 11, via a local channel h are blocked, because there are no rules that apply for these actions; Rule 42 only specifies behavior for communication actions. Function $\mathrm{ch} : \mathcal{L}_\tau \to \mathcal{H} \cup \{\bot\}$ extracts the channel label from an action. It is defined as $\mathrm{ch}(h!cs) = h$, $\mathrm{ch}(h?cs) = h$, $\mathrm{ch}(h!?cs) = h$, and $\mathrm{ch}(a) = \bot$ for $a \in \mathcal{L}_{\mathrm{basic}} \cup \{\tau\}$. Note that no renaming is applied to action ℓ in Rule 43, because this action cannot refer to local channels.

In the time transitions, the channel scope operator restricts the guard trajectories to the set of globally defined action labels and channels, as defined by Rule 44 (where θ and $\theta \restriction \mathrm{dom}(U)$ are abbreviations as defined in Section 3.7.9). In this way, the local channels are removed from the time transitions. The channel scope operator has no effect on consistency transitions.

$$\frac{(D, U \cup \{\mathbf{h}' \mapsto \mathbf{b}\}, J, R) \Vdash \langle p[\mathbf{h}'/\mathbf{h}], \sigma\rangle \xrightarrow{\rho,h!?cs,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma'\rangle,\ h \in \{\mathbf{h}'\}}{(D, U, J, R) \Vdash \langle [\![_{\mathrm{H}}\ \{\mathbf{h} \mapsto \mathbf{b}\} ::\ p\ ]\!], \sigma\rangle \xrightarrow{\rho,\tau,\mathrm{false},W,\rho'} \langle \overset{\checkmark}{[\![_{\mathrm{H}}\ \{\mathbf{h}' \mapsto \mathbf{b}\} ::\ p'\ ]\!]}, \sigma'\rangle} \quad 42$$

$$\frac{(D, U \cup \{\mathbf{h}' \mapsto \mathbf{b}\}, J, R) \Vdash \langle p[\mathbf{h}'/\mathbf{h}], \sigma \rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma' \rangle, \ \mathrm{ch}(\ell) \notin \{\mathbf{h}'\}}{(D, U, J, R) \Vdash \langle [\![_{\mathrm{H}} \{\mathbf{h} \mapsto \mathbf{b}\} :: \ p \ ]\!], \sigma \rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{[\![_{\mathrm{H}} \{\mathbf{h}' \mapsto \mathbf{b}\} :: \ p' \ ]\!]}, \sigma' \rangle} \ 43$$

$$\frac{(D, U \cup \{\mathbf{h}' \mapsto \mathbf{b}\}, J, R) \Vdash \langle p[\mathbf{h}'/\mathbf{h}], \sigma \rangle \overset{t,\rho,\theta}{\underset{\rho,A}{\longmapsto}} \langle p', \sigma' \rangle}{(D, U, J, R) \Vdash \langle [\![_{\mathrm{H}} \{\mathbf{h} \mapsto \mathbf{b}\} :: \ p \ ]\!], \sigma \rangle \overset{t,\rho,\theta \restriction \mathrm{dom}(U)}{\underset{\rho,A}{\longmapsto}} \langle [\![_{\mathrm{H}} \{\mathbf{h}' \mapsto \mathbf{b}\} :: \ p' \ ]\!], \sigma' \rangle} \ 44$$

3.8 Validation of the semantics

In Section 3.8.1, some properties of the Chi semantics are given. In Section 3.8.2, a notion of equivalence is defined, called stateless bisimilarity [15], which is similar to the well-known notion of bisimilarity [17, 13]. It is also shown that this relation is an equivalence and a congruence for all Chi operators. Some useful properties of closed Chi process terms are given in Section 3.8.3. Many of these properties express intuitions about the meaning of the Chi operators such as the commutativity and associativity of the alternative composition and the parallel composition operators. Other properties are introduced for the purpose of simplifying Chi models. The properties treated in this section add to the level of confidence one has with respect to the 'correctness' of the semantics. Since formal proofs are not yet available, all properties treated in this section must be considered as conjectures.

3.8.1 Properties of the semantics

In this section, some useful properties about the semantics of Chi are introduced. The properties are required for the remainder of the section, especially for the proofs of the properties defined in Section 3.8.3.

The following abbreviations are used in this section:

- $\langle p, \sigma_\perp, E \rangle \overset{\rho}{\rightsquigarrow}$ is defined in Section 3.3.
- $\langle p, \sigma, E \rangle \xrightarrow{\rho,\ell,b,W,\rho'}$ is an abbreviation for $\exists_{p',\sigma',E'} \langle p, \sigma, E \rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \overset{\checkmark}{p'}, \sigma', E' \rangle$
- $\langle p, \sigma, E \rangle \overset{t,\rho,\theta}{\longmapsto}$ is an abbreviation for $\exists_{p',\sigma',E'} \langle p, \sigma, E \rangle \overset{t,\rho,\theta}{\longmapsto} \langle p', \sigma', E' \rangle$

With the current set of deduction rules for the semantics of Chi the following properties of the semantics can be defined:

- The domain of the initial valuation σ equals the domain of the resulting valuation σ', and environment E equals environment E', i.e. the valuation domain and the environment are never changed in a transition.

- The left-hand (ρ) and right-hand (ρ') trajectories restricted to the domain of σ for time point zero are always the same as the initial (σ) and resulting (σ') valuation of an action transition, respectively. A similar reasoning applies to the first and last valuation of a trajectory on a time transition and the initial and resulting valuation, respectively.
- For each variable for which value $\sigma_\perp(x)$ is defined in a consistency transition, the value of that variable in a) the initial valuation $\sigma_\perp$, in b) the trajectory at time point zero, and in c) the resulting valuation σ are all the same.

The following lemma captures these facts.

Lemma 3.1. *Let p and p' be closed process terms, $\sigma_\perp$,σ,σ' be valuations, ρ, ρ' be trajectories, θ be a quadruple of guard trajectories, E and E' be environments, ℓ be an action, b be a boolean value, W be a set of variables, t be a time point, and A be a set of basic action labels. Then*

$$\begin{aligned}
\langle p, \sigma, E\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \begin{smallmatrix}\checkmark\\ p'\end{smallmatrix}, \sigma', E'\rangle \Rightarrow\ & \mathrm{dom}(\sigma) = \mathrm{dom}(\sigma') \wedge \rho_\sigma(0) = \sigma \wedge \rho'_\sigma(0) = \sigma' \\
& \wedge E = E', \\
\langle p, \sigma, E\rangle \overset{t,\rho,\theta}{\longmapsto} \langle p', \sigma', E'\rangle \Rightarrow\ & \mathrm{dom}(\sigma) = \mathrm{dom}(\sigma') \wedge \rho_\sigma(0) = \sigma \wedge \rho_\sigma(t) = \sigma' \\
& \wedge E = E', \\
\langle p, \sigma_\perp, E\rangle \overset{\rho,A}{\rightsquigarrow} \langle p', \sigma, E'\rangle \Rightarrow\ & \mathrm{dom}(\sigma_\perp) = \mathrm{dom}(\sigma) \\
& \wedge (\forall_{x\in\mathrm{dom}(\sigma_\perp),\sigma_\perp(x)\in\Lambda}\ \rho(0)(x) = \sigma_\perp(x) \wedge \sigma(x) = \sigma_\perp(x)) \\
& \wedge E = E'.
\end{aligned}$$

The Chi processes that can perform action or time transitions are consistent, and the processes resulting from the action or time transitions are consistent (the processes can perform a consistency transition).

Lemma 3.2. *Let p be a closed process term, σ be a valuation, E be an environment, ρ and ρ' be trajectories, ℓ be an action, b be a boolean value, and W be a set of variables. Then*

$$\langle p, \sigma, E\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \quad \Rightarrow \quad \langle p, \sigma, E\rangle \overset{\rho}{\rightsquigarrow}.$$

Lemma 3.3. *Let p be a closed process term, σ be a valuation, E be an environment, t be a time point, ρ be a trajectory, and θ be a quadruple of guard trajectories. Then*

$$\langle p, \sigma, E\rangle \overset{t,\rho,\theta}{\longmapsto} \quad \Rightarrow \quad \langle p, \sigma, E\rangle \overset{\rho}{\rightsquigarrow}.$$

Lemma 3.4. *Let p and p' be closed process terms, σ and σ' be valuations, E and E' be environments, ρ and ρ' be trajectories, ℓ be an action, b be a boolean value, and W be a set of variables. Then*

$$\langle p, \sigma, E\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle p', \sigma', E'\rangle \quad \Rightarrow \quad \langle p', \sigma', E'\rangle \overset{\rho'}{\rightsquigarrow}.$$

Lemma 3.5. *Let p and p' be closed process terms, σ and σ' be valuations, E and E' be environments, t be a time point, ρ be a trajectory, and θ be a quadruple of guard trajectories. Then*

$$\langle p, \sigma, E\rangle \overset{t,\rho,\theta}{\longmapsto} \langle p', \sigma', E'\rangle \quad \Rightarrow \quad \langle p', \sigma', E'\rangle \overset{\rho}{\rightsquigarrow}.$$

The following lemma shows that any variation in the set of jumping variables in the environment of a consistent Chi process has no effect on its consistency.

Lemma 3.6. *Let p and p' be closed process terms, $\sigma_\perp$ and σ be valuations, E be an environment, D be a dynamic type mapping, U be an urgency mapping, J, W be sets of Chi variables such that $J, W \subseteq \mathrm{dom}(D)$, R be a recursion mapping, ρ be a trajectory, and A be a set of basic action labels. Then*

$$\langle p, \sigma_\perp, (D, U, J, R)\rangle \overset{\rho,A}{\rightsquigarrow} \langle p', \sigma, E\rangle \quad \Leftrightarrow \quad \langle p, \sigma_\perp, (D, U, J \cup W, R)\rangle \overset{\rho,A}{\rightsquigarrow} \langle p', \sigma, E\rangle.$$

3.8.2 Stateless bisimilarity

Two closed Chi process terms are considered equivalent if they have the same behavior (in the bisimulation sense) in case both are considered from the same valuation of model variables and the same environment. We also assume that this valuation has in its domain the free occurrences of variables in the two closed Chi process terms being equivalent.

Definition 3.7 (Stateless bisimilarity)**.** A symmetric relation $R \subseteq \mathcal{P}_{\text{abstract}} \times \mathcal{P}_{\text{abstract}}$ on closed process terms is a stateless bisimulation relation if and only if for all $(p, q) \in R$, the following holds:

1. $\forall_{\sigma,E,\rho,\ell,b,W,\rho',\sigma',E'} \langle p, \sigma, E\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \checkmark, \sigma', E'\rangle$
   $\Rightarrow \langle q, \sigma, E\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle \checkmark, \sigma', E'\rangle,$
2. $\forall_{\sigma,E,\rho,\ell,b,W,\rho',p',\sigma',E'} \langle p, \sigma, E\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle p', \sigma', E'\rangle$
   $\Rightarrow \exists_{q'} \langle q, \sigma, E\rangle \xrightarrow{\rho,\ell,b,W,\rho'} \langle q', \sigma', E'\rangle \wedge (p', q') \in R,$
3. $\forall_{\sigma,E,t,\rho,\theta,p',\sigma',E'} \langle p, \sigma, E\rangle \overset{t,\rho,\theta}{\longmapsto} \langle p', \sigma', E'\rangle$
   $\Rightarrow \exists_{q'} \langle q, \sigma, E\rangle \overset{t,\rho,\theta}{\longmapsto} \langle q', \sigma', E'\rangle \wedge (p', q') \in R,$
4. $\forall_{\sigma_\perp,E,\rho,A,p',\sigma,E'} \langle p, \sigma_\perp, E\rangle \overset{\rho,A}{\rightsquigarrow} \langle p', \sigma, E'\rangle$
   $\Rightarrow \exists_{q'} \langle q, \sigma_\perp, E\rangle \overset{\rho,A}{\rightsquigarrow} \langle q', \sigma, E'\rangle \wedge (p', q') \in R,$

Two closed process terms p and q are stateless bisimilar, denoted by $p \underline{\leftrightarrow} q$, if there exists a stateless bisimulation relation R such that $(p, q) \in R$.

As a consequence of Lemma 3.1, the definition of stateless bisimilarity can be simplified considerably. Yet, with in mind future extensions of the Chi formalism, it might well be the case that these properties of the semantics are lost. Since we would prefer not to redo all the coming proofs (in such a future), this presentation was chosen.

Stateless bisimilarity is proved to be a congruence with respect to all Chi operators. As a consequence, algebraic reasoning is facilitated, since it is allowed to replace equals by equals in any context.

Theorem 3.1 (Congruence)**.** *Stateless bisimilarity is a congruence with respect to all Chi operators.*

Proof. The deduction rules of the Chi formalism satisfy the *process-tyft* format of [15]. Therefore, stateless bisimilarity is a congruence.

3.8.3 Properties of the Chi operators

In this section, some properties of the operators of Chi that hold with respect to stateless bisimilarity are discussed. Most of these correspond well with our intuitions, and hence this can be considered as an additional validation of the semantics. It is not our intention to provide a complete list of such properties (complete in the sense that every equivalence between closed process terms is derivable from those properties).

Property 3.8 (Equation, invariant and tcp process terms)**.** The following properties hold for all predicates $u \in \mathrm{Pred}(\dot{\mathcal{V}})$, continuous variables $y \in \mathcal{V}$, real valued constants $c \in \mathbb{R}$, and action labels $a \in \mathcal{L}_{\mathrm{basic}}$:

$\mathsf{eqn}\ \mathsf{true}$	$\underline{\leftrightarrow}$	$\mathsf{inv}\ \mathsf{true}$
$\mathsf{inv}\ \mathsf{true}$	$\underline{\leftrightarrow}$	$\mathsf{tcp}\ \mathsf{true}$
$\mathsf{eqn}\ y = c$	$\underline{\leftrightarrow}$	$\mathsf{eqn}\ y = c \wedge \dot{y} = 0$
$\mathsf{tcp}\ u$	$\underline{\leftrightarrow}$	$[\![_{\mathrm{A}}\ \{a \mapsto \mathsf{true}\} :: \neg u \to a : \emptyset : \mathsf{true};\ \mathsf{inv}\ \mathsf{false}\ ]\!]$

Equation / invariant / tcp process terms with predicate true are bisimilar: all are consistent and allow arbitrary delays. An equation defining a continuous variable to be equal to a constant value also defines the dotted version of the variable to be equal to zero. Finally, the tcp process term is not a primitive concept, since it can be expressed in terms of an action scope with a local, urgent action a, and a guarded action update process term using the negation of the tcp predicate as the guard of action label a, followed by an invariant false that prevents execution of the action a.

Property 3.9 (Initialization operator)**.** The following properties hold for all closed process terms $p, q \in \mathcal{P}_{\mathrm{abstract}}$ and predicates $u, u' \in \mathrm{Pred}(\dot{\mathcal{V}})$:

$\mathsf{false} \gg p$	$\underline{\leftrightarrow}$	$\bot$	$u \gg p \mathbin{[\!]} u' \gg q$	$\underline{\leftrightarrow}$	$u \wedge u' \gg (p \mathbin{[\!]} q)$
$u \gg (u' \gg p)$	$\underline{\leftrightarrow}$	$u \wedge u' \gg p$	$u \gg p \parallel u' \gg q$	$\underline{\leftrightarrow}$	$u \wedge u' \gg (p \parallel q)$

Initialization with predicate false gives an inconsistent process term. A concatenation of initializations leads to an initialization with the conjunction of the predicates. Individual initialization predicates of two process terms in alternative or parallel composition can be conjoined into a single initialization predicate operating on the alternative or parallel composition, respectively, of the two process terms.

Property 3.10 (Alternative composition)**.** The following properties hold for all closed process terms $p, q, r \in \mathcal{P}_{\mathrm{abstract}}$:

$p \mathbin{[\!]} \mathsf{eqn}\ \mathsf{true}$	$\underline{\leftrightarrow}$	p	$p \mathbin{[\!]} q$	$\underline{\leftrightarrow}$	$q \mathbin{[\!]} p$
$p \mathbin{[\!]} p$	$\underline{\leftrightarrow}$	p	$(p \mathbin{[\!]} q) \mathbin{[\!]} r$	$\underline{\leftrightarrow}$	$p \mathbin{[\!]} (q \mathbin{[\!]} r)$

The equation process term with predicate true is a zero element for alternative composition. The alternative composition is idempotent, commutative, and associative. The property $p \mathbin{[\!]} \delta \underline{\leftrightarrow} p$ does not hold. Consider, for example $p = \mathsf{eqn}\ \mathsf{true}$. Then $p \mathbin{[\!]} \delta$ cannot perform any time transitions, while p can perform arbitrary time transitions. Property $p \mathbin{[\!]} \delta \underline{\leftrightarrow} \delta$ does not hold either (but $p \mathbin{[\!]} \bot \underline{\leftrightarrow} \bot$ does hold, see Property 3.15). Consider, for example $p = \mathsf{true} \to \tau : \emptyset : \mathsf{true}$. Then $p \mathbin{[\!]} \delta$ can perform a τ transition, while δ cannot.

Property 3.11 (Sequential composition)**.** The following properties hold for all closed process terms $p, q, r \in \mathcal{P}_{\mathrm{abstract}}$, and predicates $u \in \mathrm{Pred}(\dot{\mathcal{V}})$:

$\delta;\ p$	$\underline{\leftrightarrow}$	δ			
$\mathsf{eqn}\ u;\ p$	$\underline{\leftrightarrow}$	$\mathsf{eqn}\ u$	$(p;\ q);\ r$	$\underline{\leftrightarrow}$	$p;\ (q;\ r)$
$\mathsf{inv}\ u;\ p$	$\underline{\leftrightarrow}$	$\mathsf{inv}\ u$	$(p \mathbin{[\!]} q);\ r$	$\underline{\leftrightarrow}$	$p;\ r \mathbin{[\!]} q;\ r$
$\mathsf{tcp}\ u;\ p$	$\underline{\leftrightarrow}$	$\mathsf{tcp}\ u$			

A deadlock process term followed by some other process terms is equivalent to the deadlock process term itself since the deadlock process term does not terminate successfully, i.e., deadlock is a left-zero element for sequential composition. The same holds for equation, invariant and tcp process terms. Sequential composition is associative. Alternative composition distributes over sequential composition from the left, but not from the right.

Property 3.12 (Parallel composition)**.** The following properties hold for all closed process terms $p, q, r \in \mathcal{P}_{\mathrm{abstract}}$, variables $y \in \mathcal{V}$, expressions $e \in \mathrm{Expr}(\dot{\mathcal{V}})$, and predicates $u, u' \in \mathrm{Pred}(\dot{\mathcal{V}})$:

$p \parallel q$	$\underline{\leftrightarrow}$	$q \parallel p$	$\mathsf{eqn}\ u \parallel \mathsf{eqn}\ u'$	$\underline{\leftrightarrow}$	$\mathsf{eqn}\ u \wedge u'$
$(p \parallel q) \parallel r$	$\underline{\leftrightarrow}$	$p \parallel (q \parallel r)$	$\mathsf{inv}\ u \parallel \mathsf{inv}\ u'$	$\underline{\leftrightarrow}$	$\mathsf{inv}\ u \wedge u'$
$\mathsf{eqn}\ y = e \parallel p$	$\underline{\leftrightarrow}$	$\mathsf{eqn}\ y = e \parallel p[e/y]$	$\mathsf{tcp}\ u \parallel \mathsf{tcp}\ u'$	$\underline{\leftrightarrow}$	$\mathsf{tcp}\ u \wedge u'$

Parallel composition is commutative and associative. A variable which is defined to be equal to an expression, can be replaced by its defining expression in all parallel contexts (substitution property). The parallel composition of two equation (invariant / tcp) process terms is the same as the conjunction of the equations (invariants / tcp predicates).

Note that $p[e/y]$ denotes the process term obtained by replacing in process term p all free (non-bound) expression occurrences of variable y by expression e. An expression occurrence of y is an occurrence of the variable y in a place where also a general expression could have occurred. For example, in $y \geq 1 \rightarrow a : \{y\} : y = 1$, the occurrences of y in $y \geq 1$ and in $y = 1$ are expression occurrences. The occurrence of y in $\{y\}$ is, however, not an expression occurrence, since between the braces {}, only a list of variables is allowed, and not general expressions. Therefore, $\mathsf{eqn}\ y = e \parallel y \geq 1 \rightarrow a : \{y\} : y = 1 \leftrightarrow \mathsf{eqn}\ y = e \parallel e \geq 1 \rightarrow a : \{y\} : e = 1$

Property 3.13 (Synchronizing action operator)**.** The following properties hold for all closed process terms $p, q \in \mathcal{P}_{\text{abstract}}$ and sets of basic actions $A, B \subseteq \mathcal{L}_{\text{basic}}$:

$\gamma_\emptyset(p)$	$\leftrightarrow$	p	$\gamma_A(p);\ \gamma_A(q)$	$\leftrightarrow$	$\gamma_A(p;\ q)$
$\gamma_A(\gamma_B(p))$	$\leftrightarrow$	$\gamma_{A\cup B}(p)$	$\gamma_A(p) \between \gamma_A(q)$	$\leftrightarrow$	$\gamma_A(p \between q)$

If there are no synchronizing actions, the synchronization operator has no effect. Nesting synchronization operators is equivalent to a single synchronization operator that synchronizes on the union of the sets of synchronizing actions. Assuming constancy of synchronizing actions, as in [10] where constancy of alphabets is assumed, the synchronization operator can be lifted to the top level of sequential composition and alternative composition. In other words: the synchronizing action operator distributes over sequential and alternative composition.

Property 3.14 (Channel encapsulation operator)**.** The following properties hold for all closed process terms $p, q \in \mathcal{P}_{\text{abstract}}$, predicates $u \in \text{Pred}(\dot{\mathcal{V}})$ and sets of channels $H, H' \subseteq \mathcal{H}$:

$\partial_H(\delta)$	$\leftrightarrow$	δ	$\partial_\emptyset(p)$	$\leftrightarrow$	p
$\partial_H(\mathsf{eqn}\ u)$	$\leftrightarrow$	$\mathsf{eqn}\ u$	$\partial_H(p;\ q)$	$\leftrightarrow$	$\partial_H(p);\ \partial_H(q)$
$\partial_H(\mathsf{inv}\ u)$	$\leftrightarrow$	$\mathsf{inv}\ u$	$\partial_H(p \between q)$	$\leftrightarrow$	$\partial_H(p) \between \partial_H(q)$
$\partial_H(\mathsf{tcp}\ u)$	$\leftrightarrow$	$\mathsf{tcp}\ u$	$\partial_H(\partial_{H'}(p))$	$\leftrightarrow$	$\partial_{H\cup H'}(p)$

The process terms δ, $\mathsf{eqn}\ u$, $\mathsf{inv}\ u$, and $\mathsf{tcp}\ u$ are zero elements for the channel encapsulation operator. If there are no channels to be encapsulated, the application of the channel encapsulation operator to a process term p has no effect. Encapsulation of channels distributes over the alternative composition operator and the sequential composition operator. Multiple applications of the channel encapsulation operator are equivalent to a single application where all the channels to be encapsulated are combined using union of sets of channels.

Property 3.15 (Inconsistent process term)**.** The following properties hold for all closed process terms $p \in \mathcal{P}_{\text{abstract}}$, predicates $u \in \text{Pred}(\dot{\mathcal{V}})$, sets of action labels $A \subseteq \mathcal{L}_{\text{basic}}$, sets of channels $H \subseteq \mathcal{H}$, guards $g \in \text{Pred}(\dot{\mathcal{V}})$, basic action labels $a \in \mathcal{L}_{\text{basic}} \cup \{\tau\}$, sets of variables $W \subseteq \dot{\mathcal{V}}$, and update predicates $r \in \text{Pred}(\dot{\mathcal{V}}^-)$:

$u \gg \bot$	$\leftrightarrow$	$\bot$	$\gamma_A(\bot)$	$\leftrightarrow$	$\bot$
$p \between \bot$	$\leftrightarrow$	$\bot$	$\partial_H(\bot)$	$\leftrightarrow$	$\bot$
$p \parallel \bot$	$\leftrightarrow$	$\bot$	$\bot;\ p$	$\leftrightarrow$	$\bot$

The inconsistent process term is a zero element for the initialization operator, alternative composition, parallel composition, synchronizing action operator, and the channel encapsulation operator. It is also a left-zero element for sequential composition.

4 Concrete syntax

This section presents a concise definition of the concrete syntax of Chi. The concrete syntax offers modeling equivalents for the elements of the abstract syntax, and it introduces new syntax to ensure better readability and easier modeling. The section is not meant as a precise syntax definition of the concrete syntax. For this we refer to the Chi language reference manual [11]. The main purpose of this section is to formally define the meaning of the concrete syntax by means of a mapping to the abstract syntax.

4.1 Data types

The Chi formalism is statically strongly typed. Besides the 'dynamic' type, as defined in Section 2.1.2, all variables have a 'static' type. Where the dynamic type of a variable defines the allowed trajectories for the variable (its values as a function of time), the static type defines the allowed values of the variable and the allowed operations on the variable (at any time point). The atomic types are bool (booleans), nat (natural numbers, including zero), int (integers), real (real-valued numbers), string (strings), and enum (enumerations). Type constructors operate on existing types to create structured types. The Chi formalism defines type constructors to create sets, lists, arrays, record tuples, dictionaries, functions, and distributions (for stochastic models). Channels also have a type that indicates the type of data that is communicated via the channel. Pure synchronization channels, that do not communicate data, are of the predefined type void. The Chi type system is strictly enforced in the Chi tools [20]. The type system is not formalized. Data types can be used only in the concrete syntax of Chi specifications. See for example the definition of a Chi model in Section 4.2.

4.2 Models

Below, a Chi model M is defined using the concrete syntax. In the model, $s_1, \ldots, s_k$ denote the discrete variables, $t_{\ldots}$ denote data types, $c_{\ldots}$ denote (initial) values, $x_1, \ldots, x_n$ denote the continuous variables, $z_1, \ldots, z_g$ denote the algebraic variables, ip denotes an initialization predicate that restricts the allowed values of the (dotted) variables initially (e.g. init $\dot{x} = 0$ denotes steady state initialization for x), $a_1, \ldots, a_l$ and $a'_1, \ldots, a'_{l'}$ denote the urgent and non-urgent action labels, respectively, $h_1, \ldots, h_m$ and $h'_1, \ldots, h'_{m'}$ denote the urgent and non-urgent channels, respectively, mode $X_1 = p_1, \ldots,$ mode $X_n = p_r$ denote the recursion definitions, and p_{r+1} is a process term.

model $M() =$
$[\![$ var s_1 : disc $t_{s_1} = c_{s_1}, \ldots, s_k$: disc $t_{s_k} = c_{s_k}$
, x_1 : cont $t_{x_1} = c_{x_1}, \ldots, x_n$: cont $t_{x_n} = c_{x_n}$
, z_1 : alg $t_{z_1}, \ldots, z_g$: alg t_{z_g}
, time $= c_t$
, init ip
, action $a_1, \ldots, a_l$
, action nonurg $a'_1, \ldots, a'_{l'}$
, chan $h_1 : t_{h_1}, \ldots, h_m : t_{h_m}$
, chan nonurg $h'_1 : t_{h'_1}, \ldots, h'_{m'} : t_{h'_{m'}}$
, mode $X_1 = p_1, \ldots,$ mode $X_r = p_r$
:: p_{r+1}
$]\!]$

The meaning of model M is defined as the following Chi process in the abstract language:

$$
\begin{array}{l}
\langle\ \partial_{\{h_1,\ldots,h_m,\ h'_1,\ldots,h'_{m'}\}}(ip \gg p_{r+1}) \\
,\ \{s_1 \mapsto c_{s_1}, \ldots, s_k \mapsto c_{s_k},\ x_1 \mapsto c_{x_1}, \ldots, x_n \mapsto c_{x_n},\ \mathsf{time} \mapsto c_{\mathrm{t}}\} \\
,\ (\ \{\ s_1 \mapsto \mathsf{disc}, \ldots, s_k \mapsto \mathsf{disc} \\
\quad , x_1 \mapsto \mathsf{cont}, \ldots, x_n \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont} \\
\quad , z_1 \mapsto \mathsf{alg}, \ldots, z_g \mapsto \mathsf{alg} \\
\quad \} \\
\quad ,\ \{\ a_1 \mapsto \mathrm{true}, \ldots, a_l \mapsto \mathrm{true},\ a'_1 \mapsto \mathrm{false}, \ldots, a'_{l'} \mapsto \mathrm{false},\ \tau \mapsto \mathrm{true} \\
\quad , h_1 \mapsto \mathrm{true}, \ldots, h_m \mapsto \mathrm{true},\ h'_1 \mapsto \mathrm{false}, \ldots, h'_{m'} \mapsto \mathrm{false} \\
\quad \} \\
\quad ,\ \emptyset \\
\quad ,\ \{X_1 \mapsto p_1, \ldots, X_r \mapsto p_r\} \\
\quad)\\
\rangle
\end{array}
$$

If one or more initial values from $c_{s_1}, \ldots, c_{s_k}, c_{x_1}, \ldots, c_{x_n}$ are missing, the corresponding values in the valuation $\{s_1 \mapsto c_{s_1}, \ldots, s_k \mapsto c_{s_k},\ x_1 \mapsto c_{x_1}, \ldots, x_n \mapsto c_{x_n},\ \mathsf{time} \mapsto c_{\mathrm{t}}\}$ are replaced by $\perp$. If the initial value for the time, c_{t}, is missing, it is replaced by zero in the valuation, and if the initialization declaration init *ip* is missing, the initialization predicate *ip* is taken to be the predicate true.

4.2.1 Additional assumptions and abbreviations

In the notations defined above, it is assumed that the discrete, continuous, and algebraic variables are all different. Besides the declared variables, the existence of the predefined reserved global variable time which denotes the model time is assumed. This variable cannot be declared, but its initial value can be defined. Its value can be used in expressions in the process terms $p, p_1, p_2, \ldots, p_{r+1}$. Further restrictions follow directly from the requirements that are assumed to hold for Chi processes as defined in Section 2.2.

As a shorthand, the keywords var, act, or chan are omitted when there are no variable, action, or channel declarations, respectively. In a similar way, the time, init, and mode keywords can be omitted. A declaration $s_1 : t = c_1, \ldots, s_k : t = c_k$ of a number of variables that have the same type can be abbreviated as $s_1, \ldots, s_k : t = (c_1, \ldots, c_k)$. The brackets are needed to prevent parsing problems in declarations such as var x, y : disc nat $= 1, 2, z$: disc nat, which should be interpreted as var x, y : disc nat $= (1, 2),\ z$: disc nat. Initial values can be omitted as in var s : disc real. The keyword disc is optional, as in var s : real, which declares a discrete variable of type real; and the declaration var x : cont real $= c$ of a continuous variable of type real with initial value c can be abbreviated as var x : cont $= c$. In the definition of the notation model $M() = [\![\ldots]\!]$, keywords such as var, action, chan occur in a specific order. There are however no such restrictions. Keywords can also occur multiple times. For example: model $M =$ $[\![$ action $a, b,$ var k, n : disc real, x : cont $= 1,$ chan h : nat, var z : alg real :: $\ldots$ $]\!]$, is allowed.

4.3 Process terms

The complete set of process terms $\mathcal{P}_{\mathrm{concrete}}$ of the concrete syntax is defined below by the grammar for the process terms $p \in \mathcal{P}_{\mathrm{concrete}}$.

Process term	**Process term name**

p	$::=$	p_{atom}	atomic
	$\mid$	$p\,;\ p$	sequential composition
	$\mid$	$p \,[\!]\, p$	alternative composition
	$\mid$	$p \parallel p$	parallel composition
	$\mid$	$\gamma_A(p)$	synchronizing action
	$\mid$	$*p$	loop
	$\mid$	$u \overset{*}{\rightarrow} p$	while
	$\mid$	$[\![\ \mathit{declarations} :: p\ ]\!]$	scope
	$\mid$	$l_{\text{p}}(\mathbf{x}_k, \mathbf{h}_m, \mathbf{e}_n)$	process instantiation

		Process term	**Process term name**
p_{atom}	$::=$	$\mathsf{eqn}\ u$	equation
	$\mid$	$\mathsf{inv}\ u$	invariant
	$\mid$	$\mathsf{tcp}\ u$	time can progress
	$\mid$	p_{act}	action
	$\mid$	$u \rightarrow p_{\text{act}}$	guarded action
	$\mid$	$\mathsf{now}\ p_{\text{act}}$	nondelayable action
	$\mid$	$u \rightarrow \mathsf{now}\ p_{\text{act}}$	guarded nondelayable action
	$\mid$	Δd	delay
	$\mid$	X	recursion variable

		Process term	**Process term name**
p_{act}	$::=$	skip	internal action
	$\mid$	$\mathbf{x}_n := \mathbf{e}_n$	(multi-)assignment
	$\mid$	$W : r$	update
	$\mid$	p_{sync}	synchronization
	$\mid$	$p_{\text{sync}} : \mathbf{x}_n := \mathbf{e}_n$	synchronization assignment
	$\mid$	$p_{\text{sync}} : W : r$	synchronization update

		Process term	**Process term name**
p_{sync}	$::=$	a	action label
	$\mid$	$h\,!\,\mathbf{e}_n$	send
	$\mid$	$h\,?\,\mathbf{x}_n$	receive
	$\mid$	$h\,!?\,\mathbf{x}_n := \mathbf{e}_n$	communicate

Note that the communicate process term $h\,!?\,\mathbf{x}_n := \mathbf{e}_n$ is not used for modeling. Its only use is to enable the elimination of parallel composition.

The operators are listed in descending order of their binding strength as follows $\{*,\ \overset{*}{\rightarrow}\}$, ; , $[\!]$, $\parallel$, where the operators in the set $\{*,\ \overset{*}{\rightarrow}\}$ have the same priority.

In the definitions below, omitting a guard defaults to a guard that is true, and omitting a synchronization part (action label a, send $h\,!\,\mathbf{e}_n$ or receive $h\,?\,\mathbf{x}_n$) defaults to the internal action label τ.

4.3.1 Skip process term

The skip process term is defined as an internal action with an update part consisting of an empty set of jumping variables and a predicate true.

$$\boxed{\text{skip} \triangleq \text{true} \to \tau : \emptyset : \text{true}}$$

4.3.2 Multi-assignment process term

The multi-assignment process term $\mathbf{x}_n := \mathbf{e}_n$ is defined as an internal action update process term that changes the values of the variables $x_1, \ldots, x_n$ to the values of the expressions $e_1, \ldots, e_n$, respectively.

$$\boxed{\mathbf{x}_n := \mathbf{e}_n \triangleq \text{true} \to \tau : \{\mathbf{x}_n\} : \mathbf{x}_n = \mathbf{e}_n^-}$$

Here e^- denotes the result of replacing all variables (and dotted variables) in e by their '$^-$' superscripted version, and $\mathbf{x}_n = \mathbf{e}_n^-$ denotes $x_1 = e_1^- \wedge \cdots \wedge x_n := e_n^-$. For example, process term $x := 2x + yz$ is defined as $\text{true} \to \tau : \{x\} : x = 2x^- + y^- z^-$, and process term $x, y := x + y, x - y$ is defined as $\text{true} \to \tau : \{x, y\} : (x = x^- + y^-) \wedge (y = x^- - y^-)$. Note that this rewriting rule may not be applied to a part of another atomic process term, because the process terms $p_{\text{sync}} : \mathbf{y}_n := \mathbf{e}_n$ are rewritten in a different way (see Section 4.3.5).

4.3.3 Update process term

The update process term $W : r$ is prefixed with the internal action label τ and the guard 'true'.

$$\boxed{W : r \triangleq \text{true} \to \tau : W : r}$$

4.3.4 Synchronization process terms

Omitting the update part in the action label update process term, or in the send / receive update process terms defaults to an update part with an empty set of jumping variables and a predicate true.

$$\boxed{\begin{array}{rcl} a & \triangleq & \text{true} \to a : \emptyset : \text{true} \\ u \to a & \triangleq & u \to a : \emptyset : \text{true} \\ h\,!\,\mathbf{e}_n & \triangleq & \text{true} \to h\,!\,\mathbf{e}_n : \emptyset : \text{true} \\ u \to h\,!\,\mathbf{e}_n & \triangleq & u \to h\,!\,\mathbf{e}_n : \emptyset : \text{true} \\ h\,?\,\mathbf{x}_n & \triangleq & \text{true} \to h\,?\,\mathbf{x}_n : \emptyset : \text{true} \\ u \to h\,?\,\mathbf{x}_n & \triangleq & u \to h\,?\,\mathbf{x}_n : \emptyset : \text{true} \end{array}}$$

4.3.5 Synchronization process terms with multi-assignment

The multi-assignment that is atomically combined with a synchronization process term is defined as a synchronization with an update part.

$$
\begin{array}{rcl}
a : \mathbf{y}_n := \mathbf{e}_n & \triangleq & \text{true} \rightarrow a : \{\mathbf{y}_n\} : \mathbf{y}_n = \mathbf{e}_n^- \\
u \rightarrow a : \mathbf{y}_n := \mathbf{e}_n & \triangleq & u \rightarrow a : \{\mathbf{y}_n\} : \mathbf{y}_n = \mathbf{e}_n^- \\
h\,!\,\mathbf{e}_k : \mathbf{y}_n := \mathbf{e}_n & \triangleq & \text{true} \rightarrow h\,!\,\mathbf{e}_k : \{\mathbf{y}_n\} : \mathbf{y}_n = \mathbf{e}_n^- \\
u \rightarrow h\,!\,\mathbf{e}_k : \mathbf{y}_n := \mathbf{e}_n & \triangleq & u \rightarrow h\,!\,\mathbf{e}_k : \{\mathbf{y}_n\} : \mathbf{y}_n = \mathbf{e}_n^- \\
h\,?\,\mathbf{x}_k : \mathbf{y}_n := \mathbf{e}_n & \triangleq & \text{true} \rightarrow h\,?\,\mathbf{x}_k : \{\mathbf{y}_n\} : \mathbf{y}_n = \mathbf{e}_n^-[\mathbf{x}_k/\mathbf{x}_k^-] \\
u \rightarrow h\,?\,\mathbf{x}_k : \mathbf{y}_n := \mathbf{e}_n & \triangleq & u \rightarrow h\,?\,\mathbf{x}_k : \{\mathbf{y}_n\} : \mathbf{y}_n = \mathbf{e}_n^-[\mathbf{x}_k/\mathbf{x}_k^-]
\end{array}
$$

Here $\mathbf{e}_n^-[\mathbf{x}_k/\mathbf{x}_k^-]$ denotes the result of substituting in each of the expressions $e_1^-, \ldots, e_n^-$ all occurrences of $x_1^-, \ldots, x_k^-$ by $x_1, \ldots, x_k$, respectively. For example $h\,?\,x : xs := xs +\!\!+ [x]$ is defined as $h\,?\,x : \{xs\} : xs = (xs^- +\!\!+ [x^-])[x/x^-]$, which is defined as $h\,?\,x : \{xs\} : xs = xs^- +\!\!+ [x]$.

4.3.6 Synchronization update process term

Omitting the guard defaults to the guard true.

$$
p_{\text{sync}} : W : r \quad \triangleq \quad \text{true} \rightarrow p_{\text{sync}} : W : r
$$

4.3.7 Nondelayable action process term

Prefixing an action process term with the now keyword corresponds to making the process term nondelayable by adding the tcp predicate false in an alternative composition.

$$
\mathsf{now}\ p_{\text{act}} \quad \triangleq \quad p_{\text{act}} \,[]\, \mathsf{tcp}\ \mathsf{false}
$$

The action process term p_{act} is rewritten in terms of the abstract syntax as defined above in Sections 4.3.1 until 4.3.6.

4.3.8 Guarded nondelayable action process term

A guarded nondelayable action process term is defined as an alternative composition of the guarded action process term with a tcp predicate consisting of the negated guard, to allow delaying only for as long as the guard is false.

$$
u \rightarrow \mathsf{now}\ p_{\text{act}} \quad \triangleq \quad u \rightarrow p_{\text{act}} \,[]\, \mathsf{tcp}\ \neg u
$$

The guarded action process term p_{act} is rewritten in terms of the abstract syntax as defined above in Sections 4.3.1 until 4.3.6.

4.3.9 Delay process term

By means of the delay process term Δd, a process term is forced to delay for the amount of time units specified by the value of numerical expression d, and then terminates by means of an action τ. The fact that process term Δd terminates by means of an action ensures that time-outs enforce a choice in alternative composition. The value of expression d is evaluated when the delay process term Δd is activated.

$$\begin{array}{|lcl|}\hline \Delta d & \triangleq & [\![_V\, \{t \mapsto \mathsf{cont}\}, \{t \mapsto \bot\} \\ & & :: t = d \gg (\dot{t} = -1 \,[\!]\, t \leq 0 \rightarrow \tau : \emptyset : \mathsf{true} \,[\!]\, \mathsf{tcp}\ t > 0) \\ & &]\!] \\ \hline \end{array}$$

In the definition of Δd, $t \in \mathcal{V}$ denotes a fresh variable, not occurring free in p and not occurring in d.

In the example below, the process term $\mathsf{time} \geq 2 \rightarrow t_{\mathrm{next}} := 5;\ \Delta t_{\mathrm{next}} - \mathsf{time}$ is executed in a process:

$$\mathsf{model}\ M() = [\![\ \mathsf{var}\ t_{\mathrm{next}} : \mathsf{real} :: \mathsf{time} \geq 2 \rightarrow t_{\mathrm{next}} := 5;\ \Delta t_{\mathrm{next}} - \mathsf{time}\ ]\!]$$

Rewriting the model in terms of the abstract syntax leads to:

$$\begin{array}{l} \langle\ \mathsf{time} \geq 2 \rightarrow \tau : \{t_{\mathrm{next}}\} : t_{\mathrm{next}} = 5 \\ \quad ;\ [\![_V\, \{t \mapsto \mathsf{cont}\}, \{t \mapsto \bot\} \\ \quad\quad :: t = t_{\mathrm{next}} - \mathsf{time} \gg (\dot{t} = -1 \,[\!]\, t \leq 0 \rightarrow \tau : \emptyset : \mathsf{true} \,[\!]\, \mathsf{tcp}\ t > 0) \\ \quad\quad]\!] \\ ,\ \{t_{\mathrm{next}} \mapsto \bot,\ \mathsf{time} \mapsto 0\} \\ ,\ (\{t_{\mathrm{next}} \mapsto \mathsf{disc},\ \mathsf{time} \mapsto \mathsf{cont}\}, \emptyset, \emptyset, \emptyset) \\ \rangle \end{array}$$

After execution of the guarded action update process term (assignment) at time point 2, the consistency transition defined by Rule 19 of the sequential composition operator (in combination with Rules 18 of the initialization operator and Rule 38.b of the variable scope operator) restricts the initial value of local variable t to a value that satisfies $t = t_{\mathrm{next}} - \mathsf{time}$. The process specified above then transforms into:

$$\begin{array}{l} \langle\ [\![_V\, \{t \mapsto \mathsf{cont}\}, \{t \mapsto 3\} :: (\dot{t} = -1 \,[\!]\, t \leq 0 \rightarrow \tau : \emptyset : \mathsf{true} \,[\!]\, \mathsf{tcp}\ t > 0)\ ]\!] \\ ,\ \{t_{\mathrm{next}} \mapsto 5,\ \mathsf{time} \mapsto 2\} \\ ,\ (\{t_{\mathrm{next}} \mapsto \mathsf{disc},\ \mathsf{time} \mapsto \mathsf{cont}\}, \emptyset, \emptyset, \emptyset) \\ \rangle \end{array}$$

4.3.10 Repetition operator process terms

Process term $*p$ represents the infinite repetition of process term p. Guarded repetition $u \xrightarrow{*} p$ can be interpreted as 'while u do p'.

$$
\begin{array}{lcl}
*p & \triangleq & [\![_{\mathrm{R}}\, \{X \mapsto p;\ X\} :: X\,]\!] \\
u \stackrel{*}{\rightarrow} p & \triangleq & [\![_{\mathrm{R}}\, \{X \mapsto (\ u \rightarrow \tau : \emptyset : \mathrm{true};\ p;\ X \\
 & & \quad [\!]\ \neg u \rightarrow \tau : \emptyset : \mathrm{true} \\
 & & \quad [\!]\ \mathsf{tcp}\ \mathrm{false} \\
 & & \quad)\ \} \\
 & & :: X \\
 & &]\!]
\end{array}
$$

In the definition of $*p$ and $u \stackrel{*}{\rightarrow} p$, recursion variable X denotes a fresh recursion variable not occurring free in p.

4.3.11 Scope operator process term

The scope operator process term

$$
\begin{array}{l}
[\![\ \mathsf{var}\ s_1 : \mathsf{disc}\ t_{s_1} = c_{s_1}, \ldots, s_k : \mathsf{disc}\ t_{s_k} = c_{s_k} \\
\quad ,\ x_1 : \mathsf{cont}\ t_{x_1} = c_{x_1}, \ldots, x_n : \mathsf{cont}\ t_{x_n} = c_{x_n} \\
\quad ,\ z_1 : \mathsf{alg}\ t_{z_1}, \ldots, z_g : \mathsf{alg}\ t_{z_g} \\
,\ \mathsf{init}\ ip \\
,\ \mathsf{action}\ a_1, \ldots, a_l \\
,\ \mathsf{action\ nonurg}\ a'_1, \ldots, a'_{l'} \\
,\ \mathsf{chan}\ h_1 : t_{h_1}, \ldots, h_m : t_{h_m} \\
,\ \mathsf{chan\ nonurg}\ h'_1 : t_{h'_1}, \ldots, h'_{m'} : t_{h'_{m'}} \\
,\ \mathsf{mode}\ X_1 = p_1,\ \ldots, \mathsf{mode}\ X_r = p_r \\
::\ p_{r+1} \\
]\!],
\end{array}
$$

in the concrete language is defined as the following process term in the abstract language:

$$
\begin{array}{l}
[\![_{\mathrm{V}}\, \{s_1 \mapsto \mathsf{disc}, \ldots, s_k \mapsto \mathsf{disc} \\
\quad , x_1 \mapsto \mathsf{cont}, \ldots, x_n \mapsto \mathsf{cont} \\
\quad , z_1 \mapsto \mathsf{alg}, \ldots, z_g \mapsto \mathsf{alg} \\
\quad \} \\
,\ \{s_1 \mapsto c_{s_1}, \ldots, s_k \mapsto c_{s_k},\ x_1 \mapsto c_{x_1}, \ldots, x_n \mapsto c_{x_n}\} \\
::\ ip \gg [\![_{\mathrm{A}}\, \{a_1 \mapsto \mathrm{true}, \ldots, a_l \mapsto \mathrm{true},\ a'_1 \mapsto \mathrm{false}, \ldots, a'_{l'} \mapsto \mathrm{false}\} \\
\qquad :: [\![_{\mathrm{H}}\, \{h_1 \mapsto \mathrm{true}, \ldots, h_m \mapsto \mathrm{true},\ h'_1 \mapsto \mathrm{false}, \ldots, h'_{m'} \mapsto \mathrm{false}\} \\
\qquad\quad :: [\![_{\mathrm{R}}\, \{X_1 \mapsto p_1, \ldots, X_r \mapsto p_r\} :: p_{r+1}\,]\!] \\
\qquad\quad]\!] \\
\qquad]\!] \\
]\!]
\end{array}
$$

The scope operator is used to declare a scope consisting of local discrete variables $s_1, \ldots, s_k$; local continuous variables $x_1, \ldots, x_n$; local algebraic variables $z_1, \ldots, z_g$; local urgent and non-urgent action labels $a_1, \ldots, a_l$ and $b_1, \ldots, b_{l'}$, respectively; local urgent and non-urgent channels $h_1, \ldots, h_m$ and $h'_1, \ldots, h'_{m'}$, respectively; and local recursion definitions $\mathsf{mode}\ X_1 = p_1,\ \ldots, \mathsf{mode}\ X_r = p_r$. The variables are assumed to be all different, and action labels are

assumed to be different from recursion variables. Furthermore, the same assumptions hold, and the same abbreviations can be used, as those defined for models in Section 4.2.1.

4.3.12 Process instantiation process term

Process instantiation process term $l_{\mathrm{p}}(\mathbf{x}_k, \mathbf{a}_l, \mathbf{h}_m, \mathbf{e}_n)$, where l_{p} denotes a process label, enables (re)-use of a process definition. A process definition is specified once, but it can be instantiated many times, possibly with different parameters: external variables $\mathbf{x}_k$, external action labels $\mathbf{a}_l$, external channels $\mathbf{h}_m$, and expressions $\mathbf{e}_n$.

Chi specifications in which process instantiations $l_{\mathrm{p}}(\mathbf{x}_k, \mathbf{a}_l, \mathbf{h}_m, \mathbf{e}_n)$ are used have the following structure:

$$
\begin{array}{l}
pd_1 \\
\vdots \\
pd_j \\
\\
\mathsf{model}\ \cdots = \\
[\![\ \mathsf{var}\ \ldots,\ \mathsf{action}\ \ldots,\ \mathsf{chan}\ \ldots,\ \mathsf{mode}\ \ldots :: q\ ]\!],
\end{array}
$$

where for each process instantiation $l_{\mathrm{p}}(\mathbf{x}_k, \mathbf{a}_l, \mathbf{h}_m, \mathbf{e}_n)$ occurring in process term q, a matching process definition pd_i $(1 \leq i \leq j)$ of the form

$$
\mathsf{proc}\ l_{\mathrm{p}}(\mathsf{var}\ \mathbf{x}'_k : \mathbf{t}_{x_k}, \mathsf{action}\ \mathbf{a}'_l, \mathsf{chan}\ \mathbf{h}'_m : \mathbf{t}_{h_m}, \mathsf{val}\ \mathbf{v}_n : \mathbf{t}_{v_n}) = p
$$

must be present among the j process definitions $pd_1 \ldots pd_j$. Here l_{p} denotes a process label, $\mathbf{x}_k$ denotes the 'actual external' variables $x_1, \ldots, x_k$, $\mathbf{a}_l$ denotes the 'actual external' action labels $a_1, \ldots, a_l$, $\mathbf{h}_m$ denotes the 'actual external' channels $h_1, \ldots, h_m$, $\mathbf{e}_n$ denotes the expressions $e_1, \ldots, e_n$, $\mathbf{x}'_k : \mathbf{t}_{x_k}$ denotes the 'formal external' variable definitions $x'_1 : t_{x_1}, \ldots, x'_k : t_{x_k}$, $\mathbf{a}'_l$ denotes the 'formal external' action definitions $a'_1, \ldots, a'_l$, $\mathbf{h}'_m : \mathbf{t}_{h_m}$ denotes the 'formal external' channel definitions $h'_1 : t_{h_1}, \ldots, h'_m : t_{h_m}$, and $\mathbf{v}_n : \mathbf{t}_{v_n}$ denotes the 'value parameter definitions' $v_1 : t_{v_1}, \ldots, v_n : t_{v_n}$.

The only free variables, free action labels, and free channels that are allowed in process term p are the formal external variables $\mathbf{x}'_k$ and the value parameters $\mathbf{v}_n$, the formal external actions $\mathbf{a}'_l$, and the formal external channels $\mathbf{h}'_m$, respectively. We assume that the formal external variables $\mathbf{x}'_k$ and the value parameters $\mathbf{v}_n$ are different.

Formally, the syntactic translation of process instantiation

$$
l_{\mathrm{p}}(\mathbf{x}_k, \mathbf{a}_l, \mathbf{h}_m, \mathbf{e}_n)
$$

with corresponding process definition

$$
\mathsf{proc}\ l_{\mathrm{p}}(\mathsf{var}\ \mathbf{x}'_k : \mathbf{t}_{x_k}, \mathsf{action}\ \mathbf{a}'_l, \mathsf{chan}\ \mathbf{h}'_m : \mathbf{t}_{h_m}, \mathsf{val}\ \mathbf{v}_n : \mathbf{t}_{v_n}) = p
$$

is given by

$$
\begin{array}{l}
[\![_{\mathrm{V}}\, \{v_1 \mapsto \mathsf{disc}, \ldots, v_n \mapsto \mathsf{disc}\},\ \{v_1 \mapsto \bot, \ldots, v_n \mapsto \bot\} \\
::\ \mathbf{v}_n = \mathbf{w}_n \gg \mathcal{T}_{\mathrm{abstract}}(p) \\
]\!]\ [\mathbf{x}_k, \mathbf{a}_l, \mathbf{h}_m, \mathbf{e}_n/\mathbf{x}'_k, \mathbf{a}'_l, \mathbf{h}'_m, \mathbf{w}_n],
\end{array}
$$

where $\mathcal{T}_{\mathrm{abstract}}(p)$ denotes the abstract equivalent of concrete process term p, as defined in Section 4.3.

Notation $q[\mathbf{x}_k, \mathbf{a}_l, \mathbf{h}_m, \mathbf{e}_n/\mathbf{x}'_k, \mathbf{a}'_l, \mathbf{h}'_m, \mathbf{w}_n]$ denotes the process term obtained from $q \in \mathcal{P}_{\mathrm{abstract}}$ by substitution of the (free) variables $\mathbf{x}'_k$ by $\mathbf{x}_k$, of the (free) actions $\mathbf{a}'_l$ by $\mathbf{a}_l$, of the (free) channels $\mathbf{h}'_m$ by $\mathbf{h}_m$, and of the (free) variables $\mathbf{w}_n$ by expressions $\mathbf{e}_n$.

The variables $\mathbf{w}_n$ are assumed to be fresh with respect to $\mathbf{x}'_k$ and $\mathbf{v}_n$. The substitution is defined in such a way that no variables from $\mathbf{x}_k$ or $\mathbf{e}_n$, and no actions or channels from $\mathbf{a}_l$ or $\mathbf{h}_m$, respectively, become bound. If substitution would cause new bindings, the local variable, local action or local channel that a variable, action or channel from $\mathbf{x}_k$, $\mathbf{e}_n$, $\mathbf{a}_l$ or $\mathbf{h}_m$ would become bound to, is renamed into a fresh variable, fresh action or fresh channel, respectively, before the substitution takes place.

The translation declares the value parameters $\mathbf{v}_n$ as local discrete variables with initial values $\mathbf{e}_n$. By convention, however, process term p normally does not change the values of these variables.

Bibliography

[1] J. C. M. Baeten, D. A. van Beek, and J. E. Rooda. Process algebra. In Paul A. Fishwick, editor, *Handbook of Dynamic System Modeling*, pages 19–1–19–21. Chapman & Hall/CRC, 2007.

[2] J.C.M. Baeten, D.A. van Beek, P.J.L. Cuijpers, M.A. Reniers, J.E. Rooda, R.R.H. Schiffelers, and R.J.M. Theunissen. Model-based engineering of embedded systems using the hybrid process algebra Chi. *Electronic Notes in Theoretical Computer Science*, 209:21–53, 2008.

[3] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers. Syntax and consistent equation semantics of hybrid Chi. *Journal of Logic and Algebraic Programming*, 68(1-2):129–210, 2006.

[4] D. A. van Beek and J. E. Rooda. Analysis of hybrid systems using Chi. Technical report, Eindhoven University of Technology, Department of Mechanical Engineering, The Netherlands, 2008. Lecture notes 4C650 and 4C390, to be published.

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

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

[7] H. Elmqvist, D. Brück, and M. Otter. *Dymola – Dynamic Modeling Language – User's Manual, Version 4.0*. Dynasim AB, Lund, Sweden, 2000.

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

[9] C. W. Gear. Differential-algebraic equation index transformations. *SIAM. J. Sci. Stat. Comp.*, 9:39–47, 1988.

[10] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood-Cliffs, 1985.

[11] A. T. Hofkamp, J. E. Rooda, R. R. H. Schiffelers, and D.A. van Beek. Chi 2.0 reference manual. Technical Report SE-Report 2008-02, Eindhoven University of Technology, Department of Mechanical Engineering, The Netherlands, 2008. http://se.wtb.tue.nl/sereports/.

[12] K. L. Man and R. R. H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. PhD thesis, Eindhoven University of Technology, 2006.

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

[14] H. I. Moe. *Dynamic Process Simulation, Studies on Modeling and Index Reduction*. PhD thesis, University of Trondheim, 1995.

[15] M. R. Mousavi, M. A. Reniers, and J. F. Groote. Notions of bisimulation and congruence formats for SOS with data. *Information and Computation*, 200(1):107–147, 2005.

[16] C. C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM J. Sci. Stat. Comput.*, 9(2):213–231, 1988.

[17] D. M. R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[18] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[19] Walter Rudin. *Real and Complex Analysis*. McGraw-Hill Science, 1968.

[20] Systems Engineering Group TU/e. Chi toolset. http://se.wtb.tue.nl/sewiki/chi, 2008.

[21] J. Unger, A. Kröner, and W. Marquardt. Structural analysis of differential-algebraic equation systems—theory and applications. *Computers & Chemical Engineering*, 19(8):867–882, 1995.

[22] D.A. van Beek, J.E. Rooda, R.R.H. Schiffelers, K.L. Man, and M.A. Reniers. Relating hybrid Chi to other formalisms. *Electronic Notes in Theoretical Computer Science*, 191:85–113, 2007.

A Introduction to higher index systems

Most languages for hybrid system simulation or verification cannot deal with higher index systems. Unfortunately, many models of physical systems are of a higher index. In this section, the higher index problem is explained.

The higher the index of a system of differential algebraic equations (DAEs), the more difficult it is to numerically solve the equations for the continuous variables as functions of time. Ordinary

differential equations (ODEs), which are of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$ have index 0. Adding algebraic constraints means increasing the index by at least 1. The general form of a system of DAEs is

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}, \tag{A.1}$$

where $\mathbf{x}$ denotes a (vector of) continuous variables, $\mathbf{y}$ denotes a (vector of) continuous variables that are used algebraically (the dotted version $\dot{\mathbf{y}}$ does not occur in the equations), and t denotes the time. Note that in Chi, DAEs are prefixed with the eqn keyword by means of the equation process term eqn u, and the identifier time is used instead of t. For the solution of the implicit system of equations (A.1), also the initial conditions are required. We denote the initial conditions as a valuation. For lower index systems, an initial condition $\{\dot{\mathbf{x}} \mapsto \mathbf{c}_0, \mathbf{x} \mapsto \mathbf{c}_1, \mathbf{y} \mapsto \mathbf{c}_2, t \mapsto c_3\}$ (where $\mathbf{c}_0$, $\mathbf{c}_1$, $\mathbf{c}_2$ and c_3 denote (vectors of) values), needs to satisfy only the system of equations A.1, so that $\mathbf{f}(\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, c_3) = \mathbf{0}$. Higher index systems on the other hand are characterized by the fact that they have *hidden constraints*, which are obtained by differentiation of the system of equations.

An example of an index 0 system, with, for example, initial conditions $\{x \mapsto 1, \dot{x}_0 \mapsto 0, \dot{x}_1 \mapsto 2\}$, is:

$$\dot{x}_0 = -x_1 + 1 \tag{A.2a}$$
$$\dot{x}_1 = 2 \tag{A.2b}$$

An example of an index 1 system, with, for example, initial conditions $\{x \mapsto 1, \dot{x} \mapsto 0, y \mapsto 2\}$, is:

$$\dot{x} = -x + 1 \tag{A.3a}$$
$$y = 2x \tag{A.3b}$$

An example of an index 2 system is:

$$x = 1 \tag{A.4a}$$
$$y = \dot{x} \tag{A.4b}$$

Initial conditions for this system of equations appear to be $\{x \mapsto 1, \dot{x} \mapsto 1, y \mapsto 1\}$, for example. However, the solution function for this set of equations is for x the constant function 1 ($x : \mathcal{T} \to \mathbb{R}$, where $\mathcal{T}$ and $\mathbb{R}$ are the sets of time points and real valued numbers, respectively, and $x(t) = 1$ for all $t \in \mathcal{T}$), and for $\dot{x}$, and thus also for y, the constant function 0. Since the constant function for x is differentiable, resulting in a constant derivative function of zero, no solution exists for initial conditions $\{x \mapsto 1, \dot{x} \mapsto 1, y \mapsto 1\}$. In fact, a solution of this set of equations exists only for *consistent* initial conditions, where consistent initial conditions satisfy not only the system of equations themselves, but also the hidden constraints. In the example, equation $x = 1$ implies the hidden constraint $\dot{x} = 0$, which is obtained after differentiation. Therefore, consistent initial conditions satisfy:

$$x = 1 \tag{A.5a}$$
$$y = \dot{x} \tag{A.5b}$$
$$\dot{x} = 0 \tag{A.5c}$$

which is equivalent to:

$$x = 1 \tag{A.6a}$$
$$y = 0 \tag{A.6b}$$
$$\dot{x} = 0 \tag{A.6c}$$

with (consistent) initial conditions $\{x \mapsto 1, \dot{x} \mapsto 0, y \mapsto 0\}$.

The need to differentiate (a subset of) the system of equations in order to obtain the hidden constraints and define the consistent initial conditions is related to the index of the system of equations. Several index definitions exist. In most cases, the differential index [9] is used. This index

is rather straightforward to determine, due to its constructive definition. Consider the system of equations

$$\begin{aligned} \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) &= \mathbf{0} && \text{(A.7a)} \\ \frac{d}{dt}\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) &= \mathbf{0} && \text{(A.7b)} \\ &\vdots \\ \frac{d^j}{dt^j}\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) &= \mathbf{0}, && \text{(A.7c)} \end{aligned}$$

This system can be written as

$$\begin{aligned} \mathbf{f}_0(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) &= \mathbf{0} && \text{(A.8a)} \\ \mathbf{f}_1(\ddot{\mathbf{x}}, \dot{\mathbf{y}}, \dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) &= \mathbf{0} && \text{(A.8b)} \\ &\vdots \\ \mathbf{f}_j(\mathbf{x}^{(j+1)}, \mathbf{y}^{(j)}, \ldots \ddot{\mathbf{x}}, \dot{\mathbf{y}}, \dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) &= \mathbf{0} && \text{(A.8c)} \end{aligned}$$

To derive the differential index of (A.7a), the variables $\mathbf{x}^{(j+1)}, \mathbf{y}^{(j)}, \ldots \ddot{\mathbf{x}}, \dot{\mathbf{y}}, \dot{\mathbf{x}}$ are treated as purely algebraic variables, depending on $\mathbf{x}$, $\mathbf{y}$, and t. The differential index is then defined as the smallest value of j for which System (A.8) uniquely defines $[\dot{\mathbf{x}}\ \dot{\mathbf{y}}]$ as a function of $\mathbf{x}$, $\mathbf{y}$, and t.

Systems of equations that have an index greater than one are called *higher index* systems. Consider again the system of equations (A.4) with differential index 2. A first differentiation leads to Systems (A.5) and (A.6). A second differentiation leads to $\dot{y} = 0$, so that the ODE $\dot{x} = 0$, $\dot{y} = 0$ is obtained. Even DAEs of (differential) index one may require differentiation in order to reveal hidden constraints, and to allow consistent initialization. Consider

$$\begin{aligned} \dot{x}_0 &= \dot{x}_1 && \text{(A.9a)} \\ x_0 &= 1 && \text{(A.9b)} \end{aligned}$$

Initial conditions that satisfy Equations (A.9) are $\{\dot{x}_0 \mapsto 1, \dot{x}_1 \mapsto 1, x_0 \mapsto 1, x_1 \mapsto 1\}$. These initial conditions, however, do not satisfy the hidden constraint $\dot{x}_0 = 0$, which is obtained after differentiation. Consistent initial conditions need to satisfy

$$\begin{aligned} \dot{x}_0 &= \dot{x}_1 && \text{(A.10a)} \\ x_0 &= 1 && \text{(A.10b)} \\ \dot{x}_0 &= 0, && \text{(A.10c)} \end{aligned}$$

and are, for example, $\{\dot{x}_0 \mapsto 0, \dot{x}_1 \mapsto 0, x_0 \mapsto 1, x_1 \mapsto 1\}$. The differential index, however, is one since System (A.10) can be rewritten by algebraic manipulation as the ODE $\dot{x}_0 = 0$, $\dot{x}_1 = 0$.

Another example of a higher index system is:

$$y^2 = 0 \qquad \text{(A.11)}$$

Differentiation leads to $2y\dot{y} = 0$, which together with (A.11) does not uniquely define $\dot{y}$. A second differentiation leads to $y\ddot{y} + \dot{y}^2 = 0$, which together with (A.11) leads to $\dot{y} = 0$. Although for differential equation solvers Equation (A.11) may be difficult to solve, for algebraic equation solvers, solving Equation (A.11) is not particularly difficult. Also, finding consistent initial conditions for Equation (A.11) is easy: $\{y \mapsto 0\}$.

For the determination of consistent initial conditions it is therefore more relevant to consider the smallest value of j for which System (A.8) uniquely (at least locally) defines $[\dot{\mathbf{x}}\ \mathbf{y}]$ as a function of $\mathbf{x}$, and t. This value of j is the number of times that the original DAE (A.7a) needs to be differentiated in order to be able to determine the consistent initial conditions. If a DAE

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}$$

uniquely (at least locally) defines $[\dot{\mathbf{x}}\ \mathbf{y}]$ as a function of $\mathbf{x}$ and t, the DAE does not contain hidden constraints. This is the case if and only if $\partial\mathbf{f}/\partial\mathbf{z}$ is nonsingular, where $\mathbf{z} = [\dot{\mathbf{x}}\ \mathbf{y}]$. The initial value $\mathbf{c}_1$ for variable $\mathbf{x}$ of such a DAE can be arbitrarily chosen; consistent initial conditions $\{\dot{\mathbf{x}} \mapsto \mathbf{c}_0, \mathbf{x} \mapsto \mathbf{c}_1, \mathbf{y} \mapsto \mathbf{c}_2, t \mapsto c_3\}$ need only satisfy the original equations. This can be seen as follows: differentiation of the function that uniquely defines $[\dot{\mathbf{x}}\ \mathbf{y}]$ as a function of $\mathbf{x}$ and t, yields a function $\mathbf{g}$ that defines $[\ddot{\mathbf{x}}\ \dot{\mathbf{y}}]$ as a function of $\dot{\mathbf{x}}$, $\mathbf{x}$, and t: $[\ddot{\mathbf{x}}\ \dot{\mathbf{y}}] = \mathbf{g}(\dot{\mathbf{x}}, \mathbf{x}, t)$. Therefore, for every initial condition $\{\dot{\mathbf{x}} \mapsto \mathbf{c}_0, \mathbf{x} \mapsto \mathbf{c}_1, \mathbf{y} \mapsto \mathbf{c}_2, t \mapsto c_3\}$ that satisfies the original equation, a valuation $\{\ddot{\mathbf{x}} \mapsto \mathbf{c}_4, \dot{\mathbf{y}} \mapsto \mathbf{c}_5, \dot{\mathbf{x}} \mapsto \mathbf{c}_0, \mathbf{x} \mapsto \mathbf{c}_1, t \mapsto c_3\}$ can be chosen that satisfies the additional equations. Namely $[\mathbf{c}_4\ \mathbf{c}_5] = \mathbf{g}(\mathbf{c}_0, \mathbf{c}_1, c_3)$.

A.1 Structural analysis

Another way of analyzing the properties of systems of equations is by means of *structural* analysis. Structural analysis distinguishes only zero and non-zero values [21]; only the presence of variables in equations is taken into account, not the numerical value of the corresponding coefficients in the Jacobian matrix. Structural analysis of equations is, in general, a more efficient way to detect higher index systems than numerical analysis of the Jacobian matrix. The Pantelides algorithm [16], that is based on structural analysis, identifies the minimal subset of equations that must be differentiated to be able to define consistent initial conditions. Pantelides shows that DAEs that are structurally singular with respect to $[\dot{\mathbf{x}}\ \mathbf{y}]$ require differentiation of the equations in order to reveal the hidden constraints. A system of equations is defined as structurally singular with respect to a certain set of variables if it contains a subset of equations that is structurally singular with respect to the same set of variables. A subset of equations is called structurally singular with respect to a set of variables if the number of these variables that occur in the equation subset is smaller than the number of equations in the subset. The Pantelides algorithm tries to assign each of the variables of $[\dot{\mathbf{x}}\ \mathbf{y}]$ to a different equation. If this is possible, the system of equations is structurally regular. If this is not possible, the algorithm finds the equations that need to be differentiated in order to enable consistent initialization. Structural regularity of equations with respect to $[\dot{\mathbf{x}}\ \mathbf{y}]$ is a necessary, but not sufficient, requirement to allow consistent initialization without differentiation. The reason for this is that a system that is structurally regular may still have a singular Jacobian matrix. Only a few simulators actually provide structural analysis and automatic index reduction by means of differentiation in order to reduce the index and allow consistent initialization. The Dymola [7] simulator is one of them. For the many simulators that do not provide automatic index reduction, the modeler needs to do structural analysis and index reduction him/herself, and/or use systematic modeling techniques that keep the index low. Such a technique is described in [14]. A key element of this technique is the assignment of the variables of $[\dot{\mathbf{x}}\ \mathbf{y}]$ to the equations. The chosen assignment of variables to equations is essentially a bookkeeping method; it is not part of the model. The chosen assignments could be informally indicated by means of comments to the model.

B Use of urgency

B.1 Multiple solutions for delay behavior

In the examples of Section 2.6.6, there was always exactly one solution for the delay behavior of the variables, given the length of the delay interval. In the process below, there are two solutions for the variable x: it can delay according to $x = \sqrt{\mathsf{time}}$ or $x = -\sqrt{\mathsf{time}}$. Since action labels a and b are both urgent, time can progress until time point 16 when x delays according to $x = \sqrt{\mathsf{time}}$, and time can progress until time point 1 when x delays according to $-\sqrt{\mathsf{time}}$:

$$
\begin{array}{l}
\langle\ x^2 = \mathsf{time} \parallel x \geq 4 \rightarrow a : \emptyset : \mathrm{true} \parallel\ -2 \leq x \wedge x \leq -1 \rightarrow b : \emptyset : \mathrm{true} \\
,\ \{x \mapsto \bot, \mathsf{time} \mapsto 0\} \\
,\ (\{x \mapsto \mathsf{cont}\ , \mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{true}, b \mapsto \mathrm{true}\}, \emptyset, \emptyset) \\
\rangle
\end{array}
$$

When x is an algebraic variable instead of a continuous variable, as shown below, the trajectory for x can be discontinuous. Therefore, x can delay according to $+\sqrt{\mathsf{time}}$ for the first 9 time units, until the value of x equals 3. Subsequently, x can delay according to $-\sqrt{\mathsf{time}}$ by making a discontinuous jump to -3. In this way, the process can perform arbitrary long delays. The trajectory of a continuous variable cannot make such a discontinuous jump.

$$
\begin{array}{l}
\langle\ x^2 = \mathsf{time} \parallel x \geq 4 \rightarrow a : \emptyset : \mathrm{true} \parallel\ -2 \leq x \wedge x \leq -1 \rightarrow b : \emptyset : \mathrm{true} \\
,\ \{x \mapsto \bot, \mathsf{time} \mapsto 0\} \\
,\ (\{x \mapsto \mathsf{alg}\ , \mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathrm{true}, b \mapsto \mathrm{true}\}, \emptyset, \emptyset) \\
\rangle
\end{array}
$$

B.2 Urgency and variable scoping

In the process below, an urgent channel h is declared at the process level. Two local variables x are declared by means of variable scope operators.

$$
\begin{array}{l}
\langle\ [\![_{\mathrm{V}}\ \{x \mapsto \mathsf{cont}\}, \{x \mapsto \bot\} :: x^2 = \mathsf{time} \parallel x \geq 1 \rightarrow h!\ ]\!] \\
\parallel\ [\![_{\mathrm{V}}\ \{x \mapsto \mathsf{cont}\}, \{x \mapsto \bot\} :: x^2 = \mathsf{time} \parallel x \geq 2 \rightarrow h?\ ]\!] \\
,\ \{\mathsf{time} \mapsto 0\} \\
,\ (\{\mathsf{time} \mapsto \mathsf{cont}\}, \{h \mapsto \mathrm{true}\}, \emptyset, \emptyset) \\
\rangle
\end{array}
$$

The behavior of the two local variables x is not visible at the process level, outside of the variable scopes. A variable scope can be thought of a 'black box' that hides all locally declared variables. The channel h and the *value* of its associated guard, however, *are* visible at the process level. This is because the channel h is declared globally, at the process level, and because the value of the guard defines whether or not the send or receive action via the channel is enabled, and can thus result in communication. This can be observed at the process level.

There are four solutions for the delay behavior of the process: the local variable x of the first block can delay according to $\sqrt{\mathsf{time}}$ or $-\sqrt{\mathsf{time}}$, and for each possibility, the local variable x of the second block can delay according to $\sqrt{\mathsf{time}}$ or $-\sqrt{\mathsf{time}}$. When both variables delay according to $-\sqrt{\mathsf{time}}$, the process can perform arbitrary delays. When both variables delay according to $\sqrt{\mathsf{time}}$, the process can delay until time point 4, when both guards are true. Then the synchronous execution of the send and receive actions must take place, and the process terminates.

Although the local variables x are not visible outside of the variable scopes, their effect can be observed by observing the enabledness of the send and receive actions via the global channel h. Another way of observing the behavior of the local variables is via two additional global variables y and z, which can be continuous or algebraic. For example:

$$
\langle\ [\![_{\mathrm{V}}\ \{x \mapsto \mathsf{cont}\}, \{x \mapsto \bot\} :: x^2 = \mathsf{time}, y = x \parallel x \geq 1 \rightarrow h!\ ]\!]
$$

$\| \; [\![_{\mathrm{V}} \{x \mapsto \mathsf{cont}\}, \{x \mapsto \bot\} :: x^2 = \mathsf{time}, z = x \; \| \; x \geq 2 \rightarrow h? \;]\!]$
, $\{\mathsf{time} \mapsto 0\}$
, $(\{\mathsf{time} \mapsto \mathsf{cont}, y \mapsto \mathsf{alg} \; , z \mapsto \mathsf{alg} \; \}, \{h \mapsto \mathsf{true}\}, \emptyset, \emptyset)$
$\rangle$

B.3 Explicit and implicit guards

Apart from the *explicit* guards u in process terms such as $u \rightarrow a : W : r$ and $u \rightarrow h \,!\, e : W : r$, there can also be *implicit* guards in the update predicate r. Such guards are specified by means of conditions involving variables with a '$^-$' superscript. The values of such variables, and the values of the guards, are evaluated just before the action is executed.

Implicit guards associated to urgent actions or urgent channels are either superfluous, or they can lead to deadlock. Consider for example:

$\langle \; \mathsf{eqn} \; \dot{x} = 1 \; \| \; x \geq 2 \rightarrow a : \emptyset : x^- \geq 1$
, $\{x \mapsto 0, \mathsf{time} \mapsto 0\}$
, $(\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathsf{true}\}, \emptyset, \emptyset)$
$\rangle$

Here, the implicit guard $x^- \geq 1$ is superfluous since $x \geq 2 \Rightarrow x \geq 1$. In process

$\langle \; \mathsf{eqn} \; \dot{x} = 1 \; \| \; x \geq 2 \rightarrow a : \emptyset : x^- \geq 3$
, $\{x \mapsto 0, \mathsf{time} \mapsto 0\}$
, $(\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathsf{true}\}, \emptyset, \emptyset)$
$\rangle$,

the implicit guard leads to deadlock, because at time point 2, the value of the explicit guard becomes true, preventing further delaying. Actions cannot be executed then either, because the value of the implicit guard is false.

Changing the implicit guard to an explicit one leads to:

$\langle \; \mathsf{eqn} \; \dot{x} = 1 \; \| \; x \geq 2 \wedge x \geq 3 \rightarrow a : \emptyset : \mathsf{true}$
, $\{x \mapsto 0, \mathsf{time} \mapsto 0\}$
, $(\{x \mapsto \mathsf{cont}, \mathsf{time} \mapsto \mathsf{cont}\}, \{a \mapsto \mathsf{true}\}, \emptyset, \emptyset)$
$\rangle$

In this process, the guard becomes true when $x \geq 2 \wedge x \geq 3$. Thus at time point 3, the value of the guard becomes true, preventing further delaying. The action can now be executed.

In the case of non-urgent actions and non-urgent channels, explicit and implicit guards have the same effect: the action can take place when the conjunction of the explicit and implicit guards is true. The delay behavior is not affected by the guards.