A comparative study of process algebras for hybrid systems
Khadim, U.

Published: 01/01/2006

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Dec. 2018
A Comparative Study of Process Algebras for Hybrid Systems

Uzma Khadim
u.khadim@tue.nl
Department of Mathematics and Computer Science
August 7, 2006

Contents
1 Introduction 3
2 General Characteristics of a Process algebra for specifying hybrid systems 4
3 A brief introduction to Process algebras for hybrid systems 8
  3.1 HyPA ................................. 8
  3.2 \( \phi \)-Calculus ........................... 14
  3.3 Hybrid Chi ............................ 23
  3.4 ACP_{hs} ............................. 35
4 About Discontinuities 43
5 Parallelism in Process algebras for hybrid systems 47
  5.1 Delay behaviour ........................ 48
  5.2 Action Behaviour: ...................... 51
6 System invariants 54
7 More about Time determinism 55
8 Available Tools 58
  8.1 Hybrid Chi ............................ 58
  8.2 HyPA ................................. 59
  8.3 \( \phi \)-Calculus .......................... 63
9 A Case study: A Train Gate Controller 65
  9.1 Train Gate Controller in HyPA ............. 68
  9.1.1 Simplifying \( \partial_H(\text{Trains} \parallel \text{Cntr} \parallel \text{Gate}) \) ............. 70
9.2 Train Gate Controller Specification in $\phi$-calculus 74
  9.2.1 Expanding the specification using transition rules: 76
9.3 Train Gate Controller Specification in Hybrid Chi 87
  9.3.1 Simplifying $\partial_{\text{Aio}}(T_{\text{for}} \parallel G_{\text{op}} \parallel C_{\text{idle}})$ 88
9.4 Train Gate Controller Specification in $\text{ACP}_{\text{hs}}$ 91
  9.4.1 Simplifying $\partial_H(\text{Trains} \parallel \text{Cntr} \parallel \text{Gate})$ 93

10 Conclusions 96

11 Appendix 101
  11.1 PROMELA-Hybrid code for the train gate controller specification 101
  11.2 A syntactical comparison of hybrid process algebras 103

List of Tables
1 A Syntactic Comparison of Process Algebras for Hybrid Systems 104

List of Figures
1 Time non-determinism in Hybrid Automata 57
2 Weak time determinism in Hybrid Automata-A thermostat example 57
3 Result of train gate controller simulation in hybrid Chi 60
4 HyPA Simulation Tool 63
5 Train Gate Controller Automata 67
Abstract

A hybrid system is one that exhibits both discrete and continuous behaviour. Embedded systems are typical examples of hybrid systems in which a digital component controls a continuous environment. Examples are a thermostat, storm surge barriers, fire alarms etc. With the advancing integration of software technology in our daily lives and safety critical systems, the interest in use of formal methods in specifying hybrid systems is increasing.

We take four recent process algebras for describing hybrid systems namely Process Algebra for Hybrid Systems, Hybrid Process algebra, $\phi$-Calculus and Hybrid Chi. We compare operators and constructs offered by them for specifying hybrid systems. We point out special features of each process algebra and try to determine what advantages are achieved through these features in terms of expressiveness and ease of modelling. We also present a case study of a railway gate controller that helps us to identify similar constructs of each process algebra and determine its expressiveness.

1 Introduction

We have done a comparative study of process algebras for representing hybrid systems. We take four rather recent process algebras namely Process Algebra for Hybrid Systems (by Bergstra and Middelburg, see [4]), $\phi$-Calculus (by Rounds and Song, see [11]), HyPA (by Cuipers and Reniers, see [15] and [16]) and Hybrid Chi (by Man and Schiffelers, see [17] and [18]). Our intention is as follows:

1. to develop an insight into the essential features of a process algebra for hybrid systems
2. to explore recent literature
3. to compare operators and semantics of different process algebras
4. to draw conclusions regarding usability, expressiveness, ease of modelling etc.
5. to model a simple case study in all four process algebras to accompany the results found in the last step.

The case study of a train gate controller has been taken from [4]. It is rather a simple case study with a train gate crossing. We choose it for a simple case of parallel hybrid processes that exchange messages among each other.

In the following section we proceed with essential features of process algebras for hybrid systems.
2 General Characteristics of a Process algebra for specifying hybrid systems

In this section we try to obtain a concise list of common characteristics of process algebras for hybrid systems.

- **Data:** A process algebra for hybrid systems consists of a data part in addition to process terms. This data part is composed of environment variables that represent physical entities in the environment that is described by the hybrid system under consideration.

- **Modifications:** In the physical world, entities that are modelled by environment variables vary in two ways:
  - Instantaneous changes
  - Gradual changes extended over a period of time
    - **Instantaneous changes:** Usually we find in literature that instantaneous changes to a variable accompany discrete actions of a system. By this we mean that modifying a variable usually requires a system action. Uncontrolled or arbitrary modifications will not be of much use. In order to restrict the modifications, predicates in terms of previous and new values of variables are used in process algebras. Different process algebras have different syntax for *jump-enabling* predicates accompanying actions. By jump we mean instantaneous changes to the value of a variable.
    - **Gradual changes:** Gradual changes are governed by predicates over variables that can be algebraic equations, inequalities, differential equations and differential inclusions. For example, \( 20 \leq x \leq 40 \) can be a predicate that restricts the value of the variable \( x \) between 20 and 40. Immediately the next question that comes to mind is whether the variable \( x \) is allowed to jump during the delay or not? i.e. what are the continuity restrictions on the trajectories of variables? For example, in \( ACP_{hs} \), environment variables can jump finitely often during a delay interval. A variable whose trajectory needs to be continuous during a delay interval is explicitly mentioned by the evolution operator of \( ACP_{hs} \).

Depending upon the continuity requirements of variables, we can categorize functions on time intervals into the following commonly used categories. These functions represent values of environment variables over some time interval. Consider a function \( f : I \rightarrow \Lambda \), where \( I \) is a time interval and \( \Lambda \) denotes the set of possible variable values. \( f \) gives us the values of an environment variable during the interval \( I \).

1. **Continuous functions:** \( f \) is a continuous function if it does not make sudden jumps in the interval \( I \). Roughly speaking, it
means that we can draw the graph of $f$ without lifting the pencil off the paper.

2. **Continuously differentiable functions**: $f$ is said to be continuously differentiable if both $f$ and its first derivative $f'$ are continuous.

3. **Piece-wise Continuous functions**: $f$ is a piece-wise continuous function if $I$ can be divided into finite number of subintervals such that $f$ is continuous in each subinterval of $I$. In other words, $f$ has only finitely many discontinuities (jumps) in the interval $I$.

4. **Piece-wise Continuously Differentiable functions**: In a piece-wise continuously differentiable function $f$, the subinterval $I$ can be divided into a finite number of subintervals such that the function $f$ as well as its first derivative $f'$ are continuous in each subinterval.

- **Representing passage of time**: Gradual changes extended over a period of time bring us to the question of how time passing is represented in our system specifications. Process algebras differ in their approaches towards time. One approach is to have a special variable representing time as in Hybrid Chi. The variable $\text{time}$ in Hybrid Chi refers to time elapsed since the start of a process. Another is to have a delay operator that introduces a delay between subsequent process terms (as in ACP). Yet another approach is not to have any special constructs but simply define a variable with derivative equal to one. (HyPA and $\phi$-calculus follow this approach). This variable can then act as a clock or timer.

- **Guards**: A common requirement of hybrid process algebras is the ability to express guarded actions or guarded delays. A guard is a predicate on the environment variables. A guarded action or guarded delay can only be executed if the guard evaluates to true. For example, we may need to express the following statement in a specification.

  \begin{quote}
  if speed of the vehicle becomes greater than 80 km/hr, release the pressure from the accelerator
  \end{quote}

- **Specifying assumptions about the environment**: Sometimes we may want to study the behaviour of a process under some assumptions about its environment. These assumptions can be in the form of initial conditions or as invariants on complete or part of a system. In $ACP_{hs}$ and Hybrid Chi, initial conditions and system invariants can be specified by means of special constructs. The semantics of the process algebras ($ACP_{hs}$ and hybrid Chi) ensure that the specified predicate holds as desired.

In addition to initial conditions, invariants i.e., predicates on environment variables that should remain true for the whole duration or a certain part of the system execution, can also be described in $ACP_{hs}$ and hybrid Chi.
Semantics: Another consideration of a process theory is what semantics is to be given to a process. The process algebras for hybrid systems that we study here give an operational semantics to processes. (For other semantics of hybrid processes, see [10] and [14]. [10] has specification oriented semantics and [14] has semantics in duration calculus). A process term together with a data part, usually a variable valuation, is called a process. Process algebras HyPA, ϕ-calculus, hybrid Chi and ACP_{hs} all associate a hybrid transition system with a process, with some additions/modifications to the definition of a hybrid transition system in each case.

A hybrid transition system, as defined in [16], is a tuple \((X, \Sigma, T, \varphi)\), where,

1. \(X\) is the state space;
2. \(\Sigma\) is the alphabet of the transition system. Elements of \(\Sigma\) appear as labels to transitions between states;
3. \(T\) denotes the time axis; and
4. \(\varphi\) defines the transition relation of the hybrid system.

The state space \(X\) consists of pairs of process terms and data. In a hybrid system, a state can evolve into another through discrete actions or time delays during which the variable valuation changes. Therefore \(\Sigma\) contains both a set of discrete actions (denoted by \(\Sigma_d\)) and a set of continuous trajectories (denoted by \(\Sigma_c\)). Elements of the time axis \(T\) represent duration of delays. The transition relation \(\varphi\) defined as

\[
\varphi \subseteq X \times ((T \mapsto \Sigma_c) \cup \Sigma_d) \times X,
\]

includes transitions between states through discrete actions and time delays. The function \(T \mapsto \Sigma_c\) gives the trajectory of the data part of a hybrid process during a delay.

Process Equivalence: Equivalence on processes is defined in terms of bisimulation relations on hybrid transition systems. In [19], three different notions of bisimulation for transition systems with data are given. A comparison of behaviour between processes with same data parts is made. The three notions of bisimulation are as follows:

1. State-less Bisimilarity
2. Initially state-less Bisimilarity
3. State-based Bisimilarity

The difference in these three notions is that the behaviour of the two processes can be compared for just a given data part or for all possible data parts. Consider two process terms \(P\) and \(Q\) and a set of all possible data values \(D\).
In state-less bisimilarity, at each transition step, from start till the termination, the behaviour of the two processes is compared in all possible data states. That is, \( \langle P, d \rangle \) and \( \langle Q, d \rangle \) are compared for all \( d \in D \). If \( \langle P, d \rangle \) makes a transition to \( \langle P', d' \rangle \) and \( \langle Q, d \rangle \) makes a similar transition to \( \langle Q', d' \rangle \), then \( \langle P', d' \rangle \) and \( \langle Q', d' \rangle \) are again compared for all \( d' \in D \).

In initially state-less bisimilarity, as the name indicates, only initially the behaviour of the two processes is considered in all possible data states. In all subsequent steps, the two processes are compared only in the data state resulting from the previous step. That is, \( \langle P, d \rangle \) and \( \langle Q, d \rangle \) are compared for all \( d \in D \). If \( \langle P, d \rangle \) makes a transition to \( \langle P', d' \rangle \) and \( \langle Q, d \rangle \) makes a similar transition to \( \langle Q', d' \rangle \), then \( P' \) and \( Q' \) are compared only for data state \( d' \).

In state-based bisimilarity, the behaviour of the two processes is initially compared in a given data state and then in valuations resulting from the previous transition steps.

- **Time determinism:** A consideration for the alternative composition operator of algebraic processes for systems that exhibit timed behaviour is *time determinism*. We explain below what is meant by determinism and time determinism.

According to CSP, in [1], a deterministic process is one in which whenever the process can engage in more than one events, the choice between them is resolved externally by the environment. Whereas, a non-deterministic process is one, that has a range of possible behaviors and some internal phenomena decides between the choices. The choice cannot be influenced or even observed externally, although it may be possible to infer later what choice was made.

In Milner’s CCS (see [2]), a determinate system is defined as one whose behaviour is predictable. That is, if the same experiment is repeated on a determinate system starting with the same initial conditions, then same results are expected each time.

This notion of determinism has been extended to process algebra with timings. A process is said to be timed deterministic if passage of time by itself cannot resolve possible choices. On the other hand, if while delaying, an option of behaviour can be dropped in favor of another, then the process is said to be timed non-deterministic.

Mathematically, time determinism is expressed as follows:

If \( s, s' \) and \( s'' \) are process terms and \( m \) denotes the duration of a delay, then

\[
\overset{m}{\rightarrow} s \rightarrow s' \text{ and } s \overset{m}{\rightarrow} s'' \Rightarrow s' \equiv s''
\]

In hybrid process algebras we have to cater for variable evolution during delays as well.

In literature we find systems exhibiting three types of time determinism.
1. **Strong Time determinism:** In strong time determinism, a delay cannot resolve possible choices in an alternative composition. An alternative composition of different process terms can delay (without resolving choices) under the following conditions:
   - All terms in the composition must be able to delay
   - A common variable evolution during delays is possible for all process terms. Note if all process terms allow more than one possible variable evolution, then a variable evolution is chosen non-deterministically.

   As soon as all process terms cannot delay together, the alternative composition cannot delay and choice must be resolved in favor of doing an action. If no action is possible, then the process deadlocks.

2. **Weak Time determinism:** In weak time determinism, the passage of time cannot resolve choices in an alternative composition among process terms delayable for equal durations. An alternative composition between delayable process terms can delay with a variable evolution that is possible for all process terms, without resolving choices between process terms. If all process terms allow more than one variable evolution, then the choice between these possible evolutions is resolved non-deterministically.

   The choice between process terms is resolved when one of the options remain delayable while others cannot delay. In this case, the choice is non-deterministically resolved between doing an action and a delay.

3. **Time non-determinism:** In time non-determinism, the passage of time resolves choices between delayable process terms or delayable process terms and actions just like resolving choice between actions in alternative composition. An alternative composition cannot delay while retaining choices.

   No preference is given to undelayable processes when resolving choices.

HyPA follows a time non-deterministic approach in alternative composition. \( \phi \)-Calculus and hybrid Chi have strong time determinism. \( ACP_{hs} \) has weak time determinism.

The next section discusses each process algebra and discusses both continuous and discontinuous behaviour of environment variables.

### 3 A brief introduction to Process algebras for hybrid systems

#### 3.1 HyPA

HyPA stands for hybrid process algebra. It is an extension of \( ACP \). A HyPA specification consists of a set of environment variables. Environment variables
are also called *model* variables in HyPA, indicating that they depend upon the system which is to be modelled.

The set of process terms $P$ of HyPA is given by a BNF expression as follows:

$$P ::= \delta \quad \text{Deadlock}$$
$$| \epsilon \quad \text{empty process}$$
$$| a \quad \text{discrete, atomic actions} \quad a \in A \quad \text{a set of actions}$$
$$| c \quad \text{flow clause} \quad c \in C \quad \text{a set of flow clauses}$$
$$| d \gg P \quad \text{reinitialization clause} \quad d \in D \quad \text{a set of reinitialization clauses}$$
$$| \blacktriangleright \quad \text{disrupt operator}$$
$$| \blacktriangleright \quad \text{left disrupt operator}$$
$$| P \oplus P \quad \text{alternative composition}$$
$$| P \odot P \quad \text{sequential composition}$$
$$| P \parallel P \quad \text{parallel composition}$$
$$| P \mid P \quad \text{forced communication}$$
$$| P \ll P \quad \text{left parallel operator}$$
$$| \partial_H(P) \quad \text{encapsulation operator} \quad H \subseteq A$$

HyPA contains constructs to manipulate environment variables during actions and delays of the hybrid system under consideration. We describe below the special features of HyPA.

- **Reinitialization clauses:** As actions are performed, by default the values of variables do not jump. Instantaneous changes to environment variables can be carried out through constructs called *reinitialization clauses*. A reinitialization clause is of the form $[V \mid P_r]$, where $V$ is the set of variables allowed to jump, and $P_r$ is the reinitialization predicate. $P_r$ is in terms of new and previous values of variables. For example,

$$\begin{align*}
x & : x^+ = 0 \\
r & : r^+ = 2 \times r^-
\end{align*}$$

The variables with $+$ superscript denote new values and the ones with $-$ superscript denote old values of variables $x$ and $r$. $V$ is omitted from $[V \mid P_r]$, in case no variables are allowed to jump. The reinitialization clause then acts as a guard on variable values. For example,

$$[\text{temperature}^- = 20] \gg \text{turnoff}$$

It means when the value of temperature is $20^\circ$, do action turnoff.

- **Flow clauses:** To model the behaviour of the environment variables while the system is idling, *flow clauses* are used in HyPA. A flow clause consists
of two parts, a flow predicate and a set of environment variables. It is of the form \((V \mid P_f)\). In HyPA, environment variables are allowed to jump at the start of a flow clause. Variables whose values should remain continuous as a new flow clause takes over are mentioned in the variable part of the flow clause. (Note that this is opposite to a reinitialization clause, where the variables that are allowed to jump are explicitly mentioned.) Thus in flow clause, \((V \mid P_f)\), the variables in \(V\) must not jump at the start of the flow clause. The initial jump of environment variables in a flow clause should be such that the new values of variables satisfy the flow predicate \(P_f\). After the initial jump, the values of the variables continue to vary in a way that satisfies the predicate \(P_f\). A flow predicate can be an algebraic or differential equation or a differential inclusion, etc. In a HyPA model, a reinitialization clause is often used before a flow clause to set the initial values of the continuous variables. For example,

\[
\begin{bmatrix}
\dot{x} & \dot{x}^+ = 1 \\
x & x^+ = 30
\end{bmatrix} \gg \begin{bmatrix}
\dot{x} & 0 \leq \dot{x} \leq 1 \\
x & x > 0
\end{bmatrix}
\]

The flow clause declares \(\dot{x}\) to be a continuous variable, i.e. its value cannot jump at the start of the flow. \(\dot{x}\) is assigned 1 by the reinitialization clause before the flow clause. So all possible trajectories of variable \(\dot{x}\) will start with value 1. \(x\) is also assigned a value but it is not declared to be continuous by the flow clause. Therefore it can jump to any arbitrary value greater than zero at the beginning of the delay.

In HyPA the set of environment variables, the set of discrete actions, the set of flow predicates, the set of reinitialization predicates, the set of time points and the set of possible solutions to flow and reinitialization predicates are all parameters of the HyPA theory. Therefore depending upon the system to be modelled, the notion of solution to a flow predicate can vary over absolutely continuous functions, piece-wise continuous functions, continuously differentiable functions. Similarly the time domain can also be discrete or continuous depending on the system under consideration.

- **Disrupts:** In HyPA flow clauses represent infinite, non-terminating behaviour. A disrupt operator is defined in HyPA through which an action or a new flow clause can interrupt the previous flow clause. The system then continues to behave according to the action or the flow clause following the disrupt operator.

- **Representation of time:** In HyPA there is no special operator to represent passage of time. A variable initialized to zero and with derivative equal to one can be used as a timer. For example, in the following specification, a process delays for 10 time units and then continues as process \(P\).

\[
[ t \mid t^+ = 0 ] \gg ( i \mid i = 1 ) \triangleright [ t^- = 10 ] \gg P
\]

The disrupt operator and the reinitialization clause \([ t^- = 10 ]\) intercept the flow of the system when variable \(t\) becomes 10.
**Semantics:** A HyPA process term together with a variable valuation is called a HyPA process. HyPA has a structural operational semantics. It associates a hybrid transition system with each HyPA process. Three kinds of transition relations are defined:

1. **Action transition**
2. **Time transition**
3. **Termination**

Let $Val$ be the set of all possible valuations of environment variables and $T$ be a set of time points. Let the set of all possible trajectories of variables is given by $F = [0, t] \mapsto Val$, where $t \in T$. An element of $F$ is called a flow. Let $A$ be the set of actions. The transition relations of HyPA are briefly described below:

1. **Action transition relation** $\rightarrow$: $\rightarrow \subseteq (P \times Val) \times (A \times Val) \times (P \times Val)$.

   Labels to action transitions are pairs of actions and valuations. An action transition $(p, \nu) \xrightarrow{a, \nu'} (p', \nu')$ indicates that the process $(p, \nu)$ does action $a$ in valuation $\nu'$ and becomes $(p', \nu')$. The valuation of the action label equals the valuation of the target process.

2. **Time transition relation** $\sim$: $\sim \subseteq (P \times Val) \times F \times (P \times Val)$.

   Labels to time transitions are flows from the set $F$. A time transition $(p, \nu) \xrightarrow{\sigma} (p', \nu')$, where $\text{dom}(\sigma) = [0, t]$, indicates that the process $(p, \nu)$ delays for $t$ time units and becomes $(p', \nu')$. The trajectory of environment variables during the delay is described by $\sigma$. The valuation of the target process equals the final valuation given by the flow, i.e., $\nu' = \sigma(t)$.

3. **Termination relation** $\sqrt{}$: $\sqrt{} \subseteq (P \times Val)$.

   $(p, \nu) \sqrt{}$ indicates that $(p, \nu)$ terminates immediately. An example of termination relation is $(\epsilon, \nu) \sqrt{}$, where $\nu$ is any valuation. $(\epsilon, \nu) \sqrt{}$ indicates that the empty process can terminate in any valuation.

**Bisimulations** Two types of bisimulations are defined in HyPA. One is called the robust bisimulation (denoted by $\approx_r$) that matches the notion of Stateless bisimulation. The other called bisimilarity (denoted by $\approx$) matches the notion of initially stateless bisimulation. The intuition behind the semantics of HyPA is that only the transition labels of a transition system are visible to the outside world. The bisimulation relations in
HyPA do not consider the data parts of processes under consideration but relate two processes in HyPA only on the basis of similar transition labels of the associated hybrid transition systems. In literature, always the behaviour of hybrid process terms in the same data state are compared. Since the labels of action and time transitions in HyPA include the data part (variable valuation) of the target processes therefore at each stage of bisimulation only processes with the same data parts are compared. Bisimilarity is only applicable for analysis of parts of the systems which have been linearized, i.e. the parallel operator has been removed from them. The parallel operator does not preserve bisimilarity. Two process terms that are bisimilar to each other may not remain bisimilar when composed in parallel with a third process. For example consider process terms $X$ and $Y$, where,

$$X: [| x | x^+ = 1] \gg a_1 \circ [| x^- = 1] \gg a_2$$

$$Y: [| x | x^+ = 1] \gg a_1 \circ a_2$$

The symbol $\circ$ denotes sequential composition. $X$ and $Y$ are bisimilar to each other. The second reinitialization does not change the value of variable $x$ but only acts as a guard. Let $Z$ be another process also containing environment variable $x$.

$$Z: [| x | x^+ = 2] \gg a_3$$

$X \parallel Z$ is not bisimilar to $Y \parallel Z$. As a sequence of actions $a_1a_3a_2$ is possible for $Y \parallel Z$, but performing $a_2$ after $a_3$ is not possible for $X \parallel Z$ because of the guard $[| x^- = 1]$. This happens because of variable sharing between the processes composed in parallel. In a parallel composition of $p \parallel r$, at any stage in the execution of $p$, $r$ may change the valuation that can affect the behaviour of $r$. Initially state-less bisimilarity does not cater for possible interferences that change the valuation during the execution of a process.

- **Time non-determinism:** HyPA’s alternative composition operator is time non-deterministic. This means that a choice between process terms has to be resolved immediately. A choice between two flow clauses is resolved just like it is resolved for actions in alternative composition.

For example, consider the following alternative composition:

$$\{(x \mid \dot{x} = 1) \oplus (x \mid \dot{x} = 2), \{x \mapsto 0\}\}$$

$\{x \mapsto 0\}$ represents the variable valuation. The above process can evolve for five time units in two ways:

$$\sigma_1: \{(x \mid \dot{x} = 1), \{x \mapsto 5\}\}$$

Or as follows,

$$\sigma_2: \{(x \mid \dot{x} = 2), \{x \mapsto 10\}\}$$

Here $\sigma_1$ represents a flow where $\dot{x} = 1$, and $\sigma_2$ represents a flow with $\dot{x} = 2$. 

12
Arguments:
Some arguments given in [16] in favor of time non-deterministic choice operator are as follows:

1. In hybrid systems, during delays, continuous evolution of variables can resolve choices between process terms in alternative composition just like actions do.

2. In the presence of a time non-deterministic operator, the definition of a variable abstraction operator becomes easier. In variable abstraction, the evolution of variables during delays is hidden. Therefore in an alternative composition with variable abstraction, although we do not know how the variables evolve during delay, but we know the choice enforced by the evolution.

Observations:
Some other observations about the choice operator of HyPA are as follows:

1. Due to non-determinism, an alternative composition will not deadlock because of incompatibility of flow clauses with each other.

2. The deadlock process $\delta$ of HyPA cannot perform an action or delay. The following axiom holds in HyPA

$$\delta \oplus p = p$$

In an alternative composition, deadlock is only possible when there is no possibility of doing an action or delay.

3. In an alternative choice between an action and a flow clause, no preference is given to the action.

4. In HyPA, the choice resolution cannot be delayed even if the process terms in alternative composition may allow mutual evolution of variables. Consider another HyPA expression,

$$\langle (x \mid \dot{x} = 1) \oplus (x \mid \dot{x} \geq 1), \{x \mapsto 0\}\rangle$$

Although the components of the alternative composition can perform a mutual flow, still, choice between them is resolved at the start of delay. The result of delaying for any duration $t$ must be either one flow clause or the other and not both.

For example, two possible time transitions of duration $t$ for the given alternative composition are as follows:

$$\langle (x \mid \dot{x} = 1) \oplus (x \mid \dot{x} \geq 1), \{x \mapsto 0\}\rangle \overset{\sigma_1}{\rightarrow} \langle (x \mid \dot{x} = 1), \{x \mapsto t\}\rangle$$

and,

$$\langle (x \mid \dot{x} = 1) \oplus (x \mid \dot{x} \geq 1), \{x \mapsto 0\}\rangle \overset{\sigma_1}{\rightarrow} \langle (x \mid \dot{x} \geq 1), \{x \mapsto t\}\rangle$$

where, $\sigma_1$ represents a flow with $\dot{x} = 1$. 
HyPA has a large set of axioms and some derivation rules. These axioms and derivation rules are sound with respect to robust bisimulation. A HyPA system specification can be simplified by repeatedly applying these rules and axioms. Every HyPA term can be written as a basic term which is free of the parallel operator. Elimination of parallel operator greatly simplifies a system. Two axioms that are not robust bisimilar but initially stateless bisimilar can be applied on system specifications without parallel operator. We find them useful in deriving results regarding the possible values of environment variables.

In HyPA there is no consistency concept of a hybrid system as present in some hybrid process algebras. There are no signal emissions in HyPA. Therefore the need for consistency predicates is reduced to only flow-clauses. A process \( \langle c, \nu \rangle \) is allowed to flow only if a solution to a flow-clause \( c \) exists and is reachable from valuation \( \nu \). Reinitializations can be used to initialize a system. Reinitialization as difference equations can be used as guards on environment variables, enforcing the valuation in which it is required to perform a certain action.

### 3.2 \( \phi \)-Calculus

Another process algebra for specifying and modeling hybrid systems is \( \phi \)-calculus. \( \phi \)-calculus is an extension of \( \pi \)-calculus. \( \pi \)-calculus is a process algebra to model reconfigurable systems. A reconfigurable system is one in which the sub-processes keep on making new connections among themselves. In a \( \pi \)-calculus system specification, a set of link-names is given. These names serve as channels of communication between sub-processes. Basic actions in \( \pi \)-calculus are send and receive on a link. Simultaneous execution of a send and a receive action on the same link-name result in a silent action. Messages exchanged in \( \pi \)-calculus comprise of values and link-names. Passing of a link-name to a process gives it access to other processes using the same link name. In this way, a system that changes its configuration can easily be modeled in \( \pi \)-calculus. In \( \phi \)-calculus specifications, in addition to link names, a set of environment names is given. In \( \phi \)-calculus, messages exchanged on links can also consist of environment names besides values and link names.

In this section, we refer to [11] and an unpublished later manuscript of \( \phi \)-calculus (see [12]).

The set \( P \) of \( \phi \)-calculus process terms can be defined by a BNF expression as follows:

\[
\text{(A set of environment names } X \text{ and their derivatives } \dot{X} \text{ are assumed. } x, y, z \text{ vary over environment variable names. } a, b, c \text{ vary over channel names. } \vec{x}, \vec{a} \text{ represent vectors of names. } \delta \text{ is a special action prefix with no corresponding)}
\]
\[ P ::= 0 \] Null process.
\[ \nu x P \] Local environment variable.
Declares a private environment variable \( x \) for \( P \).
\[ \nu a P \] Local channel name.
Declares a private channel \( a \) for \( P \).
\[ P \mid P \] Parallel Communication
\[ !P \] Replication. Represents infinitely many instances of \( P \) running in parallel.
\[ !P \equiv !P \mid P \equiv !P \mid P \mid P \ldots \]

\[ S \]
\[ S ::= \tau . P \] Silent action prefix
\[ \delta . P \] Delay action prefix.
\[ aF \] \( a \)-receive on channel \( a \)
and continue as abstraction \( F \).
\[ \bar{a}C \] \( \bar{a} \)-send on channel \( a \)
and continue as concretion \( C \).
\[ (S + S) \] Sums
\[ [\gamma \rightarrow \vec{x} := \vec{c}] \cdot P \] Assignment action prefix
\[ \gamma \] is a predicate on variables from \( X \) and \( \dot{X} \)
If \( \gamma \) is true the variables in \( \vec{x} \) are assigned values in expression \( \vec{c} \).
\[ [\gamma \rightarrow \langle \text{resetlist} \rangle \text{ with} \langle \text{clauselist} \rangle] \cdot P \] Reset action prefix
\( \langle \text{resetlist} \rangle \) contains an unordered list of variables from \( X \) and \( \dot{X} \).
\( \langle \text{clauselist} \rangle \) contains a set of predicates on these variables.

\[ C ::= (\nu \vec{y})(\vec{x})P \mid (\nu \vec{b})(\vec{a})P \] Concretion-a process preceded by a send action
without the channel name on which message is sent
\( \vec{x} \)- a vector of variable names
\( \vec{a} \)- a vector of channel names
\( \vec{x}, \vec{a} \) are the contents of the message passed

\[ F ::= (\vec{x})P \mid (\vec{a})P \] Abstraction-a process preceded by a receive action
without the channel name on which message is received.
\[ A ::= F \mid C \] Agents
Special features of $\phi$-calculus to represent hybrid processes are as follows:

- **Data:** The data part of a $\phi$-calculus process contains more information than generally found in the data parts of other process algebras for hybrid systems. The data part in $\phi$-calculus is called an *environment*. A $\phi$-calculus process is a pair $(E, P)$, where $E$ is the environment and $P$ is the $\phi$-calculus process term. An environment is like a store which is frequently accessed and reset by the associated process term. It consists of two parts which are given as follows:

  - the valuation of environment variables; and
  - a set of flow constraints on variables and their derivatives. The variables and their derivatives must evolve according to the constraints mentioned in the environment as the process $(E, P)$ idles.

An example of an environment from the railroad crossing specification (see section 9 for complete specification) is as follows:

$$\begin{align*}
  x & : -792, r : 10 \\
  \{\dot{x} \mid 40 \leq \dot{x} \leq 52\}, \\
  \{x \mid -1000 \leq x \leq 0\}, \\
  \{\dot{r} \mid \dot{r} = -20\}, \\
  \{r \mid 0 \leq r \leq 90\}
\end{align*}$$

A flow constraint consists of two parts.

- One a predicate on variables and their derivatives.
- Second a set of environment variables (and their derivatives) that must evolve according to the predicate.

In flow constraint, $\{x \mid -1000 \leq x \leq 0\}$, $x$ on the left, indicates the variable whose evolution is dependent on the predicate $-1000 \leq x \leq 0$. $x$ is called a resetable name of the flow constraint. The predicate may contain environment variables other than $x$.

- **Environmental actions:** Discrete modifications to the environment are carried by means of *environmental actions*. There are two types of environmental actions:

  - assignments—that modify the variable valuation.
  - resets—that modify the set of flow constraint.

For example, consider the reset action

$$\left[ x \leq -750 \rightarrow \text{reset } \dot{x}, \dot{r} \text{ with } \{\dot{x} \mid 35 \leq \dot{x} \leq 52\}, \{\dot{x} \mid \dot{x} = -9\} \right]$$

$x \leq -750$ is the guard of the reset action. $\dot{x}, \dot{r}$ is a list of variables whose constraints will be updated. $\{\dot{x} \mid 35 \leq \dot{x} \leq 52\}, \{\dot{x} \mid \dot{x} = -9\}$ is a list of
new constraints. This reset action will modify the above environment as follows:
\[
\begin{pmatrix}
  x : -792, r : 10 \\
  \{\{\dot{x} | 35 \leq \dot{x} \leq 52\}, \\
  \{x | -1000 \leq x \leq 0\}, \\
  \{\dot{r} | \dot{r} = -9\}, \\
  \{r | 0 \leq r \leq 90\}\}
\end{pmatrix}
\]

- Flows: There is no special construct in $\phi$-calculus associated with delays. The action prefixes $\bar{a}, a, \tau, \delta$, environmental actions with false guards are all delayable actions. They can delay as long as the environment associated with the process term they are prefixing can delay. During delays, the variables and their derivatives must vary according to the flow constraints in the environment. The set of possible solutions to flow constraints is a parameter of the $\phi$-calculus theory. In [12], continuously differentiable functions are allowed as possible trajectories of variables during delays. As the values of variables evolve, a false guard of an environmental action may become true. Environmental actions with true guards prefixing a process term are urgent and must be executed immediately. Environmental actions update the environment either by assigning a new value to a variable or by resetting a flow constraint. After the update, the environment can flow according to the new conditions.

A silent action followed by an environmental action with true guard is also urgent. (This is required to preserve weak bisimulation among process terms. More is explained when we discuss bisimulation for $\phi$-calculus). As a consequence, a communication on a channel becomes urgent, as soon as the guard of an environmental action following the send or receive actions (on that channel) becomes true.

Consider the following example:
\[
( \begin{pmatrix}
  x : 100 \\
  \{\{x | x \leq 500\}, \\
  \{\dot{x} | \dot{x} = 50\}\}
\end{pmatrix}, \bar{a}, [x := -1400] | a )
\]

In the given process, the environment can delay as the value of $x$ is well within the limit set by the flow constraints, but the process term cannot delay. As the environmental action $[x := -1400]$ has a true guard. (When no guard is mentioned, it amounts to a true guard. ) Therefore the communication on channel $a$ must take place immediately followed by the update of variable $x$. In this example no message is being exchanged during communication on channel $a$.

In order to make communication on a channel indefinitely delayable, the delay action $\delta$ can be prefixed to the send or receive actions on that channel. The communication on that channel can then be made urgent only
by first discharging the delay action.

\[
\left( \begin{array}{c}
\{ x : 100 \\
\{ x \mid x \leq 500 \} \\
\{ \dot{x} : \dot{x} = 50 \} \
\end{array} \right), \delta. \overline{a}. \left[ x := -1400 \right] | a
\]

The above process can delay as long as the flow constraints in the environment are satisfied. Only the execution of \( \delta \) by the process term on the left hand side of the parallel operator, the communication on channel \( a \) can be made urgent.

- **Guards:** As mentioned before environmental actions are guarded by predicates on variables and their derivatives. False guards can delay till they become true. When a guard of an environmental action prefixing a process term becomes true, then the environmental action must takes place immediately.

In order to make a send or receive action on a channel guarded, two steps need to be taken.

- An environmental action with only the desired guard (rest of the environmental action is empty) is prefixed to the send or receive action.
- The send and receive actions are arbitrarily delayable. Therefore a flow constraint limiting the flow of the environment till the guard conditions, is added in the environment of the process.

The guard placed before the send or receive action ensures that the process term delays till the guard predicate becomes true. The flow constraint ensures that the environment cannot flow further. For example, the following process will send a message on a channel \( \text{go} \) as soon as the value of variable \( \text{count} \) becomes greater than ten.

\[
\left( \left\{ \text{count} : 0 \right\} \left\{ \left\{ \text{count} \mid \text{count} \leq 10 \right\} \right\}, \left[ \text{count} = 10 \right] \overline{\text{go}} \right)
\]

- **Private Channels:** \( \nu a P \) declares a channel \( a \) private to process term \( P \). Only the sub processes of \( P \) are allowed to communicate among themselves on channel \( a \). Communication on channel \( a \), with any external process that is not a subprocess of \( P \) is not possible.

- **Private variables:** \( \nu z P \) declares a variable \( z \) that is local to process term \( P \). In order to avoid confusion with another variable of the same name, every instance of \( z \) in \( P \) is replaced by a name that is fresh in \( P \) and the environment associated with it, before any environmental action or delay involving \( z \) can take place.

Private variables and channels can be passed in messages to other processes. Passing of a private variable or channel widens its scope to the receiving process term.
• **Representation of passage of time** There are no special constructs to represent time in $\phi$-calculus. The following process term specifies a delay of 10 time units before continuing as the process term $P$.

$$\nu t \left( [ t := 0 ] . \left[ \text{reset } t, i \text{ with } \{ t \mid t \leq 10 \}, \{ i \mid i = 1 \} \right] . [ t = 10 ] . P \right)$$

• **Concretions and abstractions** A concretion is a process term which is preceded by a send action that does not mention the channel name on which message is sent. Similarly an abstraction is a process which is preceded by a receive action that does not mention the channel name on which message is received. Concretions and abstractions are called *agents*. They were introduced in $\pi$-calculus by Milner (see [3]) in order to make the semantics easier. With the help of agents, the labels of the send or receive transitions can consist of just the channel name and the details of the message exchanged are left with the resulting agents.

Examples of concretions are $\langle \vec{x} \rangle . P, \langle \vec{b} \rangle . P$ or simply $P$. In the first case a vector $\vec{x}$ of environment variables is sent. In the second case a vector $\vec{b}$ of channels is sent as a message. The concretion $P$ indicates that no message is sent during communication. Private channels or variables can also be sent in messages. The concretion $\nu \vec{y}(\vec{x}) . P$, where $\vec{y} \subseteq \vec{x}$, indicates that among the $\vec{x}$ variables sent, the variables in vector $\vec{y}$ are private to the process term $P$.

Examples of abstractions are $(\vec{y}) . P, (\vec{b}) . P$ or simply $P$. Passing of a private channel or variable in a message increases the scope of that channel or variable. For example in,

$\bar{a}(\nu b)(b) . P \mid a(c) . Q$

$b$ is a channel private to $P$. A communication between the two processes will result in the scope of $b$ being widened to include process term $Q$. i.e.,

$\nu b(P \mid Q[c \leftarrow b]),$

where $Q[c \leftarrow b]$ is a process term obtained by replacing all instances of $c$ in $Q$ by $b$.

• **Semantics:** The behaviour of a process $(E, P)$ is described by following transitions.

1. A pi-action transition.
2. An environmental action transition.
3. A delay action transition.
4. A flow transition.
1. A $\pi$-action includes sending or receiving actions on a link or the silent action $\tau$. A $\pi$-action transition changes only the process term and does not effect the environment.

We give here an example to illustrate a $\pi$-action transition:

$$( E , \bar{a}(6) . P | a(x).Q )$$

$$\xrightarrow{\pi} ( E , P | Q[x \leftarrow 6] )$$

$Q[x \leftarrow 6]$ indicates that all instances of $x$ in $Q$ are replaced by value 6.

A silent action also results when the local environment operator $\nu$ is discharged. The local variable is given a name that is fresh in the process term and the associating environment.

For example,

$$\begin{pmatrix}
  \nu z \left[ z := 60 \right] . \bar{c}(z) . 0
\end{pmatrix}
\xrightarrow{\pi}
\begin{pmatrix}
  [ z' := 60 ] . \bar{c}(z') . 0
\end{pmatrix}$$

$\emptyset$ in the above example represents that there are no constraints on the flow of environment variables.

2. An environmental action is an assignment to a variable or a reset of a flow constraint. It modifies both the environment and the process term.

$$\left\{ \begin{array}{c}
  \begin{pmatrix}
    z : 10
  \end{pmatrix},
  \nu z \left[ z := 60 \right] . \bar{c}(z) . 0
\end{pmatrix}
\\
\begin{pmatrix}
  [ z' := 60 ] . \bar{c}(z') . 0
\end{pmatrix}
\end{array} \right.$$
function when evaluated at instant zero equals the valuation in $E$ (when restricted to a domain consisting of only variables present in the valuation of $E$).

For example consider a heater controlled by a thermostat. The thermostat keeps the temperature of a room between 18° and 20°. Initially the temperature is 18°, heater is on and temperature is increasing. (This example has been taken from [4]. We have not specified the complete example.)

$$
\begin{array}{l}
T : 18 \\
\{ \{ T \mid 18 \leq T \leq 20 \}, \\
\{ T \mid \dot{T} = -T + 22 \}\}
\end{array}, \; [T = 20] \cdot \text{turnoff} \cdot \text{Th off}
$$

The environment of the above process can delay till the temperature becomes 20°. The equation $T = (22 e^t - 4) / e^t$, where $t$ denotes time, is a solution to the given flow constraints, with temperature at time zero equal to 18°. The above process can make the following flow transition:

$$
\overrightarrow{x[0,ln2]} ( \begin{array}{l}
T : 20 \\
\{ \{ T \mid 18 \leq T \leq 20 \}, \\
\{ T \mid \dot{T} = -T + 22 \}\}
\end{array}, \; [T = 20] \cdot \text{turnoff} \cdot \text{Th off})
$$

where $x$ represents the flow represented by $T = (22 e^t - 4) / e^t$. Now the environment cannot flow any further. After discharging the environmental guard $[T = 20]$, a send action on channel turnoff must take place immediately.

$$
\overrightarrow{T=20} ( \begin{array}{l}
T : 20 \\
\{ \{ T \mid 18 \leq T \leq 20 \}, \\
\{ T \mid \dot{T} = -T + 22 \}\}
\end{array}, \; \text{turnoff} \cdot \text{Th off})
$$

- **Time-determinism:** The alternative composition operator of $\phi-$calculus is strong time deterministic. The choice between alternatively composed processes is resolved at the instant when first action is performed. The choice is delayed as long as all processes can delay.

In $\phi$-Calculus, flow clauses are not a part of process terms, but a part of the environment associated with the process terms. For alternative composition, the individual delay behaviour of component process terms in the same environment is considered. Therefore a conflict due to different flow clauses cannot arise here as in hybrid Chi or $ACP_{hs}$. However the duration of the delay of alternative composition is effected by its component process terms.

Consider the following process:

$$
\overleftarrow{x[0]} ( \begin{array}{l}
\{ \dot{x} \mid \dot{x} \geq 0 \} \\
\{ x \mid x \leq 50 \}\}
\end{array}, \; (\overrightarrow{\alpha(x)} [ x \geq 30 ] \cdot P \parallel a(y).0) + b(y).Q ),
$$
where \( P \) and \( Q \) are two \( \phi \)-calculus process terms. The above process can delay as long as the value of \( x \) remains less than or equal to 30. When \( x = 30 \), the action in the first process of the alternative composition becomes urgent. The choice is then resolved in favor of any of the two process terms depending upon the first action performed.

- **Bisimulations:** Bisimulation is defined in two steps. In the first step, two phi-calculus process terms are compared with respect to discrete behaviour. Action prefixes and environmental actions are considered in this step. Environmental actions are taken to be able to do transitions like other action prefixes without regard to the environment. In the second step, two processes that are bisimilar according to the first step, are placed in same environments and their flow behaviour is compared. Two kinds of bisimulations, viz weak bisimulation and strong are defined. Weak bisimulation relates the behaviour of two process terms while abstracting from \( \tau \) action information. Strong bisimulation is the other notion of comparison. Strong bisimulation considers \( \tau \) actions also.

As mentioned before a \( \tau \) action preceding an environmental action with a true guard is made urgent to preserve weak bisimulation. Consider the following example:

\[
P = \nu a (\sigma a \cdot [x := 1] || \bar{a}) \text{ and } Q = [x := 1]
\]

In \( P \), \( \nu \) declares a private link \( a \). \( a \) is a receiving action on link \( a \). \( \bar{a} \) represents a sending action. In this example of sending and receiving no messages are being exchanged. \([x := 1]\) is an assignment with a true guard. We can see that process terms \( P \) and \( Q \) are weakly bisimilar according to the first step. That is they are weakly bisimilar with respect to action behaviour. Consider an environment \( E \):

\[
\begin{cases}
x : 0 \\
{x : \{ [x \mid \dot{x} = 1] \}}
\end{cases}
\]

Process \( Q \) cannot wait as it has an environmental action with a true guard. If \( \tau \) action preceding the assignment \([x := 1]\) is not urgent then process \( P \) can wait. Therefore the delay behaviour of two processes that are weakly bisimilar with respect to actions, can differ if \( \tau \) actions are unconditionally delayable.

There are no consistency predicates defined in phi-calculus. The flow clauses defined act like flows of hybrid automata. So an environment variable can be assigned a value that violates a flow clause but then the environment cannot flow till the variable is assigned another value or the violated flow clause is removed. The variables can also be assigned an unknown initial value \( \bot \). New variables can be added in the environment when a flow constraint is added in the environment by a reset action and it (the flow constraint) mentions a variable not already defined in \( E \). We don’t know how the evolution of a variable with \( \bot \) valuation will take place or what will be the initial value of a variable introduced through a flow constraint.
3.3 Hybrid Chi

Hybrid Chi is another process algebra for modelling hybrid systems. The algebra is provided with a number of operators to manipulate the environment variables as the system performs actions or idles over time. It has CSP like communication constructs for exchanging values among sub processes. Algebraic and differential equalities or inclusions can be hybrid Chi process terms and express evolution of variables during a delay. The alternative composition operator of hybrid Chi is strongly time deterministic. This property plays an important role in determining the duration of delays, therefore we discuss time determinism together with the delay predicates.

The set of process terms $P$ is defined with the help of a BNF expression given below:

$$p ::= W : r \gg l_a$$

<table>
<thead>
<tr>
<th>action predicate</th>
<th>$W$</th>
<th>set of variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$ predicate</td>
<td></td>
<td>$r$ predicate</td>
</tr>
<tr>
<td>$l_a$ action label</td>
<td></td>
<td>$l_a$ action label</td>
</tr>
<tr>
<td>$u$ delay predicate</td>
<td></td>
<td>$u$ predicate</td>
</tr>
<tr>
<td>$\delta$ Deadlock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\bot$ Inconsistent process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$[p]$ any delay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$u \land p$ signal emission</td>
<td></td>
<td>$u$ predicate</td>
</tr>
<tr>
<td>$p ; p$ sequential composition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b \rightarrow p$ guard</td>
<td></td>
<td>$b$ predicate</td>
</tr>
<tr>
<td>$p \parallel p$ parallel composition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h!e_n$ send process term</td>
<td>$h$ channel</td>
<td>$e_n$ expression vector</td>
</tr>
<tr>
<td>$h?x_n$ receive process term</td>
<td>$h$ channel</td>
<td>$x_n$ variable vector</td>
</tr>
<tr>
<td>$\partial_A(p)$ action encapsulation</td>
<td>$A$ Set of action labels</td>
<td></td>
</tr>
<tr>
<td>$v_H(p)$ urgent communication</td>
<td>$H$ Set of channels</td>
<td></td>
</tr>
<tr>
<td>$X$ recursion variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\iota_{J^+}(p)$ jump enabling</td>
<td></td>
<td>$J^+$ set of variables</td>
</tr>
<tr>
<td>$[v \sigma_{\bot}, C, L]$ variable scope</td>
<td>$C$ set of variables</td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\bot}$ valuation</td>
<td></td>
<td>$L$ set of variables</td>
</tr>
<tr>
<td>$[H H']p]$ channel scope</td>
<td>$H$ set of channels</td>
<td></td>
</tr>
<tr>
<td>$[R R']p]$ recursion scope</td>
<td>$R$ set of recursive definitions</td>
<td></td>
</tr>
<tr>
<td>$p_{ext}$ syntactic extensions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Features of hybrid Chi that enable specification of hybrid systems are as
follows:

- **Data**: The data part of a hybrid Chi process is more elaborate than just a variable valuation. It does not include the flow constraints as in $\phi$-calculus. The most distinguishing feature of a hybrid Chi process is that the environment variables have been divided into different categories. This categorization reflects the behaviour of a variable during delays or action execution.

The behaviour of environment variables during delays defines following five categories among variables:

1. **Discrete** ($D$): The variables belonging to this category remain constant as a hybrid Chi process delays. These variables can only be modified discreetly when the process executes an action, e.g. an assignment statement or a receive action from a channel.

2. **Continuous** ($C$): Continuous variables vary over time as the process delays with a restriction that their trajectories are absolutely continuous functions of time.

3. **Dotted Continuous** ($\dot{C}$): Dotted continuous variables are the derivatives of continuous variables. Their trajectories can be discontinuous functions of time.

4. **Algebraic** ($L$): Algebraic variables can also vary over time like continuous variables with a possibly discontinuous trajectory.

5. **A special variable ‘time’**: The variable time represents time passed since the beginning of the process.

Discrete and Continuous variables are further subdivided into jumping or non-jumping variables depending on their behaviour during actions:

1. **Jumping** ($J$): The values of variables declared as jumping change arbitrarily when the system performs an action.

2. **Non-Jumping**: Non-jumping variables remain constant as an action is performed.

Dotted continuous and Algebraic variables can always jump while an action is executed. The variable time is by definition non-jumping and continuous.

A hybrid Chi process is a triple $\langle p, \sigma, E \rangle$, where,

1. $p$ is a hybrid Chi process term;
2. $\sigma$ is a variable valuation; and
3. $E$ is the hybrid Chi environment.

The variable valuation and the environment constitute the data part of a hybrid Chi process.

The **variable valuation**, denoted by $\sigma$, consists of the valuations of the following variables:
– Discrete variables;
– Continuous variables; and
– the special variable time.

Environment (E) of a hybrid Chi process, is different from the environment in $\phi$-calculus. It consists of the following information:

– information about the categorization of variables, i.e. to which category an environment variable belongs;
– a set of channel names; and
– a definition of recursive variables

Formally an environment is a five tuple, i.e. $(C, J, H, L, R)$, where

1. $C$ is the set of continuous environment variables
2. $J$ is the set of environment variables declared as jumping
3. $L$ is the set of algebraic variables
4. $H$ is a set of channel names
5. $R$ is a set of recursive definitions.

$R$ is of the form $\{X_1 \mapsto P_1, \ldots, X_n \mapsto P_n\}$, where $X_1, \ldots, X_n$ are recursive variables and $P_1, \ldots, P_n$ are hybrid Chi process terms.

The environment, together with the variable valuation $\sigma$, completely describes the categorization of variables. For example, the set $D$ of discrete variables is $(\text{dom}(\sigma) \setminus (C \cup \{\text{time}\}))$ and the set of discrete jumping variables is $D \cap J$. The environment variables, channel names and recursive variables occurring free in the process term of a hybrid Chi process or in the defining process term of a recursive variable in $R$, must be defined either in the environment or the valuation of the hybrid Chi process.

The dotted and algebraic variables are not included in the valuation $\sigma$ in a hybrid Chi process because they can jump when an action is performed and their trajectories can be discontinuous. Therefore what course they follow in future does not depend upon their current values. But they are included in the so-called extended valuations. The extended valuation is the valuation of a process extended with valuations of dotted continuous and algebraic variables. The extended valuations are used in describing the semantics of hybrid Chi processes.

An example of a continuous function and a discontinuous derivative can be speed of a car and its acceleration. The following hybrid Chi process denotes the motion of a car. The velocity ($v$) of the car remains within the range of $50km/hr$ to $200km/hr$ and its acceleration/deceleration ($\dot{v}$) between $-10km/hr/sec$ to $10km/hr/sec$.

\[
\langle 50 \leq v \leq 200 \land -10 \leq \dot{v} \leq 10, \\
\{v \mapsto 50, \text{time} \mapsto 0\}, \\
\{(v), \emptyset, \emptyset, \emptyset, \emptyset\}\rangle
\]
The environment consists of only one variable \( v \). The variable \( v \) is declared to be non-jumping and continuous. The sets of jumping variables, algebraic variables, channels and recursive definitions in the environment are empty. The valuation gives an initial value 50 to \( v \). The process term of the above process comprises of a delay predicate on \( v \) and its derivative. What the value of \( v \) should be at an instant say 15, depends not only at the predicate \( 50 \leq v \leq 200 \), but also at the value of \( v \) as time approaches 15 seconds. But the value of \( \dot{v} \) at any time instant \( t \), is independent of its value in the vicinity of \( t \) time units.

Below we give the model of train gate controller in hybrid Chi. (See section 9 for the description of the case study and its complete hybrid Chi specification).

\[
\langle \partial_{A_{\text{Act}}} (v_{\{\text{appr,exit,raise,lower}\}}) \\
(\text{Train} \parallel \text{Gate} \parallel \text{Controller}) \\
), \\
\{x \mapsto -1400, y \mapsto -52, r \mapsto 90, d \mapsto 0, \text{time} \mapsto 0\}, \\
( \\
\{x, y, r, d\}, \emptyset, \emptyset, \\
\{\text{appr,ext,raise,lower}\}, \\
\{\text{Train} \mapsto T_{\text{far}}, \\
\text{Gate} \mapsto G_{\text{op}}, \\
\text{Controller} \mapsto C_{\text{idle}}, \\
T_{\text{far}} \mapsto \ldots, \\
G_{\text{op}} \mapsto \ldots, \\
C_{\text{idle}} \mapsto \ldots\}
\)
\]

By looking at the environment, we see that there are four environment variables \( x, y, r, d \) that are declared continuous. The sets of jumping and algebraic variables are empty. There are four channels \( \text{appr,ext,raise,lower} \) defined in the environment. The set of recursive definitions defines a number of recursive variables. Complete definitions of recursive variables are very lengthy, so we do not cover them completely here. From the valuation \( \{x \mapsto -1400, y \mapsto -52, r \mapsto 90, d \mapsto 0\} \) we realize that there are no environment variables declared as discrete.

**Action predicates:** Action predicates enable instantaneous changes to variable values. Another way to change the value of a variable discreetly is to store a value received from a channel in it. An action predicate is of the form \( W : r \gg l_a \), where \( W \) is a set of non-jumping variables, \( r \) is a predicate and \( l_a \) denotes the label \( a \) of the action. The predicate \( r \) defines a restriction in terms of current and new values of variables. The current values or values before the execution of action are denoted by \( \text{variable}^- \), a variable name with a ‘-’ superscript and the new values or the values
after the execution of action are denoted by plain variable names. If the restrictions defined in $r$ cannot be met, then the associated action cannot take place and the process term $W : r \gg I_a$ will be deadlocked. The set $W$ consists of any non-jumping variables that are explicitly allowed to jump with the given action. Consider the example of a process with an action predicate as its process term:

$$\langle \{x\} : x^- \geq 100 \land x \leq -1400 \gg \tau, \{x \mapsto 100, \text{time} \mapsto 0\}, (\{x\}, \emptyset, \emptyset, \emptyset, \emptyset) \rangle$$

It consists of a single continuous variable $x$ which is initialized to 100. The part of the predicate about the current value of $x$ acts as a guard. Only if the value of $x$ before the action satisfies the predicate $x^- \geq 100$, which is satisfied in this case, will the action $\tau$ take place and the value of $x$ will be updated to $-1400$.

- **Delay predicates:** The delay behaviors of continuous, dotted continuous and algebraic environment variables can be restricted by constructs called delay predicates. Delay predicates can be differential equations, differential inclusions or algebraic equations etc. Continuous variables are not allowed to jump initially at the start of a delay or anytime during the delay. This is different from HyPA. Just like HyPA and $\phi$-calculus, the set of possible solutions to delay predicates is a parameter of the hybrid Chi theory. The delay predicates model infinite, non-terminating behaviour. There is no disrupt operator in hybrid Chi. In hybrid Chi, we have observed that a delay predicate often appears in alternative composition with other chi constructs that determine when the delay must end. As soon as the environment variables hit the limits set by the predicate or an action or a guard alternatively composed with a delay predicate becomes urgent, the system stops delaying and the undelayable action is performed.

- **Time determinism:** The alternative composition operator of Hybrid Chi is strong time deterministic (like in $\phi$-calculus). An alternative composition of different process terms can delay as long as all terms in the composition can delay and in a way that is common to all composing terms.

**Arguments:** Some arguments given in [18] in favor of a strong time deterministic alternative choice operator are as follows:

- The strong time deterministic choice operator allows non-determinism in variable trajectories during delays. Non-determinism in variable evolutions can be modelled with the help of delay predicates that allow multiple solutions. For example, the delay predicates $\dot{x}^2 = 1$ and $20 \leq x \leq 30$ can evolve in more than one ways.
- Another argument given in [18] in favor of time deterministic choice operator is that hybrid automata has a similar choice mechanism.
Observations:

- Let \( l \) be a continuous variable. Starting from valuation \( \{ l \mapsto 0 \} \), the following process term will deadlock after 5 seconds, as \( (\dot{l} = 1 \land l \leq 5) \) cannot delay more than 5 seconds.

\[
(l = 0) \bowtie ( (\dot{l} = 1 \land l \leq 5) \parallel \dot{l} = 1 )
\]

- The following process term will perform action \( a \) after 5 seconds:

\[
(l = 0) \bowtie ( (\dot{l} = 1 \parallel l = 5 \rightarrow \emptyset : \text{true} \gg a )
\]

- The deadlock process of Hybrid Chi denoted by \( \delta \) is undelayable. Deadlock is not a unit element of alternative composition. i.e,

\[
\delta \parallel p \not\twoheadrightarrow p
\]

Deadlock in alternative composition with a delayable process is a deadlock. i.e., \( \delta \parallel a \Rightarrow \delta \).

Where as if is an undelayable process \( p \) is alternatively composed with \( \delta \), then \( \delta \parallel p \cong p \).

- An expression of the form,

\[
(\dot{l} = 1) \parallel (\dot{l} = 2)
\]

is equivalent to an inconsistent process in hybrid Chi as both the delay predicates cannot be satisfied at the same time.

**Guards:** While explaining action predicates \( W : r \gg \lambda \alpha_a \), we mentioned that part of predicate \( r \) that states a condition on the variable values before action \( a \) has taken place, acts as a guard. This arrangement, (placing a condition about current environment variables ), fulfills the requirement of putting guards before discrete actions only. Hybrid Chi also has a guard operator that can place conditionals before any hybrid Chi process term. A guarded process term is of the form \( b \rightarrow p \), where \( b \) is a predicate on environment variables and the derivatives of the continuous variables. The guards in hybrid Chi are delayable. A false guard can delay according to any delay predicate (provided that the flow requirements of variables of different categories are met), till it becomes true. A guarded process term \( b \rightarrow p \), with a true guard, can do actions or delays as the process term \( p \). Consider the following process:

\[
\langle \dot{x} = 52 \parallel x = 0 \rightarrow \{} : \text{true} \gg \text{pass},
\{x \mapsto -1400, \text{time} \mapsto 0\},
(\{x\}, \emptyset, \emptyset, \emptyset)\rangle
\]
The process term is an alternative composition between a delay predicate and a guard. The guard $x = 0$ is false initially and it can delay in any manner provided that the trajectory of variable $x$ is continuous. The delay predicate must delay according to the differential equation $\dot{x} = 52$. Since the alternatively composed process terms must agree on variable evolution and delay duration, therefore the only way for alternatively composed terms is to delay according to the differential equation $\dot{x} = 52$. The variable $x$ becomes 0 after 26.923 seconds. At time 26.923, the guard is true and action pass takes place immediately.

The semantics given to the guard operator becomes complicated with the provision that a guard that is true at the start of a trajectory, can idle in any manner, if the process term it is guarding is delayable. In such a delay, the guard must remain false during the whole trajectory except possibly at the end points of the delay. In the above example, when $x$ turns 0, the action pass must take place immediately because action predicate is a non-delayable process term. Thus urgent actions are given preference over delaying.

- **Specifying assumptions/ invariants about the environment:** In hybrid Chi, assumptions about the environment can be specified in two ways:
  - by signal emission operator
  - by delay predicates

We briefly discuss them below:

- **Signal Emission:** The signal emission operator indicates assumptions about the environment variables at any stage in a process term. It has been inspired from ACP_{ps} (see [7]). In hybrid Chi, the signal emission operator is denoted by $\triangledown$. For example, $(\text{current} \leq 0) \triangledown P$, is a process term with an initial condition $(\text{current} \leq 0)$. The process term, 
  \[
  \{\}: \text{true} \gg \text{turnon} : (\text{current} \geq 0) \triangledown P
  \]
  represents that after performing action turnon, the value of current must be greater than or equal to zero.

- **Delay Predicates:** Delay predicates act as invariants. They represent conditions about environment variables that must prevail over a certain period of time during the execution of a process. For example, 
  \[x \geq 30 \parallel [(\{} : \text{true} \gg l_a]\]
  The value of the variable $x$ must be greater than or equal to 30 at the start, during the delay and at the end of the delay of the above process term. $[(\{} : \text{true} \gg l_a]$ can delay in any manner.
A delay predicate incorporates properties of a signal emission operator. An action followed by a delay predicate cannot allocate a value to a variable that violates the delay predicate. For example, in \( \{ x \} : \text{true} \Rightarrow l_a \ ; \ x \geq 30 \), the action predicate cannot assign a value to \( x \) that is less than 30. If the only way for a process to proceed, results in a violation of a signal emission predicate or delay predicate following it, then the process cannot proceed and deadlocks. For example, \( \{ x \} : x = 20 \Rightarrow l_a \ ; \ x \geq 30 \) cannot perform action \( a \) and therefore deadlocks.

Hybrid Chi has an inconsistent process, denoted by \( \bot \) that represents a process term in a data state that does not satisfy the process terms' assumptions about the environment variables. Hybrid Chi semantics enforces conditions on transition rules that ensures that an action or delay transition never results in an inconsistent process.

**Communication**: Hybrid Chi has CSP like communication constructs. A set of channels is defined for a hybrid Chi process. Exchange of values between two process terms can take place by means of simultaneous send and receive actions on a channel. \( h!!e_n \) is the send action. \( e_n \) represents a vector of closed variables expressions. The values of \( e_1, \ldots, e_n \) are sent on channel \( h \). \( h??x_n \) denotes the corresponding receive action. \( x_n \) is a variable vector. \( h??x_n \) stores the values received from channel \( h \) in variables \( x_1, \ldots, x_n \). The labels of send, receive and communication actions are as follows.

- \( \text{isa}(h, cs) \)-send action label
- \( \text{ira}(h, cs, W) \)-receive action label
- \( \text{ca}(h, cs) \)-communication action label,

where \( h \) is a channel name, \( cs \) denotes a list of values and \( W \) is a set of variables.

An action encapsulation operator \( \partial_{A_{1\ldots n}} \) can block individual send and receive actions on internal channels of a process. The send and receive process terms are then forced to interact with each other resulting in a communication action.

It's not necessary to always exchange some values in a send or receive action. When no values are exchanged, the send \( h!! \) and receive \( h?? \) actions on a channel serve to achieve synchronization between two process terms. The environment variables behave according to their categories while send and receive actions are performed. i.e. Dotted continuous, algebraic and jumping variables can jump arbitrarily. The behaviour of variables during a send or receive action is not controlled by a predicate on environment variables as in action predicates.

The send and receive actions \( h!!e_n \) and \( h??x_n \) are undelayable. The delayable versions of send, denoted by \( h!e_n \) and receive action, denoted by
&e_n, are usually used in specifications, in order to account for possible delays. A communication on a channel with delayable send and receive actions can delay, even if both send and receive actions on that channel are available. In order to make the communication occur as soon as send and receive actions on a channel become available, the urgent communication operator \( \nu_H \) is defined. \( H \) is a set of channels. With \( \nu_H \), a send and receive action on a channel in \( H \) can delay only if no communication or synchronization can take place on that channel.

- **Any delay operator:** Applying any delay operator to a process term makes it arbitrarily delayable. A process term with any delay operator applied to it is denoted by adding square brackets to the process term. A process term \([p]\) can delay for arbitrarily long durations and the only restrictions it needs to satisfy during delays are the ones enforced by the variable categorization. The action behaviour of process term \( p \) is not changed in \([p]\).

- **Jump enabling operator:** A set of non-jumping environment variables can be declared as jumping in a process term with the help of a jump enabling operator. The process term \( \iota_{J^+}(p) \) behaves equivalent to a process term \( p \) in an environment where variables in set \( J^+ \) are declared as jumping variables. The associated environment of the process term is not modified, instead the behaviour of the process term is changed.

- **Scoping operators:** Scoping operators enable definitions of environment variables, channels and recursive variables whose scope is limited to a process term. Depending upon the desired local declaration, three kinds of scope operator process terms, each corresponds to one type of local definition, are defined. They are:
  1. Variable scope operator process term;
  2. Channel scope operator process term; and
  3. Recursion Scope operator process term

Other process terms in composition with a scope operator process term cannot access any of its local channels or local variables. Unlike \( \phi \)-calculus, the scope of a local variable or channel cannot be widened, as only values not names are exchanged in a hybrid Chi communication. In \( \phi \)-calculus, the increase in scope of a private variable or channel is achieved by passing its name to another process.

The need for local recursive definitions can be understood as follows: Consider a large system specification with a large number of sub-processes. A sub-process may need to do a specific routinely procedure that is not required by other sub-processes. A local recursive definition will confine the scope of a routine only to the sub-process that requires it.
• **Syntactic extensions:** A more user friendly syntax has been introduced for hybrid Chi process terms by means of syntactic extensions. These syntactic extensions do not increase expressiveness of hybrid Chi but improve the readability of hybrid Chi specifications. We find delay operator, modelling scope operator and process instantiation to be particularly useful syntactic extensions.

  - **Delay operators:** $\triangle_d(p)$ represents a process term that first delays for $d$ time units and then behaves as $p$.
    
    The abbreviation $\triangle d$ denotes a process term that first delays for $d$ time units and then terminates. Unlike HyPA and $\phi$-calculus, addition of clock variables to represent passage of time is not necessary. The duration of the delay can be non-deterministic also. For example the action predicate $\{t\} : t \in [\text{min}, \text{max}] \Rightarrow \tau$ defines an interval for a possible value of $t$. $\triangle t$ now indicates an arbitrary time delay within limits of $\text{min}$ and $\text{max}$.

  - **Modelling scope operator:** A modelling scope operator process term combines a variable scope operator, channel scope operator and recursive scope operator.

  - **Process instantiation:** Process instantiation and process definition enable re-use of process terms. It is different from defining a routine through a recursive definition as parameters can be passed in a process instantiation. A process instantiation process term is of the form $l_p(x_k, h_m, e_n)$, where $l_p$ denotes a process label, $x_k$ denote external variables, $h_m$ denote external channels and $e_n$ denote expressions. The process instantiation accompanies a process definition with the same name. The only free variables and channels allowed in a process definition are its formal parameters.

    Process instantiations and scoping operator process terms help in modular design of a large system. With their help, data information can be limited to a scope. This will reduce risks of undesirable side effects.

Syntactic extensions for a silent action, multiple variable assignments, process terms with repetitive behaviour, delayable send and receive and jump enabling operator have also been defined in hybrid Chi.

• **Semantics:** The hybrid Chi theory has been given a structural operational semantics that describes the behaviour of a hybrid Chi process. (We will use the word process or chi process to mean a hybrid Chi process $(p, \sigma, E)$). The operational semantics associates a hybrid transition system with every process. The equivalence defined on processes is a bisimulation relation on the underlying transition systems.

Four kinds of transition relations are described:

1. **Action transition**
2. Termination transition
3. Time transition
4. Consistency Predicates

Some notations to facilitate in defining the transition relations are given here. A set $V$ of variables and $H$ of channel names is given. $V$ includes the special variable $\text{time}$. A valuation is a partial function from a set of variables to a set of values. The set of values is denoted by $\Lambda$. It contains at least the booleans and the reals. The set of all valuations $\Sigma$ is then defined as $\Sigma : V \mapsto \Lambda$. The variable valuation of a hybrid Chi process $\sigma$ is in set $\Sigma$. Let $\dot{\Sigma}$ denote the set of dotted variables. The set of extended valuations $\dot{\Sigma}$ is defined as $\dot{\Sigma} : (V \cup \dot{V}) \mapsto \Lambda$. The set $T = \mathbb{R}_{\geq 0}$ is used to denote points in time. The set of environments is denoted by $\varepsilon$. A set $A$ of actions is given, where $A = A_{\text{label}} \cup A_{\text{comm}}$. The set $A_{\text{comm}}$ contains the labels of internal send, internal receive and communication actions. $A_{\text{comm}} = \{\text{isa}(h, cs), \text{ira}(h, cs, W), \text{ca}(h, cs)\}$, where $h$ is a channel, $cs$ is the list of values communicated and $W$ is a set of variables in which the received values are stored by the receiving process. $A_{\text{label}}$ consists of action labels other than that of communication actions. The silent action $\tau$ is in $A_{\text{label}}$.

1. The discrete behaviour is represented by action and termination transitions:
   - An action transition is a ternary relation,
     $$ \rightarrow \subseteq (P \times \Sigma \times \varepsilon) \times (\dot{\Sigma} \times A \times \dot{\Sigma}) \times (P \times \Sigma \times \varepsilon). $$
     A transition $\langle p, \sigma, E \rangle \xrightarrow{\xi, a, \xi'} \langle p', \sigma', E \rangle$ represents that a process $\langle p, \sigma, E \rangle$ in extended valuation $\xi$ performs an action $a$ and becomes process $\langle p', \sigma', E \rangle$. The extended valuation after action $a$ is $\xi'$.
   - A termination transition is a ternary relation
     $$ \rightarrow \subseteq (P \times \Sigma \times \varepsilon) \times (\dot{\Sigma} \times A \times \dot{\Sigma}) \times (\{\sqrt{\} \} \times \Sigma \times \varepsilon). $$
     A transition $\langle p, \sigma, E \rangle \xrightarrow{\xi, a, \xi'} \langle \sqrt{\}, \sigma', E \rangle$ represents that a process $\langle p, \sigma, E \rangle$ in extended valuation $\xi$ performs an action $a$ and terminates with valuation $\sigma'$. The extended valuation after action $a$ is $\xi'$.

In the above transitions, $\xi$ is $\sigma$ extended with a valuation for algebraic and dotted variables. Similarly $\xi'$ is $\sigma'$ extended with a valuation for algebraic and dotted variables.

2. A delay transition is a ternary relation,
   $$ \rightarrow \subseteq (P \times \Sigma \times \varepsilon) \times (T \times \rightarrow (\dot{\Sigma})) \times (P \times \Sigma \times \varepsilon). $$
A transition \((p, \sigma, E) \xrightarrow{\rho(s)} (p', \sigma', E)\) represents that at all times \(s \in [0,t]\) during the delay, the extended valuation is given by \(\rho(s)\). At the end of the delay the process becomes \((p', \sigma', E)\).

3. \(\xi \vdash (P \times \sigma \times \varepsilon) \times \dot{\Sigma} \). A consistency predicate \((p, \sigma, E) \xrightarrow{\xi}\), indicates that the process term \(p\) is consistent with the extended valuation \(\xi\).

The presence of extended valuations in transition relations needs more explanation. A process term \(P\) can contain a predicate containing a dotted or algebraic variables (e.g. in an action predicate, guard or signal emission) which needs to be evaluated in an extended valuation. In an action/termination transition, there can be more than one pair of initial and final extended valuations possible for a given process.

For example consider the following hybrid Chi process

\[
\langle \emptyset : z > 0 \land z = 10 \Rightarrow l_a, \{\text{time} \mapsto 0\}, (\emptyset, \emptyset, \{z\}, \emptyset, \emptyset) \rangle
\]

The environment \((\emptyset, \emptyset, \{z\}, \emptyset, \emptyset)\) consists of only one variable which is declared to be algebraic. Note that in the action predicate the set of non-jumping variables allowed to jump is empty. This is because an algebraic is by default jumping. Consider a termination transition of this process.

\[
\langle \emptyset : z > 0 \land z = 10 \Rightarrow l_a, \{\text{time} \mapsto 0\}, (\emptyset, \emptyset, \{z\}, \emptyset, \emptyset) \rangle \xrightarrow{\xi \mapsto \xi'} \langle \sqrt{\cdot}, \{\text{time} \mapsto 0\}, (\emptyset, \emptyset, \{z\}, \emptyset, \emptyset) \rangle
\]

The possible pairs of extended valuation \((\xi, \xi')\) are:

\[
\{(\xi, \xi') \mid \xi(z) > 0 \land \xi'(z) = 10 \land \xi(\text{time}) = \xi'(\text{time}) = 0\}
\]

In a time transition, \((p, \sigma, E) \xrightarrow{\rho} (p', \sigma', E)\), the trajectory \(\rho\) must satisfy restrictions due to categorization of environment variables. If \(p\) is a delay predicate or a guard and imposes further restrictions on variables, then \(\rho\) must satisfy them also. Consider the following delay transition:

\[
\langle \dot{x} = 52 \mid x = 0 \rightarrow \{\} : \text{true} >> \text{pass}, \{x \mapsto -1400, \text{time} \mapsto 0\}, (\{x\}, \emptyset, \emptyset, \emptyset) \rangle \xrightarrow{26.923} \langle \{x \mapsto -1400 + 52 \times \text{time}\} \rangle
\]

\[
\langle \dot{x} = 52 \mid x = 0 \rightarrow \{\} : \text{true} >> \text{pass}, \{x \mapsto 0, \text{time} \mapsto 26.923\}, (\{x\}, \emptyset, \emptyset, \emptyset) \rangle
\]

The evolution \(\{26.923 \mapsto \{x = -1400 + 52 \times \text{time}\}\}\) satisfies the differential equation \(\dot{x} = 52\).

Thus extended valuations provide a formal framework to reason about the instantaneous changes or evolutions of continuous, discrete, algebraic or dotted variables during a transition.
The consistency predicate indicates that certain valuations extended valuations are not desirable for a hybrid Chi process. The consistency predicate enforces restrictions like that of invariants and flows (combined) in a hybrid automaton. A process \( \langle p, \sigma, E \rangle \) is consistent with an extended valuation \( \xi \), (where \( \xi \) is \( \sigma \) extended with \( \{ \) a valuation for \( \} \) algebraic and dotted continuous variables as described in \( E \),) if the delay predicates \( u \) in \( p \) and the predicates of signal emission hold when evaluated in extended valuation \( \xi \).

The semantic rules of hybrid Chi provide proper restrictions to make sure that only consistent processes are allowed to perform an action or delay. Also the result of an action or delay transition is always a consistent process.

The following properties of the semantics can be observed:

1. During a transition the environment of a process does not change.
2. For all transitions the domain of \( \sigma \) equals the domain of \( \sigma' \).
3. For all transitions \( \langle p, \sigma, E \rangle \xrightarrow{\xi \cup \xi'} \langle p', \sigma', E \rangle \), the extended valuation \( \xi \) extends \( \sigma \), i.e. the extended valuation \( \xi \) restricted to domain of \( \sigma \), equals \( \sigma \). Similarly, the extended valuation \( \xi' \) extends \( \sigma' \).
4. For all time transitions, \( \langle p, \sigma, E \rangle \xrightarrow{t, \rho} \langle p', \sigma', E \rangle \), \( \text{dom}(\rho) = [0, t] \), and for all variables \( x \in \text{dom}(\sigma) \),
   \[
   \sigma(x) = \rho(0)(x), \text{ and } \\
   \sigma'(x) = \rho(t)(x).
   \]
5. For all consistency predicates \( \langle p, \sigma, E \rangle \xrightarrow{\xi} \), the extended valuation \( \xi \) extends \( \sigma \).

- **Bisimulations:** In hybrid Chi, state-less bisimilarity is chosen for determining equivalence of process terms. Two closed process terms are state-less bisimilar written as \( p \equiv q \), if and only if there is a state-less bisimulation relation \( R \), such that \( (p, q) \in R \).

A set of axioms to represent equivalence between process terms is also given. These axioms are sound with respect to state-less bisimilarity. The set of axioms is not comprehensive enough to allow algebraic manipulation as in HyPA or \( ACP^{srt}_{hs} \)

### 3.4 \( ACP^{srt}_{hs} \)

\( ACP^{srt}_{hs} \) is a process algebra for specification and analysis of hybrid processes. It is an extension of \( ACP \) with standard relative timing (denoted by \( ACP^{srt} \), see [8]) and \( ACP \) with propositional signals (denoted by \( ACP_{ps} \), see [7]). It has special constructs for modelling the passage of time and time-outs. In many ways HyPA and \( ACP^{srt}_{hs} \) are similar. Both are extensions of \( ACP \). Both have
a large set of axioms and two kinds of bisimulations. Both have operators for discrete and continuous manipulation of environment variables. ACP\textsuperscript{hs} incorporates more ideas and is more expressive than HyPA. In this section we try to summarize what ACP\textsuperscript{hs} has to offer.

The set of ACP\textsuperscript{hs} process terms can be described by the BNF expression as given below:

\[
P ::= \tilde{a} \quad \text{discrete, undelayable action} \\
\text{\hspace{1em}}| \quad \delta \quad \text{Deadlock} \\
\text{\hspace{1em}}| \quad \perp \quad \text{Inconsistent process} \\
\text{\hspace{1em}}| \quad \nu_{rel}(P) \quad \text{relative timeout} \\
\text{\hspace{1em}}| \quad \sigma_{rel}(P) \quad \text{relative delay} \quad r \geq 0 \\
\text{\hspace{1em}}| \quad \psi \cdot P \quad \text{signal emission} \quad \psi \quad \text{a state proposition} \\
\text{\hspace{1em}}| \quad \psi :\rightarrow P \quad \text{conditional} \quad \psi \quad \text{a state proposition} \\
\text{\hspace{1em}}| \quad \phi \cdot^\nu \ P \quad \text{signal evolution} \quad \phi \quad \text{a state proposition} \\
\text{\hspace{1em}}| \quad \chi \cdot^\nu \ P \quad \text{signal transition} \quad \chi \quad \text{a transition proposition} \\
\text{\hspace{1em}}| \quad P \cdot P \quad \text{sequential composition} \\
\text{\hspace{1em}}| \quad P + P \quad \text{alternative composition} \\
\text{\hspace{1em}}| \quad P \parallel P \quad \text{parallel composition} \\
\text{\hspace{1em}}| \quad P \mid P \quad \text{forced communication} \\
\text{\hspace{1em}}| \quad P \lpar P \quad \text{left parallel operator} \\
\text{\hspace{1em}}| \quad \partial_H(P) \quad \text{encapsulation operator} \quad H \subseteq A
\]

- **Data**: The data part of an ACP\textsuperscript{hs} process is simply a valuation of environment variables. The environment variables are not categorized as continuous, discontinuous, jumping or non-jumping as in hybrid Chi. During actions, by default all variables are jumping and can jump to arbitrary values.

- **Delay and Time out operators**: In ACP\textsuperscript{hs}, a relative delay between two process terms is described by means of a delay operator. By relative delay we mean that time is not measured from the start of the whole process, but from the termination of the preceding process term in sequential composition. The delay operator (denoted by $\sigma_{rel}^r$) has a parameter $r$ that indicates the duration of the delay. The duration is a non-negative real. For example, $\sigma_{rel}^2(\tilde{a}) \cdot \sigma_{rel}^5(p)$ is a process term that delays for two time units at the start of the process, performs action $\tilde{a}$ and then delays for 5 time units before proceeding as term $p$.

Corresponding to the relative delay operator $\sigma_{rel}^r$, hybrid Chi has syntactic extension $\Delta r$.

The relative delay and relative time out operators $\nu_{rel}$ are operators of ACP with standard relative timing (ACP\textsuperscript{sr}). Also the concept of immediate actions and immediate deadlock, represented by an action name or $\delta$ with a $^\infty$ superscript, have been taken from ACP\textsuperscript{sr}. $\tilde{a}$ must immediately
perform an action without a delay. Note that the actions of HyPA and
the actions of hybrid Chi are also undelayable.

A relative time out operator forces a process term to immediately per-
form an action at its start. If the only starting option available for a
process term is to delay then the process term deadlocks. For example,
ν_{rel}(σ_{rel}(p)) = \delta, with r > 0.

• Signal Transition operator: Discrete changes in the environment vari-
ables are modelled by means of a signal transition operator (denoted by
\textit{\textasciitilde}). The signal transition operator is very close to the renitializations
of HyPA and action predicates in Hybrid Chi. It takes a “jump predicate” as
an argument. The jump predicate of a signal transition operator is known
as a transition proposition. The values of variables before a transition are
represented by \textit{variable} and the values of variable after a transition are
represented by \textit{variable}. For example, \((x = 5) \textit{\textasciitilde} a\) represents an imme-
diate action \(a\) with no change in variable \(x\). Remember that all variables
by default are jumping. The transition proposition \(x = 5\) restricts \(x\) to
keep its value constant.

A variable update by means of a signal transition takes effect when the
term its applied to is undelayable. When applied to a delayable process, a
signal transition operator just acts as a guard on current variable values.
For example, in \((x \geq 50 \land x = 0) \textit{\textasciitilde} 5\) \(\sigma\), if value of \(x\) is
\(\geq 50\), then it delays for 3 time units. The variable update of \(x\) to 0 does not take place.

• Signal Evolution Operator: A signal evolution operator (denoted by
\textit{\textasciitilde}_v) is used to describe the evolution of variables during delays. During
delays, the trajectories of environment variables are piece-wise continu-
ously differentiable real valued functions of time. That is in a delay, a
variable can jump finitely many times. In HyPA, variables were allowed
to jump initially at the start of a delay. Whereas in \(\phi\)-calculus and hybrid
Chi (continuous) variables were not allowed to jump at all.

An evolution operator takes a predicate as an argument that further re-
stricts the behaviour of variables. The delay predicate in \(ACP_{hs}^{rel}\) is called
an evolution proposition. An evolution proposition can be a differential
equation, differential inclusion, algebraic equation, algebraic inequality
or a boolean expression, etc. The evolution operator can also declare
a set of variables to be smooth. A variable that is declared smooth by
the evolution operator must evolve according to a continuously differen-
tiable function of time during the delay. That is, a smooth variable and
its derivative are not allowed to jump during a delay. For example, in
\((x \leq 50 \land 20 \leq v \leq 100) \textit{\textasciitilde}_v \sigma_{rel}(p)\), the value of variable \(v\) must vary
continuously and stay within the limits of 20 and 100. Whereas the vari-
able \(x\) must remain below 50. The variable \(x\) and its derivative can jump
finitely many times during the delay.

A signal evolution operator takes effect when applied on a delayable pro-
cess term. When used with an undelayable process, a signal evolution
proposition is reduced to a signal emission proposition and it only needs to be satisfied at the start of the process. For example,

\[(\text{level} \leq 100) \overset{\text{level}}{\Rightarrow} (\text{refill}) \equiv (\text{level} \leq 100) \overset{\text{refill}}{\Rightarrow}\]

Here \(\text{refill}\) is an undelayable action.

In \(ACP_{hs}\), the concept of state and transition propositions and their solutions is formally defined. The definition does not restrict the type of systems that can be specified in \(ACP_{hs}\).

- **Guards:** A conditional operator is used to represent guarded actions or delays. For example,

\[
P = \sigma^5_{\text{rel}}((\text{temp} > 22) \rightarrow (\text{Process}^{\text{toff}}) + (\text{temp} \leq 19) \rightarrow (\text{Process}^{\text{ton}})) \cdot P
\]

The process \(P\) is a simple recursive process that every 5 seconds checks the temperature and follows the appropriate course.

- **Representing assumptions/invariants about the environment \(ACP_{hs}\):** like hybrid Chi, also has a signal emission operator from \(ACP_{ps}\) that can specify the assumptions about the environment at any step of the process term. Consider the following process term,

\[
(valvepos = 90^\circ \land level = 0) \overset{\text{openvalve}}{\Rightarrow} \sigma^5_{\text{rel}}((valvepos < 90^\circ \land level > 0) \overset{\text{P}}{\Rightarrow})
\]

At the start of the above process term, the valve is at 90° and the water level is zero. Five seconds after opening the valve, the valvepos is less than 90° and water level is greater than zero.

The evolution proposition of \(ACP_{hs}\), like delay predicates of hybrid Chi, represent system invariants. Consider the process term,

\[
(v < 100) \overset{v}_{\text{P}} \sigma^{10}_{\text{rel}}(P).
\]

The variable \(v\) has to remain below 100 during the complete duration of the delay. The duration of the delay is specified 10 time units. If during the delay, a situation arises in which \(v\) cannot both vary continuously and remain below 100, then the whole process will deadlock before the invariant is violated. The semantics of \(ACP_{hs}\) ensures that assumptions about the environment described by signal emissions or evolution propositions always hold. In the following process term,

\[
(x^* = 30) \overset{\text{a}}{\Rightarrow} (x > 30) \overset{x}_{\text{P}} \sigma^1_{\text{rel}}(P)
\]

action \(\text{a}\) cannot take place, as the variable update accompanying it violates the assumption \(x > 30\) following the action. Therefore the above process term is equivalent to a deadlock.
$ACP_{hs}$ also has an inconsistent process. A process term in a data state contradicting the process terms’ assumptions about the environment is equivalent to an inconsistent process.

**Integration:** $ACP_{hs}$ is extended with integration over time intervals. The intuition behind integration is to be able to model processes that have the capability of performing an action at any point in an interval. The addition of integration enables us to model non-determinism in the duration of the delay. Integration can be viewed as an alternative composition with a set of alternatives that may be uncountable. The notation $\sigma_{rel}^*$ is an abbreviation for integration over the interval $[0, \infty)$, i.e. $\sigma_{rel}^*$ represents an indefinite delay.

**Semantics:** $ACP_{hs}$ has an operational semantics that associates a hybrid transition system with a process term and a valuation of environment variables. Five types of transition relations are defined:

- action step
- time step
- termination step
- signal relations
- discontinuity relations

Some notations needed to describe different kinds of transition relations are as follows:

Let $V$ stand for a set of environment variables and $A$ be a set of actions. $\dot{V} = \{ \dot{v} \mid v \in V \}$ denotes the set of derivatives of all variables in $V$.

A *state* is a valuation of variables and their derivatives. During delays a piece-wise continuously differentiable function gives the trajectory of the variables on a non-zero time interval. We call this function a *state evolution*. $\epsilon_r$ stands for the set of all possible state evolutions on an interval of length $r$. i.e.,

$$\epsilon_r = [0, r] \to (V \to \mathbb{R}),$$

where $r$ is a positive real number. During a delay with a state evolution $\rho$, $\alpha^\rho_t$ gives the valuation of variables at any time $t$. The derivatives at any moment $t$ are given by the derivative of $\rho$ at $t$.

The set of all $ACP_{hs}$ process terms is denoted by $P$.

- An action step is a binary relation between two process terms defined for each action $a$ and each pair of states $\alpha, \alpha'$.

$$\langle \cdot, \alpha \rangle \xrightarrow{a} \langle \cdot, \alpha' \rangle \subseteq (P \times P)$$

$$\langle t, \alpha \rangle \xrightarrow{a} \langle t', \alpha' \rangle$$ indicates that process $t$ can perform action $a$ in state $\alpha$, and then continue as process $t'$ in state $\alpha'$.  

39
- A termination step is a unary relation defined on a process term for each action $a$ and each pair of states $\alpha, \alpha'$.

$$\langle \cdot, \alpha \rangle \xrightarrow{a} \langle \sqrt{\cdot}, \alpha' \rangle \subseteq P$$

$$\langle t, \alpha \rangle \xrightarrow{a} \langle \sqrt{\cdot}, \alpha' \rangle$$ indicates that in state $\alpha$, process $t$ can perform action $a$ and then terminate in state $\alpha'$.

- A time step is a binary relation between two process terms. It is defined for each duration $r \in \mathbb{R}^+$, each state evolution on duration $r$, i.e. $p \in \epsilon_r$ and for each pair of states $\alpha, \alpha'$, such that $\alpha_0^r = \alpha$ and $\alpha_r^r = \alpha'$.

$$\langle \cdot, \alpha \rangle \xrightarrow{r, p} \langle \cdot, \alpha' \rangle \subseteq (P \times P)$$

$$\langle t, \alpha \rangle \xrightarrow{r, p} \langle t', \alpha' \rangle$$ indicates that in state $\alpha$, process $t$ can idle for $r$ time units while the state evolves according to $p$, and then continue as process $t'$ in state $\alpha'$.

- A signal relation is defined on process terms for each state $\alpha$.

$$\alpha \in [s(\cdot)] \subseteq P.$$ 

$\alpha \in [s(t)]$ indicates that in state $\alpha$, the signal emitted by process $t$ holds.

The signal of a process term $t$ is a proposition that states the assumptions of the process term $t$ about the environment variables.

The signal relations in the semantics of $ACP_{hs}$ are just like the consistency relations of hybrid Chi. The signal of a process term $t$ holds in a state $\alpha$, when the state proposition of the evolution or signal emission operator evaluates to true in state $\alpha$. Consider the following examples,

* The signals of $\tilde{a}, \tilde{\delta}, \sigma_{en}(t)$, where $r$ is strictly greater than zero, are true in all states as these process terms make no assumptions about the environment.

* The signal of process $\sigma^{0}_{rel}(x)$ is the same as signal of process term $x$.

* The signal of $(\text{pressure} = 0)^{\bullet} (\text{level} \leq 30)^{\bullet} P$ is true in a state that satisfies both the proposition, $\text{pressure} = 0 \land \text{level} \leq 30$ and the signal of $P$.

- A discontinuity relation is defined on a process term for each pair of states $\alpha, \alpha'$.

$$\alpha \rightarrow \alpha' \in [d(\cdot)] \subseteq P.$$ 

$$\alpha \rightarrow \alpha' \in [d(t)]$$ indicates that discontinuities resulting from a transition from state $\alpha$ to $\alpha'$ are allowed for process $t$.

In $ACP_{hs}$, the continuity requirements of a delayable process in parallel composition with another must be protected, i.e., a process
in parallel composition with a delayable process cannot make a transition from one state to another that modifies a variable declared as continuous by the first. In order to find out whether a transition respects the continuity requirements of a process, discontinuity relations are defined. Consider the following examples,

- The discontinuities caused by an action transition from state \( \{ x \mapsto 20, \dot{x} \mapsto 1 \} \) to state \( \{ x \mapsto 25, \dot{x} \mapsto 1 \} \) are not allowed by the process term
  \[ (x \leq 30) \; \sigma_{\{x\}} \; \sigma_{\text{rel}}^5(P), \]
  as variable \( x \) is declared continuous by it.
- Where as the discontinuities caused by the same states, i.e. from \( \{ x \mapsto 20, \dot{x} \mapsto 1 \} \) to state \( \{ x \mapsto 25, \dot{x} \mapsto 1 \} \), are allowed by the process term
  \[ (x \leq 30) \; \sigma_{\emptyset} \; \sigma_{\text{rel}}^5(P). \]
  Because variable \( x \) has not been declared as continuous.
- A discontinuity resulting from a similar jump, (as in the previous two steps), is allowed for a process term
  \[ (x \leq 30) \; \sigma \; \sigma_{\text{rel}}^7(P), \]
  in which the signal evolution operator has been replaced by a signal emission.

- **Weak Time determinism:** \( ACP_{\text{hs}}^{\text{srt}} \) takes a weak time deterministic approach in alternative composition. The alternative composition of two process terms can also delay even when one of the processes cannot delay and the other is delayable.

**Arguments:** Some arguments in favor of a weak time deterministic operator given in [4] are as follows:

- The semantics of \( ACP_{\text{hs}}^{\text{srt}} \) provides for a time determinism in which a choice between different idling process terms is delayed, but a choice between different variable evolutions is not postponed. Different variable evolutions may make choices among process terms, but these choices need not be visible while idling, i.e. a non-deterministic operator is not necessary to enforce choices made due to different variable evolutions during a delay.

  For example,
  \[ (x = 0) \; \cdot \; (0 \leq \dot{x} \leq 3) \; \sigma_{\{x\}} \; \sigma_{\text{rel}}^3((x = 9) : \rightarrow \tilde{a} + (x < 9) : \rightarrow \tilde{b}) \]
  The action performed after 3 time units depends upon the evolution of \( x \) followed. Note that the above process term is syntactically equal to the following process term according to the axioms \( SRT3 \) and \( HSE7 \) of \( ACP_{\text{hs}}^{\text{srt}} \).
  \[ (x = 0) \; \cdot \; (0 \leq \dot{x} \leq 3) \; \sigma_{\{x\}} \; \sigma_{\text{rel}}^3((x = 9) : \rightarrow \tilde{a}) \]
  \[ + (0 \leq \dot{x} \leq 3) \; \sigma_{\{x\}} \; \sigma_{\text{rel}}^3((x < 9) : \rightarrow \tilde{b}) \]
Observations:

- In $ACP^{ert}_{hs}$, an expression of the form 
  
  $$(\dot{l} = 1) \cdot (x) + (\dot{l} = 2) \cdot (y)$$

  is equivalent to an inconsistent process like in hybrid Chi. The signals of the composed process terms must both be true at the same time.

- In a delay of $r$ time units, if all process terms are delayable for $r$ time units, then the variables must evolve with a trajectory that is possible for all process terms. For example,

  $$(\dot{x} = 1) \cdot (x) \sigma^5_{rel}(\tilde{a}) + (\dot{x} \geq 0) \cdot (x) \sigma^0_{rel}(\tilde{a}), \{x \rightarrow 0\}$$

  the above process can delay for 5 time units as follows:

  $$(\dot{x} = 1) \cdot (x) \sigma^5_{rel}(\tilde{a}) + (\dot{x} \geq 0) \cdot (x) \sigma^0_{rel}(\tilde{b}), \{x \rightarrow 0\}$$

  The alternative composition can also evolve for 6 time units in the following way:

  $$(\dot{x} = 1) \cdot (x) \sigma^6_{rel}(\tilde{a}) + (\dot{x} \geq 0) \cdot (x) \sigma^0_{rel}(\tilde{b}), \{x \rightarrow 0\}$$

  where $t$ represents time since the start of delay. The evolution of variables throughout the delay is $\dot{x} = t + 1$. The expression $\dot{x} = t + 1$ agrees with $\dot{x} = 1$ at time $t = 0$. As time progresses, the expressions $\dot{x} = 1$ and $\dot{x} = t + 1$ diverge.

  Therefore the time transition,

  $$(\dot{x} = 1) \cdot (x) \sigma^5_{rel}(\tilde{a}) + (\dot{x} \geq 0) \cdot (x) \sigma^0_{rel}(\tilde{b}), \{x \rightarrow 0\}$$

  that is allowed by the semantics of $ACP^{ert}_{hs}$, cannot be broken into smaller durations.

  A property of process algebras with timing, known as time-interpolation of processes, states that a time transition can always be broken into smaller time transitions.

  Mathematically, it can be expressed as follows,

  $$(s \rightarrow s') \Rightarrow \exists \langle s'', m_1, m_2 \rangle, \text{ such that } m = m_1 + m_2 \text{ and } s \rightarrow s'' \text{ and } s'' \rightarrow s'$$

  In the semantics of $ACP^{ert}_{hs}$, this property does not hold.
In an alternative composition, where one process term can delay and the other cannot, no preference is given to actions in choice resolution.

The deadlock process in $ACP_{hs}^{srt}$ is undelayable deadlock denoted by $\tilde{\delta}$. $\tilde{\delta}$ is the unit of alternative composition in $ACP_{hs}^{srt}$. The following axiom holds in $ACP_{hs}^{srt}$:

$$\tilde{\delta} + p = p$$

**Bisimulations:** Two kinds of bisimulation relations on process terms are defined.

1. One is a state-based bisimulation simply called *bisimulation*
2. The other is a state-less bisimulation called *interference-compatible bisimulation* (abbreviated as *ic-bisimulation*).

Ic-bisimulation (denoted by $\leftrightarrow$), is comparable to the robust bisimilarity of HyPA and the state-less bisimilarity of Hybrid Chi.

Bisimulation, denoted by $\equiv$, compares the behaviour of two process terms only for a given variable valuation. If two process terms $p, q$, are bisimilar for all states, i.e. $(p, \alpha) \equiv (q, \alpha)$, for all states $\alpha$, then they are called bisimulation equivalent. In this way, bisimulation equivalence, is comparable to initially state-less bisimilarity. Like in HyPA, the bisimilarity is not preserved by parallel operator, whereas ic-bisimulation is preserved by it.

$ACP_{hs}^{srt}$ is provided with a large set of axioms to perform algebraic reasoning on process terms and simplify them. Some lifting rules that help to incorporate the results of real analysis on environment variables in process terms, are also given. All process terms of $ACP_{hs}^{srt}$ can be reduced to a basic form without the parallel operator. Like HyPA, $ACP_{hs}^{srt}$ has some axioms and lifting rules that can be used only in the absence of parallel operator.

4 About Discontinuities

In a hybrid system the evolution of environment variables is not always continuous. We can study discontinuities in following different scenarios:

- Discontinuities during actions
- Discontinuities during delays
- Discontinuities as one delay predicate is taken over by another (mode switching)
- Discontinuities when process terms are composed in parallel
Discontinuities accompanying an action are incorporated in a system model by action predicates (hybrid Chi), reinitializations (HyPA), assignments or resets (phi-calculus) and signal transition propositions ($ACP_{hs}$). What about discontinuities during a delay or when one delay predicate or evolution proposition takes over another. In $ACP_{hs}$, solutions to evolution propositions are piece-wise continuously differentiable functions of time. The variables that are specifically declared as smooth by evolution operator cannot jump during the delay. In $ACP_{hs}$, one evolution proposition cannot be taken over by another evolution proposition (that offers solutions different from the last one) without executing an action in between. The actions in $ACP_{hs}$ allow arbitrary jumps of variables, therefore the problem of setting up initial conditions for the new evolution proposition does not occur. In HyPA, the solution to flow predicates is a parameter to the theory. Most practical cases require some continuity requirements on variables to be fulfilled. Therefore most solutions are continuous or continuously differentiable functions of time. In HyPA, one flow clause can be taken over by another clause through a disrupt operator. At the start of a new flow clause variables are allowed to jump arbitrarily unless specifically mentioned to remain continuous. In hybrid Chi, a delay predicate $u$ represents never terminating infinite behaviour. There is no disrupt operator in hybrid Chi. Usually a delay predicate appears in alternative composition with other process terms and the whole system ends delaying as soon as the environment variables reach the limits enforced by the delaying predicate or an action of a process term in alternative to $u$ becomes urgent. Thus, for mode switching in hybrid Chi, an action is required in between two delay predicates. The required variable discontinuities can then be taken care of in the action predicate.

In $\phi$-calculus, in mode switching the environment of a process needs to be updated by a reset action. The reset action removes the flow clauses of a variable (whose evolution needs to be switched) from the environment and replaces them by other flow clauses. The values of the variables do not jump during such a reset. If an initialization is required as new flow clauses take over, the variable has to be assigned a value. Thus mode switching and discontinuities required with it, are accomplished in two subsequent environmental actions.

In parallel composition between two or more processes that share environment variables, a situation can arise when a process tries to update a variable that is declared as continuous by another process. A feature that is present in $ACP_{hs}$ and is not present in other process algebras is to disallow modifications or discontinuities to a variable declared as continuous by a delayable process in parallel composition with another process. This is done by means of a continuity operator and discontinuity relations. A discontinuity operator when applied on a process term yields the transitions from which only those discontinuities result that are allowed by the process term. In $ACP_{hs}$, a variable is declared continuous through the evolution operator. This continuity restriction holds only if the process is delayable. The semantic rules of parallelism ensure that a process in parallel composition with a delayable process can only perform action transitions that do not modify a continuous variable. This is done with the help of discontinuity relations.
Consider a process with an environment variable $x$. The process initializes the variable $x$ to zero and then lets it continuously evolve with rate one forever. We represent this process in different process algebras:

**Hybrid Chi:**
\[ P = (x = 0) \land (\dot{x} = 1) \]

**HyPA:**
\[ P = [x | x^+ = 0] \triangleright (x | \dot{x} = 1). \]

**$\phi$-Calculus:**
\[ P = [x := 0] \cdot [\text{reset } x \text{ with } \{x | \dot{x} = 1\}] \cdot 0. \]

**$ACP_{hs}$:**
\[ P = (x = 0) \land (\dot{x} = 1) \land \sigma^*_\text{rel}(\tilde{\delta}) \]

Let’s study its behaviour in parallel composition with another process that tries to reset variable $x$ every four seconds. The process can be defined through recursion as follows:

**Hybrid Chi:**
\[ R = \Delta 4 ; \{x\} : x = 0 \triangleright \tau ; R. \]

**HyPA:**
\[ R = [t | t^+ = 0] \triangleright (t | \tilde{t} = 1) \triangleright [t | t^+ = 0] \triangleright \tau ; R. \]

**$\phi$-Calculus:**
\[ R = \nu t[t := 0] \cdot [\text{reset } t \text{ with } \{t | \tilde{t} = 1\}] \cdot R'. \]

**$ACP_{hs}$:**
\[ R = \sigma^4_{\text{rel}}((x^\bullet = 0) \cdot a \cdot R). \]

Note that in $ACP_{hs}$, we don’t need to model passage of time through a clock variable. We have an operator $\sigma_{\text{rel}}$. In hybrid Chi, the passage of time is modelled by means of a syntactic extension $\Delta t$, where $t$ is any non-negative real value. In $ACP_{hs}$, $P \parallel R$ will delay for four time units and evolve with trajectory $\dot{x} = 1$.

\[ \langle (x = 0) \land (\dot{x} = 1) \land \sigma^*_\text{rel}(\tilde{\delta}) \land \sigma^4_{\text{rel}}((x^\bullet = 0) \cdot a \cdot R), x \mapsto 0, 4 \rangle \]

Here $x \mapsto 0$ indicates the initial valuation and $x \mapsto 4$ indicates the final valuation.

At this point the process at the right hand side cannot wait any further. It must perform action $\tilde{a}$ immediately. The action $\tilde{a}$ is accompanied with a transition proposition that modifies the value of variable $x$. The process at the left hand side can still delay. Any action performed by process on the right must satisfy the continuity constraints of the process on the left. The continuity requirements, determined through discontinuity relations, of the process on the left hand side, state that the value of $x$ and its derivative cannot be modified. Therefore process $R$ cannot perform action $a$. After four time units $R$ cannot wait. Therefore the parallel composition cannot wait and the process $P \parallel R$ deadlocks after four time units.

Now we consider the evolution of $P \parallel R$ in hybrid Chi. A hybrid Chi process consists of a process term, a variable valuation and an environment. The process must start with a value of 0 for $x$. We want the trajectory of $x$ to be absolutely continuous. Therefore it is declared as a continuous non-jumping variable. The environment $E$ is ($\{x\}, \emptyset, \emptyset, \emptyset$). (Remember that an environment in hybrid
Chi is a tuple \((C, J, L, H, R)\). See section 3.3.)

\[
\langle P \parallel R, \{x \mapsto 0, \text{time} \mapsto 0\}, E \rangle^4_{\rho} \quad \text{(} \rho \text{ indicates that } \dot{x} = 1 \text{.)}
\]

\[
\sum \langle \dot{x} = 1 \parallel \{x\} : x = 0 \gg \tau ; R, \{x \mapsto 4, \text{time} \mapsto 4\}, E \rangle
\]

\[
\tau \rightarrow \langle \dot{x} = 1 \parallel R, \{x \mapsto 0, \text{time} \mapsto 4\}, E \rangle
\]

(\text{the derivation repeats itself from step (2)})

Thus after every four seconds \(x\) is initialized and during delays the process evolves with trajectory \(\dot{x} = 1\).

\(P \parallel R\) will behave the same (as it behaves in hybrid Chi) in HyPA and \(\phi\)-calculus. In \(\phi\)-calculus, we can declare \(x\) to be a local variable of process \(P\). Then \(x\) referred to in \(R\) is not the same \(x\) as in \(P\). In fact \(x\) from process \(P\), will be given a fresh name with respect to all the names in the environment and process \(P\). That way \(R\) will not be able to effect the trajectory or assign any values to local \(x\) of \(P\). The variable scope operator of hybrid Chi can confine the scope of a variable to a process term and hence can prevent other processes from accessing or modifying it. The variable declared in a variable scope operator are invisible to processes outside the scope operator.

This approach is different to that of discontinuity relations in \(ACP^{ert}_{hs}\). Declaring variables to be local have their own advantages which have been discussed in sections 3.2 and 3.3.

Continuing further with variable discontinuities that should be allowed, \(ACP^{ert}_{hs}\) is extended with a localization operator, denoted by \(\triangledown\). Semantically it has the same effect as discontinuity relations. Localization of a variable to a process will not inhibit discontinuities caused by actions from within the process, but any outside process will not be able to modify the localized variable when the given process can idle.

We illustrate the effect of this operator by means of the following example:

Consider the example of a ball moving on a smooth horizontal surface. The ball decelerates because of friction between the ball and the horizontal surface. The force of friction on the ball is given by \(F = \mu N\), where \(\mu\) is the coefficient of friction and \(N\) is the normal force at the point of contact. As the ball progresses, the material of the horizontal surface keeps on varying after regular intervals and so does the frictional constant. There is friction due to air resistance also, but we disregard this. Therefore, at any instant the deceleration of the ball is greater than that caused by the ground friction alone, i.e. \(\dot{x} \leq - \left(\gamma N/m\right)\), where \(m\) is the mass of the ball, \(\gamma\) denotes the value of frictional constant at instant time. The motion of ball with initial velocity \(\dot{x} = 200\), can be specified in \(ACP^{ert}_{hs}\) as follows:

\[
P = (\dot{x} = 200) \star (\dot{x} \leq -(\gamma N)/m) \sigma_{\{x\}}^* \sigma_{rel}(\delta)
\]
The change in frictional constant after every $r$ time units can be expressed as follows:

$$Q = \sigma_{rel}(y = f(time)) \overset{\text{modify} \mu \cdot Q}{\rightsquigarrow}$$

A parallel composition between the two processes specifies the whole system. Notice in $P || Q$, $x$ is a continuous variable. No discontinuities for $x$ are allowed when $P || Q$ is placed in composition with other processes. Where as discontinuities for $y$ are allowed. We would like to restrict the discontinuities to $y$ to only to $P || Q$. The localization operator applied to the parallel composition, $y \triangledown (P || Q)$, will update the value of $y$ after every four seconds but will not allow any third process to update $y$ after the action $\text{modify} \mu$ has taken place.

(Note that the variable time will also evolve as a continuously differentiable variable (like the variable $x$) with constant derivative 1. We omit the details of its evolution.)

## 5 Parallelism in Process algebras for hybrid systems

The extent to which parallel processes are allowed to interact with each other is an important property that determines usefulness of a process algebra. Hybrid CSP (see [10]), is probably the first process algebra for specification of hybrid systems. A provision lacking in hybrid CSP was that processes in parallel composition were not allowed to influence each others variables, i.e., in a parallel composition of $P || Q$, the set of environment variables whose changes are controlled by $P$ must be disjoint from the set of variables governed by $Q$. Process algebras for hybrid systems presented later allow more interaction between parallel processes.

$ACP_{hs}$ and HyPA are extensions of $ACP$. They inherit operators for parallel composition from $ACP$. The operators for parallel composition are as follows:

1. $P || Q$-parallel merge;
2. $P | Q$-communication merge; and
3. $P \lfloor Q$-left merge.

Primarily these operators have the same meaning as they had in $ACP$. The addition of data and data manipulation operators to process algebras $ACP_{hs}$ and HyPA add complexity to the semantics of parallel operators. Both HyPA and $ACP_{hs}$ have a large number of axioms for parallel operators. These axioms reflect interactions between reinitializations and flow clauses (in HyPA) or signal transitions and signal evolutions (in $ACP_{hs}$) of processes composed in parallel.

Hybrid Chi has only one operator, i.e. parallel merge $||$, for parallel composition. Hybrid Chi does not have a comprehensive set of axioms for the parallel operator. Left parallel operator is necessary for a finite axiomatization.
of parallel merge, but is not present in hybrid Chi. Hybrid Chi has CSP like send and receive actions. Values represented by closed variable expressions can be exchanged in communication. The labels of send, receive or communication actions cannot be used in action predicates. The definition of an action predicate allows action labels only from the set $A_{\text{label}}$, which is disjoint from $A_{\text{comm}}$ that contains the labels of send, receive and communication action. Action predicates in parallel composition will not communicate but will take place in an interleaving fashion. Without a jump predicate to control the values of variables, jumping, dotted and algebraic variables can jump arbitrarily during communication. The values received in a communication can be stored in discrete, continuous or algebraic variables. Values received cannot be stored in any dotted variables.

$\phi$-Calculus inherits its communication constructs from $\pi$-calculus. In $\phi$-calculus, values, channel and variable names can be exchanged during communication. Passing of a channel name provides reconfigurability to a system. Passing of a private variable widens the scope of the variable. $\phi$-calculus has a different approach towards instantaneous modifications to its environment. Assignment and resets in $\phi$-calculus are independent actions themselves. They cannot be attached to send, receive actions or any other actions. Like action predicates of hybrid Chi, environmental action prefixes of processes in parallel composition in $\phi$-calculus will interleave. Therefore the complexity of agreement does not arise as in reinitializations in HyPA or signal transitions in $ACP_{\text{hs}}$.

$\phi$-calculus has a replication operator. Replication of a process term $P$ is denoted by $!P$. $!P$ stands for arbitrarily many instances of process $P$ running in parallel. Syntactically,

$$!P \equiv P || !P$$

Recursion in process definitions is modelled by using the replication operator in $\phi$-calculus.

### 5.1 Delay behaviour

The process terms in parallel composition must synchronize during a delay. i.e. the process terms have the same variable evolution during the delay. A parallel composition can delay as long as all its component process terms can delay.

We discuss each process algebra one by one.

- **HyPA:**

  1. In HyPA, delays are modelled by means of flow clauses. If two flow clauses are placed parallel to each other then the resulting composition can only flow in a way that is possible for both flow clauses. Consider the following parallel composition,

$$([x | x^+ = 0] \gg (x | \dot{x} = 1)) \parallel ([x | x^+ = 20] \gg (\dot{x} \geq 0)).$$

  We denote the term on the right hand side by $P$ and the term on the left hand side by $R$. 

48
The flow clause in $P$ declares $x$ to be a continuous variable and allows it to flow only as the differential equation $\dot{x} = 1$. The flow clause in $R$ allows $x$ to jump initially and gives it more freedom in evolution. $P \parallel R$ will evolve according to $\dot{x} = 1$ and the trajectory of $x$ will start with $x = 0$.

Note that the reinitialization in process term $R$ is redundant as $x$ can jump again immediately after being assigned 20. If $x$ was not allowed to jump initially by the right hand side flow clause, then the two process terms could not agree on an initial value of $x$ and the composition could not delay. Reinitializations as well as flow clauses are taken into account when determining a common solution for delayable HyPA process terms in parallel composition.

2. A parallel merge of two process terms can also delay, if one of the process terms terminates immediately and the other can delay. i.e., a delayable process term in parallel composition with an empty process can delay. But the delayable process term and the empty process term in parallel must agree on the previous values of variables. For example,

\[
\begin{bmatrix}
x^- & \geq 50 \\
x^+ & = 0
\end{bmatrix} \gg (x \mid \dot{x} = 1) \parallel \begin{bmatrix}
x^- & = 50 \\
x^+ & = 30
\end{bmatrix} \gg \epsilon
\]

The reinitializations on the left and righthand side of parallel composition agree on the previous valuation of $x$, but not on the new values. A variable valuation update always accompanies an action or time transition. Therefore the variable update in $\begin{bmatrix}
x^- & = 50 \\
x^+ & = 30
\end{bmatrix}$ never takes place and it only acts as a guard. The above process term can delay with a trajectory of $x$ with a value equal to 0 at the start of the flow and derivative $\dot{x} = 1$.

3. Process terms composed with communication merge $|$ can delay in the same way. The left merge in HyPA, cannot delay initially. It must begin with an action of the process term on its left.

- $ACP_{hs}$: A process term in $ACP_{hs}$ equivalent to $P \parallel R$, (as defined in HyPA) can be written as follows:

\[
((x = 0) \wedge (\dot{x} = 1) \cdot \sigma_{rel}(\delta)) \parallel ((\dot{x} \geq 0) \cdot \sigma_{rel}^*\delta).
\]

We did not use signal transition operator as compared to the reinitialization constructs in HyPA, because in $ACP_{hs}$, variable values are updated according to transition propositions only when signal transition operators appear with an action. Before a delayable process, signal transitions are reduced to guards only.

A communication merge $P \mid Q$ can delay in the same way.

There is no empty process in $ACP_{hs}$. A parallel or communication merge of processes can only delay if all constituent processes can delay.
The left parallel operator of $ACP_{hs}^{rt}$ is also delayable. $P \parallel Q$ can delay if the delay behaviors of both processes synchronize. For example consider the following process term,

$$(x = 0) \cdot ((\dot{x} \geq 1) \cdot \sigma^5_{\text{rel}}(\sigma^3_{\text{re}}(x^\bullet = x \cdot \text{end}) \parallel (\dot{x} = 1) \cdot \sigma^5_{\text{rel}}(\text{finish}))$$

It can do the following time and action transitions starting with valuation $\{x \mapsto \cdot\}$:

$$\langle (x = 0) \cdot ((\dot{x} \geq 1) \cdot \sigma^3_{\text{re}}((\sigma^3_{\text{re}}(x^\bullet = x \cdot \text{end}) \parallel (\dot{x} = 1) \cdot \sigma^5_{\text{rel}}(\text{finish})), x \mapsto 0) \rangle_3 \xrightarrow{\dot{x} = 1} \langle (\dot{x} \geq 1) \cdot \sigma^5_{\text{re}}((x^\bullet = x \cdot \text{end}) \parallel (\dot{x} = 1) \cdot \sigma^5_{\text{re}}(\text{finish}), x \mapsto 2) \rangle_3 \xrightarrow{\text{end}} \langle (\dot{x} = 1) \cdot \sigma^2_{\text{re}}(\text{finish})), x \mapsto 3 \rangle$$

**Hybrid Chi:** In hybrid Chi, a parallel composition can delay only if all component process terms can synchronize during delay. Two delay predicates in parallel composition are as follows:

$$(x = 0) \cdot ((\dot{x} = 0) \cdot \sigma^3_{\text{re}}((x^\bullet = x \cdot \text{end}) \parallel (\dot{x} = 1) \cdot \sigma^5_{\text{rel}}(\text{finish})))$$

They can delay with a trajectory of $x$ with derivative of 1.

The following parallel composition will delay as long as the guard of the second component remains false. The variable $x$ will evolve with $\dot{x} = 1$ during the delay. A guard can delay in any manner and the term $[\cdot \sigma^2_{\text{rel}}]$ with any delay operator also has no restrictions on variable evolution.

$$(x = 0) \cdot ((\dot{x} = 1) \parallel (x = 50) \rightarrow \sigma^2_{\text{rel}}[\cdot \sigma^2_{\text{rel}}])$$

As soon as the guard becomes true, the send and receive actions on channel $ch$ synchronize.

**$\phi$-Calculus:** In $\phi$-calculus, the flow clauses are part of the environment. They are placed in the environment with a reset action. A reset action of the form $[\text{reset } \dot{x} \text{ with } \{\dot{x} \mid \dot{x} \geq 0\}]$ will replace any flow clauses in the environment with $\dot{x}$ as a resetable name. Therefore in a parallel composition, we cannot see the result of interaction of flow clauses of process terms in parallel as we saw in the other process algebras.

Consider the following process:

$$(\emptyset, [x := 0], [\text{reset } \dot{x} \text{ with } \{\dot{x} \mid \dot{x} = 1\}], 0 \parallel [x := 20], [\text{reset } \dot{x} \text{ with } \{\dot{x} \mid \dot{x} \geq 0\}], 0)$$

We start the process with an empty environment, i.e., there are no variables or flow clauses yet in the environment. All environmental actions are with true guards. They can occur in any interleaving fashion. The process term $0 \parallel 0$ is delayable. The evolution and the initial value of $x$ during the delay depends on the assignments and resets occurring last.
In order to see the interaction of two flow clauses, \( \dot{x} \) needs to be removed from the reset list of reset actions. This could be easily achieved if empty reset lists were allowed in reset actions.

A parallel composition of process terms can delay as long as each constituent of the parallel composition can delay. In \( \phi \)-calculus, a silent action followed by an environmental action with true guard is urgent. Therefore it may happen that individual process terms can delay but their parallel composition cannot. For example, consider the following process term

\[
\nu a (\nu x \cdot x := 0). (\text{reset } \dot{x} \text{ with } \{\dot{x} = 1\}) . \dot{a}(x) . 0 \parallel a(y).[\gamma].0).
\]

In an empty environment after the assignment and reset actions, we get the following process:

\[
(\left(x : 0 \{\{\dot{x} | \dot{x} = 1\}\}\right), \nu a (\dot{a}(x) . 0 \parallel a(y) . [\gamma] . 0)).
\]

The above process can communicate on channel \( a \) immediately or it can delay before communication. The maximum duration for which the above process can postpone communication on channel \( a \), depends on the predicate \( \gamma \). As soon as \( \gamma \) is true, communication must take place.

5.2 Action Behaviour:

- **HyPA:** We discuss the action behaviour of parallel operators of HyPA one by one:

1. \( P \parallel Q \):
   - \( P \parallel Q \) can interleave the action behaviour of \( P \) and \( Q \). i.e., \( P \parallel Q \) can first do an action of \( P \) and continue as a parallel merge of process term following the action in \( P \) and \( Q \). Vice versa, \( P \parallel Q \) can first do an action of \( Q \) and continue as a parallel merge of process term following the action in \( Q \) and \( P \).
   - It can also synchronize communicating actions. In HyPA as in \( ACP \), there are no special send and receive actions. The process algebra is provided with a communication function, defined as \( \gamma : A \times A \rightarrow A \), where \( A \) is a set of actions. The communication function defines which actions can communicate. It is a partial function. The communication defined is handshaking and synchronous, i.e. only two actions can communicate and simultaneous execution of actions is required for communication. An encapsulation operator blocks the individual execution of communicating actions to enforce communication.

In communication, \( P \) and \( Q \) can must agree on any variable reinitializations accompanying their first actions. For example, the following process term is deadlocked.

\[
\partial_{\{a,b\}}([x | x^+ = 30] \triangleright a \parallel [x | x^+ = 40] \triangleright b),
\]

(where \( \gamma(a, b) = c \))
2. $P|Q$: A communication merge can delay like the parallel merge.
While performing actions it behaves like a parallel merge with the restriction that the first action performed is the result of communication between first actions of $P$ and $Q$. In case the given communication function does not define the communication between first actions of $P$ and $Q$, the communication merge deadlocks. Actions cannot communicate with the empty process term. $P$ and $Q$ can must agree on any variable reinitializations accompanying their first actions. After the first communication, $P|Q$ continues as the parallel merge of the processes following the first actions in $P$ and $Q$.

3. $P \parallel Q$: $P \parallel Q$ cannot delay initially. The first action performed by $P \parallel Q$ must be an initial action of $P$. After the first action, $P \parallel Q$ continues as the parallel merge of the process term following the first action in $P$ and the process term $Q$.

4. In $ACP_{hs}^{srt}$ and HyPA, data can be exchanged between process terms during communication as follows:
If some data element $d$ belonging to a data set $D$, is to be sent during communication, then a summation of parameterized actions with a parameter of type D, over all elements of $D$ is used.

For example,

$\partial_H(\text{send}(d_0) \odot P \parallel \Sigma_{d \in D} \text{rec}(d) \odot Q(d))$,

is equivalent to

$\text{comm}(d_0) \odot \partial_H(P \parallel Q(d_0))$

with the communication function and $H$ defined as follows:

$\gamma(\text{send}(d), \text{rec}(d)) = \text{comm}(d)$, for all $d \in D$

$H = \{\text{send}(d), \text{rec}(d) \mid d \in D\}$.

This will have an effect similar to

$h!!d_0 ; P \parallel h??x ; Q(x)$

in hybrid Chi.

- $ACP_{hs}^{srt}$: The action behaviour of parallel operators of $ACP_{hs}^{srt}$ is like the action behaviour of HyPA parallel operators. The parallel merge can interleave and synchronize communicating actions. In communication, like HyPA, communicating process terms must agree on any changes in variable valuations accompanying their first actions. For example,

$\partial_{(a,b)}((x^\bullet = 30) \triangleright \tilde{a} \mid (x^\bullet = 40) \triangleright \tilde{b})$,

(where $\gamma(a,b) = c$)

Process terms in the above example cannot communicate as they do not agree on variable valuations after their first actions.

There are some differences in the parallel operators of HyPA and $ACP_{hs}^{srt}$. They are as follows:
1. The left parallel operator of $ACP_{hs}$ is delayable. $P \parallel Q$ can delay if the delay behaviors of both processes synchronize. It can perform an action of $P$.

2. In parallel or left merge, i.e. $P \parallel Q$ and $P \parallel \{Q\}$, if one of the processes performs an action first, then any change in variable values accompanying the action should be such that the signal of the other process in parallel composition remains true after the action. The signal of a process term is a proposition that states what assumptions about the environment are made by a process term. As explained in section 3.4, assumptions about the environment are specified through a signal emission or an evolution proposition.

Consider the following process term. The action $a$ cannot take place before action $\tilde{b}$, as the variable update accompanying it falsifies the signal of process term on the right.

$$(x^* = 4) \triangleright a \parallel (x \leq 3) \triangleright \tilde{b}$$

Another example of restriction due to signal of a process are as follows:

$$(x = 0) \triangleright ((x^* = 4) \triangleright a \parallel (\dot{x} = 1 \land x \leq 3) \triangleright \sigma_{rel}^1(\tilde{b}))$$

Action $\tilde{a}$ cannot be performed as the signal of the right hand process requires $x$ to be less than or equal to three.

In HyPA there are no signal emission operator or signal relations as in $ACP_{hs}$. Therefore, there is no such restriction on a HyPA process. The following process term can do action $a$ before action $b$.

$$[x | x^+ = 4] \triangleright a \parallel [x | x^- \leq 3] \triangleright b$$

The reinitialization $[x | x^- \leq 3]$ acts only as a guard and not as a signal emission. After performing action $a$, the above process term will deadlock, as the guard on the right becomes false.

3. In $P \parallel Q$ or $P \parallel \{Q\}$, if $P$ performs an action first and $Q$ is a delayable process, then $P$ must respect the continuity requirements of $Q$. The continuity requirements of $Q$ state that any variable and its derivative declared as continuous by (the evolution operator of) $Q$ cannot be modified by $P$. In the following process term action $a$ cannot take place as it tries to update the value of $x$ which is declared as continuous by the process on the right.

$$(x = 0) \triangleright ((x^* = x + 1) \triangleright a \parallel (x \leq 3) \triangleright \sigma_{rel}^3(\tilde{b}))$$

The above process term will deadlock, whereas the process term given below has an option of performing $\tilde{c}$ . The variable update accompanying action $c$ updates variable $v$, which has not been declared as a smooth variable by the middle process term.

$$(x^* = x + 1) \triangleright a \parallel (x \leq 3 \land v \geq 10) \triangleright \sigma_{rel}^3(\tilde{b}) \parallel (v^* = 20) \triangleright \tilde{c}$$
• **Hybrid Chi**: The parallel operator of hybrid Chi can interleave action predicates and synchronize matching send and receive actions. Send and receive actions on the same channel with identical values sent and received are matching. Hybrid Chi also has a signal emission operator like $ACP_{hs}^{\text{port}}$. Hybrid Chi specifications can also express assumptions about the environment in a signal emission proposition or a delay predicate. Hybrid Chi has consistent action semantics, i.e. a process of hybrid Chi is never allowed to enter in an inconsistent state. An inconsistent state results when the values of environment variables do not agree with the delay predicate or the predicate of signal emission operator in a process.

Like signal relations of $ACP_{hs}^{\text{port}}$, the semantics of hybrid Chi has consistency relations. A consistency relation (see section 3.3) describes whether the signal emission or delay predicates of a process term are satisfied by a variable valuation or not. A consistent equation semantics enforces that a process $(x := y \parallel y = 1)$ behaves the same as $(x := 1 \parallel y = 1)$.

Now consider another process term.

\[
\{x\} : x = 0 \gg \tau ; \{x\} : x = 4 \gg l_a \parallel \dot{x} = 1 \land x \leq 3
\]

Let $x$ be a continuous, non-jumping environment variable. The above process term will behave as follows:

- Internal action $\tau$ is executed and $x$ is assigned value 0.
- $\{x\} : x = 4 \gg l_a$ cannot perform action $a$, as the predicate requires $x$ to be assigned a value of 4. This is inconsistent with the process term on the right.

Matching send and receive actions may interleave or communicate. In order to enforce communication an encapsulation operator that blocks individual sends and receives is used.

• **$\phi$-Calculus**: In $\phi$-calculus, there are no consistency predicates or signal relations like in Hybrid chi or $ACP_{hs}^{\text{port}}$. $(E, P \parallel Q)$ can do any action that $(E, P)$ or $(E, Q)$ can do. The environmental actions, delay action prefix, silent action prefix and send or receive actions can interleave in parallel composition.

Send and receive actions on a private channel cannot interleave and are forced to communicate.

### 6 System invariants

One manifestation of a delay predicate of hybrid Chi or evolution proposition of $ACP_{hs}^{\text{port}}$ in representing system invariants is as follows:

Suppose you are involved in the development of the specification of a large system with many sub-processes. Different people develop specifications of sub
processes. The parallel composition of all these specifications constitutes the complete system. The safety criteria of the system requires that a certain condition on the environment variables always holds. One way to represent that this safety condition is always satisfied, is to place a predicate representing the safety condition in parallel with the parallel composition of all the sub processes. The semantics of hybrid Chi or $ACP_{hs}$, with the help of signal relations and consistency predicates ensures that an action or delay transition violating this constraint can never occur.

If $u$ represents a safety condition, and $P, Q, R,$ represent specifications of sub processes, then in hybrid Chi the following process term represents the system together with the invariant $u$:

\[ u \parallel P \parallel Q \parallel R \parallel \ldots \]

In $ACP_{hs}$, the desired system can be represented as follows:

\[ u \sigma^*_\text{rel}(\tilde{\delta}) \parallel P \parallel Q \parallel R \parallel \ldots \]

In HyPA, we cannot enforce a system invariant by a similar construction. When we place a flow clause with flow predicate $u$ in parallel to other HyPA specifications,

\[ (u) \parallel P \parallel Q \parallel R \parallel \ldots \]

then the parallel composition can only delay if all of the process terms $P, Q, R, \ldots$ agree with $u$ throughout the delay duration. But the semantics of HyPA does not prevent $P, Q$ or $R$ from performing actions with variable updates that violate $u$.

For example,

\[ (x \leq 100) \parallel [x \mid x^+ = 200] \gg a \]

There are no consistency predicates in HyPA to prevent action $a$.

In $\phi-$calculus, flow clauses are part of the environment. We can place the desired invariant $u$ in the environment through a reset action in parallel to other $\phi-$calculus specifications. But as in HyPA, this does not prevent process terms in parallel composition to assign values to variables that violate predicate $u$.

### 7 More about Time determinism

In $ACP_{hs}$ and hybrid Chi, arguments in favor of time determinism say that their alternative choice operator can still model non-determinism. For example,

- A process that has a number of options for variable evolutions can be modelled. Consider a process with an option to evolve either like $\dot{x} = 1$
or like $\dot{x} = 2$, where $x$ is a continuous variable.

$$\begin{align*}
ACP_{hs}^\text{sort}: \\
(\dot{x} = 1 \lor \dot{x} = 2) \sigma_{\text{rel}}(\delta)
\end{align*}$$

Hybrid chi:

$\dot{x} = 1 \lor \dot{x} = 2$

$\phi - \text{calculus}$:

$$\begin{align*}
( \{(x : 0) \mid \dot{x} = 1 \lor \dot{x} = 2\} ) \cdot 0
\end{align*}$$

• Based on different variable evolutions during a delay, different actions can be chosen in future. for example

$$\begin{align*}
ACP_{hs}^\text{sort}: \\
P &= (x = 0) \& (\dot{x} = 1 \lor \dot{x} = 2) \sigma_{\text{rel}}(x) \\
&\quad \sigma_{\text{rel}}((x = 2): (x = \ast x) \sigma_{\text{rel}}(\delta) \\
&\quad +(x = 4) \rightarrow (x = \ast x) \sigma_{\text{rel}}(\delta)
\end{align*}$$

$\chi$:

$$\begin{align*}
P &= (x = 0) \& (\dot{x} = 1 \lor \dot{x} = 2) \\
&\quad \begin{cases}
(x : 0, t : 0) \\
(x : 0, t : 0)
\end{cases}
\end{align*}$$

$\phi$:

$$\begin{align*}
P &= (x = 0, t = 0) \rightarrow (x = \ast x) \rightarrow (x = \ast x) \\
&\quad \begin{cases}
x = 2 \land t = 2 \rightarrow \text{true} \Rightarrow a : P \\
x = 4 \land t = 2 \rightarrow \text{true} \Rightarrow b : Q
\end{cases}
\end{align*}$$

Hybrid process algebras frequently refer to hybrid automata when explaining alternative composition. In a hybrid automaton, there are edges and locations. Edges correspond to guarded discrete actions and locations correspond to environment flows. A process is allowed to delay in a location as long as the invariant and flow conditions of the location are satisfied. A process can also jump to a new location through a discrete actions as soon as the guard of its edge becomes true. The hybrid automaton corresponding to the above example is given in figure 7. Where the locations $P$ and $Q$ refer to locations corresponding to process terms $P$ and $Q$ respectively.

Consider another example of a hybrid automaton representing a thermostat process. This example has been taken from [9]. The process starts with an initial temperature of $20^\circ$ in Off location. The temperature continuously decreases at rate $\dot{T} = -0.1T$. The process is allowed to delay in location Off as long as the temperature remains $18^\circ$ or above. It is also allowed to perform a silent action and jump to location On as soon as the temperature falls below $19^\circ$. The thermostat automaton is given in figure 7.

Therefore, we find that in hybrid automata, actions are not preferred over delays if both are possible. This choice resolution between an action and a delay is closer to weak time determinism than strong time determinism.
Figure 1: Time non-determinism in Hybrid Automata

Figure 2: Weak time determinism in Hybrid Automata - A thermostat example
8 Available Tools

Tools play an important role in enhancing the usability of a formalism. $ACP_{bs}$ has no tools for simulation or verification of its specifications. A brief introduction of tools available for other process algebras is as follows:

8.1 Hybrid Chi

The tool set available for hybrid Chi specifications consists of a hybrid Chi simulator and a translator of hybrid Chi specifications into hybrid automata.

- **Chi2HA translator**: A subset of hybrid Chi language has been formally translated into a class of hybrid automata closely resembling I/O hybrid automata. The tool “Chi2HA” automatically translates hybrid Chi specifications into hybrid automata. A tool PHaVER, (Polyhedral hybrid automaton verifier) is then used to analyze the translations. We have not used this tool as it was not available publicly.

- **Hybrid Chi Python Simulator**: A symbolic simulator for hybrid Chi specifications is available. Hybrid chi specifications can be first compiled and then simulated. The compiler points out syntax errors. For example, hybrid Chi language that has been implemented is strongly typed. Types such as real, int, nat (stands for natural numbers) and bool are available for environment variables. One can get compiler errors such as “cannot resolve an expression” due to type mismatch.

  The semantic errors are detected by the simulator. There are some restrictions that need to be followed in the specifications for simulation and not all constructs of hybrid Chi are available for use. For example, algebraic variables, variable scope operator and the delay operator are not available.

  A complete list of restrictions on specifications for simulations is available on hybrid Chi web site (http://se.wtb.tue.nl/sewiki/chi/hybrid_chi_python_simulator) and it keeps on evolving. A list of available tool chains, documentation manuals and links to mailing lists for chi users are available on this website http://se.wtb.tue.nl/sewiki/chi/start. Hybrid chi language simulated by the simulator has evolved from discrete and timed versions of chi. Therefore this website contains information about tooling and documentation of discrete-event and timed chi also.

  A number of observation about the simulator are as follows:

  - Discrete, continuous variables and dotted variables are available for use in specifications.
  - Declared variables need to be initialized.
  - For simulating delays, an ordinary differential equation of the form $\dot{x} = \ldots$ for each continuous variable is required in delay predicates.
  - The simulator has a manual as well as automatic mode. In manual mode, the simulator prompts the user for resolving alternative choices.
and for specifying the duration of delays. In automatic mode, the simulator makes these choices itself.

- The simulator gives following textual information:
  - the hierarchy of different constructs or structure of a specification,
  - the choices available at an instant in time,
  - Maximum durations available for a delay, and
  - discrete and communication actions performed

- A graph of variable evolution during delays is plotted by using gnuplot. One can choose which environment variables one wants to plot.

- Consistency predicates for hybrid Chi processes have been implemented.

- Maple is used as a solver to solve differential equations, predicate expressions and to find the maximum duration of delays.

We have observed that in automatic mode, the simulator always selects the time duration of zero duration if available.

In interactive mode, the user needs to be careful while choosing the time duration of delays, as a value greater than the maximum duration results in abrupt program abortion. As suggested by the developer of the tool, when dealing with real numbers, if possible one should use rational numbers of the form integer/integer instead of numbers with decimals.

Sometimes the simulator gets stuck on a very small time duration like $0.2000000000e^{-8}$. Variable values do not change during such a small duration. The simulator keeps on offering this duration to the user as a possible time transition till the user selects an action in alternative choice with the time transition.

We include the simulation result for hybrid Chi specification for train gate controller in figure 8.1, after running the simulation in automatic mode for 40 seconds. (See section 9 for the statement of the case study). In the figure $x$ represents the distance of the train from the gate (initialized to $-1400$), $r$ represents the angle of the gate (initialized to $90^\circ$) and $d$ represents a possible delay by controller in forwarding a message to gate.

8.2 HyPA

- The linearization tool: Linearization is a process by which an algebraic specification is transformed into a specification without parallelism. A linearized specification is a direct representation of all behaviors possible by a system. It is usual to have manipulation tools that can only work with linearized system specifications. In HyPA, axioms allowing safety analysis of environment variables in HyPA models can be only be used with process terms without a parallel operator.
Figure 3: Result of train gate controller simulation in hybrid Chi
Linearization of a HyPA specification can be done manually by repeated application of axioms. We would like to mention here that manual linearization of even a small algebraic specification can be a very long and tedious task. Therefore tools for linearization can be very useful.

In [20], details of two linearization algorithms for HyPA specifications are given. One linearization algorithm is to be used with the μCRL toolkit (see [23]) and the other does not use any external tools. A new operator called the variable abstraction operator has been introduced in [20]. This operator is required by the second linearization algorithm for linearizing large HyPA specifications. A number of dummy variables are introduced in the specification during the application of the linearization algorithm. The behaviour of these variables is abstracted from the final linearized specification when the algorithm outputs its result. A simple tool with textual interface, implementing the second algorithm is available. We tried to linearize the train gate controller specification given in section 9 using the linearization tool. The tool gave a number of error messages that we failed to resolve. A number of HyPA specifications that have been linearized using the tool are included in the distribution.

• The simulator There is a simulator available for simulating HyPA specifications. [21] fully describes the working and development of this simulator. The simulator simulates by repeatedly calculating the first possible transitions of a HyPA model through a function called the first transition function. This function is based on the operational semantics of HyPA. Any HyPA specification can be simulated (including the unguarded ones).

Several observations regarding the simulator are as follows:

− The format of the input file is different from that of the linearization tool. Some additions such as variable declarations and clause formalisms are added to an input file. This information helps the simulator to identify the variables and some syntax errors.

− Two clause formalisms specific to mathematica (see website http://www.wolfram.com/products/mathematica/index.html) have been developed. They define the format for reinitialization and flow clauses. They are as follows:

  * Mathematica clause formalism
  * Symbolic mathematica clause formalism

The mathematica formalism uses numerical methods to solve clauses. This poses certain problems as exact values are difficult to reach. Therefore equalities among variables may never become true.

A construct \((a == b)\), with variables \(a\) and \(b\), should be replaced by a construct of the form \(|a - b| \leq e\), with \(e\) representing a very small value.
The evolution of variables against time or against each other during a delay can be viewed in a graph. The user can decide the types and interval ranges of a graph.

Discrete actions can also be viewed against a time axis.

Alternative choices at any instant are visible in a window. A lookahead parameter can be set by the user. This parameter indicates the time duration for which a user can see the result of all possible alternatives (i.e., variable evolution or action behaviour) before they are actually chosen by the user.

Like the simulator of hybrid Chi, there are two modes one random or automatic, and the other step by step, available for simulation.

* In random simulation choices are made by the simulator. The visualizations of the graph and discrete actions are updated continuously. A stop criterion for a random simulation can be given. The stop criterion currently implemented in the simulator is an action. Another idea proposed in [21] for a stop criterion was a condition on variables. But this is not implemented yet. In random mode, the simulator resolves the choices between alternatives and picks a random duration of a delay. The user can adjust the simulator to always select the maximum possible duration of the delay.

* In step by step mode, the simulator prompts the user to resolve an alternative choice or to choose the duration of a delay. A blue adjustable bar in the variable graph indicates the maximum duration of the delay.

The user can undo an action or time step.

Ordinary first order linear differential equations and inequalities can be used in flow clauses for simulation.

Not all constructs of Mathematica are available. Using logical functions of Mathematica such as `Implies` or `Not` in flow clauses results in an error message that informs that either `&&` or `=` or `<>` (stands for inequality) or inequalities expected.

If the variables have not been initialized or the system of equations in flow clauses or reinitialization clauses is underspecified, the simulator prompts the user to add more restrictions.

The user can zoom in and out of the variable evolution graph. Zooming in and out of a graph can take a long time.

The simulator tool is interactive. It gives a number of options to the user and provides a good visualization of options available, actions performed and variables’ evolution. The simulator can help detecting errors in specifications. An error in the steam boiler specification given in [16] (chapter 3) was detected by the simulator. By running its simulation, it was observed that water level can become negative initially.
The structure of the specification being simulated is not visible in any diagram. In case of an underspecified system, sometimes a user has to enter the additional restrictions a number of times before simulator continues with the simulation.

A snapshot of the tool with the simulation of train gate controller in HyPA is given in figure 8.2.

Figure 4: HyPA Simulation Tool

8.3 \( \phi \)–Calculus

SPHIN is a model checker for hybrid system specifications in \( \phi \)–calculus. SPHIN is an extension of the model checker SPIN. A simulator only simulates an instance of a specification, i.e. one of the possible behaviors of a system. By repeated simulations, one can gain more insight into a specification and increase one's confidence about its correctness. Model checking is a verification technique where one can make sure that certain safety and liveness conditions for a specification always hold. For example, in the train gate controller specification, we want to be sure that the gate is always fully closed whenever the train is within a certain distance from it. This condition comes under the category of safety conditions.

For using the tool SPHIN, it is necessary to be familiar with the SPIN model checker. The input language to SPHIN is PROMELA-Hybrid. PROMELA-
Hybrid is a superset of PROMELA (the input language for SPIN). Additions in PROMELA have been made to allow analog variables declarations. And also to represent their instantaneous and continuous modifications during the execution of specification.

A typical Hybrid statement is of the form:

```
promela_statement when_clause reset_clause
```

The `when_clause` states conditions on analog variables, which must be satisfied for the PROMELA statement to be executed. The `reset_clause` states the new conditions for an analog variable. The reset can only be implemented if the PROMELA statement is executable and the when-clause is true.

Three intervals are associated with an analog variable on its declaration. These represent a range of possible initial values, a range of possible derivative values and invariants on an analog variable. The semantics of PROMELA-hybrid is different in this respect from $\phi$-calculus that a range of possible initial values for the variable and its derivative are possible.

The SPHIN model checker allows more non-determinism than the simulators of hybrid Chi and HyPA.

In model-checking, all possible states of a system during an execution of the specification need to be represented. In SPHIN, the possible valuations of analog variables are represented as polyhedrons in states. The Parma Polyhedra Library (PPL) is used to calculate new polyhedra as continuous variables change.

A number of examples including a train gate controller, Fischer’s mutual exclusion algorithm, 3-robot bucket brigade and flocking agents have been specified in PROMELA-hybrid and verified using the SPHIN model checker. These examples are available with the SPHIN distribution.

SPHIN is a text based tool. It gives messages about the actions performed and the evolution of the variable polyhedron. It is not easy to interpret the dimensions of the polyhedron from the textual messages. The user can use both the simulation and verification mode of SPHIN. Messages regarding variable polyhedron can also be suppressed to concentrate on the sequence of action executed.

The train gate controller specification in PROMELA-Hybrid distributed with the SPHIN distribution has been taken from [9], with maximum possible delay of the controller equal to 3 seconds. Our train gate controller specification is slightly different with a train that moves faster and also a gate that lowers and raises at a higher rate. (Please see section 9). The maximum possible delay allowed in our train gate controller system is 5 seconds. We tried to verify the safety condition that the gate is always closed when the train is less than 350 meters away from it. The following statement is included in the monitor process.

```
assert(false) when \{x in (-350,100], g in (0,)} \{reset{}
```

With the delay of 5 seconds, we notice that a monitor process always raises a `false` assertion at depth 23. This is contrary to the results obtained through
manual calculation. Linearizing the train gate controller specification in different process algebras indicates that gate is always closed when the train is within 506 meters from it.

We notice that with a delay of 5 seconds, relaxing the safety condition to even \( x = 0 \) raises a false assertion.

The SPHIN verifier gives messages about the actions performed and variables evolution. After a certain execution depth, we cannot follow the sequence of execution due to at times reverse transitions made by the verifier in the state space. The SPIN verifier writes a trail file that can be simulated and any errors detected during the verification can be explored. This option which is called guided simulation is very useful in debugging. It is not available with SPHIN verifier yet.

A number of improvements have been suggested in [13] and [24]. These suggestions include establishing a formal connection between \( \phi \)-calculus and PROMELA-Hybrid, adding guided simulation and random simulation support in SPHIN, providing analog state visualization and a GUI interface for SPHIN.

The PROMELA-hybrid code for the train gate controller specification given in section 9 is given in the appendix.

9 A Case study: A Train Gate Controller

We take a case study of train gate controller from [4]. This case study is similar to but not the same as the train gate controller in [9].

The system of train gate controller has three components:

- train;
- gate; and
- controller

We describe below the behaviour of each component.

- **The train**: When a train approaches the gate from a great distance its speed is between 48 m/s and 52 m/s. As soon as it passes the detector placed at 1000m backward from the gate, an appr signal is sent to the controller. The train may now slow down, but its speed stays between 40m/s and 52 m/s, and pass the gate. As soon as it passes the detector placed at 100m forward from the gate, an exit signal is sent to the controller. A new train may come after the current one has passed the second detector, but only at a distance greater than or equal to 1500 m.

- **The gate**: The gate is able to receive lower and raise signals from the controller at any time. As soon as the gate receives a lower signal, it lowers from 90° to 0° at a constant rate of \(-20°\) per second. As soon as it receives a raise signal, it raises from 0° to 90° at the same rate in the opposite direction.
• **The controller:** The controller is able to receive appr and exit signals from the train detectors at any time. When the controller receives an appr signal, it takes less than 5 s before a lower signal is sent to the gate. When the controller receives an exit signal, it takes less than 5 s before a raise signal is sent to the gate. Because of fault tolerance considerations, appr signals should always cause the gate to go down, and exit signals should be ignored while the gate is going down.

It is assumed that initially there is no train at a distance smaller than 1400m from the gate, the gate is open, and the controller is idling. Moreover, it is assumed that each single train changes its speed only smoothly.

We represent this case study in each process algebra and try to simplify the resulting specifications.

The train gate controller specifications have following environment variables:

- $x$ – for the distance of train from gate
- $r$ – for the angle of gate with the ground
- $d$ – for possible delay of controller.
- $y$ – for speed of the train.

We also give the hybrid automata corresponding to each component in figure 9.

In transition labels, $x,r,d$ denote values of variables before the transition. The primed variables $x', r', d'$ represent values of variables in the new location after the transition.

All the environment variables must evolve smoothly during delays and must not jump arbitrarily during actions. In $\phi$-calculus and hybrid Chi, a separate variable $y$ is used to represent the speed of the train. On the other hand, in HyPA and $ACP_{hs}$, the variables $x, r, d$ and their derivatives are sufficient to represent the train gate controller system. In hybrid Chi the trajectory of dotted continuous variables can be discontinuous. With the delay predicates $48 \leq \dot{x} \leq 52$ and $40 \leq \dot{x} \leq 52$, the speed of the train can jump during delays.

Also dotted variables in hybrid Chi are jumping by definition. Therefore, we declare a continuous environment variable $y$ and set it equal to $\dot{x}$ during delays. $y$ being continuous, it cannot jump during delays or during actions.

In [12], it is mentioned that trajectories of variables can be continuously differentiable functions of time. Therefore both $x$ and $\dot{x}$ will evolve continuously during delays. During send, receive or silent action prefixes, the environment of a $\phi$-calculus process is not modified. During environmental resets, the set of flow constraints is modified but valuation of the environment is not altered.

The valuation does not consist of derivatives of variables. In order to make sure that $\dot{x}$ is not altered during resets, we declare a variable $y$ and add a flow constraint $\dot{x} = y$ to the environment. Since $y$ cannot jump during environment resets therefore $\dot{x}$ cannot jump as well.
Figure 5: Train Gate Controller Automata
9.1 Train Gate Controller in HyPA

\[ \text{Trains} = [x, \dot{x} | x^+ \leq -1400 \land 48 \leq \dot{x}^+ \leq 52] \Rightarrow T^{\text{far}} \]

\[ T^{\text{far}} = (x, \dot{x} | x \leq -1000 \land 48 \leq \dot{x} \leq 52) \]

\[ \quad \Rightarrow [x^- = -1000] \Rightarrow s(\text{appr}) \odot T^{\text{near}} \]

\[ T^{\text{near}} = (x, \dot{x} | -1000 \leq x \leq 0 \land 40 \leq \dot{x} \leq 52) \]

\[ \quad \Rightarrow [x^- = 0] \Rightarrow T^{\text{past}} \]

\[ T^{\text{past}} = (x, \dot{x} | 0 \leq x \leq 100 \land 40 \leq \dot{x} \leq 52) \]

\[ \quad \Rightarrow [x^- = 100] \Rightarrow s(\text{exit}) \odot [x, \dot{x} | x^+ \leq -1400 \land 48 \leq \dot{x}^+ \leq 52] \Rightarrow T^{\text{far}} \]

\[ \text{Gate} = [r, \dot{r} | r^+ = 90 \land \dot{r}^+ = 0] \Rightarrow G^{\text{op}} \]

\[ G^{\text{op}} = (r, \dot{r} | r = 90 \land \dot{r} = 0) \]

\[ \Rightarrow (r(\text{lower}) \odot [\dot{r} | \dot{r}^+ = -20] \Rightarrow G^{\text{dn}} \oplus r(\text{raise}) \odot G^{\text{op}}) \]

\[ G^{\text{dn}} = (r, \dot{r} | 0 \leq r \leq 90 \land \dot{r} = -20) \]

\[ \Rightarrow ([r^- = 0] \Rightarrow [\dot{r} | \dot{r}^+ = 0] \Rightarrow G^{\text{cl}} \]

\[ \oplus r(\text{raise}) \odot [\dot{r} | \dot{r}^+ = 20] \Rightarrow G^{\text{up}} \odot r(\text{lower}) \odot G^{\text{dn}}) \]

\[ G^{\text{up}} = (r, \dot{r} | 0 \leq r \leq 90 \land \dot{r} = 20) \]

\[ \Rightarrow ([r^- = 90] \Rightarrow [\dot{r} | \dot{r}^+ = 0] \Rightarrow G^{\text{op}} \]

\[ \oplus r(\text{raise}) \odot G^{\text{up}} \odot r(\text{lower}) \odot [\dot{r} | \dot{r}^+ = -20] \Rightarrow G^{\text{dn}}) \]

\[ G^{\text{cl}} = (r, \dot{r} | r = 0 \land \dot{r} = 0) \]

\[ \Rightarrow (r(\text{lower}) \odot G^{\text{cl}} \oplus r(\text{raise}) \odot [\dot{r} | \dot{r}^+ = 20] \Rightarrow G^{\text{op}}) \]

\[ C^{\text{ndr}} = C^{\text{idle}} \]

\[ C^{\text{idle}} = (\text{true}) \Rightarrow (r(\text{appr}) \odot [d, \dot{d} | d^+ = 0 \land \dot{d}^+ = 1] \Rightarrow C^{\text{dn}} \]

\[ \oplus r(\text{exit}) \odot [d, \dot{d} | d^+ = 0 \land \dot{d}^+ = 1] \Rightarrow C^{\text{up}}) \]

\[ C^{\text{dn}} = (d, \dot{d} | 0 \leq d \leq 5 \land \dot{d} = 1) \]

\[ \Rightarrow (r(\text{appr}) \odot C^{\text{dn}} \oplus r(\text{exit}) \odot C^{\text{dn}} \oplus s(\text{lower}) \odot C^{\text{idle}}) \]

\[ C^{\text{up}} = (d, \dot{d} | 0 \leq d \leq 5 \land \dot{d} = 1) \]

\[ \Rightarrow (r(\text{appr}) \odot [d | d^+ = 0] \Rightarrow C^{\text{dn}} \oplus r(\text{exit}) \odot C^{\text{up}} \oplus \]

\[ s(\text{raise}) \odot C^{\text{idle}}) \]

68
where,

\[ s(\text{appr}) | r(\text{appr}) = c(\text{appr}) \]
\[ s(\text{exit}) | r(\text{exit}) = c(\text{exit}) \]
\[ s(\text{raise}) | r(\text{raise}) = c(\text{raise}) \]
\[ s(\text{appr}) | r(\text{lower}) = c(\text{lower}) \]

The gate controller system is specified by:

\[ \partial H = (\text{T} \text{ra} \text{ins} | \text{Cnt} \text{r} | \text{Gate}), \]

where

\[ H = \{ s(d) \mid d \in \{ \text{appr}, \text{exit}, \text{raise}, \text{lower} \} \} \cup \{ r(d) \mid d \in \{ \text{appr}, \text{exit}, \text{raise}, \text{lower} \} \} \]
9.1.1 Simplifying $\partial H(\text{Trains} \parallel \text{Cntr} \parallel \text{Gate})$

The expression $\partial H(\text{Trains} \parallel \text{Cntr} \parallel \text{Gate})$ is simplified by repeated application of axioms of HyPA.

$$\partial H(\text{Trains} \parallel \text{Cntr} \parallel \text{Gate}) = \begin{bmatrix} x^+ \leq -1400 \\ 48 \leq x^+ \leq 52 \\ r^+ = 90 \\ \dot{r}^+ = 0 \\ x \leq -1400 \\ 48 \leq \dot{x} \leq 52 \\ r = 90 \\ \dot{r} = 0 \\ x^- = -1000 \end{bmatrix} \gg c_1(\text{appr}) \circ$$

$$\partial H(T_{near} \parallel G^{op} \parallel \text{Cidle} \parallel \dot{r} = 0 \land \dot{r}^+ = 1) \gg C^{dn} \parallel G^{op}$$

$$\partial H(T_{near} \parallel G^{op} \parallel [d, \dot{d} \mid d^+ = 0 \land \dot{d}^+ = 1]) \gg C^{dn}$$

$$\partial H(T_{near} \parallel C^{idle} \parallel [\dot{d} \mid \dot{d}^- = 1]) \gg C^{dn}$$

$$\oplus \quad \begin{bmatrix} d, \dot{d} \mid d^+ = 0 \\ \dot{d} = 0 \\ r^- = 90 \\ x^- = -1000 \end{bmatrix} \gg c_2(\text{lower})$$

$$\oplus \partial H(T_{near} \parallel C^{idle} \parallel [\dot{r} \mid r^+ = -20]) \gg G^{dn}$$

$$\oplus \partial H(T_{near} \parallel C^{idle} \parallel [\dot{r} \mid \dot{r}^+ = -20]) \gg G^{dn} \parallel C^{idle}$$
\[ \left. \frac{\partial H}{\partial T_{\text{near}}} \right|_{\text{\tiny \|C_{idle}\|}} \cong \frac{20}{G_{\text{dn}}} \right|_{\text{\tiny \|C_{idle}\|}} = \begin{bmatrix} \dot{r} \\ x \\ \dot{x} \\ r \\ \dot{r} \end{bmatrix} \begin{align*} \dot{r} &= 90 \\ \dot{r} &= -20 \\ -1000 \leq x \leq -740 \\ -1000 \leq x \leq 0 \\ 40 \leq \dot{x} \leq 52 \\ 0 \leq r \leq 90 \\ \dot{r} &= -20 \\ -820 \leq x \leq -506 \end{align*} \]
A repeated application of axioms of HyPA simplifies the specification and it is obvious how the system is going to behave in different stages. While simplifying the specification, we take into account what can happen when two trains follow each other at minimum possible distance. That is a new train arrives at 1400 m backwards from the gate as soon as the previous train crosses the 100 m detector. From the calculations, we can see that the gate is always closed, whenever the train is within 500 m from the gate.

The bisimulation axioms of HyPA that do not preserve parallel composition are applied to those parts of the specification that are no longer within the scope of a parallel operator.
The simplification of the specification by hand comes out to be a very lengthy procedure and prone to errors. Errors can be due to incorrect application of HyPA axioms or mistakes in noting down process terms as they expand. In this case study, the real analysis of the system is easy. When we expanded the specification, we had in mind a notion of how the system should behave. That helped us in early detection of mistakes in expansion. In more complicated systems, it might take a very long time before a mistake in calculations is detected.

We were unable to resolve a runtime error in the linearization tool while applying it on train gate controller specification. It would be very convenient to automatically linearize HyPA specifications of the size comparable to train gate controller or larger.

9.2 Train Gate Controller Specification in $\phi$-calculus

We give below the train gate controller specification in $\phi$-calculus.

$$Train = \nu xy \left[ (x, y) := (-1400, 52) \right] . T_{far}$$

$$T_{far} = \left[ \text{reset } x, \dot{x}, y \text{ with } \{ x \mid x \leq -1000 \}, \{ y \mid 48 \leq y \leq 52 \}, \{ \dot{x} \mid \dot{x} = y \} \right] . \left[ x = -1000 \right] . appr. T_{near}$$

$$T_{near} = \left[ \text{reset } x, y \text{ with } \{ x \mid -1000 \leq x \leq 0 \}, \{ y \mid 40 \leq y \leq 52 \} \right] . \left[ x = 0 \right] . T_{past}$$

$$T_{past} = \left[ \text{reset } x \text{ with } \{ x \mid 0 \leq x \leq 100 \} \right] . \left[ x = 100 \right] . exit . \left[ (x, y) := (-1400, 52) \right] . T_{far}$$

$$Gate = \nu r \left[ r := 90 \right] . G^{up}$$

$$G^{up} = \left[ \text{reset } r, \dot{r} \text{ with } \{ r \mid \dot{r} = 0 \}, \{ r \mid r = 90 \} \right] . \left( \text{lower.} G^{dn} + \text{raise.} G^{up} \right)$$

$$G^{dn} = \left[ \text{reset } r, \dot{r} \text{ with } \{ r \mid r \geq 0 \}, \{ r \mid \dot{r} = -20 \} \right] . \left( \{ r = 0 \} . G^{cl} + \text{lower.} G^{dn} + \text{raise.} G^{up} \right)$$

$$G^{up} = \left[ \text{reset } r, \dot{r} \text{ with } \{ r \mid \dot{r} = 20 \}, \{ r \mid r \leq 90 \} \right] . \left( \{ r = 90 \} . G^{op} + \text{raise.} G^{up} + \text{lower.} G^{dn} \right)$$

$$G^{cl} = \left[ \text{reset } r, \dot{r} \text{ with } \{ r \mid \dot{r} = 0 \}, \{ r \mid r = 0 \} \right] . \left( \text{lower.} G^{cl} + \text{raise.} G^{up} \right)$$

74
\[
\begin{align*}
Con\text{tr} &= \nu dC^{idle} \\
C^{idle} &= \text{appr}. [d := 0] . C^{dn} + \text{exit}. [d := 0] . C^{up} \\
C^{dn} &= [\text{reset } d, d \text{ with } \{d | d = 1\}, \{d | d \leq 5\}] \\
&\quad . (\text{lower}. [\text{reset } d, d \text{ with } \{\text{TRUE}\}] . C^{idle} \\
&\quad + \text{exit}. C^{dn} + \text{appr}. C^{dn}) \\
C^{up} &= [\text{reset } d, d \text{ with } \{d | d = 1\}, \{d | d \leq 5\}] \\
&\quad . (\text{raise}. [\text{reset } d, d \text{ with } \{\text{TRUE}\}] . C^{idle} \\
&\quad + \text{exit}. C^{up} + \text{appr}. [d := 0] . C^{dn})
\end{align*}
\]

The train gate controller is a parallel composition of the three processes starting in an empty environment. The channels appr, exit, lower, raise are made local to the process.

\[ (\emptyset, \nu\langle\text{appr, exit, raise, lower}\rangle(\text{Train} \mid \text{Gate} \mid \text{Con}\text{tr})) \]

It is also possible to define recursion variables of Train, Gate and Con\text{tr} by using the replication operator of \(\phi\)-calculus. We find the present approach of expressing recursion variables more readable.

Some observations about the specification are as follows:

- The variables are given specific values in the \(\phi\)-calculus environment. Any restrictions on the variables are added as flow constraints. Other process algebras allow their variables to initialize or jump to an arbitrary value satisfying a condition. For example, in HyPA specification, as the train crosses the 100\text{m} detector, \(x\) is updated according to the condition \(x ≤ -1400\). In \(\phi\)-calculus, \(x\) is updated to a particular value \(-1400\). To model that assignment of \(-1400\) to \(x\) can be delayed, we can make use of the delay action prefix as follows:
  - First introduce a dummy channel \(a\) in the specification.
  - Rewrite the recursion variable \(T^{past}\) as follows:
    \[
    T^{past} = [\text{reset } x \text{ with } \{x | 0 \leq x \leq 100\}] \\
    . [x = 100] . \text{exit}. \\
    . [y := 0] . [\text{reset } y \text{ with } \{y | y = 0\}] \\
    \delta. a. [\{(x, y) := (-1400, 52)\}] . T^{far}
    \]
- Put \(!a\) in parallel with the train gate controller specification. The train gate controller specification is recursive. Therefore we put infinitely many instances of the receive action \(\bar{a}\) in parallel with the specification.

The required process is as follows:

\[ (\emptyset, \nu\langle\text{appr, exit, raise, lower, }a\rangle(\text{Train} \mid \text{Gate} \mid \text{Con}\text{tr} \mid \!a)) \]
The variable $x$ varies with $\dot{x} = y$. We assign 0 to $y$ and reset the flow constraint of $y$, in order to make the environment with valuation $x : 100$ delayable.

The process term $\delta. a. \[ (x, y) := (-1400, 52) \] .T_{far}$ is arbitrarily delayable. It must discharge $\delta$ before the communication on channel $a$ and the assignment $\[ (x, y) := (-1400, 52) \]$ can become urgent.

This arrangement will have the same effect as initializing $x$ according to the predicate $x^+ \leq -1400$. However the latter approach is much more intuitive.

- As soon as the controller has sent the lower or raise signal to the gate, the flow constraints $\{d \mid d \leq 5\}$ needs to be removed from the environment, to allow the environment to delay further. The approach we follow is to remove both $\{d \mid d \leq 5\}$ and $\{\dot{d} \mid \dot{d} = 1\}$ and replace it by a constraint $\{TRUE\}$. 

9.2.1 Expanding the specification using transition rules:

There are no axioms in $\phi$-calculus. Therefore algebraic manipulation cannot be applied to simplify $\nu(\text{appr,exit,raise,lower})(\text{Train} \mid \text{Gate} \mid \text{Contr})$. We can study the behaviour of $\nu(\text{appr,exit,raise,lower})(\text{Train} \mid \text{Gate} \mid \text{Contr})$ starting in an empty environment by using the action and time transition rules of $\phi$-calculus.

The actual derivation comes out to be very lengthy. In order to condense it, we will often combine a number of action transition steps together. For example, $\tau, \tau, \tau$ implies that three silent steps are performed one after the other.

$\text{[}\ x = 0\text{]} \cdot \text{[reset} \ \dot{x} \ \text{with} \ \{\dot{x} \mid \dot{x} = 1\}\text{]}$ implies two environmental actions, first just a guard and second a reset take place one after the other.

Since a specific value needs to be given to environment variables, we choose the maximum speed, i.e. 52, and the minimum distance of the train from the gate, i.e. $-1400 \text{m}$. In time transitions, we also take the maximum delay of 5 seconds for the controller, in order to see how the specification behaves under extreme conditions.

Let $\vec{A}$ denote the channel vector $\langle \text{appr,exit,raise,lower} \rangle$.

\[
(\emptyset, \nu(\vec{A}))(\text{Train} \mid \text{Gate} \mid \text{Contr}) \xrightarrow{\tau, \tau, \tau, \tau} (\emptyset, \nu(\vec{A}))(\ [ (x, y) := (-1400, 52) ] \cdot T_{far} \mid C_{idle} \mid [ r := 90 ] \cdot G_{op})
\]
\[
\begin{bmatrix}
\{ \text{reset } d, \dot{d} \text{ with } \{ \dot{d} | d = 1 \}, \{ d | d \leq 5 \} \}
\end{bmatrix}
\]

\[
\begin{cases}
x : -1000, y : 52 \\
r : 90, d : 0 \\
\{ \{ x \mid -1000 \leq x \leq 0 \}, \\
\{ y \mid 40 \leq y \leq 52 \}, \\
\{ \dot{x} \mid \dot{x} = y \}, \\
\{ \dot{r} \mid \dot{r} = 0 \}, \\
\{ d \mid d = 1 \}, \\
\{ d \mid d \leq 5 \} \}
\end{cases}
\]

\[
\nu(\tilde{A}(\{ x = 0 \}, T_{\text{past}} | \\
\text{lower. } \{ \text{reset } d, \dot{d} \text{ with } \{ \text{TRUE} \} \}. C_{\text{idle}} \\
+ \text{exit. } C_{dn} + \text{appr. } C_{dn}) | \\
\text{lower. } G_{dn} + \text{raise. } G_{op})
\]

\[
\begin{bmatrix}
\{ \text{reset } d, \dot{d} \text{ with } \{ \text{TRUE} \} \}
\end{bmatrix}
\]

\[
\begin{cases}
x : -740, y : 52 \\
r : 90, d : 5 \\
\{ \{ x \mid -1000 \leq x \leq 0 \}, \\
\{ y \mid 40 \leq y \leq 52 \}, \\
\{ \dot{x} \mid \dot{x} = y \}, \\
\{ \dot{r} \mid \dot{r} = 0 \}, \\
\{ d \mid d = 1 \}, \\
\{ d \mid d \leq 5 \} \}
\end{cases}
\]

\[
\nu(\tilde{A}(\{ x = 0 \}, T_{\text{past}} | \\
\text{lower. } \{ \text{reset } d, \dot{d} \text{ with } \{ \text{TRUE} \} \}. C_{\text{idle}} \\
+ \text{exit. } C_{dn} + \text{appr. } C_{dn}) | \\
\text{lower. } G_{dn} + \text{raise. } G_{op})
\]

\[
\begin{bmatrix}
\{ \text{reset } d, \dot{d} \text{ with } \{ \text{TRUE} \} \}
\end{bmatrix}
\]

\[
\begin{cases}
x : -740, y : 52 \\
r : 90, d : 5 \\
\{ \{ x \mid -1000 \leq x \leq 0 \}, \\
\{ y \mid 40 \leq y \leq 52 \}, \\
\{ \dot{x} \mid \dot{x} = y \}, \\
\{ \dot{r} \mid \dot{r} = 0 \}, \\
\{ d \mid d = 1 \}, \\
\{ d \mid d \leq 5 \} \}
\end{cases}
\]

\[
\nu(\tilde{A}(\{ x = 0 \}, T_{\text{past}} | \\
\text{lower. } \{ \text{reset } d, \dot{d} \text{ with } \{ \text{TRUE} \} \}. C_{\text{idle}} \\
| G_{dn})
\]

\[
\begin{bmatrix}
\{ \text{reset } d, \dot{d} \text{ with } \{ \text{TRUE} \} \}
\end{bmatrix}
\]

\[
\begin{cases}
x : -740, y : 52 \\
r : 90, d : 5 \\
\{ \{ x \mid -1000 \leq x \leq 0 \}, \\
\{ y \mid 40 \leq y \leq 52 \}, \\
\{ \dot{x} \mid \dot{x} = y \}, \\
\{ \dot{r} \mid \dot{r} = 0 \}, \\
\{ \text{TRUE} \}
\end{cases}
\]

\[
\nu(\tilde{A}(\{ x = 0 \}, T_{\text{past}} | \\
C_{\text{idle}} | G_{dn})
\]

78
\[\begin{align*}
  &x(0.9) \cdot 730, \\
  &\begin{cases}
    x : 0, y : 52 \\
    r : 0, d : 5 \\
    \{x | -1000 \leq x \leq 1000\}, \\
    \{y | 40 \leq y \leq 52\}, \\
    \{\dot{x} | \dot{x} = y\}, \\
    \{r | \dot{r} = 0\}, \\
    \{r | r = 0\}, \\
    \{TRUE\}
  \end{cases}, \\
  &\nu(\bar{A})([x = 0].T_{\text{post}} \mid C_{\text{idle}}) \\
  &\nu(\bar{A})([x = 0].T_{\text{post}} \mid C_{\text{idle}}) \\
  &\nu(\bar{A})([x = 0].T_{\text{post}} \mid C_{\text{idle}}) \\
  &\nu(\bar{A})([x = 0].T_{\text{post}} \mid C_{\text{idle}})
\end{align*}\]
\( [(x,y):=(-1400,52), [d:=0] \)

\[
\begin{pmatrix}
  x : -1400, y : 52 \\
  r : 0, d : 0 \\
  \{ \{ x \mid 0 \leq x \leq 100 \} , \\
  \{ y \mid 40 \leq y \leq 52 \} , \\
  \{ \dot{x} \mid \dot{x} = y \} , \\
  \{ r \mid \dot{r} = 0 \} , \\
  \{ r \mid r = 0 \} , \\
  \{ \text{TRUE} \} \}
\end{pmatrix}
\]

\( \nu(\vec{A})(T_{far} \mid C^{up} \mid (\text{lower.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)

\( \nu(\vec{A})([x=1000].\text{appr.T}^{near} \mid C^{up} \mid (\text{raise.G}^{cl} + \text{raise.G}^{up})) \)
\[
\begin{align*}
\tau & \equiv (\text{raise | raise}) \\
\nu(\bar{A})([x = 1000].\text{appr}.T_{\text{near}} | \\
& (\text{reset } d, \dot{d} \text{ with } \{\text{TRUE}\} \cdot C_{\text{idle}})) \\
& + \text{exit}.C^{\text{up}} + \text{appr}.[d := 0]C^{\text{dn}}) \\
& | (\text{lower}.G^{\text{cl}} + \text{raise}.G^{\text{up}})) \\
\end{align*}
\]
At this stage, the first train has exited the gate, i.e. it has crossed the detector at 100m, from the gate. The controller has sent the raise signal to the gate and the gate is rising. At the same time a new train arrives. It reaches the \(-1000\) m detector and sends an appr signal to the gate. From this stage, we derive transitions for two possible scenarios.

**Scenario 1:** The controller takes 5 seconds before it sends lower to the gate. The gate is fully opened when it receives the lower signal.
The derivation is then repeated from step (A).

Scenario 2: The controller sends the signal immediately. The gate is not fully open when it receives the lower signal from the controller.

\[
\begin{align*}
\nu(\bar{A})([x = 0], T_{past} & | C_{idle} \\
& | ( [ r = 0 ] . G_{cl+} \\
& + exit. C_{dn} + appr. C_{dn} \\
& | ([ r = 90 ] . G_{up} + raise. G_{up} + lower. C_{dn}) )
\end{align*}
\]

\[
\begin{align*}
\nu(\bar{A})([x = 0], T_{past} & | C_{idle} \\
& | ( [ r = 0 ] . G_{cl+} \\
& + exit. C_{dn} + appr. C_{dn} \\
& + exit. C_{dn} + appr. C_{dn} )
\end{align*}
\]

\[
\begin{align*}
\nu(\bar{A})([x = 0], T_{past} & | C_{idle} \\
& | ( [ r = 0 ] . G_{cl+} \\
& + exit. C_{dn} + appr. C_{dn} \\
& + exit. C_{dn} + appr. C_{dn} )
\end{align*}
\]

The derivation is then repeated from step (A).
From this stage onwards we can find repeating patterns in the derivation of transitions.

We have used the transition rules to expand the train gate controller specification. While applying the transition rules, we have to make certain choices as we cannot cover all possible scenarios. For example there are infinitely many time transitions of duration less than or equal to five. We select a scenario with boundary conditions on variables. We choose the maximum train speed and the maximum possible delay by the controller. We obtain the same result as in HYPA, that the gate is closed when ever the train is within 500 m of the gate.

The SPHIN model checker is doing the same thing that we are trying to do manually here. That is, generating a state space and verifying it against a safety or liveness condition.
9.3 Train Gate Controller Specification in Hybrid Chi

The train gate controller process is specified as follows (the recursion variables $T_{\text{far}}$, $G_{\text{op}}$ and $C_{\text{idle}}$ are defined separately):

\[
\begin{aligned}
&\langle \quad \partial_{\text{Aia}} (v_{\{\text{appr,exit,raise,lower}\}} \\
&\quad (\text{Train} \parallel \text{Gate} \parallel \text{Controller} \\
&\quad ) \quad ), \\
&\quad \{x \mapsto -1400, y \mapsto -52, r \mapsto 90, d \mapsto 0, \text{time} \mapsto 0\}, \\
&\quad \{x, y, r, d\}, \emptyset, \emptyset, \\
&\quad \{\text{appr,exit,raise,lower}\}, \\
&\quad \{\text{Train} \mapsto T_{\text{far}}, \quad \\
&\quad \text{Gate} \mapsto G_{\text{op}}, \quad \\
&\quad \text{Controller} \mapsto C_{\text{idle}}\} \quad \rangle
\end{aligned}
\]

There are four environment variables in the specification. All are declared continuous. The set of action labels $A_{\text{ia}}$, includes send and receive action labels on channels appr, exit, raise and lower. The encapsulation operator $\partial_{\text{Aia}}$ blocks individual send and receive actions on these channels.

- $T_{\text{far}} = x \leq -1000 \land 48 \leq \dot{x} \leq 52 \land y = \dot{x}$
- $T_{\text{near}} = -1000 \leq x \leq 0 \land 40 \leq \dot{x} \leq 52 \land y = \dot{x}$
- $T_{\text{past}} = 0 \leq x \leq 100 \land 40 \leq \dot{x} \leq 52 \land y = \dot{x}$
- $G_{\text{op}} = \dot{r} = 0 \land r = 90 \parallel \text{[raise??]} \parallel G_{\text{op}}$
- $G_{\text{dn}} = r \geq 0 \land \dot{r} = -20$
- $G_{\text{up}} = r \leq 90 \land \dot{r} = 20$
- $G_{\text{cl}} = \dot{r} = 0 \land r = 0 \parallel \text{[lower??]} \parallel G_{\text{cl}}$

87
We note that in hybrid Chi, in addition to the send and receive actions, there are actions with labels \( \text{pass, ready}_{dn}, \text{ready}_{up} \). We mentioned earlier (in section 3.3), that delay predicates in hybrid Chi represent infinite behaviour and strong time determinism of hybrid Chi plays an important role in the termination of delays. Delays are placed in alternative composition with guards or delayable action predicates. Action with labels \( \text{pass, ready}_{dn}, \text{ready}_{up} \) are necessary in their respective places to terminate delays and resolve choices in alternative compositions. The \( \emptyset \) in action predicates, for example in \( \emptyset : \text{true} \triangleright \text{ready}_{dn} \), indicates that no variables are allowed to jump.

9.3.1 Simplifying \( \partial_{A_{ia}}(T_{far} \parallel G_{op} \parallel C_{idle}) \)

In hybrid Chi, the set of axioms given is small and is not enough to simplify \( \partial_{H}(T_{far} \parallel G_{op} \parallel C_{idle}) \). In [18], an elimination theorem for the parallel operator is given. We simplify the train gate controller system using this elimination theorem. At places, we also cancel impossible options (for example an action preceded by a false guard) using the semantic rules of hybrid Chi.

\[
\partial_{A_{ia}}(T_{far} \parallel G_{op} \parallel C_{idle}) \\
= \begin{cases}
\dot{y} = 0 \land r = 90 \\
\dot{x} \leq -1000 \land 0 \leq 48 \leq \dot{x} \leq 52 \land y = \dot{x} \\
[ x = -1000 \rightarrow ca(appr, c_0, x_0) ] \\
\partial_{A_{ia}}(\{ d \} : d = 0 \triangleright \tau ; C_{dn} \parallel G_{op} \parallel T_{near} ) \\
[ x = 1000 \rightarrow \delta \ldots (A) \\
(\text{where } \partial_{A_{ia}}(appr!! ; (C_{idle} \parallel G_{op} \parallel T_{near}) \equiv \delta))
\end{cases}
\]
\[ \partial_{\text{Aia}}(\{d\} : d = 0 \gg \tau ; C_{\text{dn}} \parallel G_{\text{op}} \parallel T_{\text{near}}) \]
\[ \Leftrightarrow -1000 \leq x \leq 0 \land 40 \leq \dot{x} \leq 52 \land y = \dot{x} \]
\[ \land \dot{r} = 0 \land r = 90 \]
\[ \land \{d\} : d = 0 \gg \tau ; \]
\[ \partial_{\text{Aia}}(C_{\text{dn}} \parallel G_{\text{op}} \parallel T_{\text{near}}) \ldots (B) \]
\[ \partial_{\text{Aia}}(C_{\text{dn}} \parallel G_{\text{op}} \parallel T_{\text{near}}) \]
\[ \Leftrightarrow -1000 \leq x \leq 0 \land 40 \leq \dot{x} \leq 52 \land y = \dot{x} \]
\[ \land \dot{r} = 0 \land r = 90 \]
\[ \land d = 1 \land d \leq 5 \]
\[ \land [\text{ca(lower, } e_0, x_0)] ; \]
\[ \partial_{\text{Aia}}(C_{\text{idle}} \parallel G_{\text{dn}} \parallel T_{\text{near}}) \ldots (C) \]
\[ \partial_{\text{Aia}}(C_{\text{idle}} \parallel G_{\text{cl}} \parallel T_{\text{near}}) \]
\[ \Leftrightarrow -1000 \leq x \leq 0 \land 40 \leq \dot{x} \leq 52 \land y = \dot{x} \]
\[ \land \dot{r} = -20 \land r \geq 0 \]
\[ \land r = 0 \rightarrow (\emptyset : \text{true} \gg \text{ready}\_{\text{dn}} ; \]
\[ \partial_{\text{Aia}}(C_{\text{idle}} \parallel G_{\text{cl}} \parallel T_{\text{near}}) \ldots (D) \]
\[ \partial_{\text{Aia}}(C_{\text{idle}} \parallel G_{\text{cl}} \parallel T_{\text{past}}) \]
\[ \Leftrightarrow 0 \leq x \leq 100 \land 40 \leq \dot{x} \leq 52 \land y = \dot{x} \]
\[ \land r = 0 \land \dot{r} = 0 \]
\[ \land [x = 100 \rightarrow \text{ca(exit, } e_0, x_0)] ; \]
\[ \partial_{\text{Aia}}(\{d\} : d = 0 \gg \tau ; C_{\text{up}} \parallel G_{\text{cl}} \parallel \{x, y\} : x \leq -1400 \gg \tau ; T_{\text{far}}) \ldots (F) \]
\[ \partial_{\text{Aia}}(\{d\} : d = 0 \gg \tau ; C_{\text{up}} \parallel G_{\text{cl}} \parallel \{x, y\} : x \leq -1400 \gg \tau ; T_{\text{far}}) \]
\[ \Leftrightarrow \dot{r} = 0 \land r = 0 \]
\[ \land \{d\} : d = 0 \gg \tau ; \]
\[ \partial_{\text{Aia}}(C_{\text{up}} \parallel G_{\text{cl}} \parallel \{x, y\} : x \leq -1400 \gg \tau ; T_{\text{far}}) \]
\[ \land \{x, y\} : x \leq -1400 \gg \tau ; \]
\[ \partial_{\text{Aia}}(\{d\} : d = 0 \gg \tau ; C_{\text{up}} \parallel G_{\text{cl}} \parallel T_{\text{far}}) \ldots (G) \]
\[ \partial_{\text{Aia}}(\{d\} : d = 0 \gg \tau ; C_{\text{up}}) \]
\[ \parallel G^{\text{cl}} \parallel T^{\text{far}} \]
\[ \Rightarrow \dot{r} = 0 \land r = 0 \]
\[ \parallel x \leq -1400 \land 48 \leq \dot{x} \land y = \dot{x} \leq 52 \]
\[ \parallel \{d\} : d = 0 \gg \tau ; \]
\[ \partial_{\text{Aia}}(C_{\text{up}} \parallel G^{\text{cl}} \parallel T^{\text{far}}) \ldots (H) \]

\[ \partial_{\text{Aia}}(C_{\text{up}} \parallel G^{\text{cl}} \parallel T^{\text{far}}) \]
\[ \parallel \{x, y\} : x \leq -1400 \gg \tau ; \]
\[ \Rightarrow \dot{r} = 0 \land r = 0 \]
\[ \parallel d \leq 5 \land \dot{d} = 1 \]
\[ \parallel \{ ca(\text{raise}, e_0, x_0) \} ; \partial_{\text{Aia}}(C^{\text{idle}} \parallel G^{\text{up}} \parallel \{x, y\} : x \leq -1400 \gg \tau ; T^{\text{far}}) \]
\[ \parallel \{x, y\} : x \leq -1400 \gg \tau ; \]
\[ \partial_{\text{Aia}}(C^{\text{idle}} \parallel G^{\text{up}} \parallel T^{\text{far}}) \ldots (I) \]

\[ \partial_{\text{Aia}}(C^{\text{idle}} \parallel G^{\text{up}} \parallel T^{\text{far}}) \]
\[ \parallel \{x, y\} : x \leq -1400 \gg \tau ; \]
\[ \Rightarrow r \leq 90 \land \dot{r} = 20 \]
\[ \parallel \{x, y\} : x \leq -1400 \gg \tau ; \]
\[ \partial_{\text{Aia}}(C^{\text{idle}} \parallel G^{\text{up}} \parallel T^{\text{far}}) \ldots (J) \]

\[ \partial_{\text{Aia}}(C^{\text{idle}} \parallel G^{\text{up}} \parallel T^{\text{far}}) \]
\[ \parallel \{x, y\} : x \leq -1400 \gg \tau ; \]
\[ \Rightarrow r \leq 90 \land \dot{r} = 20 \]
\[ \parallel \{x, y\} : x = -1000 \rightarrow ca(\text{appr}, e_0, x_0) \} ; \]
\[ \partial_{\text{Aia}}(\{d\} : d = 0 \gg \tau ; G^{\text{dn}} \parallel G^{\text{up}} \parallel T^{\text{near}}) \]
\[ \parallel \{x, y\} : x = -1400 \gg \tau ; \]
\[ \partial_{\text{Aia}}(C^{\text{idle}} \parallel G^{\text{op}} \parallel T^{\text{far}}) \ldots (K) \]

(See relation (A) for \( \partial_{\text{Aia}}(C^{\text{idle}} \parallel G^{\text{op}} \parallel T^{\text{far}}) \))
We find the elimination theorem useful in simplifying the train gate controller specification. As mentioned before, at different stages we make use of the operational semantics to resolve choices in alternative composition. A more comprehensive set of axioms of hybrid Chi together with the elimination theorem can make the whole procedure algebraic. A weaker notion of bisimulation than state-less bisimulation will also be needed to incorporate axioms for real analysis of systems, similar to the ones present in HyPA and ACPhs.

A point to be noted here is that the present elimination theorem cannot eliminate parallel composition from all hybrid Chi processes. Therefore work in that direction needs to be done, in addition to adding more axioms, before algebraic reasoning can be automatized in hybrid Chi.

9.4 Train Gate Controller Specification in ACPhs

The components train, gate and controller are defined below.
Trains = (x ≤ −1400) \land T_{far}

T_{far} = (x ≤ −1000 \land 48 ≤ \dot{x} ≤ 52) \land_{(x)} \sigma_{rel}((x = −1000) \rightarrow (x^{*} = \star x \land \dot{x}^{*} = \star \dot{x}) \land_{s_{1}}(\lnot \text{appr}) \land T_{near})

T_{near} = (−1000 ≤ x ≤ 0 \land 40 ≤ \dot{x} ≤ 52) \land_{(x)} \sigma_{rel}((x = 0) \rightarrow (x^{*} = \star x \land \dot{x}^{*} = \star \dot{x}) \land \lnot \text{pass} \land T_{past})

T_{past} = (0 ≤ x ≤ 100 \land 40 ≤ \dot{x} ≤ 52) \land_{(x)} \sigma_{rel}((x = 0) \rightarrow (x^{*} ≤ −1400) \land s_{1}(\lnot \text{exit}) \land T_{far})

Gate = (r = 90) \lor G^{op},

G^{op} = (r = 90 \land \dot{r} = 0) \land_{(r)}

σ_{rel}((r^{*} = \star r) \land r_{2}(\lnot \text{lower}) \land G^{dn})
+ σ_{rel}((r^{*} = \star r) \land r_{2}(\text{raise}) \land G^{op}))

G^{dn} = (0 ≤ r ≤ 90 \land \dot{r} = −20) \land_{(r)}

σ_{rel}((r^{*} = \star r) \land r_{2}(\text{lower}) \land G^{dn})
+ σ_{rel}((r^{*} = \star r) \land r_{2}(\text{raise}) \land G^{op}) +
σ_{rel}((r = 0) \rightarrow ((r^{*} = \star r) \land (\lnot \text{ready}_{dn}) \land G^{cl}))

G^{up} = (0 ≤ r ≤ 90 \land \dot{r} = 20) \land_{(r)}

σ_{rel}((r^{*} = \star r) \land r_{2}(\text{lower}) \land G^{dn})
+ σ_{rel}((r^{*} = \star r) \land r_{2}(\text{raise}) \land G^{op}) +
σ_{rel}((r = 90) \rightarrow ((r^{*} = \star r) \land (\lnot \text{ready}_{up}) \land G^{cl}))

G^{cl} = (r = 0 \land \dot{r} = 0) \land_{(r)}

σ_{rel}((r^{*} = \star r) \land r_{2}(\text{lower}) \land G^{cl})
+ σ_{rel}((r^{*} = \star r) \land r_{2}(\text{raise}) \land G^{up}))
\[
\begin{align*}
\text{Contr} & \quad = (d = 0) \wedge C^{\text{idle}} \\
C^{\text{idle}} & \quad = (\dot{d} = 0)^{\bullet} r_{(d)} \\
& \quad + \sigma^{\bullet}_{\text{rel}}((d = 0)^{\bullet} r_{1}(\text{appr}) \cdot C^{\downarrow}) + \sigma^{\bullet}_{\text{rel}}((d = 0)^{\bullet} r_{1}(\text{exit}) \cdot C^{\uparrow}) \\
C^{\uparrow} & \quad = (0 \leq d \leq 5 \wedge \dot{d} = 1)^{\bullet} r_{(d)} \\
& \quad + \sigma^{\bullet}_{\text{rel}}((d = 0)^{\bullet} r_{1}(\text{appr}) \cdot C^{\downarrow}) + \sigma^{\bullet}_{\text{rel}}((d = 0)^{\bullet} r_{1}(\text{exit}) \cdot C^{\uparrow}) + \\& \quad + \sigma^{\bullet}_{\text{rel}}((d = \bullet), s_{1}(\text{lower}) \cdot C^{\text{idle}}) \\
C^{\downarrow} & \quad = (0 \leq d \leq 5 \wedge \dot{d} = 1)^{\bullet} r_{(d)} \\
& \quad + \sigma^{\bullet}_{\text{rel}}((d = \bullet), s_{1}(\text{lower}) \cdot C^{\downarrow}) + \sigma^{\bullet}_{\text{rel}}((d = 0)^{\bullet} r_{1}(\text{exit}) \cdot C^{\downarrow}) + \\& \quad + \sigma^{\bullet}_{\text{rel}}((d = \bullet), s_{1}(\text{lower}) \cdot C^{\text{idle}}) \\
\end{align*}
\]

\[
\gamma(s(d), r_{i}(d)) = c_{i}(d), \quad \text{where } d \in \{\text{appr, exit, raise, lower}\} \\
i \in \{1, 2\}
\]

The gate controller system is specified by:

\[
\partial_{H}(\text{Trains} \parallel \text{Cntr} \parallel \text{Gate}),
\]

where

\[
H = \{s(d) \mid d \in \{\text{appr, exit, raise, lower}\}\} \cup \{r(d) \mid d \in \{\text{appr, exit, raise, lower}\}\}
\]

All variables \(x, r, d\) vary smoothly during delays. The specification of \(ACP_{\text{hs}}^{\text{srt}}\) requires actions \(\text{pass}, \text{ready}_{\text{dn}}, \text{and } \text{ready}_{\text{up}}\), because in \(ACP_{\text{hs}}^{\text{srt}}\), an evolution operator offering one solution to variable trajectories during a delay, cannot be followed by another evolution operator offering a different variable evolution. Consider for example \(C^{\downarrow}\). If the gate is going down with the rate \(-20\), then when the gate is fully closed, in order to change the rate of going down to 0, an action \(\text{ready}_{\text{dn}}\) is required. Note in \((r = \bullet r) \cdot \text{ready}_{\text{dn}}\), the derivative of \(r\) is not required to remain constant. Therefore, \(\dot{r}\) can jump arbitrarily. The derivative \(\dot{r}\) will jump such that the evolution proposition following action \(\text{ready}_{\text{dn}}\) becomes true.

An action is required for mode switching in \(ACP_{\text{hs}}^{\text{srt}}\).

### 9.4.1 Simplifying \(\partial_{H}(\text{Trains} \parallel \text{Cntr} \parallel \text{Gate})\)

We introduce recursion variables \(X_{0}, X_{1}, \ldots, X_{11}\).

Let \(V = \{x, r, d\}\).
We apply axioms and lifting rules of $ACP_{hs}$ in the simplification of $\partial_H(Trains \parallel Cntr \parallel Gate)$.

\[
X_0 = \partial_H(Trains \parallel Cntr \parallel Gate) = (x \leq -1400 \land d = 0 \land r = 90) \cdot X_1
\]
\[
X_1 = \partial_H(T_{far} \parallel C_{idle} \parallel G_{op}) = (x \leq -1000 \land 48 \leq \dot{x} \leq 52 \land \dot{d} = 0 \land r = 90 \land \dot{r} = 0) \cdot \nu_\forall
\]
\[
(\partial_H(T_{far} \parallel C_{idle} \parallel G_{op}))
\]

Let $\phi_1$ denote proposition $(x \leq -1000 \land 48 \leq \dot{x} \leq 52 \land \dot{d} = 0 \land r = 90 \land \dot{r} = 0)$.

For a set of variables $V$, $C_V$ denotes the transition proposition $v^* = \cdot \nu v \land \nu^* = \cdot \dot{v}$, for every $v \in V$.

\[
X_1 = \phi_1 \circ V \left( \int_{x \in [400/52, \infty]} \sigma_\nu^d(\partial_H(((x = -1000) \dashv (C_{x}) \cdot s_1(\text{appr}) \cdot T_{near} \parallel \nu \cdot r_1(\text{appr}) \cdot C_{dn} + (d^* = 0) \cdot \nu \cdot r_1(\text{exit}) \cdot C_{up})) \parallel G_{op})))
\]
\[
= \phi_1 \circ V \left( \int_{x \in [400/52, \infty]} \sigma_\nu^t((x = -1000) \dashv (C_{x,r} \land d^* = 0) \cdot \nu \cdot c_1(\text{appr}) \cdot \partial_H(T_{near} \parallel C_{dn} \parallel G_{op})))
\]
\]

Let $\phi_2$ denote proposition $(-1000 \leq x \leq 0 \land 40 \leq \dot{x} \leq 52 \land \dot{d} = 1 \land r = 90 \land \dot{r} = 0)$.

\[
X_2 = \phi_2 \circ V \left( \int_{x \in [0, 5]} \sigma_\nu^t((d^* = 0 \land r^* = \cdot \nu \cdot r \land C_{x}))
\]
\[
\nu \cdot c_2(\text{lower}) \cdot \partial_H(T_{near} \parallel C_{idle} \parallel G_{dn}))
\]
\]

Let $\phi_3$ denote proposition $(-1000 \leq x \leq 0 \land 40 \leq \dot{x} \leq 52 \land \dot{d} = 0 \land r \geq 0 \land \dot{r} = -20)$.

\[
X_3 = \phi_3 \circ V \left( \int_{x \in [0, 20]} \sigma_\nu^{90/20}((r = 0) \dashv (r^* = \cdot \nu \cdot r \land C_{d,x}))
\]
\[
\nu \cdot \text{ready}_{dn} \cdot \partial_H(G_{cl} \parallel C_{idle} \parallel T_{near}))
\]

Let $t \in [0, 5]$ denote the time taken by the controller to send the lower signal in $X_2$. Then the total time elapsed since the train crossed the 1000m detector is $t + 90/20$. 90/20 is the time required by the gate to close fully. Let $t' = t + r/20$, where $r$ denotes the angle of the gate.

\[
X_4^{t'} = \partial_H(G_{cl} \parallel C_{idle} \parallel T_{near})^{t'}
\]

Let $\phi_4$ denote proposition $(-1000 \leq x \leq 0 \land 40 \leq \dot{x} \leq 52 \land \dot{d} = 0 \land r = 0 \land \dot{r} = 0)$. 

94
\[ X_4' = \phi_4 \cdot \nu_f \left( \int_{f \in [1000/52-\tau, 1000/40-\tau]} \sigma_{rel}^f \right) \\
(x = 0) \rightarrow (C_{(x, r, d)} \cdot \mathbf{pass} \cdot \partial_H (T^{pass} \parallel C^{idle} \parallel G^{cl}))) \]

\[ X_5 = \partial_H (T^{pass} \parallel C^{idle} \parallel G^{cl}) \]

\[ X_6 = \phi_6 \cdot \nu_f \left( \int_{f \in [100/52, 100/40]} \sigma_{rel}^f \right) \\
(x = 100) \rightarrow (C_{r} \wedge x^* \leq -1400 \wedge d^* = 0) \equiv c_1 (exit) \cdot \\
\partial_H (T^{T_{far}} \parallel C^{up} \parallel G^{cl})) \]

where \( \phi_6 \) denotes \( 0 \leq x \leq 100 \wedge 40 \leq \dot{x} \leq 52 \wedge \dot{d} = 0 \wedge r = 0 \wedge \dot{r} = 0 \).

Let \( \phi_6 \) denote \( x \leq -1000 \wedge 48 \leq \dot{x} \leq 52 \wedge \dot{d} = 1 \wedge d \leq 5 \wedge r = 0 \wedge \dot{r} = 0 \).

\[ X_7 = \partial_H (T^{T_{far}} \parallel C^{idle} \parallel G^{up}) \]

Let \( \phi_7 \) denote \( x \leq -1000 \wedge 48 \leq \dot{x} \leq 52 \wedge \dot{d} = 0 \wedge d = 0 \wedge 0 \leq r \leq 90 \wedge \dot{r} = 20 \).

Let \( t \) denote the time taken by the controller to send the raise signal. The gate requires 4.5 seconds to fully open. A new train requires at least \( 400/52 = 7.69 \) seconds to reach the 1000m detector. If time taken by the controller \( t \) is greater than approx 3.19, then a new train may arrive and send an approach signal before the gate is fully open.

The minimum time required by a new train to send an appr signal to the controller, after the controller has sent the raise signal to the gate is, \((400/52) - t\).

\[ X_7 = \phi_7 \cdot \nu_f \left( \int_{f \in [400/52-\tau, 90/20]} \sigma_{rel}^f \right) (x = -1000) \rightarrow \\
(C_{(x, r)} \wedge d^* = 0) \equiv c_1 (appr) \cdot \partial_H (T^{near} \parallel C^{dn} \parallel G^{up}) \\
+ \sigma_{rel}^{90/20} (r = 90) \rightarrow (C_{(x, d)} \wedge r^* = r) \equiv c_2 (ready_{up}) \cdot \partial_H (T^{T_{far}} \parallel C^{idle} \parallel G^{up})) \]

\[ X_9 = \partial_H (T^{T_{far}} \parallel C^{idle} \parallel G^{op}) = X_1 \]

\[ X_8 = \partial_H (T^{near} \parallel C^{dn} \parallel G^{up}) \]

Let \( \phi_8 \) denote proposition \((-1000 \leq x \leq 0 \wedge 40 \leq \dot{x} \leq 52 \wedge d \leq 5 \wedge \dot{d} = 1 \wedge 0 \leq r \leq 90 \wedge \dot{r} = 20\)).

\( X_8 \) denotes the case when a new train arrives before the gate was fully open. Let \( t \) denote the delay elapsed since the controller sent the last raise signal to the gate. If \( t < 90/20 \), i.e. time elapsed since the last raise signal was sent is
less than 4.5 seconds, then the controller may issue a lower signal before the
gate is fully opened.

\[
X_8 = \phi_8 \circ_v \int_{t \in [0, 90/20 - t]} \sigma_{rel}^* \left( (C_{(v)} \land d^* = 0 \land r^* = \bullet r) \uparrow c_2(\text{lower}) \right)
\]
\[
\cdot \partial_H(T_{near} \parallel C_{idle} \parallel G_{dn})
\]
\[
+ \sigma_{rel}^{90/20 - t} \left( (r = 90) \rightarrow (C_{(z,d)} \land r^* = \bullet r) \uparrow \text{ready}_{up} \right)
\]
\[
\cdot \partial_H(T_{near} \parallel C_{dn} \parallel G_{op})
\]

\[
X_{10} = \partial_H(G_{op} \parallel T_{near} \parallel C_{dn})
\]
\[
= X_2
\]

\[
X_{11} = \partial_H(G_{dn} \parallel T_{near} \parallel C_{idle})
\]
\[
= X_3
\]

\text{ACP}_{hs} \text{ has a comprehensive set of axioms and lifting rules. Work needs to be done towards tool support in automatically applying algebraic reasoning to } \text{ACP}^{\text{port}}_{hs} \text{ specifications.}

10 Conclusions

Summary

We have done a study of process algebras for hybrid systems. We included process algebras HyPA, \(\phi\)-calculus, hybrid Chi and \(\text{ACP}^{\text{port}}_{hs}\) in our study. We tried to build a general list of requirements for hybrid system specifications. We studied each hybrid algebra process algebra and identified their operators and constructs that fulfill a general requirement as well as features specific to each process algebra. We discussed the discontinuities that essentially appear in environment variables in different stages of a hybrid system. We gave special attention towards how parallel processes are allowed to behave in each process algebra. Tools available for each hybrid process algebras were also explored. Finally, we did a case study of a train gate controller in all hybrid process algebras.

We did not include one commonly known process algebra for hybrid systems in our study, i.e. hybrid CSP. Hybrid CSP (see [10]) has a specification oriented semantics. Definitions of some basic operators such as \textit{chop} operator (denoted by \(\cdot\)) and \textit{product} operator (denoted by \(\times\)) are also given in [10]. These definitions are then used in defining the semantics of more complicated operators such as sequential compositions and disrupts. We are not clear about the meaning of these basic operators in hybrid CSP. It is unclear to us as to what happens to the environment variables in \(P \sim Q\), as \(Q\) takes over.

There has been a lot of research on how to improve the formal representation of hybrid systems. Apart from process algebras, a lot of work has been done on modelling of hybrid system through hybrid automata. We find that among process algebras studied, each process algebra can express a wide variety of
hybrid systems. There are pros and cons for the approach followed by each process algebra which are discussed below:

- **HyPA:**
  **Pros:**
  - HyPA has a relatively small number of operators and a simpler semantics. Therefore it is easier to use than other process algebras.
  - HyPA has a large set of axioms for algebraic manipulation. Two axioms for deriving results about environment variables evolution are also present.
  - Work has been done on automatic linearization of HyPA specifications
  - A simulator for HyPA specifications is available.
  **Cons:**
  - System invariants cannot be enforced in hHyPA specifications.

- **φ-calculus:**
  **Pros:**
  - φ-calculus also has a small number of operators and a simple semantics. No constructs to enforce delay. Delay constraints are part of the environment.
  - Local variables and local channels can be declared
  - One can easily model reconfigurable processes.
  - A hybrid model checker SPHIN available for model checking φ−calculus specifications.
  **Cons:**
  - Like HyPA, system invariants cannot be enforced in φ-calculus specifications.
  - The use of delay action prefix is not very intuitive.
  - Variables must be initialized to a single value instead of just giving an initial condition.

- **Hybrid chi**
  **Pros:**
  - Hybrid Chi has a large number of operators. The process instantiation enables reuse of processes. Scoping operators enable modular design of large specifications.
Hybrid Chi evolved from discrete event and timed chi. Therefore it has quite a number of users. A large number of case studies have been done in hybrid Chi.

A simulator for hybrid Chi specifications is available.

Formal translation of a subset of hybrid Chi into I/O hybrid automata has been done. A tool for automatic conversion of hybrid Chi models into hybrid automata is available.

Cons:

- It has very detailed and complicated semantics and a very large number of operators.
- A small set of axioms is given which is not enough for simplification of specifications.

ACP_{hs}

Pros:

- ACP_{hs} has a large set of axioms for algebraic manipulation. The axioms and lifting rules for deriving results about environment variables evolution are very useful in simplification of linear specifications.
- ACP_{hs} incorporates innovative theoretical ideas of practical application that others do not have. For example, discontinuity relations and localization operator.

Cons:

- the discontinuity operator makes the semantics of parallelism complicated.
- No simulation or linearization tools available for ACP_{hs}.

Acknowledgements We would like to thank Michel Reniers, William Rounds, Pieter Cuijpers, Ka Lok Man and Ramon.R.H. Schiffelers for their kind cooperation in helping us understand their research works. Michel Reniers, Ramon.R.H. Schiffelers, Hosung Song are also thanked for providing us the tools.

I would like to thank my supervisor Jos Baeten for taking a keen interest in this project and his excellent guidance throughout its term.

References


11 Appendix

11.1 PROMELA-Hybrid code for the train gate controller specification

/*
 * Train-Gate-Controller example for SPHIN
 */

#define DELAY 5 /* controller delay */
#define hskip (1+1)

/* "skip" can't be used in a hybrid statement */

/* channel definition for synchronization */
chan app = [0] of {int};
chan exit = [0] of {int};
chan raise = [0] of {int};
chan lower = [0] of {int};

/* declared globally to allow monitor process to check the values. */
analog x = { [-1400,-1400], [48,52], [-1000] }; analog g
= [90,90], [0,0], [90,90];
analog t = { [0,0], [0,0], [0,0] };

active proctype train() {
  far: app!0 when { x in [-1000,-1000] } reset { x = { , [40,52], [0] } };
  Near: hskip when { x in [0,0] } reset { x = { , [40,52], [100] } };
  past: atomic { exit!0 when { x in [100,100] } reset { x = { [-1400,-1400],
                      [48,52], [-1000] } }; goto far } }

active proctype gate() { open:
  do
    :: raise?_:
      :: atomic { lower?_ when {} reset { g = { , [-20,-20], [0] } };
        goto lowering }
  od;

  lowering:
  do
    :: lower?_
      :: atomic { raise?_ when {} reset { g = { , [20,20], [90] };
           goto raising }
    :: atomic { hskip when { g in [0,0] } reset { g = { , [0,0], [0,0] } };
         goto closed }
  od;


101
raising:
do:: raise?_
:: atomic { lower?_ when {} reset { g = { , [-20,-20], [0,] } }; goto lowering }
:: atomic { hskip when { g in [90,90] } reset { g = { , [0,0], [90,90] } }; goto open }
od;
closed:
do:: lower?_
:: atomic { raise?_ when {} reset { g = { , [20,20], [,90] } }; goto raising }
od
}
active proctype controller() { idle:
do:: atomic { app?_ when {} reset { t = { [0,0], [1,1], [,DELAY] } }; goto about_to_lower }
:: atomic { exit?_ when {} reset { t = { [0,0], [1,1], [,DELAY] } }; goto about_to_raise }
od;
about_to_lower:
do:: atomic { exit?_ when {} reset { t = { [0,0], [1,1], [,DELAY] } }; goto about_to_raise }
:: atomic { lower!0 when {} reset { t = { , [0,0], } }; goto idle }
od;
about_to_raise:
do:: atomic { app?_ when {} reset { t = { [0,0], [1,1], [,DELAY] } }; goto about_to_lower }
:: atomic { raise!0 when {} reset { t = { , [0,0], } }; goto idle }
od
}
#define Dmin -350
active proctype monitor() {
assert(false) when {x in [Dmin,100], g in (0,)} reset {}
}
11.2 A syntactical comparison of hybrid process algebras
<table>
<thead>
<tr>
<th>1. Empty process</th>
<th>doesn’t exist</th>
<th>$\epsilon$</th>
<th>skip- a syntactic extension</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>an abbreviation for $\emptyset : \text{true} \gg \tau$</td>
</tr>
<tr>
<td>2. Deadlock</td>
<td>$\delta$-undelayable deadlock</td>
<td>$\delta$-undelayable deadlock</td>
<td>$\delta$-undelayable deadlock</td>
</tr>
<tr>
<td></td>
<td>$\delta$-undelayable deadlock</td>
<td>$\delta$-undelayable deadlock</td>
<td>0- delayable deadlock</td>
</tr>
<tr>
<td>3. Inconsistent Process</td>
<td>$\perp$</td>
<td>doesn’t exist</td>
<td>$\perp$</td>
</tr>
<tr>
<td>4. Actions</td>
<td>$\bar{a}$</td>
<td>$a$</td>
<td>$\emptyset$: True $\gg l_a$, $l_a$ a is the action label, atomic, undelayable</td>
</tr>
<tr>
<td></td>
<td>atomic, undelayable</td>
<td>atomic, undelayable</td>
<td>$J, L, \dot{C}$ change arbitrarily</td>
</tr>
<tr>
<td></td>
<td>state changes arbitrarily</td>
<td>no change in state</td>
<td>No change in state</td>
</tr>
<tr>
<td>5. Communication actions</td>
<td>same as actions. A partial comm function ($\gamma$) given, that defines which actions communicate</td>
<td>same as actions. A partial comm function ($\gamma$) given, that defines which actions communicate</td>
<td>$\text{ch} ? ? x_n$: receive values from ch, store in var $x_1, \ldots, x_n$</td>
</tr>
<tr>
<td></td>
<td>No data exchanged</td>
<td>No data exchanged</td>
<td>$\text{ch}! e_n$: send values of expr. $e_1, \ldots, e_n$ on ch.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Values can be exchanged in messages</td>
</tr>
<tr>
<td>6. Flow clauses \ Evolution Op.</td>
<td>$\phi \wedge V \ P$</td>
<td>$c$ of the form $(V \mid P_f)$</td>
<td>$u$</td>
</tr>
<tr>
<td></td>
<td>$\phi$ – a delay predicate</td>
<td>$P_f$ – a delay predicate</td>
<td>simply an algebraic, differential, equation or inequality</td>
</tr>
<tr>
<td></td>
<td>variables $\in V$ remain continuously differentiable during the delay</td>
<td>variables $\in V$ cannot jump at the start of delay</td>
<td>doesn’t exist</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$a(\vec{x})$: receive references from a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\bar{a}(\vec{x})$: send references $\vec{x}$ on a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Values and names can be exchanged</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$a$ – link name, $\vec{x}$ – vector of values or names</td>
</tr>
<tr>
<td></td>
<td>$ACP_{hs}$</td>
<td>HyPA</td>
<td>Hybrid Chi</td>
</tr>
<tr>
<td>-----</td>
<td>---------------------</td>
<td>--------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>7</td>
<td>Instantaneous</td>
<td>$\chi \triangleright P$</td>
<td>$d \triangleright P$</td>
</tr>
<tr>
<td></td>
<td>modifications</td>
<td>$\chi$ — a transition predicate</td>
<td>$d$ — of the form $[V</td>
</tr>
<tr>
<td></td>
<td>in terms of old and</td>
<td>in $ACP_{hs}$</td>
<td>Variable $\in V$ allowed to jump</td>
</tr>
<tr>
<td></td>
<td>new values of</td>
<td>$P_t$ — a jump predicate like $\chi$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>variables</td>
<td>$\in ACP_{hs}$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Delay operator</td>
<td>$\sigma_{rel}^r (P)$</td>
<td>doesn’t exist</td>
</tr>
<tr>
<td></td>
<td>$\sigma_{rel}^r (P)$</td>
<td>$\sigma_{rel}^r$ — relative delay</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$r$ — delay duration</td>
<td>operator</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>doesn’t exist</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Time out</td>
<td>$\nu_{rel} (P)$</td>
<td>Doesn’t exist</td>
</tr>
<tr>
<td></td>
<td>$\nu_{rel} (P)$</td>
<td>$\nu_{rel}$ — relative time out</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\nu_{rel} (P)$ cannot delay initially</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>must perform an action first</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Signal emission</td>
<td>$\psi \wedge P$</td>
<td>Doesn’t exist</td>
</tr>
<tr>
<td></td>
<td>$\psi$ — a predicate on variables and their derivatives</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AC$P_{hs}^{art}$</td>
<td>HyPA</td>
<td>Hybrid Chi</td>
</tr>
<tr>
<td>---</td>
<td>-----------------</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>11. Guards</td>
<td>$\psi \rightarrow P$&lt;br&gt;$\psi$ - a predicate on variables and their derivatives&lt;br&gt;Undelayable guard</td>
<td>Doesn’t exist</td>
<td>$b \rightarrow P$&lt;br&gt;$b$ - a predicate on variables and dotted variables&lt;br&gt;Delayable guard</td>
</tr>
<tr>
<td>12. Disrupt Operator</td>
<td>Doesn’t exist</td>
<td>$P \triangleright Q$ - $Q$ can interrupt $P$ anytime during its execution&lt;br&gt;$\triangleright$ - Left disrupt&lt;br&gt;$P \triangleright Q$ - First a part of $P$ is executed&lt;br&gt;After that, $P \triangleright$ behaves as normal disrupt.</td>
<td>Doesn’t exist</td>
</tr>
<tr>
<td>13. Alternative Composition</td>
<td>$P + Q$&lt;br&gt;Weak Time deterministic</td>
<td>$P \oplus Q$&lt;br&gt;Time non-deterministic&lt;br&gt;a choice b/w $P$ and $Q$ is resolved immediately</td>
<td>$P \parallel Q$&lt;br&gt;Strongly Time deterministic&lt;br&gt;the choice is resolved upon execution of first action</td>
</tr>
<tr>
<td>14. Action Prefixing</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
</tr>
<tr>
<td></td>
<td>$ACP_{\phi}$</td>
<td>HyPA</td>
<td>Hybrid Chi</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>15. Concretions / abstractions</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
</tr>
</tbody>
</table>
| 16. Parallelism | $P \parallel Q$ - parallel merge  
$P \parallel Q$ - left merge  
$P | Q$ - Communication merge | $P \parallel Q$ - parallel merge  
$P \parallel Q$ - left merge  
$P | Q$ - Communication merge | $P \parallel Q$  
Parallel Operator | $P \parallel Q$  
Parallel Operator |
| 17. Encapsulation Operator | $\partial_H P$  
$H$ - a set of actions  
all $a \in H$ are blocked | $\partial_H P$  
$H$ - a set of actions  
all $a \in H$ are blocked | $\partial_H P$  
$H$ - a set of actions  
all $a \in H$ are blocked | Doesn’t exist |
| 18. Local Variable Operator | Doesn’t exist | Doesn’t exist | $\nu x P$  
declares a variable $x$ which is local to $P$ |
| 18. Local Channel Operator | Doesn’t exist | Doesn’t exist | $\nu a P$  
declares a channel $a$ which is local to $P$ |
<table>
<thead>
<tr>
<th></th>
<th>ACP&lt;sub&gt;hs&lt;/sub&gt;</th>
<th>HyPA</th>
<th>Hybrid Chi</th>
<th>φ-Calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>18. Local Recursive definition operator</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
<td>([R \mathcal{R} \mid p]) \newline recursion scope operator \newline declares a recursive definition \newline (R) local to (p)</td>
<td>Doesn’t exist</td>
</tr>
<tr>
<td>19. Replication</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
<td>!(P) – stands for infinitely many instances of (P) running in parallel</td>
</tr>
<tr>
<td>20. Urgent communication</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
<td>(v_H(P)) \newline (H) – a set of channels \newline makes all channels in (H) urgent \newline a send or receive on an urgent channel must communicate as soon as possible</td>
<td>Doesn’t exist</td>
</tr>
<tr>
<td>21. Integration</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
</tr>
<tr>
<td>22. Localization operator</td>
<td>(x \nabla P) \newline localizes discontinuities of (x)</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
<td>Doesn’t exist</td>
</tr>
</tbody>
</table>