HYBRIDGE

Distributed Control and Stochastic Analysis of Hybrid Systems
Supporting Safety Critical Real-Time Systems Design

WP4: Compositional Specification of Stochastic Hybrid Systems

On Control of Complex Stochastic Hybrid Systems

Stefan Strubbe, Jan Willem Polderman, Agung Julius and Arjan van der Schaft[1]

April, 2005

Version: *1.0*
Task number: *4.4*
Deliverable number: *D4.4*
Contract: *IST-2001-32460 of European Commission*

[1] University of Twente

DOCUMENT CONTROL SHEET

Title of document: *On Control of Complex Stochastic Hybrid Systems*

Authors of document: *Stefan Strubbe, Jan Willem Polderman, Agung Julius and Arjan van der Schaft*

Deliverable number: *D4.4*

Contract: *IST-2001-32460 of European Comission*

Project: *Distributed Control and Stochastic Analysis of Hybrid Systems Supporting Safety Critical Real-Time Systems Design (HYBRIDGE)*

DOCUMENT CHANGE LOG

Version #	Issue Date	Sections affected	Relevant information
0.1	Dec 2004	All	First draft
0.2	January 2005	All	Draft for internal review
1.0	April 2005	1,3,5	Review comments incorporated.

Version 1.0		Organisation	Signature/Date
Authors	Stefan Strubbe	UTwente	
	Jan Willem Polderman	UTwente	
	Agung Julius	UTwente	
	Arjan van der Schaft	UTwente	
Internal reviewers	Henk Blom	NLR	

On Control of Complex Stochastic Hybrid Systems[1]

Stefan Strubbe, Jan Willem Polderman, Agung Julius and Arjan van der Schaft

April 2005

[1]Public deliverable 17 for the EU project HYBRIDGE (IST-2001-32460)

Contents

Chapter 1

Introduction

1.1 Scope of the report

This report deals with some fundamental issues in the control of complex stochastic hybrid systems.

Let us start with a disclaimer. The report does *not* deal with any aspect of *optimal control* of stochastic hybrid systems, although there does exist a substantial body of knowledge about this topic. In particular we refer to the work by Davis on optimal control of Piecewise-Deterministic Markov processes [7], and the work by Ghosh and Arapostathis [11].

Instead our focus will be on structural issues in the control of stochastic hybrid systems which are strictly related to the *complexity* of the system, as well as to the delineation of some sort of *limits of performance* of the controlled complex system. Common theme will be to look at control as the *addition of extra components to the complex system*, that is, *control by interconnection* or, more precisely, *control by composition* since the nature of composition (and the related communication structure) is fundamental in complex hybrid systems, being stochastic or not.

We will be far from presenting a complete theory. Instead we will report on three rather loosely related recent lines of research which in our opinion all will play a role in a larger body of theory which is waiting to be developed. The first chapter treats a basic control by composition problem for hybrid automata, without explicit taking into account stochasticity. The second chapter is concerned with an extension of the framework of CPDPs as recently developed within Workpackage 4, see in particular Deliverables [32] and [33]. The main issue addressed here is an extension of the possibilities for communication in the composition of CPDPs, and appears to be important both for modeling (see the presented ATM example) and control by composition. The third chapter studies conditions for stability of complex switched linear systems, which can be seen as a necessary requisite for a theory of stabilization of complex hybrid systems. This chapter convincingly shows how the analysis and control of complex hybrid systems profitably uses tools from automata theory.

1.2 Summary of contents

In the first chapter the problem of control by composition is treated for Hybrid Behavioral Automata (HBA). HBAs are automata that make use of the distinction between active and passive transitions. Therefore HBAs are closely related to CPDPs, although HBAs do not

involve stochastic aspects. The control problem can be seen as a combination of the control by interconnection problem for the continuous dynamics of each location of a complex hybrid system (modelled as an HBA), i.e. each location has its own control objective. To solve the problem, the controller needs to know at all times in which location the process is. In order to gain that knowledge, the controller uses passive transitions to observe the switchings between the locations of the process. Under certain conditions of the transition structure of the process, the control problem can be solved.

In the second chapter, we define value-passing CPDPs. In the composition of two value-passing CPDPs, one CPDP can communicate the value of its continuous state to the other CPDP. The other CPDP may use this information to switch to another location. The reset map of this switch may also depend on the information that was communicated. This idea is formalized via a special kind of synchronization of active transitions. Synchronization of two active transitions is not possible in the CPDP framework (with composition operator $||$) of [33]. However, with the composition operator $|_A^P|$, also described in [33] for active/passive transition systems (but not for CPDPs), it is possible to express active-active synchronization. Therefore, in this chapter we define the composition operator $|_A^P|$ for value-passing CPDPs (after having defined value-passing CPDPs), such that value passing can be expressed via this operator.

After the definitions of value-passing CPDPs and $|_A^P|$ for value-passing CPDPs, we apply (the composition of) value-passing CPDPs to a part of the air traffic management example 'Free Flight' (described in [25] and [10]). The use of value passing for compositional specification can be clearly seen in this example. The section ends with a discussion on the adequacy of our concept of value-passing from a compositional analysis point of view.

The final section of the second chapter is concerned with the connection between value-passing CPDPs and PDPs. We give an algorithm that can be used to convert a closed value-passing CPDP to a PDP. We prove that, when the algorithm terminates, the value-passing CPDP expresses a PDP behavior and the corresponding PDP is given by the results of the algorithm. As an example, the algorithm is applied to the 'Free Flight' example of the second section. The result is that the algorithm terminates (in five rounds), which shows that the value-passing CPDP expresses a PDP behavior.

The third chapter presents a stability analysis approach for a class of hybrid automata. It is assumed that the dynamics in each location of the hybrid automaton is linear and stable, and that the guards on the transitions are hyperplanes in the state space. For each pair of ingoing and outgoing transitions in a location a conservative estimate is made of the gain via a Lyapunov function for the dynamics in that location. It is shown how the choice of the Lyapunov function can be optimized to obtain the best possible estimate. The calculated conservative gains are used in defining a so-called gain automaton that forms the basis of an algorithmic criterion for the stability of the hybrid automaton.

Chapter 2

Control by composition using the active/passive framework

2.1 Introduction

In this chapter we discuss the hybrid behavioral automata (HBA) model. In particular, we are interested in the interconnections of such structure and its relation with controller design problem.

One of the defining features of the HBA is the introduction of the passive transitions. With the passive transitions in the model, we can model uni-directional discrete synchronizations between automata, in contrast with the common bi-directional synchronizations commonly used in other models.

There are other models that capture the spirit of uni-directional synchronization, for example, I/O automata [21] and its extension, hybrid I/O automata [19, 20]. The differences between the these frameworks and our framework are that in our framework synchronization with multiple active (output) agents is allowed and that *input enabledness* is embedded in the composition semantics rather than imposed on top of the structure as an assumption.

Formulation of control as interconnection has been recently advocated in behavioral system theory [38, 27]. This had led to some fundamental results, generalizing classical results in the input-output framework; see [37, 4] for the linear case and [29], [28] for extensions to other system classes. The same paradigm also appeared in computer science, for example, in the submodule construction problem [22].

In this chapter, a control problem is presented. A solution, derived from the construction of canonical controllers presented in [28] is derived. We also present and discuss some conditions under which the proposed controller solves the problem.

The layout of this chapter is as follows. In the next section we will present the structure of hybrid behavioral automata. Some notions of interconnection and projection of HBA are given in Section 3. In Section 4, we discuss the control problem and propose a solution, which is designed according to the canonical controller in [28]. In Section 5, the connection with CPDP is disussed and some concluding remarks are given.

2.2 Definitions and Notations

We begin by defining hybrid behavioral automata and their behaviors. First, note that throughout this paper we use a general totally ordered set $\mathbb{T}$ as the underlying time axis for the trajectories inside each location. Instances of $\mathbb{T}$ can be $\mathbb{R}$, $\mathbb{R}_+$ or other chosen sets. This time axis is not to be mistaken with the *hybrid time trajectory,* which is defined later.

A hybrid behavioral automaton (HBA) $\mathbb{A}$ is a septuple $(L, W, A, T, P, Inv, \mathfrak{B})$, where

- L is the set of locations or discrete states,
- W is the set of continuous variables taking values in $\mathbf{W}$,
- A is the set of labels,
- T is the set of active jumps/transitions. Each jump is given as a pentuple $(l, \mathbf{a}, l', G, R)$, where l is the origin location, $\mathbf{a}$ is the label of the jump, l' is the target location, $G := (\gamma, g)$ is the guard of the jump, where $\gamma : \mathfrak{B}(l) \times \mathbb{T} \to$codomain$(\gamma)$ and $g \subset$codomain(γ), and $R : \mathfrak{B}(l) \times \mathbb{T} \to 2^{\mathfrak{B}(l')}$ is the reset map of the jump.
- P is the set of passive jumps/transitions. We represent each passive jump as a quadruple $(l, \mathbf{a}, l', R)$. Passive jumps are not guarded.
- Inv maps each location $l \in L$ to a pair $Inv(l) := (\nu, V)$, where $\nu : \mathfrak{B}(l) \times \mathbb{T} \to$codomain$(\nu)$ and $V \subset$codomain(ν).
- $\mathfrak{B}$ maps each location to its continuous behavior. A behavior is a subset of $W^{\mathbb{T}}$.

The guard G and the invariant $Inv(l)$ as introduced above are instances of *dynamic predicates*. In general, a dynamic predicate is a pair $C := (\psi, \Psi)$,

$$\psi : \mathfrak{B} \times \mathbb{T} \to \text{codomain}(\psi),$$
$$\Psi \subset \text{codomain}(\psi).$$

$\mathfrak{B}$ signifies a behavior over the general (ordered) time axis $\mathbb{T}$. A pair $(w, t) \in \mathfrak{B} \times \mathbb{T}$ is said to satisfy the dynamic predicate C if $\psi(w, t) \in \Psi$. We denote it by $(w, t) \models C$. The negation of this statement is denoted by $(w, t) \not\models C$.

We assume that the maps γ, R, and ν are *causal*. A map $x : \mathfrak{B} \times \mathbb{R} \to X$ is causal if for any w_1 and w_2 in $\mathfrak{B}$ and $\tau \in \mathbb{R}$, the following implication holds.

$$\left(w_1(t)|_{t \leq \tau} = w_2(t)|_{t \leq \tau} \right) \Longrightarrow \left(x(w_1, t) = x(w_2, t) \right).$$

In order to describe the evolution of such automaton, we need to define a suitable timeline structure. In this case, we use a slightly modified version of *hybrid time trajectory*, introduced in [36]. The idea behind the hybrid time trajectory is as follows. We have continuously evolving dynamical system, but punctuated by jumps. Because of this, we choose a timeline that consists of intervals of $\mathbb{T}$. Each interval acts as a timeline for describing the evolution between jumps.

Definition 1. *A hybrid time trajectory $\tau = \{I_i\}_{i=0}^{N}$ is a finite or infinite sequence of intervals of $\mathbb{T}$, such that:*

- $I_0 = [\tau_0, \tau_0']$ *or* $(-\infty, \tau_0']$, $\tau_0 \leq \tau_0' \in \mathbb{T}$,
- $I_i = [\tau_i, \tau_i']$ *for* $i < N$ *and, if* $N < \infty$, $I_N = [\tau_N, \tau_N']$ *or* $I_N = [\tau_N, \tau_N')$,
- *for all* i, $\tau_i \leq \tau_i' = \tau_{i+1}$.

A hybrid time trajectory $\tau' = \{I_i'\}_{i=0}^{N'}$ is said to be a *prefix* of another time trajectory $\tau = \{I_i\}_{i=0}^{N}$ if $N' \leq N$ and $I_i' = I_i$ for all $0 \leq i \leq N'$. A hybrid time trajectory $\tau = \{I_i\}_{i=0}^{N}$ is said to be *infinite* if $N = \infty$ or $\tau_N' = \infty$.

In line with this timeline structure, we describe the evolution of an HBA. A hybrid trajectory is denoted as (τ, ξ). Here τ is a hybrid time trajectory and ξ maps the interval $\{I_n\}_{n \geq 0}$ in τ to a triple (l_n, w_n, j_n), where $l_n \in L$, $w_n : I_n \to \mathbf{W}$, and $j_n \in (T \cup P)$ or $j_n = \varnothing$. The case where $j_n = \varnothing$ may happen only on the last interval of τ.

A hybrid trajectory (τ', ξ') is said to be a *prefix* of another hybrid trajectory (τ, ξ) if τ' is a prefix of τ and $\xi' = \xi$ on τ'. A hybrid trajectory (τ, ξ) is called *infinite* if τ is infinite.

A hybrid trajectory (τ, ξ) is included in the *hybrid behavior* $\mathcal{A}$ of the automaton $\mathbb{A}$ if the following conditions are satisfied for all $n \geq 0$.

1. $w_n \in \mathfrak{B}(l_n)$,
2. $j_n = (l_n, \mathbf{a}, l_{n+1}, G_n, R_n)$, for some $\mathbf{a} \in A$,
3. $j_n \in T$,
4. $(w_n, \tau_n') \models G_n$,
5. $\tau_n' \leq \inf\{t \mid t \geq \tau_n \text{ and } (w_n, t) \not\models Inv(l_n)\}$,
6. $w_{n+1} \in R_n(w_n, \tau_n')$.

Such trajectory is called a *valid* trajectory. Notice that we do not have passive jumps in a valid trajectory.

A hybrid trajectory (τ, ξ) is included in the *potential behavior* $\bar{\mathcal{A}}$ if it satisfies conditions 1,2,4,5 and 6 above, together with a relaxed version of condition 3, $j_n \in (T \cup P)$. The intuitive idea is that we only include hybrid trajectories without any passive jumps in $\mathcal{A}$, while in $\bar{\mathcal{A}}$ we also allow passive jumps. Obviously $\mathcal{A} \subset \bar{\mathcal{A}}$. Notice that, by the way they are defined, $\mathcal{A}$ and $\bar{\mathcal{A}}$ are *prefix closed*. This means that if a hybrid trajectory (τ, ξ) is in $\mathcal{A}$ (or $\bar{\mathcal{A}}$), then any of its prefixes (τ', ξ') is also in $\mathcal{A}$ (or $\bar{\mathcal{A}}$).

Throughout this chapter we shall also use the following shorthand notation. We write $l \xrightarrow{\mathbf{a}} l'$ to denote the existence of an active transition going from location $l \in L$ to location $l' \in L$ with label $a \in A$. The existence of a passive transition with the same characteristics is denoted by $l \overset{\mathbf{a}}{\dashrightarrow} l'$. The notations $l \xrightarrow{\mathbf{a}}$ and $l \overset{\mathbf{a}}{\dashrightarrow}$ denote the existence of $l' \in L$ such that $l \xrightarrow{\mathbf{a}} l'$ and $l \overset{\mathbf{a}}{\dashrightarrow} l'$ respectively. The absence of such transitions are denoted as $l \not\xrightarrow{\mathbf{a}} l'$, $l \not\overset{\mathbf{a}}{\dashrightarrow} l'$, $l \not\xrightarrow{\mathbf{a}}$, and $l \not\overset{\mathbf{a}}{\dashrightarrow}$ respectively.

If the information about the guard and the reset map is also included in the notation, we write $l \xrightarrow{(\mathbf{a}, G, R)} l'$ to denote $(l, \mathbf{a}, l', G, R) \in T$, and $l \overset{(\mathbf{a}, R)}{\dashrightarrow} l'$ if $(l, \mathbf{a}, l', R) \in P$.

2.3 Composition and projection operators

Two HBA, $\mathbb{A}_1$ and $\mathbb{A}_2$ characterized by $(L_i, W, A, T_i, P_i, Inv_i, \mathfrak{B}_i)$, $i = 1, 2$, can be interconnected to form another HBA $\mathbb{A} = \mathbb{A}_1 \parallel \mathbb{A}_2$. The automaton $\mathbb{A}$ is characterized by the septuple $(L, W, A, T, P, Inv, \mathfrak{B})$, where

$$\begin{aligned} L &= L_1 \times L_2, \\ \mathfrak{B}((l_1, l_2)) &= \mathfrak{B}_1(l_1) \cap \mathfrak{B}_2(l_2), \\ Inv((l_1, l_2)) &= (\nu, V), \end{aligned}$$

such that $Inv((l_1, l_2))$ is satisfied if and only if both $Inv_1(l_1)$ and $Inv_2(l_2)$ are satisfied.

The set T and P consist of pentuples and quadruples, $((l_1, l_2), \mathbf{a}, (l_1', l_2'), G_T, R_T)$ and $((l_1, l_2), \mathbf{a}, (l_1', l_2'), R_T)$ respectively, such that the following interconnection semantics are commutatively satisfied.

$$\frac{l_1 \xrightarrow{(\mathbf{a},G,R)} l_1', l_2 \not\xrightarrow{\mathbf{a}}}{(l_1,l_2) \xrightarrow{(\mathbf{a},G,R)} (l_1',l_2)} \qquad \frac{l_1 \xrightarrow{(\mathbf{a},G,R_1)} l_1', l_2 \xrightarrow{(\mathbf{a},R_2)} l_2'}{(l_1,l_2) \xrightarrow{(\mathbf{a},G,R_1 \cap R_2)} (l_1',l_2')}$$

$$\frac{l_1 \not\xrightarrow{\mathbf{a}}, l_2 \xrightarrow{(\mathbf{a},R)} l_2'}{(l_1,l_2) \xrightarrow{(\mathbf{a},R)} (l_1,l_2')} \qquad \frac{l_1 \xrightarrow{(\mathbf{a},R_1)} l_1', l_2 \xrightarrow{(\mathbf{a},R_2)} l_2'}{(l_1,l_2) \xrightarrow{(\mathbf{a},R_1 \cap R_2)} (l_1',l_2')}$$

By taking intersection of reset maps, we mean intersecting their respective images.

The interconnection operation described here possesses some ideal properties that make it suitable for establishing modularity for hybrid behaviors, namely *commutativity* and *associativity* [30].

Notice that all (continuous) variables are involved in the synchronization. This type of interconnections is called *total interconnections*. It is also possible to define *partial interconnections*, where only a part of the variables are synchronized.

Two HBA, $\mathbb{A}_1$ and $\mathbb{A}_2$ characterized by $(L_i, W_i \cup Z, A, T_i, P_i, Inv_i, \mathfrak{B}_i)$, $i = 1, 2$, are partially interconnected over Z to form another HBA $\mathbb{A} = \mathbb{A}_1 \parallel_Z \mathbb{A}_2$. The automaton $\mathbb{A}$ is characterized by the septuple $(L, W_1 \cup W_2 \cup Z, A, T, P, Inv, \mathfrak{B})$, where

$$\begin{aligned} L &= L_1 \times L_2, \\ \mathfrak{B}((l_1, l_2)) &= \mathfrak{B}_1(l_1) \parallel_Z \mathfrak{B}_2(l_2), \\ Inv((l_1, l_2)) &= (\nu, V), \text{ where} \end{aligned}$$

$$\begin{aligned} &Inv_i(l_i) =: (\nu_i, V_i), \ i = 1, 2, \\ &\nu((w_1, w_2, z), t) = (\nu_1((w_1, z), t), \nu_2((w_2, z), t)), \text{ and} \\ &V = V_1 \times V_2. \end{aligned}$$

The set T and P consist of pentuples and quadruples, $((l_1, l_2), \mathbf{a}, (l_1', l_2'), G_T, R_T)$ and $((l_1, l_2), \mathbf{a}, (l_1', l_2'), R_T)$ respectively, such that the following interconnection semantics are commutatively satisfied.

$$\frac{l_1 \xrightarrow{(\mathbf{a},G,R)} l_1', l_2 \not\xrightarrow{\mathbf{a}}}{(l_1,l_2) \xrightarrow{(\mathbf{a},G,R)} (l_1',l_2)} \qquad \frac{l_1 \xrightarrow{(\mathbf{a},G,R_1)} l_1', l_2 \xrightarrow{(\mathbf{a},R_2)} l_2'}{(l_1,l_2) \xrightarrow{(\mathbf{a},G,R_1 \parallel_Z R_2)} (l_1',l_2')}$$

$$\frac{l_1 \not\xrightarrow{\mathbf{a}}, l_2 \xrightarrow{(\mathbf{a},R)} l_2'}{(l_1,l_2) \xrightarrow{(\mathbf{a},R)} (l_1,l_2')} \qquad \frac{l_1 \xrightarrow{(\mathbf{a},R_1)} l_1', l_2 \xrightarrow{(\mathbf{a},R_2)} l_2'}{(l_1,l_2) \xrightarrow{(\mathbf{a},R_1 \parallel_Z R_2)} (l_1',l_2')}$$

By performing partial interconnection on the reset maps we mean partially interconnecting their respective images in the behavioral sense[1].

Let $\mathbb{A} = (L, W \cup Z, A, T, P, Inv, \mathfrak{B})$. We can project the automaton to the set of variables W, written as $\pi_W\mathbb{A}$, by defining $\pi_W\mathbb{A} = (L, W, A, \pi_W T, \pi_W P, \pi_W Inv, \pi_W\mathfrak{B})$, where

$$\pi_W\mathfrak{B} := \{w \mid \exists z \text{ such that } (w, z) \in \mathfrak{B}\}.$$

The projected set of active transitions $\pi_W T$ consists of pentuples $(l, \mathbf{a}, l', \pi_W G, \pi_W R)$, with $(l, \mathbf{a}, l', G, R) \in T$. The projected guard and reset map are defined as follows. For any $w \in \pi_W\mathfrak{B}(l)$ and $t \in \mathbb{T}$, the pair (w, t) satisfies $\pi_W G$ if and only if there is a $z \in Z^{\mathbb{T}}$ such that $(w, z) \in \mathfrak{B}(l)$ and (w, z, t) satisfies G. For any $w \in \pi_W\mathfrak{B}(l)$ and $t \in \mathbb{T}$, the trajectory $w' \in \mathfrak{B}'(l')$ is included in $R'(w, r)$ if and only if there are z and z' such that

$$\begin{aligned}(w, z) &\in \mathfrak{B}(l),\\ (w', z') &\in \mathfrak{B}(l'),\\ (w', z') &\in R(w, z, t).\end{aligned}$$

The projected set of passive transitions $\pi_W P$ consists of quadruples $(l, \mathbf{a}, l', \pi_W R)$, with $(l, \mathbf{a}, l', R) \in P$. The projected invariant $\pi_W Inv$ is such that for any $l \in L$, $w \in \pi_W\mathfrak{B}(l)$ and $t \in \mathbb{R}$, the pair (w, t) satisfies $\pi_W Inv(l)$ if and only if there exists a z such that $(w, z) \in \mathfrak{B}(l)$ and (w, z, t) satisfies $Inv(l)$.

We also define another notion of projection, which we call *factorization*. The idea behind it is as follows.

Interconnecting two automata results in an automaton whose discrete dynamics is somewhat larger (i.e. more locations and more transitions) than those of the components. By factorizing the interconnected automata, we aim to see the 'effect' of the interconnection on the individual component.

Take two HBA $\mathbb{A}_i = (L_i, W, A, T_i, P_i, Inv_i, \mathfrak{B}_i)$, $i = 1, 2$. Let $\mathbb{A} = (L_1 \times L_2, W, A, T, P, Inv, \mathfrak{B}) = \mathbb{A}_1 \parallel \mathbb{A}_2$. Factorizing $\mathbb{A}$ with respect to its component $\mathbb{A}_1$, denoted as $\pi_{\mathbb{A}_1}\mathbb{A}$ can be done as follows. First, we construct the following equivalent relation. For any (l_{1i}, l_{2i}) and (l_{1j}, l_{2j}) in $L_1 \times L_2$,

$$(l_{1i}, l_{2i}) \approx (l_{1j}, l_{2j}) \text{ iff } (l_{1i} = l_{1j}).$$

Each equivalent class of $\approx$ represents a location in L_1 and is named accordingly.

The action of the factorization $\pi_{\mathbb{A}_1}$ to $\mathbb{A}$ results in $\pi_{\mathbb{A}_1}\mathbb{A} = (L_1, W, A, T', P', Inv', \mathfrak{B}')$, where

$$\begin{aligned}T' = \{(l_i, \mathbf{a}, l_j, G, R) \mid \exists l_i' \in l_i, l_j' \in l_j,\\ \text{such that } (l_i', \mathbf{a}, l_j', G, R) \in T\},\\ P' = \{(l_i, \mathbf{a}, l_j, R) \mid \exists l_i' \in l_i, l_j' \in l_j,\\ \text{such that } (l_i', \mathbf{a}, l_j', R) \in P\},\\ \mathfrak{B}'(l_i) = \bigcup_{l \in l_i} \mathfrak{B}(l),\end{aligned}$$

and the invariant $Inv'(l_i)$ is such that any pair $(w, t) \in \mathfrak{B}'(l_i) \times \mathbb{T}$ satisfies it iff (w, t) satisfies at $Inv(l)$ for at least one $l \in l_i$.

[1] $(R_1 \parallel_Z R_2) := \{(w_1, w_2, z) \mid (w_1, z) \in R_1 \text{ and } (w_2, z) \in R_2\}$

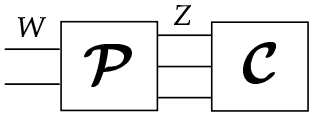

Figure 2.1: Control with partial interconnection.

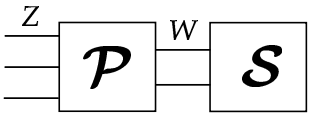

Figure 2.2: The controller $\mathcal{C}_{\text{can}} = \pi_Z(\mathcal{P} \|_W \mathcal{S})$.

Factorizing $\mathbb{A}$ with respect to $\mathbb{A}_1$ gives us information about the effect of the interconnection to $\mathbb{A}_1$. This is particularly useful when interconnection is seen as controlling [28]. In this point of view, a plant model (in this case it is $\mathbb{A}_1$) and a desired specification $\mathbb{S}$ are given. The problem is to find a controller, in this case $\mathbb{A}_2$, such that $\pi_{\mathbb{A}_1}(\mathbb{A}_1 \parallel \mathbb{A}_2) = \mathbb{S}$. This formulation is very closely related to control as seen from behavioral point of view [38].

2.4 The control problem and passive canonical controller

The behavioral approach to control theory sees control as interconnection. Given a plant (in term of behavior) $\mathcal{P}$ and a specification $\mathcal{S}$ (also in term of behavior), the problem of finding a controller that achieves the desired closed-loop behavior is translated to the problem of finding a controller behavior $\mathcal{C}$, such that $\mathcal{P} \parallel \mathcal{C} = \mathcal{S}$ [38, 28, 29]. The symbol $\parallel$ signifies behavior interconnection [38, 27]. This formulation is closely related to the submodule construction problem in computer science [22].

In most of the problems, however, not all variables of the plant are available for interconnection with the controller. Most problems deal with *partial interconnections*. This type of interconnection can be described as in Figure 2.1. In this figure, W represents the set of variables on which the specification is expressed, while Z represents those used in the interconnection with the controller. Note that these two sets are not necessarily disjunct. Partial interconnections are denoted by adding a subscript to the composition operator. Thus, the structure in Figure 2.1 is $\mathcal{P} \|_Z \mathcal{C}$.

Such problems for general behaviors have been treated in [29, 28], where a construction for *canonical controllers* is given. This construction is shown in Figure 2.2. A canonical controller $\mathcal{C}_{\text{can}}$ constructed in this way solves the problem provided that the plant $\mathcal{P}$ and the specification $\mathcal{S}$ satisfy a couple of conditions, which can be thought of as some generalized controllability and observability conditions. In this case, we then have

$$\pi_W (\mathcal{P} \|_Z \mathcal{C}_{\text{can}}) = \mathcal{S}.$$

In the following, we introduce the notion of *rooted hybrid behavioral automata*. Simply speaking, the root of an automata is the location in which all trajectories are assumed to start. Thus, the root acts as discrete initial condition for the evolution. An HBA $\mathbb{A}$ that has a root l is denoted as $\mathbb{A}(l)$. To this rooted automaton we can associate a *rooted hybrid behavior* $\mathcal{A}(l)$, such that a hybrid trajectory $(\tau, \xi) \in \mathcal{A}(l)$ if $(\tau, \xi) \in \mathcal{A}$ and[2] $l_0 = l$. The *rooted potential behavior* $\bar{\mathcal{A}}(l)$ is defined in a similar fashion.

We are going to treat the following control problem:

Given a plant in terms of a rooted HBA $\mathbb{P}(l) = (L, W \cup Z, A, T_p, \varnothing, Inv_p, \mathfrak{B}_p)$, and a specification $\mathbb{S}(l) = (L, W, A, T_s, \varnothing, Inv_s, \mathfrak{B}_s)$. Notice that we assume that both the plant and the specification do not have any passive transition. The problem is to find a controller $\mathbb{C}(l)$, which is also expressed in terms of rooted HBA, such that

$$(\pi_W \circ \pi_{\mathbb{P}})\,(\mathcal{P} \parallel_Z \mathcal{C})(l, l) = \mathcal{S}(l).$$

The operators $\pi_{\mathbb{P}}$ and π_W denote factorization of the interconnected automaton with respect to $\mathbb{P}$ and projection of the continuous dynamics to the W variables. □

A solution, which is adopted from [29, 28], is proposed. The idea is as follows. We shall use a controller that has no active transitions. Such controller is called a *passive controller*. The controller has the same set of locations as the plant, and the set of passive transitions in the controller is the same as the set of active transitions in the plant less the information about the guard. The controller is a rooted HBA $\mathbb{C}(l) = (L, Z, A, \varnothing, P_c, Inv_c, \mathfrak{B}_c)$. Since we assume no active transitions in the controller, we also assume the invariant Inv_c is a dynamic predicate that is always satisfied. Moreover, the behavior $\mathfrak{B}_c$ is defined as follows.

$$\mathfrak{B}_c(l) = \pi_Z(\mathfrak{B}_p(l) \parallel_W \mathfrak{B}_s(l)).$$

Notice that this construction is similar to that in [29, 28].

Theorem 1. *The proposed controller $\mathbb{C}(l)$ solves the control problem,*

$$(\pi_W \circ \pi_{\mathbb{P}})\,(\mathcal{P} \parallel_Z \mathcal{C})(l, l) = \mathcal{S}(l),$$

if the following conditions hold.

(c1) *For any $l \in L$, $\mathfrak{B}_s(l) \subset \pi_W\,(\mathfrak{B}_p(l))$.*

(c2) *For any $l \in L$ and any pairs $(w, z), (\tilde{w}, z) \in \mathfrak{B}_p(l)$, the following implication holds.*

$$(w \in \mathfrak{B}_s(l)) \Rightarrow (\tilde{w} \in \mathfrak{B}_s(l))$$

(c3) *The set of active transitions $T_s = \pi_W T_p$.*

(c4) *The invariant $Inv_s = \pi_W Inv_p$.*

(c5) *For any $l \in L$, $\mathbf{a} \in A$, there can be at most one transition in T_p that starts in location l with label $\mathbf{a}$.*

[2]Recall that l_0 is the location of the first interval of τ.

(c6) *For any* $l \in L$, $t \in \mathbb{T}$, $(l, \mathbf{a}, l', G, R) \in T_p$ *and any pairs* $(w, z), (w, \tilde{z}) \in \mathfrak{B}_p(l)$, *the following implications hold*

$$(w, z, t) \models G \Leftrightarrow (w, \tilde{z}, t) \models G, \tag{2.2a}$$
$$(w, z, t) \models Inv_p(l) \Leftrightarrow (w, \tilde{z}, t) \models Inv_p(l), \tag{2.2b}$$
$$(w', z') \in R(w, z, t) \Leftrightarrow (w', \tilde{z}') \in R(w, z, t)$$
$$\textit{for all } (w', z'), (w', \tilde{z}') \in \mathfrak{B}_p(l'). \tag{2.2c}$$

Proof. In this proof we shall denote $\mathbb{P} \parallel_Z \mathbb{C} := \mathbb{Q}$ for brevity. Consequently, its behavior is denoted as $\mathcal{Q}$. Assume that all the conditions above hold. Based on (c1) and (c2), we can infer[3]

$$\mathfrak{B}_s(l) = \pi_W \left(\mathfrak{B}_p(l) \parallel_Z \mathfrak{B}_c(l) \right), \forall l \in L. \tag{2.3}$$

Take any hybrid trajectory $(\tau, \xi) \in \mathcal{S}(l)$. We shall prove that $(\tau, \xi) \in (\pi_W \circ \pi_{\mathbb{P}})\, \mathcal{Q}(l, l)$. Let $\tau = \{I_i\}_{i=0}^{N}$. Since $(\tau, \xi) \in \mathcal{S}(l)$, the following conditions must be satisfied.for all $N \geq n \geq 0$

$$l_0 = l, \tag{2.4a}$$
$$w_n \in \mathfrak{B}_s(l_n), \tag{2.4b}$$
$$j_n = (l_n, \mathbf{a}_n, l_{n+1}, G_n, R_n) \in T_s, \tag{2.4c}$$
$$(w_n, \tau_n') \models G_n, \tag{2.4d}$$
$$\tau_n' \leq \inf\{t \mid t \geq \tau_n \text{ and } (w_n, t) \not\models Inv_s(l_n)\}, \tag{2.4e}$$
$$w_{n+1} \in R_n(w_n, \tau_n'). \tag{2.4f}$$

We argue that there is a trajectory $(\tau, \tilde{\xi}) \in \mathcal{Q}(l, l)$ such that for all $N \geq n \geq 0$

$$\tilde{l}_0 = (l, l), \tag{2.5a}$$
$$\tilde{l}_n = (l_n, l_n), \tag{2.5b}$$
$$\tilde{w}_n = (w_n, z_n) \in \mathfrak{B}_p(l_n) \parallel_Z \mathfrak{B}_c(l_n), \tag{2.5c}$$
$$\tilde{j}_n = (\tilde{l}_n, \mathbf{a}_n, \tilde{l}_{n+1}, \tilde{G}_n, \tilde{R}_n) \in T_q, \text{ where} \tag{2.5d}$$
$$(l_n, \mathbf{a}_n, l_{n+1}, \tilde{G}_n, \tilde{R}_n) \in T_p, G_n = \pi_W \tilde{G}_n, R_n = \pi_W \tilde{R}_n, \tag{2.5e}$$
$$(\tilde{w}_n, \tau_n') \models \tilde{G}_n, \tag{2.5f}$$
$$\tau_n' \leq \inf\{t \mid t \geq \tau_n \text{ and } (\tilde{w}_n, t) \not\models Inv_p(l_n)\}, \tag{2.5g}$$
$$\tilde{w}_{n+1} = (w_{n+1}, z_{n+1}) \in \tilde{R}_n(w_n, z_n, \tau_n'). \tag{2.5h}$$

These relations are obtained by the following. Equation (2.5e) is obtained through (2.4c) and (c3), then (2.5d) is obtained using the definition of partial interconnection. This implies (2.5b). Equation (2.5c) is obtained using (2.3). Next, (2.5f) is implied by $G_n = \pi_W \tilde{G}_n$, while (2.5g) is implied by (c4) and (c6). Finally, (2.5h) is implied by (c3), and due to (2.2c) z_{n+1} is not restricted, since any z_{n+1} such that $(w_{n+1}, z_{n+1}) \in \mathfrak{B}_p(l_{n+1})$ will satisfy (2.5h). This guarantees that the whole argument can be established iteratively, starting with $n = 0$. Given the existence of such $(\tau, \tilde{\xi}) \in \mathcal{Q}(l, l)$, it follows that $(\tau, \xi) \in (\pi_W \circ \pi_{\mathbb{P}})\, \mathcal{Q}(l, l)$. Hence we have that

$$\mathcal{S}(l) \subset (\pi_W \circ \pi_{\mathbb{P}})\, \mathcal{Q}(l, l). \tag{2.6}$$

[3]Please see [29, 28] for a proof.

To show the converse, take any $(\tau, \xi) \in (\pi_W \circ \pi_{\mathbb{P}})\, \mathcal{Q}(l, l)$. We shall show that $(\tau, \xi) \in \mathcal{S}(l)$. Again, we let $\tau = \{I_i\}_{i=0}^{N}$. Since $(\tau, \xi) \in (\pi_W \circ \pi_{\mathbb{P}})\, \mathcal{Q}(l, l)$, there is a $(\tau, \tilde{\xi}) \in \mathcal{Q}(l, l)$ such that for all $N \geq n \geq 0$, (2.5a)-(2.5h) hold. The reasoning is as follows. The controller $\mathbb{C}(l)$ has only passive transitions, therefore any transition in $(\mathbb{P} \parallel_Z \mathbb{C})\,(l, l)$ must be a synchronization between a passive transition of $\mathbb{C}$ and an active transition of $\mathbb{P}$. Using (c5) and the definition of partial interconnection, we can infer (2.5b). This equation then implies (2.5c). Further, (2.5b) and the definition of π_W imply (2.5d), (2.5e), (2.5f), (2.5g), and (2.5h). Again, due to (2.2c) z_{n+1} is not restricted, therefore ensuring that the whole argument can be established iteratively. Now we shall prove that $(\tau, \xi) \in \mathcal{S}(l)$ by showing that (2.4a) - (2.4f) hold. First, (2.4b) is implied by (2.3), then (c3) and (2.5e) imply (2.4c), (2.4d) and (2.4f). Finally, (2.4e) is implied by (c4). Now, we have established that

$$\mathcal{S}(l) \supset (\pi_W \circ \pi_{\mathbb{P}})\, \mathcal{Q}(l, l), \tag{2.7}$$

and hence completed the proof. □

Let us now discuss the conditions posed in the theorem, i.e. (c1) - (c6). The first two conditions arise as sufficient conditions to guarantee that we can establish (2.3). These conditions are adopted from [29, 28]. In the same reference, some variants of the conditions are also presented. Conditions (c3) - (c5) are essential, because of the passive nature of the controller. Since the controller only has passive transitions, it cannot be used to influence the active transitions in the plant, hence (c3) and (c4). Condition (c5) is there to guarantee that the location of the controller can accurately follow that of the plant, without any possibility of being misleaded due to some nondeterminism. Finally, condition (c6) is there to guarantee that the choice for z_n does not influence the choice for z_{n+1}. As explained above, this in turn guarantees that the whole reasoning can be done iteratively (interval wise).

2.5 HBA and CPDP

The connection between HBA and CPDP is that they are both automata frameworks using active/passive transitions. The differences are: HBA is not stochastic (i.e. there are are no spontaneous transitions and the reset maps are non-stochastic). Except for the stochastic aspects, HBA is a more general framework than CPDP: instead of the differential equations in CPDP, HBA uses the more general behavioral approach and instead of letting the reset map depend on the current state only, the HBA reset maps may depend on the entire trajectory in the current location.

The objective of the control problem, treated in this chapter, is to control the continuous dynamics in each location. This means that in fact each location has its own control objective. In order to achieve this control objective, the controller needs to know at all times in which location the process is. This is where the active/passive framework is used: as soon as the process switches to another location with an active transition labelled a, the controller observes this switch via a passive transition labelled $\bar{a}$.

The extension of this control problem to a stochastic hybrid system context is, although promising, a non-trivial one. One problem is that in the current framework of CPDPs interaction of the continuous dynamics with a controller dynamics is not possible. However, since the interconnection of plant and continuous controller will normally result in ordinary

closed-loop differential equations, such an extension is perfectly allowable from a PDP point of view. Thus allowing these kinds of continuous interactions for CPDP opens possibilities to treat the control problem of this chapter also in the CPDP context.

Chapter 3

Value-passing CPDPs

In the CPDP-model as it is defined in [33], it is not possible that one component can inform another component about the value of its continuous state. In Dynamically Colored Petri Nets, this is possible (see [9]) and later in this chapter we will see an example where this feature is used. In this chapter we introduce a new CPDP-model (see [34]), in which this feature is also present. We chose to follow a standard method of data-communication, called *value-passing*. Value-passing has been defined for different models like LOTOS ([18]). Value-passing can be seen as a natural extension to (the standard) communication through shared events because it is also expressed through "shared events"/"synchronization of transitions".

3.1 Definition of the value-passing CPDP

We introduce a new definition for CPDP, where communication of data is possible. A CPDP is a tuple $(L, V, W, v, w, F, G, \Sigma, A, P, S)$, where

- L is a countable set of locations
- V is a set of state variables. With $d(y)$ for $y \in V$ we denote the dimension of variable y. $y \in V$ takes its values in $\mathbb{R}^{d(y)}$.
- W is a set of output variables. With $d(y)$ for $y \in W$ we denote the dimension of variable y. $y \in W$ takes its values in $\mathbb{R}^{d(y)}$.
- $v : L \to 2^V$ maps each location to a subset of V, which is the set of state variables of the corresponding location
- $w : L \to 2^W$ maps each location to a subset of W, which is the set of output variables of the corresponding location
- F assigns to each location l and each $y \in v(l)$ a mapping from $\mathbb{R}^{d(y)}$ to $\mathbb{R}^{d(y)}$, i.e. $F(l, y) : \mathbb{R}^{d(y)} \to \mathbb{R}^{d(y)}$.
- G assigns to each location l and each $y \in w(l)$ a mapping from $\mathbb{R}^{d(x_1)+\cdots+d(x_m)}$ to $\mathbb{R}^{d(y)}$, where x_1 till x_m are the state variables of location l.
- Σ is the set of communication labels. $\bar{\Sigma}$ denotes the 'passive' mirror of Σ and is defined as $\bar{\Sigma} := \{\bar{a} | a \in \Sigma\}$.

- A is a finite set of active transitions and consists of 5-tuples (l, a, l', G, R) and 6-tuples (l, a, l', G, R, val), denoting a transition from location $l \in L$ to location $l' \in L$ with communication label $a \in \Sigma$, guard G, reset map R and (if present) value-passing identifier val. G is a subset of the active output variable space. val can be equal to either $!Y$ or $?Y$. For the case $!Y$, Y is a subset of $w(l)$, meaning that this transition can pass the values of the variables from Y to other transitions in other components. For the case $?Y$, $Y \subset \mathbb{R}^n$ for some $n \in \mathbb{N}$, meaning that this transition asks for inputs with values in $\mathbb{R}^n$ (which should be passed/offered by other transitions in other components). Only inputs that are in Y are accepted. The reset map R assigns to each point in $G \times Y$ (for the case $(l, a, l', G, ?Y)$) or to each point in G (for all other cases) for each state variable $y \in v(l')$ a probability measure on the invariant (and its Borel sets) of y for location l'.

- P is a finite set of passive transitions of the form $(l, \bar{a}, l', R)$. R is defined on all interior points.

- S is a finite set of spontaneous (also called Poisson) transitions and consists of 5-tuples (l, λ, a, l', R), denoting a transition from location $l \in L$ to location $l' \in L$ with communication label $a \in \Sigma$, jump-rate function λ and reset map R. The jump rate λ is a mapping from Inv_l to $\mathbb{R}_+$. R is defined on all interior points of l as it is done for spontaneous transitions.

Composition is defined via the following composition rules. Rule r1data is concerned with value-passing, the other rules are not.

$$r1\text{data}.\frac{\mathrm{l_1} \overset{\mathrm{a,G_1,R_1,v_1}}{\longrightarrow} \mathrm{l_1'}, \mathrm{l_2} \overset{\mathrm{a,G_2,R_2,v_2}}{\longrightarrow} \mathrm{l_2'}}{\mathrm{l_1}|^{\mathrm{P}}_{\mathrm{A}}|\mathrm{l_2} \overset{\mathrm{a,G_1|G_2,R_1\times R_2,v_1|v_2}}{\longrightarrow} \mathrm{l_1'}|^{\mathrm{P}}_{\mathrm{A}}|\mathrm{l_2'}}(\mathrm{a} \in \mathrm{A}, \mathrm{v_1|v_2} \neq \perp).$$

Here, $v_1|v_2$ is defined as: $v_1|v_2 :=!Y$ if $v_1 =!Y$ and $v_2 :=?Y'$ or if $v_2 =!Y$ and $v_1 :=?Y'$ (with $\dim(Y')=\dim(Y)$); $v_1|v_2 :=?(Y_1 \cap Y_2)$ if $v_1 =?Y_1$ and $v_2 =?Y_2$ and $\dim(Y_1)=\dim(Y_2)$; $v_1|v_2 := \perp$ otherwise. $\perp$ here means that v_1 and v_2 are not compatible. $G_1|G_2$ is defined as follows: $G_1|G_2 := (G_1 \cap Y) \times G_2$ if $v_1 =!Y'$ and $v_2 =?Y$; $G_1|G_2 := G_1 \times (G_2 \cap Y)$ if $v_1 =?Y$ and $v_2 =!Y'$; $G_1|G_2 := G_1 \times G_2$ if $v_1 =?Y_1$ and $v_2 =?Y_2$. Here, $G \cap Y$, which is abuse of notation, means $\{y \in G | y \downarrow_z \in Y\}$, where z is the set of variables received by the $?Y$ transition.

In these definitions of $v_1|v_2$ and $G_1|G_2$ we see an interplay between the output guards G_1,G_2 and the input guards Y_1,Y_2: In the synchronization of an $(l_1, a, l_1', G_1, R_1, !z)$ transition with a $(l_2, a, l_2', G_2, R_2, ?Y)$ transition, Y restricts the guard G_1 such that the z-part of G_1 lies within Y. This restriction can not be coded in $v_1|v_2$ (as it is done in the $?Y_1$-$?Y_2$-case), therefore we need to code it in the output guards. The composition rules that do not concern value passing are as follows.

$$r1.\frac{l_1 \overset{a,G_1,R_1}{\longrightarrow} l_1', l_2 \overset{a,G_2,R_2}{\longrightarrow} l_2'}{l_1|^P_A|l_2 \overset{a,G_1\times G_2,R_1\times R_2}{\longrightarrow} l_1'|^P_A|l_2'}(a \in A),$$

$$r2.\frac{l_1 \overset{a,G_1,R_1}{\longrightarrow} l_1', l_2 \overset{\bar{a},R_2}{\longrightarrow} l_2'}{l_1|^P_A|l_2 \overset{a,G_1\times Id,R_1\times R_2}{\longrightarrow} l_1'|^P_A|l_2'}(a \notin A),$$

$$r3.\frac{l_1 \overset{a,G_1,R_1}{\longrightarrow} l_1', l_2 \overset{\bar{a}}{\not\longrightarrow}}{l_1|_A^P|l_2 \overset{a,G_1\times Id,R_1\times Id}{\longrightarrow} l_1'|_A^P|l_2}(a \notin A).$$

$$r4.\frac{l_1 \overset{\bar{a},R_1}{\longrightarrow} l_1'}{l_1|_A^P|l_2 \overset{\bar{a},R_1\times Id}{\longrightarrow} l_1'|_A^P|l_2}(\bar{a} \notin P).$$

$$r5.\frac{l_1 \overset{\bar{a},R_1}{\longrightarrow} l_1', l_2 \overset{\bar{a},R_2}{\longrightarrow} l_2'}{l_1|_A^P|l_2 \overset{\bar{a},R_1\times R_2}{\longrightarrow} l_1'|_A^P|l_2'}(\bar{a} \in P),$$

$$r6.\frac{l_1 \overset{\bar{a},R_1}{\longrightarrow} l_1', l_2 \overset{\bar{a}}{\not\longrightarrow}}{l_1|_A^P|l_2 \overset{\bar{a},R_1\times Id}{\longrightarrow} l_1'|_A^P|l_2}(\bar{a} \in P)$$

$$r7.\frac{l_1 \overset{\lambda,a,R_1}{\longrightarrow} l_1'}{l_1|_A^P|l_2 \overset{\hat{\lambda},a,R_1\times Id}{\longrightarrow} l_1'|_A^P|l_2},$$

where for $\xi = (\xi_1, \xi_2)$, $\hat{\lambda}(\xi) = \lambda(\xi_1)$. For an explanation of the communication possibilities (and the composition rules) of the $|_A^P|$ operator, we refer to [33].

We chose to define a guard (of an active transition) as a subset of the output space. On the one hand, this can be seen as a restriction with respect to state-guards (although we can always make output-copies of state variables). On the other hand, the notion of output-guards might be more appropriate when dealing with equivalences that depend on external behaviors (like bisimulation).

We chose not to assigns guards or value-passing to passive transitions. This is to keep things as simple as possible. Technically it is possible to assigns guards to passive transitions, but this will blow up the number of composition rules and the number of transitions in the composite system: Two passive transitions with shared event $\bar{a}$ and with guards G_1 and G_2 will result in three transitions with guards $G_1 \times G_2$, $G_1 \times \neg G_2$ and $\neg G_1 \times G_2$. Synchronization of two active transitions with guards always results in only one transition. A similar thing happens if we assign value-passing to passive transitions. We might consider using (output) value passing for spontaneous transitions.

3.2 Free Flight in air traffic example

In this section we give a CPDP model for a part of the Pilot-flying-DCPN model. See [25] and [10] for more information about this model and its broader context. This model consists of several local Petri nets (LPNs). We concentrate on one local Petri net called *Current goal*. This LPN interacts with several other LPNs from Pilot-flying and also with LPNs outside Pilot-flying (for example with the LPN *HMI PF* from the DCPN called ACCMS). We chose for concentrating on *current goal* because there are interesting interactions present in this LPN and the question is whether we can model these interactions as well in CPDP or not. In our CPDP model we model the component *current goal* almost as detailed as the DCPN

model. For the rest, we only model LPNs that are connected to the *current goal*. These components will be modelled very abstractly (only the parts that are interesting as far as *current goal* is concerned).

3.2.1 System description

We describe component *Current goal* detailed and we describe components *Task performance*, *Memory*, *Audio alert* and *HMI-PF* abstractly. We start with the abstract components and end with *Current goal*.

Component *HMI-PF* is a failure-indicator. There are five systems that can fail: Engine, navigation, ASAS, ADS-B Rec, ADS-B Transm. The *HMI-PF* system itself can also fail, then we say that it is in non-working mode (otherwise it is in working mode). If one (or more) of the five systems fail, this can be seen by the pilot from this *HMI-PF* component.

Component *Audio alert* sends a signal as soon as there is an alert. In the signal is coded which goal needs to be achieved by the pilot: C1 - collision avoidance, C2 - emergency actions, C3 - conflict resolution, C4 - navigation vertical, C5 - navigation horizontal, C6 - preparation route change, C7 - miscellaneous. In case of C2, it is also specified which failure causes the emergency (this can be one of the five systems indicated by *HMI-PF* or failure number six: other emergency).

Component *Memory* memorizes which goals need to be achieved. If the pilot is busy achieving a goal while *Audio alert* signals that another goal (with lower priority than the current goal) needs to be achieved, then this new goal can be stored in the memory. As soon as the pilot is ready with the current goal, he can retrieve from the memory which subset of C1 till C7 needs to be done (and in case C2 which failure is present).

Component *Task performance* is the largest LPN of the DCPN *Pilot flying*, it monitors which task is being performed by the pilot. It has many interactions with other LPNs (inside and outside *Pilot flying*). We only describe it abstractly. In order to achieve one goal, the pilot needs to perform six tasks: T1 - monitoring, T2 - monitoring and decision, T3 - coordination, T4 - execution, T5 - execution monitoring, T6 - monitoring and goal prioritization. For reasons of convenience we say that when the pilot is ready with the six tasks (and waiting for the next goal to be achieved) he is performing task T7 - end task. Because there are seven goals, each consisting of seven tasks, it is clear why this component has a large state space and many interactions. In our CPDP-model we abstract the seven times seven states back to two states: A endtask-state (the pilot is not performing a task) and a task-state (the pilot is performing some task for some goal).

Component *Current goal* monitors which goal (C1 - C7) is now being achieved by the pilot. This component has interaction with (and only with) the four components described above. Thus, the four components described above form (abstractly) the entire interaction-environment of *current goal*. The state of *current goal* reflects the current goal which is one of C1 - C7 and in case C2 it also reflects the failure that is currently being resolved. The state of this component can be changed in two ways:

1. A signal is received from component *Audio alert*: If the goal that is coded in this signal has higher priority (C1 is highest, C7 is lowest) than the goal that is currently worked on, then the pilot will stop achieving the current goal and will switch to the new goal from *Audio alert*. If the goal from the signal has lower priority than the current goal, then the pilot will not switch and the new goal from *Audio alert* will be stored in memory.

2. The pilot is ready for achieving a new goal (i.e. *Task performance* is in state *endtask*): Then, if *HMI-PF* is indicating that there are no failures, the pilot can read the memory to see if there are goals on stack that need to be achieved. If this stack is not empty, the pilot chooses a goal from this stack randomly (if there is more than one goal in the stack).

If the *Current goal* state changes, then this needs to be communicated to the component *Task performance* which has then to jump to the first task of the new goal. If we sum up the interactions of component *Current goal*, then, at the abstraction level of our model, this component receives information from components *HMI-PF* (are there some indicators on?), *Audio alert* (is there an alert signal?), *Memory* (what is on stack?) and *Task performance* (are we in state *endtask* and thus ready for a new goal?). And component *Current goal* sends information to the components *Memory* (if a signal with lower priority is received from *Audio alert*) and *Task performance* (if the *Current goal* state has changed and new tasks need to be performed).

3.2.2 The CPDP model

We model component *Current goal* and its abstracted interaction-environment as five value-passing CPDPs which we interconnect via composition operators of the $|^P_A|$ type. The graphical representation of the five CPDPs can be seen in Figure 3.1. We now specify the CPDPs *HMI-PF*, *Audio alert*, *Memory*, *Task performance* and *Current goal*, which are interconnected as follows:

$$((CurrentGoal||_{A_1}AudioAlert)||_{A_2}Memory)||_{A_3}TaskPerformance,$$

with $A_1 := \{alert\}$, $A_2 := \{getmem, storemem\}$ and $A_3 := \{alertchng, memchng\}$.

CPDP *HMI-PF* has one location with one variable named C_{HMI}. The value of this variable indicates whether there is a failure in one of the five systems (indicated by *HMI-PF*). C_{HMI} consists of five components C^i_{HMI} $(i = 1, 2, 3, 4, 5)$ which all have either value $true$ or $false$ (with $true$ indicating a failure for the corresponding system). There is only one transition which is an unguarded active transition from the only location to itself with label $getHMI$ and with output C_{HMI}. This transition is used only to send the state information to the component *Current goal*, therefore the reset map of this transition does not change the state C_{HMI}. Note that for the CPDPs in this ATM-example, we do not define output variables. We assume that for every state variable used in active transitions we have an output variable copy defined.

CPDP *Audio alert* has one location with two variables named k and q. $k \in \{1, 2, 3, 4, 5, 6\}$ and $q \in \{1, 2, 3, 4, 5, 6\}$. These values represent the interrupt goal (and failure in case $k = 2$).

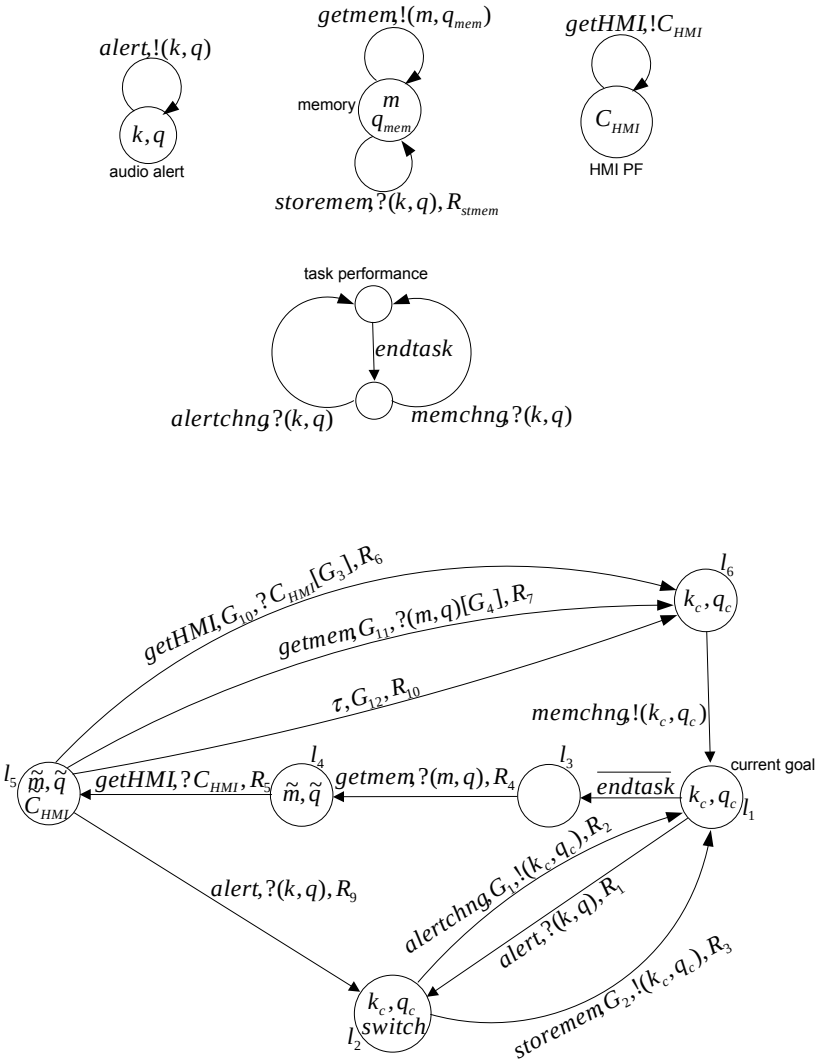

Figure 3.1: CPDP pilot flying model

There is one active transition with label *alert* and with outputs k and q. This transition should normally be guarded (where the guard is satisfied as soon as an alert signal should be sent), but at the abstraction level of our model we do not model this. Also the reset map of this transition is not specified here.

CPDP *Memory* has one location with two variables named m and q_{mem}. m is a variable with seven components (m_1 till m_7 for the goals C1 till C7) which can have value ON and OFF. (In the DCPN there is also the value $LATER$ for m_4 and m_5 which we do not consider in the CPDP). q_{mem} is a variable with six components (for the six failures) taking values in $\{0, 1\}$. There are two active transitions. The unguarded transition with label *getmem* and output m and q_{mem} is used to send information to *Current goal*, therefore the reset map leaves the state unaltered. The unguarded transition with label *storemem* and input k and q is used by *Current goal* to change the memory state. (Note that we write $?(k, q)$ to denote

inputs of the combined state-space of k and q which is $?\mathbb{R}^2$ because $k, q \in \mathbb{R}$). The reset map R_{stmem} of this transition changes m_k (with k the received input) to ON and changes q^q_{mem} (with q the received input) to 1.

CPDP *Task performance* has two locations, $Idle$ and $Busy$, both without variables. When the system switches from $Busy$ to $Idle$, the active transition with label *endtask* is executed. The system can switch from $Idle$ to $Busy$ via two transitions: 1. Via the active input transition with label *alertchng* and inputs k and q. This happens when *Current goal* executes an active output transition with label *alertchng* due to having received a signal from *Audio alert*. (Normally *Task performance* should use the information from the inputs k and q via the reset map of the transition, but we do not model that at our level of abstraction). 2. Via the active input transition with label *memchng* and inputs k and q. This happens when *Current goal* executes an active output transition with label *memchng* due to the situation where the pilot is idling and a new goal is retrieved by *Current goal* from the memory.

CPDP *Current goal* is the only CPDP that we have modelled in detail. *Current goal* has six locations, named l_1 till l_6. We will now describe each location:

- Location l_1 has two variables named k_c and q_c. The process is in this location when one of the goals is being achieved (i.e. *Task performance* is in location $Busy$) and the values of k_c and q_c represent the current goal and (in case $k_c = 2$) current failure. There are two outgoing transitions: 1. An unguarded active input transition to l_2 labelled *alert* with inputs k and q, synchronizing on an *alert* signal from *Audio alert*, with reset map

$$R_1 := \begin{cases} k_c := k, q_c := q, switch := true & \text{if k} < \text{k}_\text{c} \\ k_c := k_c, q_c := q_c. switch := false & \text{else.} \end{cases}$$

  2. A passive transition to l_3 labelled $\overline{endtask}$, synchronizing on an *endtask* signal from *Task performance*.

- The process is in location l_2 if (1) after having received the *alert* signal, the current goal needs to be changed (according to the *alert* signal) or (2) the interrupt goal (from the *alert* signal) needs to be stored in memory. (1) is the case when $switch = true$, (2) is the case when $switch = false$. Therefore, $G_1 := \{(k_c, q_c, switch) | switch = true\}$, $G_2 := \{(k_c, q_c, switch) | switch = false\}$, with G_1 the guard of the active output transition labelled *alertchng* with outputs k_c and q_c and reset map R_2 and with G_2 the guard of the active output transition labelled *storemem* with outputs k_c and q_c and reset map R_3. R_2 and R_3 are the same and do the following reset: $k_c := k_c$, $q_c := q_c$. Note that, under maximal progress, the process jumps immediately to location l_1 as soon as it arrives in location l_2, causing also a synchronizing transition in either *Task performance* (with label *alertchng*) or *Memory* (with label *storemem*).

- The process arrives in location l_3 after the *endtask* signal, which means that the pilot should check the memory whether there are other goals that need to be achieved. With the unguarded active input transition with label *getmem* and inputs m and q and reset map R_4, the process jumps to location l_4 while retrieving the memory state (m, q). The reset map R_4 stores this (m, q) in $(\tilde{m}, \tilde{q})$.

- Before executing a goal from the memory, the pilot should first check *HMI-PF* to see whether there are indications for failing devices. This happens in the transition to l_5 on the label $getHMI$ while retrieving the *HMI-PF* state C_{HMI}. The reset R_5 stores C_{HMI} together with $\tilde{m}$ and $\tilde{q}$ in the state of l_5.

- From location l_5 there is an active transition to l_6 with label τ and guard $G_{12} := \{(\tilde{m}, \tilde{q}, \tilde{C}_{HMI})|\ \tilde{C}^i_{HMI} = true$ for some $i = 1, 2, 3, 4, 5$ or $\tilde{m}^i = ON$ for some $i < 7\}$. Under maximal progress, this τ-transition is taken immediately after arriving in l_5 when the *Memory* and *HMI-PF* states give reason to work on a new goal. The reset map R_{10} resets $k_c := 2, q_c := r$ if $S := \{i|i \leq 5, \tilde{C}^i_{HMI} = true\} \neq \emptyset$, where r is randomly chosen from the set S, otherwise R_{10} resets $k_c := \min\{i|m_i = ON\}, q_c := 0$. If the guard G_{12} is not satisfied in l_5, then this means that the pilot should wait until an *alert* signal is received or until either the *Memory* state or the *HMI-PF* state changes such that the pilot should work on a new goal. On an *alert* signal from *Audio alert* the transition to l_2 is taken where R_9 is equal to R_1. The active input transition to l_6 labelled *getmem* waits till the *Memory* state has changed such that the input-guard G_4 is satisfied, where $G_4 := \{(m, q)|m^i = ON$ for some $2 \neq i < 7\}$. The reset map R_7 resets $k_c := \min\{i|m_i = ON\}, q_c := 0$. The active input transition to l_6 labelled $getHMI$ waits till the *HMI-PF* state has changed such that the input-guard G_3 is satisfied, where $G_3 := \{C_{HMI}|C^i_{HMI} = true$ for some $i = 1, 2, 3, 4, 5\}$. The reset map R_6 resets $k_c := 2, q_c := r$ with r randomly chosen from $S := \{i|i \leq 5, \tilde{C}^i_{HMI} = true\} \neq \emptyset$.

- If the process arrives in location l_6, then this means that the state of l_6 represents the goal that should immediately be worked on by the pilot. Therefore, the unguarded active transition to l_1 labelled *memchng* is taken immediately (under maximal progress). The outputs k_c and q_c are accepted by the *memchng* transition in *Task performance*. The reset map of the output *memchng* transition copies the state of l_6 to the state of l_1.

3.2.3 Discussion

We now discuss the CPDP 'pilot flying' model. We compare it to the DCPN model and we discuss the suitability of the value-passing concept for the kinds of interactions that are present in the DCPN model.

Besides mere sending and receiving of data, a value-passing transition in one component can also be used to put guards on the outputs of other components. These guards are called input-guards. In Figure 3.1 the input-guards are G_3 and G_4. The transition with G_3 can only be taken if a process in some other component satisfies guard G_3. (In this case the other component is *HMI-PF*). One could question whether it is possible to specify location l_5 (and its outgoing transitions) if we are not allowed to use input-guards: If we are not allowed to use input-guards, then the guard G_3 should be modelled within the component *HMI-PF* such that as soon as guard G_3 switches from not-satisfied to satisfied (or vice versa) an active signal should be broadcast by *HMI-PF*. This signal can then be received via a passive (or active) transition by *Current goal* who then knows that guard G_3 has become satisfied. There seems to be a trade-off here. If we allow input-guards in the CPDP-framework, then modelling becomes easier (we need less transitions) but the CPDP-language becomes more complex which might be disadvantageous for (compositional) analysis of CPDP processes.

If we compare the *Current goal* CPDP with the *Current goal* LPN then we can roughly say that the $G_{interrupt}$ transition in the LPN corresponds to the CPDP cycles $l_1 \xrightarrow{alert} l_2 \xrightarrow{alertchng} l_1$ and $l_1 \xrightarrow{alert} l_2 \xrightarrow{storemem} l_1$. while the $G_{subsequent}$ transition in the LPN corresponds to the cycles $l_1 \xrightarrow{\overline{endtask}} l_3 \xrightarrow{getmem} l_4 \xrightarrow{getHMI} l_5 \xrightarrow{\alpha} l_6 \xrightarrow{memchng} l_1$, where $\alpha \in \{\tau, getmem, getHMI\}$. Here we see the same kind of trade-off that we saw with the input-guards: To model the complex DCPN transition $G_{subsequent}$ in CPDP, we need transitions like $\overline{endtask}$, $getmem$ and $getHMI$ to gather specific information from other components, while in the DCPN, this information is immediately available (via enabling arcs) to the transition (without using extra transitions). The more complex and more expressive interaction possibilities in DCPN make easy modelling (with less transitions) while the restricted interaction possibilities of CPDP make modelling less easy (although possible) but makes compositional analysis easier.

An automata-analogy of the direct access to state (or output) values of other components present in DCPN transitions is the idea of input/output variables (as for example in [20]). If we then connect two automata where X is an input variable of the first automaton and an output variable of the second automaton, then the first automaton has direct access to the values of X in the other automaton. If we import this idea into CPDP and make m, q_{mem} and C_{HMI} input variables of *Current goal*, then *Current goal* always knows the values of these variables, and transitions used for gathering information (like the ones with $getmem$ and $getHMI$ between l_3 and l_5) are not necessary anymore. Thus, importing this idea into CPDP makes modelling of certain types of interaction much easier. The question is however, what price do we have to pay for importing such a strong/expressive interaction-mechanism? We think that such a strong mechanism is to strong to allow development of compositional analysis techniques. To make the mechanism less strong we could restrict the possibilities of using the data of input variables. For example, if we allow that input-data is used in guards and reset maps but we do not allow that input-data is used in the differential equations, then the expressiveness of the mechanism is decreased in a significant way which perhaps makes the trade-off question interesting again?

If we use input/output variables in CPDP, then the component *Current goal* of Figure 3.1 could be modelled as simple as it is pictured in Figure 3.2 (where we assume that *Current goal* has access to the variables m and q_{mem} and C_{HMI}). Then R_1 resets $k_c := 7$. $k_c = 7$ is then part of G_1. The rest of G_1 are conditions on the variables m, q_{mem} and C_{HMI} such that a new goal can be started. Then R_2 resets k_c and q_c according to the new goal. Thus, by using input/output variables, the transition structure of *Current goal* can be much simplified, but in order to do that, we need to allow communication/interaction outside the transition structures (i.e. in value-passing, we keep the communication restricted to the synchronization of transitions).

3.3 Value-passing CPDPs and PDPs

Value-passing CPDPs exhibit non-determinism. There are two sources for this non-determinism. The first source are the guards. A guard says when a transition may be taken, but it does not say when it has to be taken. By using the maximal progress strategy, we can resolve the non-determinism caused by the guards. The second source is the fact that multiple transitions

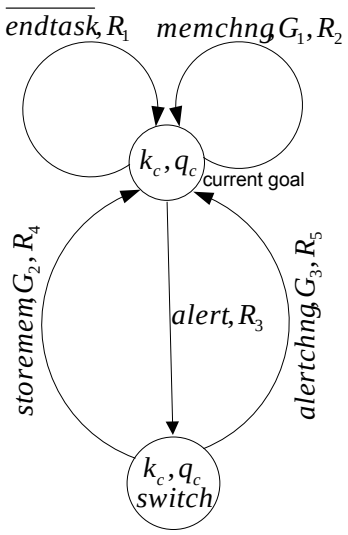

Figure 3.2: Current goal with input/output variables

may be enabled at the same state (also in case of using maximal progress). To resolve this non-determinism we could use an 'adversary' which probabilistically chooses a transition at any state of jumping. More formally said, an adversary assigns to each state a probability measure on the set of enabled active transitions of that state. (The choice function as defined in CPDP (without value-passing) is an adversary).

Does, under maximal progress, a closed value-passing CPDP with adversary behave like a PDP? We claim that, for CPDPs that have no spontaneous transitions, this is the case if the following two conditions are satisfied

1. There are no input-transitions
2. There exists an $N \in \mathbb{N}$ such that each multi-step transition consists of less than N single transitions.

Note that the paths of a CPDP have the cadlag-property: if there is multi-step transition, then the value of the path at that time instant is the target state after the last transition. Is there a way to automatically check the second condition? We now give an algorithm which can check this claim and which also shows that under this condition (assumed that there are no input-transitions) the CPDP exhibits a PDP behavior. (It is clear that if one of the two conditions above is not satisfied, the CPDP does not exhibit a PDP behavior).

Suppose we have a CPDP X with location set L and active transition set T. The CPDP operates under maximal progress and under adversary A. We write $A_x(\alpha)$ for the probability that active transition α is taken when an active transition is executed at state x. We assume that the CPDP has no spontaneous transitions. The case 'with spontaneous transitions' is treated at the and of this section. For any $n \in \mathbb{N}$ we now define T_s^n and T_u^n. T_s^n is a set of stable transitions of multiplicity n and T_u^n is a set of unstable transitions of multiplicity n. A stable transition is a transition that always jumps to the stable state-space of the target

location. The stable state-space is the part of the state-space where no guards are satisfied. An unstable transition always jumps to the unstable state-space (which is the complement of the stable state-space). We show that any transition (of any multiplicity) can be split up into stable and unstable transitions (without changing the stochastic behavior).

We introduce the concept of total reset map. $R_{tot}(B, x)$ denotes the probability of jumping into $B \in \mathcal{B}(E)$ when an active jump takes place at state x. We have that

$$R_{tot}(B, x) = \sum_{\alpha \in \{l_x \to\}} [A_x(\alpha) R_\alpha(B \cap Inv(l'_\alpha), x)],$$

where $\{l_x \to\}$ is the set of all active transitions that leave the location of x. We split R_{tot} up into the stable total reset map $R_{tot,s}$ and the unstable total reset map $R_{tot,u}$, where $R_{tot,s}(B, x) = R_{tot}(B \cap E_s, x)$ and $R_{tot,u}(B, x) = R_{tot}(B \cap E_u, x)$, where E_s and E_u denote respectively the stable and unstable state-spaces of the state-space E of X.

Take any $\alpha \in T$. We define G_{α_s} as the set of all $x \in G_\alpha$ such that $R_\alpha(Inv_s(l'_\alpha), x) \neq 0$, where $Inv_s(l'_\alpha)$ is the stable part of the invariant of the target location of α. Then for all $x \in G_{\alpha_s}$ we define

$$R_{\alpha_s}(B, x) := \frac{R_\alpha(B \cap Inv_s(l'_\alpha), x)}{R_\alpha(Inv_s(l'_\alpha), x)},$$

$$A_x(\alpha_s) := A_x(\alpha) R_\alpha(Inv_s(l'_\alpha), x).$$

If $G_{\alpha_s} \neq \emptyset$, then we add transition α_s with guard G_{α_s} and reset map R_{α_s} to T_s^1. The adversary works on α_s as $A_x(\alpha_s)$ (as defined above).

We define G_{α_u} as the set of all $x \in G_\alpha$ such that $R_\alpha(Inv_u(l'_\alpha), x) \neq 0$. For all $x \in G_{\alpha_u}$ we define

$$R_{\alpha_u}(B, x) := \frac{R_\alpha(B \cap Inv_u(l'_\alpha), x)}{R_\alpha(Inv_u(l'_\alpha), x)},$$

$$A_x(\alpha_s) := A_x(\alpha) R_\alpha(Inv_u(l'_\alpha), x).$$

If $G_{\alpha_u} \neq \emptyset$, then we add transition α_u with guard G_{α_u} and reset map R_{α_u} to T_u^1. The adversary works on α_u as $A_x(\alpha_u)$ (as defined above). It can be easily seen that replacing α by α_s and α_u, does not change the total stable and total unstable reset map. Then, the total stable (unstable) reset map is constructed only from the stable (unstable) transitions.

We introduce the following notations. $P_x(B \circ \beta \circ \alpha)$ denotes the probability that, given that an active jump takes place at state x, transition α is executed followed directly by transition β jumping into the set $B \in \mathcal{B}(Inv(l'_\beta))$. It can be seen that we then have

$$P_x(B \circ \beta \circ \alpha) = A_x(\alpha) \int_{\hat{x} \in G_\beta} A_{\hat{x}}(\beta) R_\beta(B, \hat{x}) \mathrm{d}R_\alpha(\hat{x}, x).$$

We will now inductively determine the sets T_s^n and T_u^n. Suppose the sets T_s^{n-1} and T_u^{n-1} and T_s^1 and T_u^1 are given. Now, for any $\alpha \in T_u^{n-1}$, $\beta \in T_s^1 \cup T_u^1$ such that $l'_\alpha = l_\beta$, we define $G_{\beta \circ \alpha}$ as all $x \in G_\alpha$ such that $R_\alpha(G_\beta, x) \neq 0$. Then, for all $x \in G_{\beta \circ \alpha}$ we define

$$A_x(\beta \circ \alpha) := P_x(Inv(l'_\beta) \circ \beta \circ \alpha),$$

$$R_{\beta\circ\alpha}(B,x) := \frac{P_x(B\circ\beta\circ\alpha)}{A_x(\beta\circ\alpha)}.$$

If $G_{\beta\circ\alpha} \neq \emptyset$ and $\beta \in T_s^1$ then we add transition $\beta\circ\alpha$, with guard, reset map and adversary as above, to T_s^n. If $G_{\beta\circ\alpha} \neq \emptyset$ and $\beta \in T_u^1$ then we add transition $\beta\circ\alpha$, with guard, reset map and adversary as above, to T_u^n. If for $z \in \{s,u\}$ we define

$$R_{tot,z}^n(B,x) := \sum_{\alpha\in\{l_x\rightarrow\}\cap T_z^n} [A_x(\alpha)R_\alpha(B\cap Inv(l'_\alpha),x)],$$

then it can be seen that for any $n \in \mathbb{N}$ we have

$$R_{tot}(B,x) = \sum_{i=1}^{n}[R_{tot,s}^i(B,x)] + R_u^n(B,x),$$

with other words, if X^n denotes a copy of CPDP X, except that the active transition set of X^n equals $T_s^1 \cup T_s^2 \cup \cdots \cup T_s^n \cup T_u^n$, then the total reset maps of X^{n-1} and X^n are the same for all n.

Theorem 2. *Let X^n be derived from X as above. Let $R_{tot,s}^n$ denote the total stable reset map of X^n. X exhibits a PDP behavior if and only if $R(E,x) := \lim_{n\rightarrow\infty} R_{tot,s}^n(E,x) = 1$ for all $x \in E_u$. Then, the corresponding PDP of X has total reset map R.*

Proof. From the text above, it is clear that if $R(E,x) = 1$ for all x, then this is the total reset map of the PDP corresponding to X. If for some $x \in E$, $R(E,x) < 1$, then it can be seen that this must mean that there exists an active transition α in X with multiplicity infinity such that $A_x(\alpha) > 0$. This means that (from x) there is a deadlock probability (i.e. time does not progress anymore) greater than zero, which means that X does not exhibit a PDP behavior. □

Corollary 1. *If for some $n \in N$ we have that $T_u^n = \emptyset$, then the multiplicity of the transitions of X is bounded by n and X exhibits a PDP behavior, where the corresponding PDP is equal to the corresponding PDP of X^n (which can be constructed because all transitions of X^n have multiplicity one).*

Now we can say a few things about the case that the CPDP X has spontaneous transitions. Suppose that the multiplicity of the active transitions of X is bounded by n. Let $\hat{X}^n$ be a copy of X^n together with the following spontaneous transitions: For any spontaneous transition (l,λ,l',R) of X we add to $\hat{X}^n$ the transition $(l,\lambda,L,\hat{R})$, where, for $B \in \mathcal{B}(E)$,

$$\hat{R}(B,x) := R(B\cap Inv_s(l')) + \sum_{\alpha:l\rightarrow}\int_{\hat{x}\in G_\alpha} A_{\hat{x}}(\alpha)R_\alpha(B\cap Inv(l'_\alpha))\mathrm{d}R(\hat{x},x),$$

where $\alpha : l \rightarrow$ denotes the set of all active transitions of X^n that leave location l. Note that $(l,\lambda,L,\hat{R})$ is not a standard CPDP transition, but a transition that represents a Poisson process in location l with jump-rate λ and with reset map $\hat{R}$, which can jump to multiple locations. Therefore we write L instead of l' in the tuple of the transition. It can be seen now that the corresponding PDP of X is the corresponding PDP of X^n together with these non-standard spontaneous transitions.

3.3.1 Examples of value-passing-CPDP to PDP conversion

We follow the algorithm written above to check whether the CPDP ATM-example of Section 3.2, which has no spontaneous transitions, can be converted to a PDP.

Example (ATM). We assume that the system is closed (i.e. no more components will be connected). This means that we remove the passive transitions in the composite CPDP (which are some $\overline{endtask}$ transitions). It can be seen that the composite CPDP does not have active input-transitions. We assume that time will elapse in the locations of *audioalert* and $taskperformance$. Both may have (different) extra dynamic of the form $\dot{x} = f(x)$, then the guards of transitions *alert* and *endtask* depend on x. We assume that the transitions *alert*, *alertchng* and *memchng* are stable. Note that location l_1 is stable, that locations l_2,l_3,l_4 and l_6 are unstable and that location l_5 has both a stable and an unstable state space.

First we look at T_s^1: the stable parts of the transitions of multiplicity one. For this example we have

$$T_s^1 = \{storemem, alertchng, memchng, getHMI_{s,45}\},$$

where these names correspond to the transitions with the same label: *storemem* represents the transition from l_2 to l_1 synchronized with the transition with the same label in component *memory*. $getHMI_{s,45}$ corresponds to the stable part, which is the part that does not jump into guard G_{12}, of the transition between l_4 and l_5 synchronizing with the transition in *HMI-PF*, etc. Because R_5 makes a copy of C_{HMI},m and q, we get that the guard of $getHMI_{s,45}$ equals $Inv(l_4)\backslash G_{12}$ and the guard of $getHMI_{u,45}$, the unstable part, equals G_{12}. Furthermore, we have for this example

$$T_u^1 = \{alert_{12}, alert_{52}, getmem_{34}, getmem_{56}, getHMI_{u,45}, getHMI_{56}, endtask\},$$

$$T_s^2 = \{alertchng \circ alert_{12}, alertchng \circ alert_{52}, storemem \circ alert_{12}, storemem \circ alert_{52},$$
$$memchng \circ \tau, memchng \circ getHMI, memchng \circ getmem, getHMI_s \circ getmem\},$$

where $getHMI_s \circ getmem$ denotes the transition with multiplicity two that consists of *getmem* from l_3 to l_4 followed directly by the stable part of $getHMI$ from l_4 to l_5, etc.

$$T_u^2 = \{getmem \circ endtask, getHMI_u \circ getmem, \tau \circ getHMI\},$$

$$T_s^3 = \{memchng \circ \tau \circ getHMI_u, getHMI_s \circ getmem \circ endtask\},$$
$$T_u^3 = \{getHMI_u \circ getmem \circ endtask, \tau \circ getHMI_u \circ getmem\},$$
$$T_s^4 = \{memchng \circ \tau \circ getHMI_u \circ getmem\},$$
$$T_u^4 = \{\tau \circ getHMI_u \circ getmem \circ endtask\}.$$
$$T_s^5 = \{memchng \circ \tau \circ getHMI_u \circ getmem \circ endtask\},$$
$$T_u^5 = \emptyset.$$

We see, when X denotes the composite CPDP, that X^5 (i.e. the CPDP that has active transitions $(\cup_{i=1}^5 T_s^i) \cup T_u^5$) has no unstable transitions. This means that X^5 can directly be converted to a PDP, which then is the corresponding PDP of X.

To prove that the composite CPDP of this ATM example can be converted to a PDP, it would also have been enough to show that the CPDP does not have cycles such that the locations of the cycle all have unstable parts. It is clear that a cycle in component *Current goal* should include location l_1, which is a stable location. It can easily be seen that in the composite CPDP the two (product)locations that contain l_1 are both stable and that any cycle in the composite CPDP should contain one of these two locations. Therefore this composite CPDP does not have transitions with multiplicity infinity and should therefore be convertable to a PDP. (However, if we want to specify this PDP, we still have to do the algorithm or something similar). □

Because the algorithm terminates on the ATM-example above, we know that the ATM-example has a PDP behavior. However, it is possible that the algorithm does not terminate, while the CPDP does exhibit a PDP behavior. We now give an example of this.

Example. Let CPDP X have one location, l_1. The state-space of l_1 is $[0,1]$, the continuous dynamics of l_1 is the clock dynamics, i.e. $\dot{x} = 1$. From l_1 to l_1 there is one active transitions with guard G and reset map R. $G = [\frac{1}{2}, 1]$. For $x \in G$, $R(\{0\}, x) = \frac{1}{2}$ and $R(A, x) = |A \cap [\frac{1}{2}, 1]|$ for $A \in \mathcal{B}([0,1] \backslash \{0\})$. This means that from an x in G, the reset map jumps to 0 with probability $\frac{1}{2}$ and jumps uniformly into $[\frac{1}{2}, 1]$ with probability $\frac{1}{2}$. It can easily be seen that for X we have that $T_u^n \neq \emptyset$ for all $n \in \mathbb{N}$. This means that the algorithm explained above does not terminate for this example. Still, according to Theorem 2, X expresses a PDP behavior, because for $x \in G$, $R([0,1], x) = \lim_{n \to \infty} R_{tot,s}^n([0,1], x) = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} + \cdots = 1$. □

Chapter 4

Stability Analysis for Hybrid Automata using Conservative Gains

4.1 Introduction

The study of hybrid systems, which are systems whose behavior can be seen as the result of the interaction between discrete and continuous dynamics, has given rise to a wealth of different models (see [31] for an overview). The hybrid automata model [2, 12] is an attractive candidate as it is very general and can be seen as a straightforward extension of the timed automata model [3] that has been extensively studied, especially in computer science, resulting in a large body of results on (automated) analysis and design. A hybrid automaton consists of an automaton with locations and transitions (or switches) between the transitions, together with continuous dynamics in the locations (usually described by differential equations) and constraints on both locations and transitions.

The question of stability for hybrid automata has been known to be nontrivial. It is possible to find very simple examples (see e.g. [5]) that illustrate that even if the dynamics in each location is stable, still the global behavior of the hybrid automaton may be unstable. By now there are many results on the stability of hybrid systems (for an overview see [8, 16, 23]). Many of these results address the question of the stability of a set of locations under arbitrary switching between these locations [23, 39], for instance by checking the existence of a Lyapunov function common to all locations [16, 1]. Such a criterion would be too strong in general for hybrid automata, since there one is not interested in all possible switchings, but only in the switchings allowed by the transitions of the automaton.

A more suitable criterion for our purposes is that of multiple Lyapunov functions [5, 6, 16]. Here one assumes the existence of a Lyapunov function in each location, together with the so-called non-increasing sequence property: if a location is visited again, the value of the Lyapunov function should be less than it was at the previous time the location was entered. This is a sufficient condition for the stability of a hybrid automaton. However, the problem is that is not clear how the non-increasing sequence can be effectively checked in general, since it would in principle require checking all possible behaviors of a hybrid system.

Therefore we take another approach that exploits the automaton structure of the hybrid automaton while taking advantage of the existence of a Lyapunov function in each location. In Section 4.2 we give some basic definitions about hybrid automata and stability, and define the class of hybrid automata that we study. In Section 4.3 we show how to calculate a

conservative estimate of the gain using Lyapunov functions. It is shown how an optimal Lyapunov function can be chosen. In Section 4.4 we show how to use conservative gains in constructing an automaton that is used in an algorithm that detects non-contractive cycles in a hybrid automaton. The absence of such non-contractive cycles is a sufficient condition for the stability of the hybrid automaton.

4.2 Hybrid automata and stability

The hybrid automaton model [2, 12] extends the classical notion of automaton by incorporating continuous dynamics in the locations, together with constraints at both locations and transitions.

Definition 4.2.1. A *hybrid automaton* is a tuple $H = (X, L, Init, Inv, f, E, Guard, Assign, \Sigma)$ where:

- $X \subseteq \mathbb{R}^n$ is the *continuous state space* ranged over by the state vector x.
- L is a finite set of *locations*.
- $Init \subseteq L \times \mathbb{R}^n$ is a set of *initial location state pairs*.
- $Inv : L \to 2^X$ assigns to each location ℓ an *invariant* to be satisfied by state x while in location ℓ.
- $f : L \to (X \to \mathbb{R}^n)$ assigns to each location ℓ a *continuous vector field* f_ℓ such that the state x should satisfy $\frac{d}{dt}x = f_\ell(x)$.
- $E \subseteq L \times L$ is the set of *transitions*, also called *switches*.
- $Guard : E \to 2^X$ assigns to each transition a *guard* that has to be satisfied by state x if the transition is taken.
- $Assign : E \to (X \to X)$ assigns to each transition an *assignment* that may alter state x when the transition is taken.
- Σ a set of *transition labels*. We assume a labeling function $lab : E \to \Sigma$ and refer to transitions by their labels (assuming uniqueness).

We make a few additional assumptions:

- We assume that $Init = L' \times \mathbb{R}^n$ for a set $L' \subseteq L$ of initial locations, so that for a given initial location we can start with any state (which is technically convenient when studying stability).
- We assume that there are no invariants, i.e., Inv maps each location to the trivial condition *true*. This means that transitions are never forced, and it is possible to remain in a location forever.
- we assume that the dynamics in each location is linear, so $\frac{d}{dt}x = f_\ell(x) = A_l x$, $A_\ell \in \mathbb{R}^{n \times n}$.

- We assume that for each transition e the guard is a hyperplane defined by an equation of the form $v_e^{\mathrm{T}} x = 0$ for some $v_e \in \mathbb{R}^n$.
- We assume that the state is left unchanged by transitions (also called *continuous switching*), so for each transition e, $Assign_e(x) = x$.

We call a hybrid automaton that satisfies these assumptions a *Linear Continuous Hyperplane* (LCH) hybrid automaton.

Example 4.2.2. Consider the hybrid automaton consisting of four locations, $\ell_1, \ldots, \ell_4$. The dynamics in location ℓ_i is given by $\frac{d}{dt}x = A_{\ell_i}x$, $A_{\ell_i} \in \mathbb{R}^{2\times 2}$, $i = 1, \ldots 4$. The following events can occur: $E = \{(\ell_1, \ell_2), (\ell_1, \ell_4), (\ell_2, \ell_1), (\ell_2, \ell_3), (\ell_3, \ell_4), (\ell_4, \ell_2)\}$, to
which correspond labels a to f respectively. To each event there corresponds a switching line L_{ij}. For instance if the automaton is in location ℓ_2 there are two possible transitions: to ℓ_1 and to ℓ_3. The transition to ℓ_1 is enabled if and only if $x \in L_{21}$, whereas the transition to ℓ_3 is possible when $x \in L_{23}$.

Definition 4.2.3. A *hybrid trace* of an LCH hybrid automaton is a finite or infinite sequence of the form $\sigma = x_1 e_1 x_2 e_2 \ldots x_{m-1} e_{m-1} x_m$, with an associated monotonically increasing timing sequence $\tau_0 \tau_1 ... \tau_m$ (with $\tau_0 = 0$, $\tau_i \in \mathbb{R} \cup \{\infty\}$), such that

- each e_i is a transition from location ℓ_i to location ℓ_{i+1}
- each x_i is a mapping from $[\tau_{i-1}, \tau_i]$ to $\mathbb{R}^n$ satisfying $\frac{d}{dt}x_i = A_{\ell_i}x_i$
- initial and switching constraints and assignments are respected, so $(\ell_1, x_1(0)) \in Init$, and for all $1 \leq i \leq m-1$: $v_{e_i}^{\mathrm{T}} x_i(\tau_i) = 0$ and $x_i(\tau_i) = x_{i+1}(\tau_i)$.

Notice that in our setting Zeno behavior cannot occur. The reason for this is that transitions from one location to the other are enabled only if the state has progressed from one hyperplane to the other. The amount of time that this takes is invariant under scaling and is completely determined by the angle between the hyperplanes and the dynamics in the location. Since the number of locations and the number of transition enabling hyperplanes is finite it follows that there is a uniform minimal dwell time.

Definition 4.2.4. An LCH hybrid automaton is *stable* iff $\forall \epsilon > 0\ \exists \delta > 0 : \|x^0\| < \delta \Rightarrow$ for all hybrid traces $x_1 e_1 x_2 e_2 \ldots$ with $x_1(0) = x^0$ and $\forall i \forall t \in [\tau_{i-1}, \tau_i] : \|x_i(t)\| < \epsilon$. An automaton that is not stable is called *unstable*.

It is well known that even if for each location ℓ the dynamics is stable, so the matrix A_ℓ is stable (i.e., all eigenvalues have nonnegative real part, and eigenvalues with zero real part are semisimple, see [27]), still the hybrid automaton can be unstable (see e.g. [5] for a simple example). We say that a hybrid automaton has *stable locations* if for each location ℓ the matrix A_ℓ is stable. Now our problem is the following: find sufficient conditions for the stability of an LCH hybrid automaton with stable locations.

4.3 Conservative estimate of gains via Lyapunov functions

Suppose a location ℓ is entered via a transition a with a state vector x_a and is left via a transition b with a state vector x_b. An indication as to how the location contributes to the

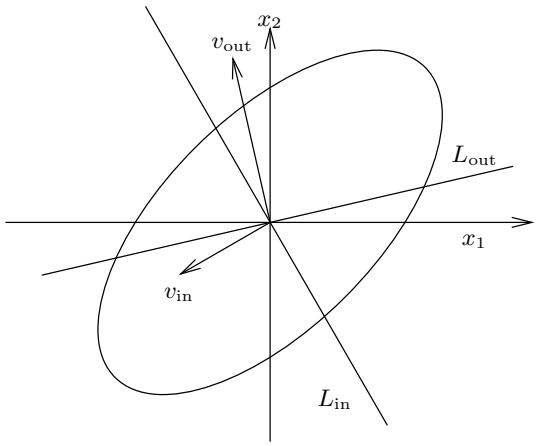

Figure 4.1: Switching lines in a location

stability or instability is the ratio of the norm of the outbound state and the inbound state. A ratio below one is in favor of stability whereas a ratio above one points at instability. Of course, since the ratio depends on the actual trace and state trajectory it does not provide a feasible stability indicator. Therefore we propose to use an upperbound that only depends on the particular location and its corresponding inbound and outbound transitions.

Definition 4.3.1. Consider an LCH hybrid automaton H with stable locations. With each location ℓ we associate a symmetric positive definite matrix P_ℓ such that $A_\ell^{\mathrm{T}} P_\ell + P_\ell A_\ell \leq 0$. Let e_{in} represent a transition to ℓ and e_{out} a transition from ℓ and let L_{in}, given by $v_{\text{in}}^{\mathrm{T}} x = 0$, and L_{out}, given by $v_{\text{out}}^{\mathrm{T}} x = 0$, denote the corresponding switching hyperplanes. Define ellipsoids E_{in} and E_{out} as $E_{\text{in}} = \{x \in L_{\text{in}} \mid x^{\mathrm{T}} P_\ell x = 1\}$ and $E_{\text{out}} = \{x \in L_{\text{out}} \mid x^{\mathrm{T}} P_\ell x = 1\}$. The corresponding gain $\alpha_{\text{in/out}}$ is defined as

$$\alpha_{\text{in/out}} = \max_{x_{\text{i}} \in E_{\text{in}}, x_{\text{o}} \in E_{\text{out}}} \frac{x_{\text{o}}^{\mathrm{T}} x_{\text{o}}}{x_{\text{i}}^{\mathrm{T}} x_{\text{i}}} \tag{4.1}$$

Obviously, since $V(x) = x^{\mathrm{T}} P_\ell x$ is a Lyapunov function for $\frac{d}{dt} x = A_\ell x$ we have that any trajectory that enters the location through L_{in} and leaves it through L_{out} has the property that the ratio of the norms of outbound and inbound states is upperbounded by $\sqrt{\alpha_{\text{in/out}}}$.

Two questions arise. Firstly, given P_ℓ how can we calculate $\alpha_{\text{in/out}}$? Secondly, it appears that the choice of P_ℓ affects $\alpha_{\text{in/out}}$. How to choose P_ℓ such that $\alpha_{\text{in/out}}$ is minimal? The latter question is treated in Section 4.3.2.

4.3.1 Calculation of the gains

Let us explain how the calculation for given P_ℓ works. For ease of exposition we treat the two dimensional case first, see Figure 4.1.

The switching lines are given by $v_{\text{in}}^{\mathrm{T}} x = 0$ and $v_{\text{out}}^{\mathrm{T}} x = 0$ respectively. Let $\tilde{v}_{\text{in}}$ and $\tilde{v}_{\text{out}}$ be orthogonal to v_{in} and v_{out} respectively. Then it is not difficult to verify that

$$\alpha_{\text{in/out}} = \frac{\tilde{v}_{\text{out}}^{\mathrm{T}} \tilde{v}_{\text{out}}}{\tilde{v}_{\text{in}}^{\mathrm{T}} \tilde{v}_{\text{in}}} \frac{\tilde{v}_{\text{in}}^{\mathrm{T}} P_\ell \tilde{v}_{\text{in}}}{\tilde{v}_{\text{out}}^{\mathrm{T}} P_\ell \tilde{v}_{\text{out}}}. \tag{4.2}$$

If we choose $\tilde{v}_{\text{in}}$ and $\tilde{v}_{\text{out}}$ both on the same level curve, then (4.2) reduces to $\frac{\tilde{v}_{\text{out}}^{\text{T}}\tilde{v}_{\text{out}}}{\tilde{v}_{\text{in}}^{\text{T}}\tilde{v}_{\text{in}}}$.

For dimensions $n > 2$ the situation is a bit more complicated since the maximization in (4.2) comes into play.

First notice that

$$\alpha_{\text{in/out}} = \frac{\max_{x_{\text{o}} \in E_{\text{out}}} x_{\text{o}}^{\text{T}} x_{\text{o}}}{\min_{x_{\text{i}} \in E_{\text{in}}} x_{\text{i}}^{\text{T}} x_{\text{i}}}. \tag{4.3}$$

Next, e.g., the numerator of (4.3) can easily be calculated as follows. First assume that v_{out} has norm one (otherwise normalize), then determine an orthogonal matrix V_{out} such that the first column of V_{out} is v_{out}. Define $P_{\text{out}} = V_{\text{out}}^{\text{T}} P_\ell V_{\text{out}}$ and $\tilde{E}_{\text{out}} = \{z \in \mathbb{R}^n \mid z_1 = 0, z^{\text{T}} P_{\text{out}} z = 1\}$. Then

$$\max_{x \in E_{\text{out}}} x^{\text{T}} x = \max_{z \in \tilde{E}_{\text{out}}} z^{\text{T}} z = \frac{1}{\lambda_{\min}(\tilde{P}_{\text{out}})}, \tag{4.4}$$

where $\tilde{P}_{\text{out}}$ is obtained from P_{out} by deleting the first row and and the first column. Furthermore $\lambda_{\min}(P)$ denotes the smallest eigenvalue of matrix P. In a similar way the denominator of (4.3) is obtained, resulting in

$$\alpha_{\text{in/out}} = \frac{\lambda_{\max}(\tilde{P}_{\text{in}})}{\lambda_{\min}(\tilde{P}_{\text{out}})}. \tag{4.5}$$

4.3.2 Optimizing the choice of Lyapunov function

The gains that provide a (conservative) stability indicator obviously depend on the Lyapunov functions in each location. Loosely, the fit of the level curves with respect to the trajectories determines the amount of conservatism. The better the fit, the less conservative the gain. Since Lyapunov functions are not unique, this suggests that we might be able to choose in each location a Lyapunov function that is optimal with respect to the switching planes. In this section we explain how this can indeed be done. We confine ourselves to quadratic Lyapunov functions. We show that for a given stable matrix A the set of quadratic Lyapunov functions is convex and compact. Furthermore the stability gain corresponding to A and given switching hyperplanes depends continuously on the Lyapunov function so that at least the optimum exists. We illustrate the effectiveness of these results by means of a two dimensional example.

Let $A \in \mathbb{R}^{n \times n}$ be a stable matrix. We are interested in the set of quadratic Lyapunov functions, or, more precisely, the set of level curves corresponding to quadratic Lyapunov functions. To enable the comparison of different Lyapunov functions we choose a nonzero $x_0 \in \mathbb{R}^n$ and define the set

$$\Omega_{x_0} = \{P \in \mathbb{R}^{n \times n} \mid A^{\text{T}} P + P A \leq 0, x_0^{\text{T}} P x_0 = 1\}.$$

In fact, Ω_{x_0} is a parametrization of the level curves corresponding to quadratic Lyapunov functions and level unity.

Lemma 4.3.2. *Let $A \in \mathbb{R}^{n \times n}$ and let $x_0 \in \mathbb{R}$ be a nonzero vector that does not belong to an A-invariant subspace of dimension at most $n-1$. Let U be an open neighborhood of $0 \in \mathbb{R}$. Then* $\operatorname{span}_{t \in U}(\exp(At))x_0 = \mathbb{R}^n$.

Proof. Assume the contrary. Then there exists nonzero $z \in \mathbb{R}^n$ such that $z^T \exp(At) x_0 = 0$ for all $t \in U$. Repeated differentiation and substituting $t = 0$ then yields

$$z^T A^k x_0 = 0 \quad k \geq 0. \tag{4.6}$$

Define $\mathcal{V} = \operatorname{span}_{k\geq 0}\{A^k x_0\}$. Obviously, $\mathcal{V}$ is an A-invariant subspace. Moreover $z^T\mathcal{V} = 0$ and therefore $\dim \mathcal{V} \leq n-1$. This is a contradiction and the statement follows. □

Lemma 4.3.3. *Let $v_1, \ldots, v_n$ be a basis of $\mathbb{R}^n$ and let $c \in \mathbb{R}$ be a positive constant. Define $\Omega = \{P = P^T \geq 0 \mid v_i^T P v_i \leq c, \quad i = 1, \ldots, n\}$. Then Ω is bounded.*

Proof. It suffices to prove that there exists a constant $M > 0$ such that for all $x \in \mathbb{R}^n$ with $x = \sum_{j=1}^{n} \lambda_i v_i$ and $\sum_{j=1}^{n} \lambda_j^2 = 1$ we have that $x^T P x \leq M$, that is the quadratic forms $x^T P x$ are uniformly (with respect to Ω) bounded on the unit sphere. Choose any such x. Then

$$x^T P x = (\sum_{i=1}^{n} \lambda_i v_i)^T P (\sum_{i=1}^{n} \lambda_i v_i) = \sum_{i=1}^{n} \lambda_i^2 v_i^T P v_i + \sum_{i\neq j} \lambda_i \lambda_j v_i^T P v_j \tag{4.7}$$

$$\leq c \sum_{i=1}^{n} \lambda_i^2 + \frac{1}{2} \sum_{i\neq j} \lambda_i \lambda_j (v_i^T P v_i + v_j^T P v_j) \leq c + c \sum_{i\neq j} \lambda_i \lambda_j \tag{4.8}$$

$$\leq c + \frac{1}{2} c \sum_{i\neq j} (\lambda_i^2 + \lambda_j^2) = c + c(n-1) = cn. \tag{4.9}$$

Where we used that for any two vectors v, w: $v^T P w + w^T P v \leq v^T P v + w^T P w$.

□

Theorem 4.3.4. *Let $A \in \mathbb{R}^{n\times n}$ be a stable matrix and let $x_0 \in \mathbb{R}$ be a nonzero vector. Define $\Omega = \{P \in \mathbb{R}^{n\times n} \mid A^T P + P A \leq 0$ and $x_0^T P x_0 = 1\}$.*

1. *If x_0 does not belong to a proper A-invariant subspace then Ω is compact.*
2. *Every $P \in \Omega$ is positive semi-definite.*
3. *Ω is convex.*

Proof. Notice that since A is stable the set Ω is non-empty. Choose any $P \in \Omega$. Due to the condition on x_0 the trajectory $x(t) = \exp(At)x_0$ spans $\mathbb{R}^n$. Since x satisfies $\frac{d}{dt}x = Ax$ it follows that

$$x(t)^T P x(t) \leq x_0^T P x_0 = 1 \quad \forall t \geq 0. \tag{4.10}$$

Choose time instants $t_1, \ldots, t_n$ such that $\operatorname{span}(x(t_1), \ldots, x(t_n)) = \mathbb{R}^n$. It follows from Lemma 4.3.3 (with $c = 1$) that Ω is bounded.

To see that Ω is closed, choose a sequence $P_k \in \Omega$ such that $\lim_{k\to\infty} P_k = P$. Then

$$A^T P + P A = \lim_{k\to\infty} A^T P_k + P_k A =: -Q_k. \tag{4.11}$$

By assumption $Q_k \geq 0$ and therefore also $\lim_{k\to\infty} Q_k =: Q \geq 0$. We conclude that $P \in \Omega$. This proves the first statement.

The second and the third part are obvious. □

Example 4.3.5. Let the dynamics in a given location be given by $\frac{d}{dt}x = Ax$ with

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

The switching lines are given by

$$L_{\text{in}} = \lambda \underbrace{\begin{bmatrix}0\\1\end{bmatrix}}_{a_{\text{in}}} \quad L_{\text{out}} = \lambda \underbrace{\begin{bmatrix}1\\0\end{bmatrix}}_{a_{\text{out}}} \tag{4.12}$$

As explained in Section 4.3.1, for a given Lyapunov function $V(x) = x^{\mathrm{T}}Px$ the gain is defined as

$$\alpha_P = \frac{a_{\text{out}}^{\mathrm{T}} a_{\text{out}}}{a_{\text{in}}^{\mathrm{T}} a_{\text{in}}} \frac{a_{\text{in}}^{\mathrm{T}} P a_{\text{in}}}{a_{\text{out}}^{\mathrm{T}} P a_{\text{out}}} = \frac{p_{22}}{p_{11}} \tag{4.13}$$

The set $\Omega_{a_{\text{in}}}$ is depicted in Figure 4.2. To find the optimal Lyapunov function we want

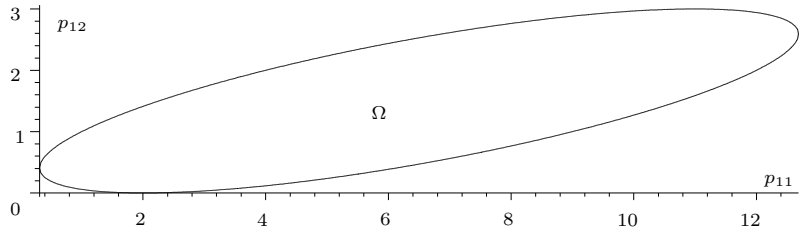

Figure 4.2: The set $\Omega_{a_{\text{in}}}$ of all Lyapunov functions.

to minimize α_P over $\Omega_{a_{\text{in}}}$. For this example this amounts to the minimization of $\frac{1}{p_{11}}$ or, equivalently, the maximization of p_{11}. Theorem 4.3.4 guarantees the optimum exists. Using Maple we found two extreme Lyapunov functions. One that minimizes α_P and one that maximizes α_P:

$$P_{\text{min}} = \begin{bmatrix}12.70 & 2.59\\ 2.59 & 1\end{bmatrix} \quad P_{\text{max}} = \begin{bmatrix}.32 & .41\\ .41 & 1\end{bmatrix} \tag{4.14}$$

The corresponding minimum and maximum values of the gains are

$$\alpha_{\text{min}} \approx 0.38 \quad \alpha_{\text{max}} \approx 2.61 \tag{4.15}$$

In Figure 4.3 the level curves of the two Lyapunov functions are drawn together with the phase portrait of $\frac{d}{dt}x = Ax$. This clearly shows the benefit of using the non-uniqueness of Lyapunov function. The one that minimizes α is obviously dramatically less conservative than the one that maximizes α.

Remark 4.3.6. In higher dimensions the minimization of the gain may be cumbersome to perform. Indeed, the expression in (4.5) is a non-convex function of P. Notice, however, that $\lambda_{\text{max}}(P)$ is convex and $\lambda_{\text{min}}(P)$ is concave. A suboptimal solution is therefore obtained by disregarding either the numerator or the denominator in (4.5).

4.4 Gain automata and detection of non-contractive cycles

Suppose we have a hybrid automaton with stable locations. If for each location that can be visited more than once the conservative gain is ≤ 1, then it can be seen that the hybrid automaton is stable: since the number of locations in a trace that are visited only once (seen

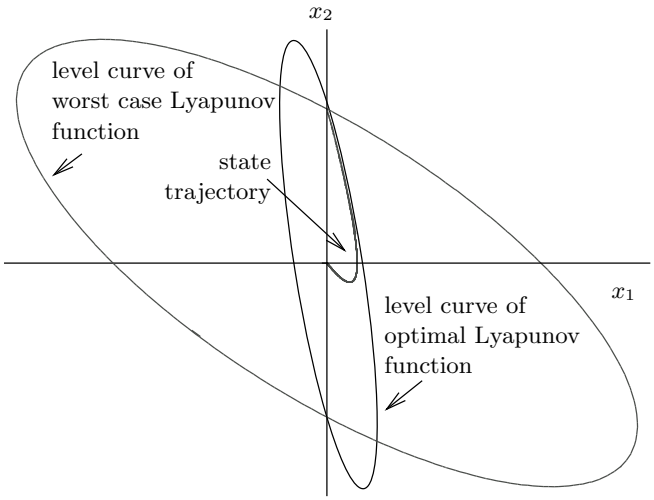

Figure 4.3: Level curves of the extreme Lyapunov functions

as a function of a trace) is bounded, there is a bound to the gains corresponding to the traces. However, such a condition is unnecessarily restrictive as it does not take into account situations where a higher gain in one location is compensated by a lower gain in another location. So we need a more sophisticated condition.

Definition 4.4.1. Let H be a hybrid automaton, then a *contractive cycle* of H is a sequence of transitions $e_1 e_2 \dots e_m$ such that each e_i is a transition from ℓ_i to ℓ_{i+1}, with $\ell_1 = \ell_{m+1}$, and $\alpha_{e_1 e_2} \cdot \alpha_{e_2 e_3} \cdot \ldots \cdot \alpha_{e_m e_1} \leq 1$.

Theorem 4.4.2. *Let H be an LCH hybrid automaton with stable locations. Then: H is unstable $\Rightarrow$ H has a non-contractive cycle.*

Proof. Seeking a contradiction, assume that all cycles are contractive. Choose $\epsilon > 0$ and let $\delta_k > 0$ be such that $\lim_{k\to\infty} \delta_k = 0$. Assume that for every k there exists a trace $\sigma_k = x_{1,k} e_{1,k} x_{2,k} \cdots e_{M-k-1k} x_{m_k,k}$, with

$$\|x_{1,k}(0)\| < \delta_k \quad \|x_{m_k,k}(t_k)\| \geq \epsilon \tag{4.16}$$

Remove from each trace σ_k all cycles to obtain $\tilde{\sigma}_k$. Denote by $\tilde{\eta}_k$ the event sequence corresponding to $\tilde{\sigma}_k$. Since the number of events is finite and since $\tilde{\sigma}_k$ does not contain cycles, it follows that there exist an infinite number $\tilde{\sigma}_{k_i}$s and an event sequence $\tilde{\eta}$ such that for all i:

$$\tilde{\eta}_{k_i} = \tilde{\eta} = e_1\, e_2 \cdots e_{M-1} \tag{4.17}$$

Hence every $\tilde{\sigma}_{k_i}$ is of the form

$$\tilde{\sigma}_{k_i} = x_1^{k_i} e_1 x_2^{k_i} e_2 \cdots x_{M-1}^{k_i} e_{M-1} x_M^{k_i} \tag{4.18}$$

with

$$x_j^{k_i} : \left[\tau_{k_i,j}, \tau'_{k_i,j}\right] \to \mathbb{R}^n$$

trajectories in location ℓ_j. If for some j $\tau'_{k_i,j} < \tau_{k_i,j+1}$, then a cycle has been removed in between. Since all cycles are contractive in the sense of Definition 4.4.1 we conclude that

$$\|x(\tau_{k_i,j+1})\| \leq \|x(\tau'_{k_i,j})\|. \tag{4.19}$$

Of course, if $\tau'_{k_i,j} = \tau_{k_i,j+1}$, then (4.19) is trivially satisfied. Notice that since $\|x_{m_k,k}(t_k)\| \geq \epsilon$ and $\|x_k(0)\| < \tilde{\delta}_k$ we also have that

$$\|x_M^{k_i}(\tau'_{k_i,M})\| \geq \epsilon \quad \|x_1^{k_i}(\tau_{k_i,1})\| < \delta_{k_i} \tag{4.20}$$

Now, choose $\tilde{\delta}_M > 0$ such that for any trajectory x in location ℓ_M we have

$$\|x(0)\| < \tilde{\delta}_M \Rightarrow \|x(t)\| < \epsilon. \tag{4.21}$$

Assume that $\tilde{\delta}_j$ has been defined. Choose $\tilde{\delta}_{j-1}$ such that for every trajectory x in location ℓ_j we have

$$\|x(0)\| < \tilde{\delta}_{j-1} \Rightarrow \|x(t)\| < \tilde{\delta}_j. \tag{4.22}$$

Take i such that $\delta_{k_i} < \tilde{\delta}_1$. It follows that

$$\|x_1^{k_i}(\tau'_{k_i,1})\| < \tilde{\delta}_2$$

from which we conclude

$$\|x_2^{k_i}(\tau'_{k_i,2})\| < \tilde{\delta}_3.$$

Repeating this argument we finally get:

$$\|x_M^{k_i}(\tau'_{k_i,M})\| < \epsilon. \tag{4.23}$$

This contradicts the first inequality in (4.20) and therefore not all cycles of H can be contractive. This concludes the proof. □

Theorem 4.4.2 provides us with a sufficient condition for stability, namely the absence of non-contractive cycles. In order to check for non-contractive cycles we first transform a hybrid automaton into another type of automaton.

Definition 4.4.3. A *gain automaton* is a tuple $GA = (S, S^0, G)$ where

- S is the set of nodes,
- S^0 is the set of initial nodes,
- $G \subseteq S \times \mathbb{R}^+ \times S$ is the set of edges labeled with gains.

Definition 4.4.4. Let H be a hybrid automaton, then the gain automaton for H is defined by $GA(H) = (S_H, S_H^0, G_H)$ where

- The nodes of the gain automaton are the transitions of H, so $S_H = E$.
- The initial nodes S_H^0 are the transitions from an initial location of H.
- For each pair of transitions e, e' in H such that $\xrightarrow{e} l \xrightarrow{e'}$ there is an edge $e \xrightarrow{\alpha_{ee'}} e'$ in G_H.

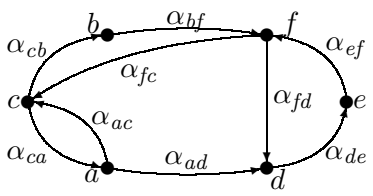

Figure 4.4: Example of a gain automaton

Example 4.4.5. Let H be the hybrid automata of Example 4.2.2, then the gain automaton $GA(H)$ is given in Figure 4.4.

We define an algorithm on the gain automaton of a hybrid automaton for the detection of non-contractive cycles. This algorithm is inspired by the well-known algorithm for transforming an automaton into an equivalent regular expression (see e.g.[13, 17]). It works by successively deleting nodes of the gain automaton, while transforming the edges. The basic steps of the algorithm are:

Node elimination: a node is eliminated. Each possible pair of an incoming and outgoing edge of this node leads to a new edge, labeled with the product of the gains, as illustrated in Figure 4.5(a).

Double edge elimination: if two edges have the same initial and final node they are transformed into a single edge, labeled with the maximum of the gains, as illustrated in Figure 4.5(c).

Loop edge analysis: it is possible that deleting a node creates a loop edge, as illustrated in Figure 4.5(b). If the gain of such a loop edge is > 1 (we call this a *non-contractive* loop edge) the algorithm is terminated. Otherwise, the loop edge is removed.

Algorithm 4.4.6. *Input: a gain automaton GA.*
check all loop edges;
if *a non-contractive loop edge is found*
then *terminate;*
remove all loop edges;
repeat
eliminate a state;
eliminate all resulting double edges;
analyze all resulting loop edges;
if *a non-contractive loop edge is found*
then *terminate;*
remove all loop edges
until *there is only one state*

Theorem 4.4.7. *Let H be an LCH hybrid automaton with stable locations. Then: Algorithm 4.4.6 detects a non-contractive loop edge in $GA(H)$ $\Leftrightarrow$ H contains a non-contractive cycle.*

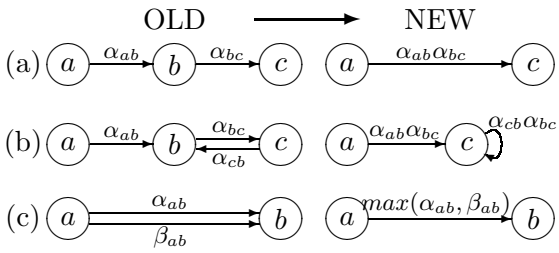

Figure 4.5: Basic steps of the algorithm

Proof. $\Rightarrow$: trivial since each non-contractive loop edge corresponds to a non-contractive cycle.

$\Leftarrow$: suppose there is a non-contractive cycle in H. This means that in the gain automaton there is a cycle where the product of the gain labels is bigger than 1; call such a cycle a non-contractive gain cycle.

Now each removal step in the algorithm preserves the existence of a non-contractive gain cycle: this is immediately clear for node elimination and double edge elimination. For loop edge elimination it also holds, since if a non-contractive gain cycle contains a contractive loop edge, then the gain cycle without the contractive loop edge remains non-contractive.

So at each step of the algorithm the presence of a non-contractive gain cycle is unaltered. Suppose there is a non-contractive gain cycle. If it were not detected by a non-contractive loop edge detection then the algorithm would yield a gain automaton with a single state and no transitions, so not containing a non-contractive gain cycle, which contradicts the fact that at each step the presence of a non-contractive gain cycle is unaltered. So if H has a non-contractive cycle it will be detected by detecting a non-contractive loop edge. □

The number of nodes in $GA(H)$ is quadratic in the number of nodes of H, and the complexity of Algorithm 4.4.6 is linear in the number of nodes of GA, so the complexity of Algorithm 4.4.6 is quadratic in the number of nodes of H. This means we have a computationally efficient way of checking the sufficiency condition for stability, viz. the absence of non-contractive cycles.

Remark 4.4.8. Our analysis applies, *mutatis mutandis* equally well to *asymptotic stability*. If the dynamics in each location is asymptotically stable then the hybrid automaton is asymptotically stable if all cycle are strictly contractive, i.e., if the product of all gains along the cycle is strictly smaller than one.

Chapter 5

Overview of the results of WP4

In WP4 of the Hybridge project we have developed an automaton framework called CPDP for compositional specification and analysis of stochastic hybrid systems. A CPDP is a PDP-type system which means that it is a mixture of deterministic motion (determined by ordinary differential equations) and random jumps. Besides random jumps (*spontaneous transitions* in CPDP terminology) there are also active and passive transitions. These transitions are labelled with discrete events and are used for interaction between different CPDPs. We have defined a set of composition operators $|^P_A|$. Different types of interaction can be expressed using different sets A and P. We have given an operational semantics for $|^P_A|$ and the result of composing two CPDPs is again a CPDP. This means that the behavior of multiple CPDPs running in parallel and interacting via active and passive events can be expressed as a single CPDP and therefore theory for CPDPs will also apply to composition of CPDPs. The latest developments within the CPDP framework are as follows.

- The framework has been extended with value-passing which allows that interacting CPDPs can send/receive data of the continuous process to each other. We have shown that this type of interaction can be expressed by adding a few composition rules to the parallel composition operator $|^P_A|$.
- In [35] an algorithm is given for determining the maximal bisimulation of a CPDP. With this algorithm the state space of a CPDP can be reduced in an optimal way such that analysis of the original CPDP can be done on the state reduced CPCP. Bisimulation can be done in a compositional way by finding bisimilar state reduced components and placing back the state reduced components in the composition. Bisimulation, which is a tool that makes analysis easier, can be done on the level of the components and is therefore a compositional analysis technique.

The direction we want to point out for future research on CPDP is: compositional analysis. One form of compositional analysis is formed by bisimulation techniques. It would be interesting to find subclasses of CPDPs that on the one hand are broad enough to contain many interesting complex systems and on the other hand are such that the computation of the bisimulation algorithm of [35] is decidable.

Bibliography

[1] Andrei A. Agrachev and Daniel Liberzon. Lie-algebraic stability criteria for switched systems. *SIAM Jour. Ctr. Opt.*, 40:253–269, 2001.

[2] Alur, R., C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems I*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer Verlag, 1993.

[3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[4] M. N. Belur and H. L. Trentelman, *Stabilization, pole placement and regular implementability*, IEEE Trans. Automatic Control **47(5)** (2002), 735–744.

[5] Michael S. Branicky. Stability of switched and hybrid systems. In *Proc. 33rd IEEE Conf. Decision and Control*, pages 3498–3503, Orlando, FL, 1994.

[6] Michael S. Branicky. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Trans. Aut. Contr.*, 43:475–482, 1998.

[7] M. H. A. Davis. *Markov Models and Optimization.* Chapman & Hall, London, 1993.

[8] DeCarlo, Raymond A., Michael S. Branicky, Stefan Pettersson, and Bengt Lennartson. Perspectives and results on the stability and stabilizability of hybrid systems. In *Proc. of the IEEE*, volume 88, pages 1069–1082, 2000.

[9] M.H.C. Everdij and H.A.P. Blom, *Modelling hybrid state Markov processes through Dynamically and Stochastically Coloured Petri Nets*, HYBRIDGE public deliverable D2.4, www.nlr.nl/public/hosted-sites/hybridge, NLR, 2004.

[10] M.H.C. Everdij, M.B. Klompstra, H.A.P. Blom, and B. Klein Obbink, *Compositional specification of a multi-agent system by Dynamically Coloured Petri Nets*, HYBRIDGE public deliverable D9.2, www.nlr.nl/public/hosted-sites/hybridge, NLR, 2004.

[11] M.K. Ghosh, A. Arapostathis, and S. Marcus. Ergodic control of switching diffusions. In *SIAM J. Contr. Optimization*, volume 35(6), pages 1952–1988, 1997.

[12] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings LICS'96*, pages 278–292, 1996.

[13] Rajeev Motwani Hopcroft, John E. and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Adisson-Wesley, second edition, 2001.

[14] Mikael Johansson and Anders Rantzer. Computation of piecewise quadratic Lyapunov functions for hybrid systems. *IEEE Trans. Aut. Contr.*, 43:555–559, 1998.

[15] Xenofon D. Koutsoukos and Panos J. Antsaklis. Design of stabilizing switching control laws for discrete and continuous-time linear systems using piecewise-linear Lyapunov functions. Technical Report ISIS Technical Report ISIS-2001-002, ISIS, 2001.

[16] Daniel Liberzon and A. Stephen Morse. Basic problems in stability and design of switched systems. *IEEE Control Systems Magazine*, 19:59–70, 1999.

[17] Peter Linz. *An Introduction to Formal Languages and Automata*. Jones and Bartlett publishers, third edition, 2001.

[18] M. Haj-Hussein L. Logrippo, M. Faci, *An introduction to lotos: Learning by examples*, Comp. Networks and ISDN Systems **23(5)** (1992), 325–342.

[19] N. A. Lynch, R. Segala, and F. W. Vaandrager, *Hybrid I/O automata revisited*, Proceedings 4th Int'l Workshop on Hybrid Systems : Computation and Control, Springer-Verlag, 2001, pp. 403–417.

[20] N. A. Lynch, R. Segala, and F. W. Vaandrager, *Hybrid I/O automata*, Information and Computation **185(1)** (2003), 105–157.

[21] N. A. Lynch and M. R. Tuttle, *An introduction to input/output automata*, CWI Quarterly **2(3)** (1989), 219–246.

[22] P. Merlin and G. V. Bochmann, *On the construction of submodule specifications and communication protocols*, ACM Trans. Programming Language and Systems **5(1)** (1983), 1–25.

[23] Anthony N. Michel. Recent trends in the stability analysis of hybrid dynamical systems. *IEEE Transactions on Circuits and Systems - I*, 45:120–134, 1999.

[24] Giancarlo Ferrari-Trecate Mignone, Domenico and Manfred Morari. Stability and stabilization of piecewise affine and hybrid systems: an LMI approach. In *Proceedings 39th IEEE CED*, pages 504–509, 2000.

[25] B. Klein Obbink, *Description of advanced operation: Free flight*, HYBRIDGE public deliverable D9.1, www.nlr.nl/public/hosted-sites/hybridge, NLR, 2004.

[26] Stefan Pettersson and Bengt Lennartson. Stability and robustness for hybrid systems. In *Proc. 35th IEEE Conf. Decision and Control*, pages 1202–1207, Kobe, Japan, 1996.

[27] J.W. Polderman and J.C. Willems. *Introduction to mathematical systems theory: a behavioral approach*, volume 26 of *Texts in Applied Mathematics*. Springer, New York NY, USA, 1998.

[28] A. J. van der Schaft and A. A. Julius, *Achievable behavior by composition*, Proceedings 41st IEEE Conf. Decision and Control (Las Vegas), IEEE, 2002, pp. 7–12.

[29] A. J. van der Schaft, *Achievable behavior of general systems*, Systems & Control Letters, vol. 49, pp. 141–149, 2003.

[30] A. J. van der Schaft and J. M. Schumacher, *Compositionality issues in discrete, continuous, and hybrid systems*, Int. J. Robust Nonlinear Control **11(5)** (2001), 417–434.

[31] A.J. van der Schaft and J.M. Schumacher. *An Introduction to Hybrid Dynamical Systems*, volume 251 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag, London, 2000.

[32] S.N. Strubbe and A.J. van der Schaft, *CPDP: A compositional framework for stochastic hybrid systems of the PDP type*, HYBRIDGE public deliverable D4.2, www.nlr.nl/public/hosted-sites/hybridge, Twente University, 2003.

[33] S.N. Strubbe and A.J. van der Schaft, *Semantics and interaction-structures for the CPDP-model*, HYBRIDGE public deliverable D4.3, www.nlr.nl/public/hosted-sites/hybridge, Twente University, 2004.

[34] S.N. Strubbe and A.J. van der Schaft, *Value-passing and stochastic semantics for Communicating Piecewise Deterministic Markov Processes*, submitted to CDC 2005 conference.

[35] S.N. Strubbe and A.J. van der Schaft, *Algorithmic bisimulation for Communicating Piecewise Deterministic Markov Processes*, submitted to CDC 2005 conference.

[36] C. Tomlin, J. Lygeros, and S. Sastry, *A game theoretic approach to controller design for hybrid systems*, Proceedings of the IEEE **88(7)** (2000), 949–970.

[37] H. L. Trentelman and J. C. Willems, *H-infinity control in a behavioral context: The full information case*, IEEE Trans. Automatic Control **44(3)** (1999), 521–536.

[38] J. C. Willems, *On interconnections, control and feedback*, IEEE Transactions on Automatic Control **42** (1997), 326–339.

[39] Anthony N. Michel Ye, Hui and Ling Hou. Stability theory for hybrid dynamical systems. *IEEE Trans. Aut. Contr.*, 43:461–474, 1998.