Action and predicate safety of hybrid processes

Action and Predicate Safety of Hybrid Processes

P.J.L. Cuijpers and M.A. Reniers

April 5, 2004

Abstract

In this paper, we study two kinds of safety properties for hybrid processes, namely safety for actions and safety for predicates on model variables. We give an algebraic specification of these safety properties using the process algebra HyPA, and show how to reduce the question of safety of a linear process specification to the question of safety of its sub-processes. As an example, we study a variant of Fischer's protocol, in which there can be a relative error between the clocks that are used.

1 Introduction

Safety, put simply, means that a certain property, which is considered bad, does not hold at any time, during any of the possible executions of a system. The analysis of safety properties of a model of a system, is an important way to study correctness of a design or implementation. A famous example is that, when studying a design of a nuclear plant, one might want to verify that certain high temperatures in the plant (causing meltdown) never occur during normal execution of the system. Another example, is that when a certain resource is shared by two systems, but has limited capacity, then the systems may not use that resource at the same time. This is is often called 'mutual exclusion'. Safety for actions and safety for predicates, are two special kinds of safety properties. We have safety for a certain action, if this action cannot be executed from any of the reachable states of a system, while safety for a predicate (or more precisely, a propositional predicate on the model variables), means that this predicate is never satisfied during any of the executions of the system. The high-temperature example and the mutual-exclusion example can be expressed, both as safety for an action, or as safety for a predicate, depending on the precise way in which the plant and the protocol are modeled. An important aspect of safety for actions and safety for predicates, is that they can be verified locally, for each of the reachable states of a system.

In this report, we describe a method for the analysis of safety for actions and safety for predicates of hybrid systems [27, 16, 21], i.e. systems that display a mixture of physical and computational behavior. Due to the growing use of embedded systems, where software is used to control mechanical and electrical devices, research in this area has received a lot of interest in the past few years [1, 4, 7, 12, 15, 14, 27]. As a specific example of our safety analysis method, we analyze a variant of the well known Fischer's protocol for mutual exclusion [18]. Although, admittedly, not intrinsically hybrid, this protocol does allow us to show some of the strengths of our method.

In our study of Fischer's protocol, we focus on the possibility, that not all clocks that are used in the protocol, are implemented in exactly the same way. The error between clocks can be approached in two different ways. One may express the error that may occur in every clock separately, giving rise to a bounded differential inclusion for each clock (for example of the form $\dot{x} \in [0.5, 1.5]$ for an absolute error of 0.5), or one may study the difference between clocks, and hence focus on a relative error (giving rise, for example, to an equation of the form $\dot{x}_1 \leq 1.5 \cdot \dot{x}_2 \ \wedge \ \dot{x}_2 \leq 1.5 \cdot \dot{x}_1$). The latter, calls for a hybrid approach to safety analysis, because the analysis question induces a dependence between the different clocks. This is the problem we focus on. The analysis method we use, is based on process algebraic reasoning principles.

Process algebra, is a branch of computer science, in which computational systems are described in an algebraic way, and analyzed using symbolic manipulation. Very recently, process algebras have been developed in which hybrid systems can be modeled and analyzed [17, 9, 6, 23]. In this report, we give a short presentation of such a hybrid process algebra, called HyPA [9], and then focus on the analysis of safety for actions and safety for predicates of hybrid processes in this algebra. The observation that safety for actions and safety for predicates can be verified locally, makes it possible, for so-called linear process descriptions [2, 13, 26], to reduce the question of safety of a process, to a number of questions on the safety of subprocesses. Furthermore, for the example of Fischer's protocol, we explicitly carry out the transformation into a linear process description, and show some of the analysis on the subprocesses. In this way, we prove mutual exclusion for certain given initial conditions.

The structure of this report is as follows. In section 2, we give a brief introduction to the syntax of HyPA, and informally explain its semantics. We also discuss the concept of linear process descriptions, since it plays a crucial role in this report. Due to space considerations, we refer to [10, 9] for the formal semantics and most of the axiomatization. When we perform equational reasoning in this report, we will attempt to give the intuitions on the steps of reasoning, rather than to give the full axiomatic derivations. In section 3, we discuss the semantic model of hybrid transition systems briefly, in order to give the reader enough background to understand some of the definitions further on in the report. In section 4, we formalize the notions of safety for actions and safety for predicates, and explain how these can be stated as a process algebraic specification. Also, we give the basic axioms needed to do calculations on this specification. In section 5, we discuss a method for the analysis of safety properties of linear process descriptions in HyPA, and in section 6, this method is used for the analysis of our variant of Fischer's protocol, under given initial conditions on the parameters of the protocol. In section 7, we discuss the results that we have found, and give recommendations for future research.

2 Hybrid process algebra

In this section, we discuss the syntax of HyPA, and informally explain its semantics. In appendix A the full semantics is given for reference, but for a detailed explanation of this semantics we refer to [9]. The syntax of HyPA is an extension of the process algebra ACP [2], with the disrupt operator from LOTOS [8] and with variants of the flow clauses and re-initialization clauses from the event-flow formalism introduced in [27]. The signature of HyPA consists of the following constant and function symbols:

1. deadlock δ,
2. empty process ϵ,
3. discrete actions $a \in \mathcal{A}$,
4. flow clauses $c \in C$,
5. a family of process re-initialization operators $(d \gg \_)_{d \in D}$,
6. alternative composition $\_ \oplus \_$,
7. sequential composition $\_ \odot \_$,

8. disrupt $\_ \blacktriangleright \_$ and left-disrupt $\_ \rhd \_$,
9. parallel composition $\_ \parallel \_$, left-parallel composition $\_ \lfloor\!\lfloor \_$, and forced-synchronization $\_ \mid \_$,
10. a family of encapsulation operators $(\partial_H(\_))_{H \subseteq \mathcal{A}}$

The binding order of these operators is as follows: $\odot$, $\blacktriangleright$, $\rhd$, $d \gg$, $\parallel$, $\lfloor\!\lfloor$, $\mid$, $\oplus$, where alternative composition binds weakest, and sequential composition binds strongest.

The atomic processes δ (called *deadlock*) and ϵ (called *empty process*) are used to model a deadlocking process and a (successfully) terminating process, respectively. The atomic *discrete actions* are used as an abstract model for discrete, computational behavior. The set $\mathcal{A}$ of discrete actions is considered a parameter of the theory and can be instantiated at will by the user of the algebra.

Flow clauses are used to model continuous, never terminating, physical behavior by describing how the model variables $\mathcal{V}_{\mathrm{m}}$ are allowed to change through time. Both the model variables $\mathcal{V}_{\mathrm{m}}$ (including the domains they range over) and the appropriate notion of time T (some totally ordered set with a least element denoted 0) are problem-specific and should be given by the modeler. The sets $\mathcal{V}_{\mathrm{m}}$ of model variables and T of time points are therefore considered parameters of the theory as well. The domain $\mathcal{V}(x)$ of a model variable $x \in \mathcal{V}_{\mathrm{m}}$ is specified by the modeler at the first introduction of the variables. In this report, the specification of domains is left out since, most of the time, it is obvious from the context. We write $\mathcal{V} = \bigcup_{x \in \mathcal{V}_{\mathrm{m}}} \mathcal{V}(x)$ for the union of all variable domains, and $\mathit{Val} = \mathcal{V}_{\mathrm{m}} \to \mathcal{V}$ for the set of variable valuations.

A flow clause describes a set of flows, where a flow is a (partial) function of time T with a closed-interval domain starting from 0, to the valuations Val of model variables (with $x \in \mathcal{V}_{\mathrm{m}}$ taking value in the range $\mathcal{V}(x)$). The flows that are described by a flow clause are called the solutions of that flow clause. The set of all flows with a closed-interval domain starting in 0 is $\mathcal{F} = \{ f \in T \mapsto \mathit{Val} \mid \mathrm{dom}(f) = [0,t] \text{ for some } t \in T \}$. Formally, a flow clause is a pair $(V \mid P_{\mathrm{f}})$ of a set of model variables $V \subseteq \mathcal{V}_{\mathrm{m}}$, signifying which variables are not allowed to jump at the beginning of a flow, and a flow predicate $P_{\mathrm{f}} \in \mathcal{P}_{\mathrm{f}}$. The set of jump variables turns out to be important, amongst others, in the context of higher index systems (see [10] and [24] for some examples), where flows can sometimes be discontinuous. We consider the set of flow predicates $\mathcal{P}_{\mathrm{f}}$ and the notion of solution $\models_{\mathrm{f}} \subseteq \mathcal{F} \times \mathcal{P}_{\mathrm{f}}$, that defines which flows are considered solutions of a flow predicate, parameters of the theory. Furthermore, in this report we assume that the notion of solution is closed under concatenation. Formally, we require that if $\sigma \models_{\mathrm{f}} P_{\mathrm{f}}$, $\sigma' \models_{\mathrm{f}} P_{\mathrm{f}}$ and $\sigma(t) = \sigma'(0)$, with $\mathit{dom}(\sigma) = [0,t]$, then $\sigma'' \models_{\mathrm{f}} P_{\mathrm{f}}$, with σ'' defined as the concatenation $\sigma''(\tau) = \sigma(\tau)$ if $\tau < t$ and $\sigma''(\tau) = \sigma'(\tau - t)$ if $\tau \geq t$. This is not an unusual assumption in systems theory, see for example [25]. More importantly, it turns out to be a convenient assumption for the axiomatization of the notion of equivalence we use in this report. We define a valuation-flow pair (ν, σ) to be a solution of a flow clause $(V \mid P_{\mathrm{f}})$ (denoted $(\nu, \sigma) \models_c (V \mid P_{\mathrm{f}})$) if $\sigma \models_{\mathrm{f}} P_{\mathrm{f}}$ and $\nu(x) = \sigma(0)(x)$, for all $x \in V$. The set of all flow clauses is denoted C. We usually leave out the brackets for V, and even omit it (and the '|' delimiter) if it is empty.

In the examples in this report, we parameterize HyPA using differential equations (i.e. predicates on the model variables $\mathcal{V}_{\mathrm{m}}$ and their time-derivatives $\dot{\mathcal{V}}_{\mathrm{m}}$) as flow predicates. For a differential equation $\dot{x} = f(x, u)$, the notion of solution assumes x and u to be piecewise continuous functions such that $x(t) = x(0) + \int_0^t f(x(\tau), u(\tau))\, d\tau$ for all $t \in \mathit{dom}(x)$. Indeed, this notion of solution is closed under concatenation.

A process re-initialization $d \gg p$ models the behavior of p where the model variables are submitted to a discontinuous change as specified by the re-initialization clause d. A re-initialization clause describes a set of re-initializations, where a re-initialization is a pair of valuations representing the values of the model variables prior to and immediately after the re-initialization. Such re-initializations are called solutions of the re-initialization clause. The set of all re-initializations

$Val \times Val$ is denoted $\mathcal{R}$. A re-initialization clause is a pair $[V \mid P_{\mathrm{r}}]$ of a set of model variables $V \subseteq \mathcal{V}_{\mathrm{m}}$ and a re-initialization predicate P_{r}. The set V models which variables are allowed to change. Note that this is precisely opposite to flow clauses, where V denotes those variables that do not change. This has turned out to be the most convenient choice of notation, for our modeling purposes. For similar reasons as before, the set of re-initialization predicates $\mathcal{P}_{\mathrm{r}}$ and the notion of solution $\models_{\mathrm{r}} \subseteq \mathcal{R} \times \mathcal{P}_{\mathrm{r}}$, that defines which re-initializations are considered solutions of a re-initialization predicate, are considered parameters of the theory. We define a re-initialization (ν, ν') to be a solution of a re-initialization clause $[V \mid P_{\mathrm{r}}]$ (denoted $(\nu, \nu') \models_d (V \mid P_{\mathrm{r}})$) if $(\nu, \nu') \models_{\mathrm{r}} P_{\mathrm{r}}$ and $\nu(x) = \nu'(x)$, for all $x \notin V$. The set of all re-initialization clauses is denoted D.

In the examples in this report, we parameterize HyPA using predicates on the previous ($\mathcal{V}_{\mathrm{m}}^{-} = \{x^{-} \mid x \in \mathcal{V}_{\mathrm{m}}\}$) and future ($\mathcal{V}_{\mathrm{m}}^{+} = \{x^{+} \mid x \in \mathcal{V}_{\mathrm{m}}\}$) values of the model variables as re-initialization predicates. The notion of solution, obviously, is such that $(\nu, \nu') \models_{\mathrm{r}} P_{\mathrm{r}}(x^{-}, x^{+})$ if and only if $P_{\mathrm{r}}(\nu(x), \nu'(x))$ evaluates to *true*.

The alternative composition $p \oplus q$ models a (non-deterministic) choice between the processes p and q. The sequential composition $p \odot q$ models a sequential execution of processes p and q. The process q is executed after (successful) termination of the process p.

The disrupt $p \blacktriangleright q$ models a kind of sequential composition where the process q may take over execution from process p at any moment, without waiting for its termination. This composition is invaluable when modeling two flow clauses executing one after the other, since the behavior of flow clauses never terminates. The left-disrupt $p \rhd q$ is mainly needed for calculation and axiomatization purposes, rather than for modeling purposes. For example, it occurs in the axiomatization of reachability in section 4. The left-disrupt first executes a part of the process p and then behaves as a normal disrupt.

The parallel composition $p \parallel q$ models concurrent execution of p and q. The intuition behind this concurrent execution is that discrete actions are executed in an interleaving manner, with the possibility of synchronization (as in ACP, where synchronization is called communication), while flow clauses are forced to synchronize, and can only synchronize if they accept the same solutions. The synchronization of actions takes place using a (partial, commutative, and associative) communication function $\gamma \in \mathcal{A} \times \mathcal{A} \mapsto \mathcal{A}$. For example, if the actions a and a' synchronize, the resulting action is $a'' = a \gamma a'$. This communication function is considered a parameter of the theory. Actions cannot synchronize with flow clauses, and in a parallel composition between those, the action executes first. Re-initializations synchronize their solutions only if the processes on which they act synchronize.

As with the left-disrupt, the operators left-parallel composition and forced-communication are mainly introduced for calculation purposes. The left-parallel composition $p \mathbin{\|\!\_} q$ denotes that p performs a discrete action first (if possible), and then it behaves as normal parallel composition, and the forced-synchronization $p \mid q$ denotes how the first behavior (either a discrete action or a part of a flow) of p and q is synchronized, after which it behaves as normal parallel composition.

Encapsulation $\partial_H(p)$ models that certain discrete actions (from the set $H \subseteq \mathcal{A}$) are blocked during the execution of the process p. This operator is often used in combination with the parallel composition to model that synchronization between discrete actions is enforced. In section 4, we introduce a variant of the encapsulation operator, aimed at blocking executions of processes that violate certain predicates on model variables.

From the signature of HyPA, terms can be constructed using variables from a given set of process variables $\mathcal{V}_{\mathrm{p}}$ (with $\mathcal{V}_{\mathrm{p}} \cap \mathcal{V}_{\mathrm{m}} = \emptyset$), as usual. These are referred to as terms or *open terms*. Terms in which no process variables occur, are called *closed terms*. Finally, all the terms should be interpreted in the light of a set E of recursive definitions of the form $X : p$, where X is a special process variable, called recursion variable, and p is an open term, possibly containing recursion variables. Recursion is a powerful way to model repetition in a process. The set $\mathcal{T}(\mathcal{V}_\mathrm{r})$ denotes the set of all open terms in which only recursion variables are used. These terms we refer to as *process terms*.

One of the more fundamental theorems about HyPA, presented in [9], is that every closed term can be rewritten into a so-called *basic term*. These basic terms clearly show that parallel compositions can be eliminated from all closed terms.

Definition 1 (Basic terms) *A* basic term *is a closed term of the following form:*

$$N ::= d \gg \epsilon \mid d \gg a \odot N \mid d \gg c \rhd N \mid N \oplus N,$$

where $a \in \mathcal{A}$, $c \in C$, and $d \in D$.

Theorem 1 (Elimination) *Every closed term is derivably equal to a basic term.*

Proof See [9]. ⊠

Continuing from this, we conjecture that many recursive specifications can be transformed, using similar techniques, into a so-called linear recursive specification, or linear process definition.

Definition 2 (Linear Recursive Specification) *A* linear recursive specification*, also called* linear process definition*, is a recursive specification of the form*

$$X_i : \left(\bigoplus_{j \in J(i)} d_j \gg \epsilon \right) \oplus \left(\bigoplus_{j \in J'(i)} d'_j \gg a_j \odot X_j \right) \oplus \left(\bigoplus_{j \in J''(i)} d''_j \gg c_j \rhd X_j \right) ,$$

where the X_i represent a family of recursion variables, parameterized by $i \in I$, and every $J(i)$, $J'(i)$, $J''(i) \subseteq I$ is a finite subset of I. The notation $\bigoplus_{j \in J} p_j$ is used as a shorthand for a finite alternative composition of processes p_j.

Current work by Peter van den Brand and Michel Reniers is focussed on performing such a transformation automatically for a certain class of recursive specifications, based on the work of [26] for the process algebra μCRL.

3 Hybrid transition systems

The general semantic model that underlies HyPA, is that of hybrid transition systems [11]. The model is based on that of ordinary labeled transition systems, but with two transition relations instead of one. Computational behavior is modeled through action transitions, which, intuitively, take no time to execute, and are labeled by the names of the actions and valuations they represent. Physical behavior is modeled through the use of flow transitions, which represent the behavior of model variables during the passage of time. The formalism is closely related to the hybrid automata of [20].

Definition 3 (Hybrid Transition System) *A* hybrid transition system *is a tuple $< X, A, \Sigma, \mapsto, \rightsquigarrow, \checkmark >$, consisting of a* state space $X = \mathcal{T}(\mathcal{V}_\mathrm{r}) \times \mathit{Val}$, *containing pairs of process terms and valuations, a* set of action labels $A = \mathcal{A} \times \mathit{Val}$, *containing action names and valuations, a* set of flow labels $\Sigma = \mathcal{F}$, *containing flows, and* transition relations $\mapsto \subseteq X \times A \times X$ *and* $\rightsquigarrow \subseteq X \times \Sigma \times X$, *labeled by actions and flows respectively. Lastly, there is a* termination predicate $\checkmark \subseteq X$.

We use the notation $<x> \overset{a}{\mapsto} <x'>$ for a transition $(x, a, x') \in\mapsto$ with $x, x' \in X$ and $a \in A$. Similarly, we use $<x> \overset{\sigma}{\rightsquigarrow} <x'>$ for a transition $(x, \sigma, x') \in\rightsquigarrow$ with $\sigma \in \Sigma$, and for arbitrary transitions, we use $<x> \xrightarrow{l} <x'>$ instead of $(x, l, x') \in\mapsto \cup \rightsquigarrow$ and $l \in A \cup \mathcal{F}$. Finally, termination is denoted $<x>\checkmark$ instead of $x \in \checkmark$.

The notion of equivalence on processes we use in this report, is based on the notion of bisimilarity on hybrid transition systems.

Definition 4 (Bisimilarity on hybrid transition systems) *A relation $R \subseteq (\mathcal{T}(\mathcal{V}_r) \times Val) \times (\mathcal{T}(\mathcal{V}_r) \times Val)$ on the state space of a hybrid transition system, is a* simulation *relation if*

- *for all $<p,\nu>, <q,\mu> \in X$ such that $<p,\nu> R <q,\mu>$, we find $<p,\nu>\checkmark$ implies $<q,\mu>\checkmark$;*
- *for all $<p,\nu>, <p',\nu'>, <q,\mu> \in X$ such that $<p,\nu> R <q,\mu>$ and $l \in A \cup \Sigma$, we find $<p,\nu> \xrightarrow{l} <p',\nu'>$ implies there exists $<q',\mu'> \in X$ such that $<q,\mu> \xrightarrow{l} <q',\mu'>$ and $<p',\nu'> R <q',\mu'>$.*

A relation is a bisimulation relation if it is a symmetric simulation relation. Two process terms p and q are bisimilar, denoted $p \leftrightarrow q$, if there exists a bisimulation relation $\mathcal{R}$ such that for all $\nu \in Val$ we find $<p,\nu>\mathcal{R}<q,\nu>$.

Note, that bisimilarity reflects the intuition that only the labels of transitions, and the possibility of termination, are visible to the environment of a process. Nevertheless, for HyPA, notion of bisimilarity coincides with the notion of *initially stateless bisimilarity* defined in [22], in which the valuations of variables in the state are also considered visible.

Theorem 2 *Two process terms p and q are bisimilar, if and only if they are initially stateless bisimilar in the sense of [22].*

Proof A relation $\mathcal{R}$ is a statebased bisimulation relation, if it is a bisimulation relation with the additional property that $<p,\nu>\mathcal{R}<q,\mu>$ implies $\nu = \mu$. Two process terms p and q, are initially stateless bisimilar, if there exists a statebased bisimulation relation $\mathcal{R}$ such that $<p,\nu>\mathcal{R}<q,\nu>$ for every $\nu \in Val$.

Obviously, initially stateless bisimilarity implies bisimilarity. Furthermore, in [10], it is shown that the semantics of HyPA is such that $<x,\nu> \xrightarrow{l} <x',\nu'>$ and $<y,\mu> \xrightarrow{l} <y',\mu'>$ implies $\nu' = \mu'$, for any $x, x', y, y', \nu, \nu', \mu$ and μ'. In other words, that the valuation of the variables in a state can be deduced from the labeling of transitions to that state. Using this, it is straightforward to show, that given a bisimulation relation $\mathcal{R}$ that witnesses $p \leftrightarrow q$, the relation $\mathcal{R} \cap \{(<x,\nu>, <y,\nu>) \mid x, y \in \mathcal{T}(\mathcal{V}_r),\ \nu \in Val\}$ is a statebased bisimulation relation witnessing initially stateless bisimilarity of p and q. ⊠

It is important to know, that in [10, 9] a stronger notion of equivalence was used. For calculations in the context of parallel composition, namely, it is important that the used equivalence is robust with respect to interference caused by processes executed in parallel. This is not the case for the notion of (bi-)simulation presented above. As an example, one might study the following discrete systems.

$$\begin{aligned} X &: [\ x \mid x^+ = 1\ ] \gg a_1 \odot [\ x^- = 1\ ] \gg a_2 \\ Y &: [\ x \mid x^+ = 1\ ] \gg a_1 \odot a_2 \end{aligned}$$

Under the notion of (initially stateless) bisimilarity, we find that $X \leftrightarrows Y$, since the value of x is set to 1 by the first action a_1, the second re-initialization of X is irrelevant. However, placing these processes in parallel with the process

$$Z \quad : \quad \left[\ x \ \middle| \ x^+ = 2 \ \right] \gg a_3$$

clearly shows a difference. The sharing of variables, leads to the result that the sequence a_1 followed by a_3 gives a deadlock situation for $X \parallel Z$, while the action a_2 can still occur for $Y \parallel Z$. The interference of Z shows a difference between X and Y.

Interference can be modeled as a function $\iota : \mathit{Val} \to \mathit{Val}$. In order for the equivalence to be robust for interference, we enforce that the bisimulation relation is closed with respect to any interference function ι. Observe that we apply the same interference function to both variable valuations.

Definition 5 (Robust) *A relation $R \subseteq (\mathcal{T}(\mathcal{V}_\mathrm{r}) \times \mathit{Val}) \times (\mathcal{T}(\mathcal{V}_\mathrm{r}) \times \mathit{Val})$ is* robust *if for all $\langle p, \nu \rangle, \langle q, \mu \rangle \in X$ such that $\langle p, \nu \rangle \, R \, \langle q, \mu \rangle$ we find $\langle p, \iota(\nu) \rangle \, R \, \langle q, \iota(\mu) \rangle$ for all interferences $\iota \in \mathit{Val} \to \mathit{Val}$.*

Now, in a similar way as before, robust bisimulation relations give rise to a notion of robust equivalence on process terms. We say that two process terms $p, q \in \mathcal{T}(\mathcal{V}_\mathrm{r})$ are robustly bisimilar, denoted $p \leftrightarrows\!\!\!\!\!= q$, if there exists a robust bisimulation relation $\mathcal{R}$ such that $<p, \nu>\mathcal{R}<q, \nu>$ for all valuations $\nu \in \mathit{Val}$. For HyPA, this notion of equivalence coincides with the notion of *stateless bisimilarity* in [22].

Theorem 3 *Two process terms p and q are robustly bisimilar if and only if they are stateless bisimilar in the sense of [22].*

Proof Similar to the proof of theorem 2. $\boxtimes$

To conclude, three important properties of these equivalences, that are used implicitly throughout this paper, are the following.

Theorem 4 *Robust bisimilarity is a congruence for parallel composition. Hence, if $p \leftrightarrows\!\!\!\!\!= q$, then $p \parallel r \leftrightarrows\!\!\!\!\!= q \parallel r$ for all process terms r.*

Proof See [9]. $\boxtimes$

Theorem 5 *Robust bisimilarity and (initially stateless) bisimilarity are congruences for all operators of HyPA other than parallel composition.*

Proof This is straightforward to verify using the congruence formats proposed in [22] on the operational semantics of HyPA given in [9]. $\boxtimes$

Theorem 6 *Robust bisimilarity is strictly finer than (initially stateless) bisimilarity. Hence, if we have $p \leftrightarrows\!\!\!\!\!= q$ then we may conclude $p \leftrightarrows q$.*

Proof Trivial. Any robust bisimulation relation is a bisimulation relation. $\boxtimes$

4 Specification of safety

In the introduction, we have already recognized two special kinds of safety properties. Safety properties in terms of actions, and safety properties in terms of model variables. Safety in terms of actions means that certain 'bad' actions never happen, while safety in terms of model variables means that certain 'bad' valuations never occur. In this section, we give a formal definition and a process algebraic specification of safety for actions and safety for predicates. This means, that we construct a process algebraic equation, in such a way that a hybrid transition system has a certain safety property if and only if the process term associated with this transition system, is a solution to that equation. In the next section, we use this specification as a starting point for our analysis.

The predicates P_{m} that describe the aforementioned bad valuations, are taken from the set $\mathcal{P}_{\mathrm{m}}$ of predicates on model variables (i.e. we do not use $\dot{x}$, x^- or x^+ in these predicates). We write $\nu \models_{\mathrm{m}} P_{\mathrm{m}}$, for a valuation $\nu \in \mathit{Val}$, to denote that ν satisfies P_{m}. We start out by formalizing the notions of safety for actions and safety for predicates on the semantic level, using the following three definitions.

Definition 6 (Reachable set) *Given a hybrid transition system, the set $\mathcal{R}(x)$ of* reachable states *from a state x, is defined as the smallest set such that:*

- *$x \in \mathcal{R}(x)$;*
- *If $x' \in \mathcal{R}(x)$ and $<x'> \xrightarrow{l} <x''>$, for some l, then $x'' \in \mathcal{R}(x)$.*

Definition 7 (Safety for actions) *A state x is safe for the actions in $H \subseteq \mathcal{A}$, if for every $x' \in \mathcal{R}(x)$ and every transition $<x'> \overset{a,\nu}{\mapsto} <x''>$ we find $a \notin H$. A process term p is safe for the actions in H if, for every valuation $\varsigma \in \mathit{Val}$, the state $<p, \varsigma>$ is safe for the actions in H.*

Definition 8 (Safety for predicates) *A state x is safe for a predicate $P_{\mathrm{m}} \in \mathcal{P}_{\mathrm{m}}$ on the model variables $\mathcal{V}_{\mathrm{m}}$, if for every $x' \in \mathcal{R}(x)$ we find, firstly, that for every transition $<x'> \overset{\sigma}{\rightsquigarrow} <x''>$ we have $\sigma(t) \not\models_{\mathrm{m}} P_{\mathrm{m}}$ for any $t \in \mathit{dom}(\sigma)$, and secondly, that for every transition $<x'> \overset{a,\nu}{\mapsto} <x''>$ we have $\nu \not\models_{\mathrm{m}} P_{\mathrm{m}}$. A process term p is safe for a predicate P_{m} if, for every valuation $\varsigma \in \mathit{Val}$, the state $<p, \varsigma>$ is safe for P_{m}.*

Note, especially, that the definitions of safety we have given here, only look at what happens in the labels of the transition system. This is in line with our intuition that only the labels of a transition system are visible to the outside world. The semantics of HyPA is chosen in such a way that the valuation in an action label, and the last valuation of a flow label, coincide with the valuation in the state that is reached. So we may still draw some conclusions about what happens in the states. However, the valuation in the initial state is not taken into account.

Algebraically, we can analyze safety for actions, using the encapsulation operator. If and only if p is safe for H, then removing all actions in H from p will have no effect. In other words, in this case p and the encapsulation of H over p are equivalent.

Theorem 7 (Specification of safety for actions) *A process term p is safe for actions in H if and only if $p \leftrightarrows \partial_H(p)$.*

Proof See appendix B. ⊠

Analogously to the encapsulation operator for actions, we define an encapsulation operator $\partial_{P_{\mathrm{m}}}()$ for predicates $P_{\mathrm{m}} \in \mathcal{P}_{\mathrm{m}}$ on model variables, that will aid us in the algebraic specification of safety for predicates. The operational semantics of this operator is given in table 1. Note, that the predicate encapsulation operator, only blocks the execution of transitions, that have labels that violate the predicate. It does not block violating valuations in the state, and hence may terminate in a state that violates the predicate. The semantical rules for encapsulation of actions have also been given, for reference.

Table 1: Semantical rules for encapsulation and predicate encapsulation

$$\frac{<p,\nu> \overset{\sigma}{\rightsquigarrow} <p',\nu'>}{<\partial_H(p),\nu> \overset{\sigma}{\rightsquigarrow} <\partial_H(p'),\nu'>} \qquad \frac{<p,\nu> \overset{\sigma}{\rightsquigarrow} <p',\nu'>, \forall_{t\in dom(\sigma)}\ \sigma(t) \not\models_{\mathrm{m}} P_{\mathrm{m}}}{<\partial_{P_{\mathrm{m}}}(p),\nu> \overset{\sigma}{\rightsquigarrow} <\partial_{P_{\mathrm{m}}}(p'),\nu'>}$$

$$\frac{<p,\nu> \overset{a,\nu'}{\mapsto} <p',\nu''>, a \notin H}{<\partial_H(p),\nu> \overset{a,\nu'}{\mapsto} <\partial_H(p'),\nu''>} \qquad \frac{<p,\nu> \overset{a,\nu'}{\mapsto} <p',\nu''>,\ \nu' \not\models_{\mathrm{m}} P_{\mathrm{m}}}{<\partial_{P_{\mathrm{m}}}(p),\nu> \overset{a,\nu'}{\mapsto} <\partial_{P_{\mathrm{m}}}(p'),\nu''>}$$

$$\frac{<p,\nu>\checkmark}{<\partial_H(p),\nu>\checkmark} \qquad \frac{<p,\nu>\checkmark}{<\partial_{P_{\mathrm{m}}}(p),\nu>\checkmark}$$

An axiomatization of predicate encapsulation, for robust bisimilarity, is given in table 2. For reference, the axioms regarding encapsulation of actions are given as well. We use a, c and d for arbitrary actions, flow clauses and re-initialization clauses, and use x and y as process variables. H and P_{m} denote arbitrary sets of actions and predicates on model variables. In the axiomatization we use the following notational conventions.

- P_{m}^- denotes a re-initialization predicate such that $(\nu,\nu') \models_{\mathrm{r}} P_{\mathrm{m}}^-$ iff $\nu \models_{\mathrm{m}} P_{\mathrm{m}}$. For the re-initialization clause $[P_{\mathrm{m}}^-]$ we therefore obtain $(\nu,\nu') \models_d [P_{\mathrm{m}}^-]$ iff $\nu = \nu' \ \wedge\ \nu \models_{\mathrm{m}} P_{\mathrm{m}}$.
- P_{m} is used also as a flow predicate such that $\sigma \models_{\mathrm{f}} P_{\mathrm{m}}$ iff $\sigma(t) \models_{\mathrm{m}} P_{\mathrm{m}}$ for every $t \in dom(\sigma)$. For the flow clause (P_{m}) we therefore obtain $(\nu,\sigma) \models_c (P_{\mathrm{m}})$ iff $\nu \models_{\mathrm{m}} P_{\mathrm{m}} \ \wedge\ \sigma(t) \models_{\mathrm{m}} P_{\mathrm{m}}$ for every $t \in dom(\sigma)$.
- $d \sim d'$ denotes a re-initialization clause such that $(\nu,\nu'') \models_d d \sim d'$ iff there exists $\nu' \in Val$ such that $(\nu,\nu') \models_d d$ and $(\nu',\nu'') \models_d d'$. This models a concatenation of re-initializations. For the re-initialization predicates $P_{\mathrm{r}}(x^-,x^+), P_{\mathrm{r}}'(x^-,x^+) \in \mathcal{P}_{\mathrm{r}}$ we have:

$$[\ x \mid P_{\mathrm{r}}(x^-,x^+)\ ] \sim [\ x \mid P_{\mathrm{r}}'(x^-,x^+)\ ] = [\ x \mid \exists_y\ P_{\mathrm{r}}(x^-,y) \wedge P_{\mathrm{r}}'(y,x^+)\ ]\ .$$

Theorem 8 *The axiomatization in table 2 is sound for robust bisimilarity.*

Proof Soundness of the axioms is proven in appendix C. That bisimilarity and robust bisimilarity are congruences for the predicate encapsulation operator, which is important for the actual use of the axiomatization, becomes immediately clear using the formats proposed in [22]. ⊠

Using predicate encapsulation, we can obtain a similar theorem on safety for predicates as we found on safety for actions.

Table 2: Axiomatization of encapsulation and predicate encapsulation

$\partial_H(\delta)$	$\overset{\leftrightarrow}{=}$	δ	$\partial_{P_\mathrm{m}}(d \gg \delta)$	$\overset{\leftrightarrow}{=}$	δ
$\partial_H(\epsilon)$	$\overset{\leftrightarrow}{=}$	ϵ	$\partial_{P_\mathrm{m}}(d \gg \epsilon)$	$\overset{\leftrightarrow}{=}$	$d \gg \epsilon$
$\partial_H(a)$	$\overset{\leftrightarrow}{=}$	δ if $a \in H$	$\partial_{P_\mathrm{m}}(d \gg a)$	$\overset{\leftrightarrow}{=}$	$\big(d \sim [\ \neg P_\mathrm{m}^-\ ]\big) \gg a$
$\partial_H(a)$	$\overset{\leftrightarrow}{=}$	a if $a \notin H$			
$\partial_H(c)$	$\overset{\leftrightarrow}{=}$	c	$\partial_{P_\mathrm{m}}(d \gg c)$	$\overset{\leftrightarrow}{=}$	$d \gg \Big(c \wedge \Big(\ \neg P_\mathrm{m}\ \Big)\Big)$
$\partial_H(d \gg x)$	$\overset{\leftrightarrow}{=}$	$d \gg \partial_H(x)$			
$\partial_H(x \oplus y)$	$\overset{\leftrightarrow}{=}$	$\partial_H(x) \oplus \partial_H(y)$	$\partial_{P_\mathrm{m}}(x \oplus y)$	$\overset{\leftrightarrow}{=}$	$\partial_{P_\mathrm{m}}(x) \oplus \partial_{P_\mathrm{m}}(y)$
$\partial_H(x \odot y)$	$\overset{\leftrightarrow}{=}$	$\partial_H(x) \odot \partial_H(y)$	$\partial_{P_\mathrm{m}}(x \odot y)$	$\overset{\leftrightarrow}{=}$	$\partial_{P_\mathrm{m}}(x) \odot \partial_{P_\mathrm{m}}(y)$
$\partial_H(x \rhd y)$	$\overset{\leftrightarrow}{=}$	$\partial_H(x) \rhd \partial_H(y)$	$\partial_{P_\mathrm{m}}(x \rhd y)$	$\overset{\leftrightarrow}{=}$	$\partial_{P_\mathrm{m}}(x) \rhd \partial_{P_\mathrm{m}}(y)$

Theorem 9 (Specification of safety for predicates) *A process term p is safe for a predicate $P_\mathrm{m} \in \mathcal{P}_\mathrm{m}$ on model variables, if and only if $p \underline{\leftrightarrow} \partial_{P_\mathrm{m}}(p)$.*

Proof Similar to the proof of theorem 7. ⊠

Note, that the axiomatization of encapsulation also holds for bisimilarity, since that is a weaker notion of equivalence than robust bisimilarity. Using the axiomatization, we then find, trivially, that the processes δ and ϵ are always safe, because they cannot perform any transitions. Furthermore, basic compositions ($\oplus$, $\odot$, $\rhd$) of safe processes are again safe. In particular, if the process terms p and q are safe for S, we find:

$$\begin{array}{rcccl} p \oplus q & \underline{\leftrightarrow} & \partial_S(p) \oplus \partial_S(q) & \underline{\leftrightarrow} & \partial_S(p \oplus q) \\ p \odot q & \underline{\leftrightarrow} & \partial_S(p) \odot \partial_S(q) & \underline{\leftrightarrow} & \partial_S(p \odot q) \\ p \rhd q & \underline{\leftrightarrow} & \partial_S(p) \rhd \partial_S(q) & \underline{\leftrightarrow} & \partial_S(p \rhd q) \end{array}$$

with $\partial_S(\_)$ standing for either action or predicate encapsulation.

Typically, the analysis of safety involves the calculation of reachable states of a system. However, the notion of robust bisimilarity that was used in [9], is far too strong to allow efficient calculations on reachable states. As a consequence we need to use the weaker notion of (initially stateless) bisimilarity, which may not be used in the context of parallel composition. In the axiomatization of bisimilarity, we use the following two additional notations:

- $d^!$ denotes a re-initialization clause such that $(\nu', \nu'') \models_d d^!$ if and only if there exists $\nu \in \mathit{Val}$ with $(\nu, \nu') \models_d d$ and $\nu' = \nu''$. This models a boolean condition reflecting the reachable valuations of d, starting from an arbitrary valuation. As an example, for the re-initialization predicate $P_\mathrm{r}(x^-, x^+) \in \mathcal{P}_\mathrm{r}$ we have $[\ x \mid P_\mathrm{r}(x^-, x^+)\ ]^! = [\ \exists_y\ P_\mathrm{r}(y, x^+)\ ]$.
- $D(c)$ denotes a re-initialization clause such that $(\nu, \nu') \models_d D(c)$ if and only if there exists $\sigma \in \mathcal{F}$ with $\mathit{dom}(\sigma) = [0, t]$ and $(\nu, \sigma) \models_c c$ and $\nu' = \sigma(t)$. This models a re-initialization reflecting the possible transitions of c. Recall, that consecutive transitions are also reflected, due to the assumption that the solution of flow-predicates is closed under concatenation. As

an example, for an algebraic differential equation of the form $\dot{x} = f(x,u)$ we have:

$$D\left(\begin{array}{c} x \\ u \end{array} \middle| \; \dot{x} = f(x,u) \right) = \left[\begin{array}{c} x \\ u \end{array} \middle| \; \exists_{t,\xi,\nu} \left\{ \begin{array}{l} dom(\xi) = dom(\nu) = [0,t] \\ \forall_{0 \le t' \le t}\ \xi(t') = x^- + \int_0^{t'} f(\xi(\tau), \nu(\tau))d\tau \\ u^- = \nu(0) \ \wedge \ u^+ = \nu(t) \\ x^+ = \xi(t) \end{array} \right. \right] .$$

where ξ and ν are piecewise continuous and piecewise differentiable.

The axioms listed in table 3, reflect how, after a transition, only the valuations that are actually reachable are of importance to the equivalence of process terms.

Table 3: Axioms for (initially stateless) bisimilarity

$d \gg a \odot x$	$\leftrightarrows$	$d \gg a \odot d^! \gg x$
$d \gg c \rhd x$	$\leftrightarrows$	$d \gg c \rhd (d \sim D(c))^! \gg x$

Theorem 10 *The axioms given in table 3 are sound for (initially stateless) bisimilarity.*

Proof This is proven in appendix D. ⊠

In the next section, we show how to use the axiomatization of HyPA from [9], together with the axioms given above, for the analysis of safety properties of linear hybrid processes.

5 Algebraic safety analysis of linear hybrid processes

The problem we face in this section, is to show safety for actions or safety for predicates, for the variable X_0 in a linear recursive specification

$$X_i : \left(\bigoplus_{j \in J(i)} d_j \gg \epsilon \right) \oplus \left(\bigoplus_{j \in J'(i)} d'_j \gg a_j \odot X_j \right) \oplus \left(\bigoplus_{j \in J''(i)} d''_j \gg c_j \rhd X_j \right) .$$

In particular, we study the question whether $X_0 \leftrightarrows \partial_{P_\mathrm{m}}(X_0)$, for a predicate P_m on model variables. The question whether $X_0 \leftrightarrows \partial_H(X_0)$, for a set of actions H, can be treated in a similar way. We perform the following calculation, using, amongst others, the axioms for (initially stateless) bisimilarity, and for predicate encapsulation, given in section 4.

$$
\begin{aligned}
\partial_{P_{\mathrm{m}}}(X_i) \;&\mathrel{\underline{\rightleftarrows}}\; \partial_{P_{\mathrm{m}}}\left(\begin{array}{l}
\bigoplus_{j\in J(i)} d_j \gg \epsilon \;\oplus \\
\bigoplus_{j\in J'(i)} d'_j \gg a_j \odot X_j \;\oplus \\
\bigoplus_{j\in J''(i)} d''_j \gg c_j \rhd X_j
\end{array}\right) \\
&\Leftrightarrow\; \partial_{P_{\mathrm{m}}}\left(\begin{array}{l}
\bigoplus_{j\in J(i)} d_j \gg \epsilon \;\oplus \\
\bigoplus_{j\in J'(i)} d'_j \gg a_j \odot (d'_j)^{!} \gg X_j \;\oplus \\
\bigoplus_{j\in J''(i)} d''_j \gg c_j \rhd \left(d''_j \sim D(c_j)\right)^{!} \gg X_j
\end{array}\right) \\
&\mathrel{\underline{\rightleftarrows}}\; \left\{\begin{array}{l}
\bigoplus_{j\in J(i)} \partial_{P_{\mathrm{m}}}(d_j \gg \epsilon) \;\oplus \\
\bigoplus_{j\in J'(i)} \partial_{P_{\mathrm{m}}}\left(d'_j \gg a_j \odot (d'_j)^{!} \gg X_j\right) \;\oplus \\
\bigoplus_{j\in J''(i)} \partial_{P_{\mathrm{m}}}\left(d''_j \gg c_j \rhd \left(d''_j \sim D(c_j)\right)^{!} \gg X_j\right)
\end{array}\right. \\
&\mathrel{\underline{\rightleftarrows}}\; \left\{\begin{array}{l}
\bigoplus_{j\in J(i)} d_j \gg \epsilon \;\oplus \\
\bigoplus_{j\in J'(i)} \left(d'_j \sim \left[\ \neg P_{\mathrm{m}}^{-}\ \right]\right) \gg a_j \odot \partial_{P_{\mathrm{m}}}\left((d'_j)^{!} \gg X_j\right) \;\oplus \\
\bigoplus_{j\in J''(i)} d''_j \gg \left(c_j \wedge \left(\ \neg P_{\mathrm{m}}\ \right)\right) \rhd \partial_{P_{\mathrm{m}}}\left(\left(d''_j \sim D(c_j)\right)^{!} \gg X_j\right)
\end{array}\right.
\end{aligned}
$$

From this calculation, we may easily deduce the following.

Theorem 11 *The question whether X_i is safe for a predicate P_{m}, i.e. whether $X_i \Leftrightarrow \partial_{P_{\mathrm{m}}}(X_i)$, reduces to the following questions about subprocesses of X_i.*

- $d'_j \gg a_j \Leftrightarrow \left(d'_j \sim \left[\ \neg P_{\mathrm{m}}^{-}\ \right]\right) \gg a_j$, *for all* $j \in J'(i)$ *and*
- $(d'_j)^{!} \gg X_j \Leftrightarrow \partial_{P_{\mathrm{m}}}\left((d'_j)^{!} \gg X_j\right)$, *for all* $j \in J'(i)$, *and,*
- $d''_j \gg c \Leftrightarrow d''_j \gg (c \wedge \left(\ \neg P_{\mathrm{m}}\ \right))$, *for all* $j \in J''(i)$, *and,*
- $(d''_j \sim D(c_j))^{!} \gg X_j \Leftrightarrow \partial_{P_{\mathrm{m}}}\left((d''_j \sim D(c_j))^{!} \gg X_j\right)$, *for all* $j \in J''(i)$.

Proof Substitution of these equivalences in the calculation above, easily gives the desired result $X_i \Leftrightarrow \partial_{P_{\mathrm{m}}}(X_i)$. Notice, that $D(c) = D(c \wedge (\neg P_{\mathrm{m}}))$ if and only if $c = c \wedge (\neg P_{\mathrm{m}})$. ⊠

Apparently, if we study safety of X_0, this depends only on the safety of the direct subprocesses of X_0, and on the safety of the X_j (with $j \in J'(0) \cup J''(0)$) under certain initial conditions. Naturally, in the process of checking safety of X_j, loops may occur in which the process is called again under different initial conditions. Gathering all initial conditions that may occur this way, leads to the following recursive definition, for the total conditions R_i.

- $R_0 = \left[\ \mathit{true}\ \right]$;
- $\forall_{j\in J'(i)}\ (R_i \sim d'_j)^{!} \Rightarrow R_j$.
- $\forall_{j\in J''(i)}\ (R_i \sim d''_j \sim D(c_j))^{!} \Rightarrow R_j$.

If we now assume, for every i, that

- $\left(R_i \sim d'_j\right) \gg a_j \Leftrightarrow \left(R_i \sim d'_j \sim \left[\ \neg P_{\mathrm{m}}^{-}\ \right]\right) \gg a_j$, for all $j \in J'(i)$, and
- $\left(R_i \sim d''_j\right) \gg c_j \Leftrightarrow \left(R_i \sim d''_j\right) \gg (c_j \wedge \left(\ \neg P_{\mathrm{m}}\ \right))$, for all $j \in J''(i)$,

then we can derive the following equivalences:

$$R_i \gg X_i \quad \Leftrightarrow \quad \left\{ \begin{array}{l} \bigoplus_{j\in J(i)}\ R_i \sim d_j \gg \epsilon\ \oplus \\ \bigoplus_{j\in J'(i)}\ \big(R_i \sim d'_j \sim [\ \neg P^-_{\mathrm{m}}\ ]\big) \gg a_j \odot R_j \gg X_j\ \oplus \\ \bigoplus_{j\in J''(i)}\ \big(R_i \sim d''_j\big) \gg \Big(c_j \wedge \Big(\ \neg P_{\mathrm{m}}\ \Big)\Big) \rhd R_j \gg X_j \end{array} \right.$$

$$\partial_{P_{\mathrm{m}}}(R_i \gg X_i) \quad \Leftrightarrow \quad \left\{ \begin{array}{l} \bigoplus_{j\in J(i)}\ R_i \sim d_j \gg \epsilon\ \oplus \\ \bigoplus_{j\in J'(i)}\ \big(R_i \sim d'_j \sim [\ \neg P^-_{\mathrm{m}}\ ]\big) \gg a_j \odot \partial_{P_{\mathrm{m}}}(R_j \gg X_j)\ \oplus \\ \bigoplus_{j\in J''(i)}\ \big(R_i \sim d''_j\big) \gg \Big(c_j \wedge \Big(\ \neg P_{\mathrm{m}}\ \Big)\Big) \rhd \partial_{P_{\mathrm{m}}}(R_j \gg X_j) \end{array} \right.$$

Obviously, $R_i \gg X_i$ and $\partial_{P_{\mathrm{m}}}(R_i \gg X_i)$ are both solutions of the same guarded recursive specification with respect to bisimilarity. Hence, using the recursive specification principle (see appendix E and [2, 9]), that states that a solution of a guarded recursive specification is unique modulo (initially stateless) bisimilarity, we may conclude $R_i \gg X_i \Leftrightarrow \partial_S(R_i \gg X_i)$ and especially $X_0 \leftrightarrows [\ \mathit{true}\ ] \gg X_0 \leftrightarrows R_0 \gg X_0 \Leftrightarrow \partial_{P_{\mathrm{m}}}(R_0 \gg X_0) \leftrightarrows \partial_{P_{\mathrm{m}}}(X_0)$. Hence, X_0 is safe.

We have shown, that the question of safety for actions or safety for predicates of a recursion variable X_0 in a linear recursive specification, depends only on the safety of the closed terms $\big(R_i \sim d'_j\big) \gg a_j$ and $\big(R_i \sim d''_j\big) \gg c$ with $i \in I$ and $j \in J'(i)$, or $j \in J''(i)$, respectively. Admittedly, the iteration leading to the initial conditions R_i is possibly infinite. One reason for this is the fact that the set I, from which the indices are chosen, can be infinite. Another reason may be that the set I is finite, but that every iteration in the definition of R_i introduces new possible initial conditions. Luckily, for the example of Fischer's protocol that we study in the next section, the iteration terminates. Future research should, perhaps, concentrate on finding solutions for the possibly infinite computation. One possible direction, in case I is finite, is to use over-approximations of R_i, as is done in the method of predicate abstraction described in [1]. Another possible direction, in case I is infinite, is to use induction techniques to derive R_i algebraically.

6 Safety of a hybrid variant of Fischer's protocol

A classical case study for safety analysis, is Fischer's time based mutual exclusion protocol [18]. The protocol, often studied using timed automata [19, 3, 1], consists of a number of identical processes (up to their parameters) containing four modes each. The goal of the protocol is to ensure that no two processes enter a certain access mode at the same time. In our study, we will restrict ourselves to the case where there are only two participants. The case in which we have more participants, is expected to be safe under similar conditions, but more research is needed to further that claim.

Although the protocol is not intrinsically hybrid, the variant we study in this section calls for a hybrid approach. In its original form, the protocol uses timing restrictions on both processes to guarantee safety. Our goal is to study in which way the safety of the protocol is affected, by a difference in speed between the clocks of the two processes. In particular, we study a relative error between the clocks. This introduces a dependency between the clocks, that makes the problem hybrid in nature. The clocks that are used in our variant of the protocol can run with arbitrary speed, assuming that there is a bounded relative error between the speeds with which they run.

In figure 1 we have depicted a hybrid automaton, modeling one of the participating processes. Below, we give the associated recursive specification in HyPA. In our version of the protocol, we require only that clocks (modeled by the x_i variables) are increasing ($\dot{x}_i \geq 0$) and that there is at most a proportional error $e \geq 1$ in the derivatives of the clocks of each of the processes. The protocol is parameterized by the constant bounds d and D on clocks, and by the constant

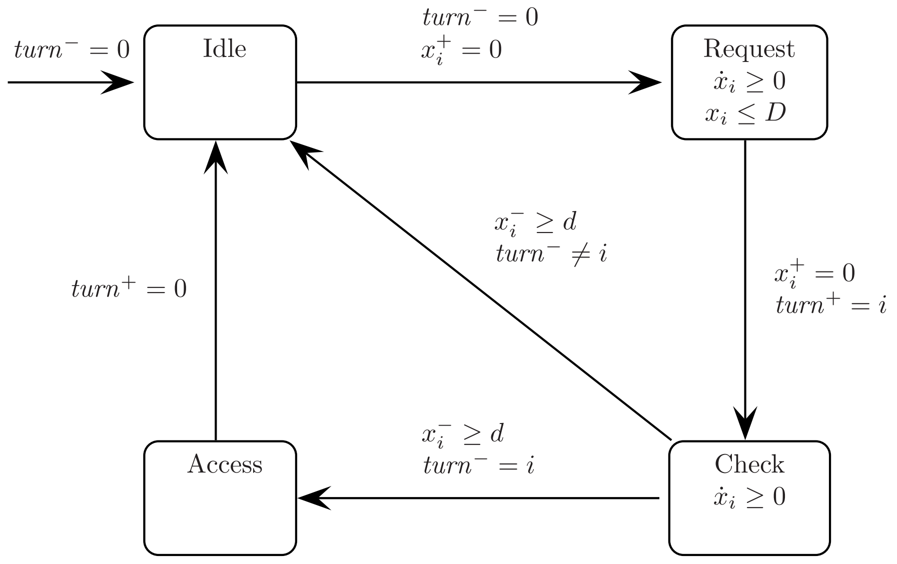

Figure 1: One process in Fischer's protocol

proportional error e. The idea of a relative error, is modeled as a separate constraint C, and turns out to be sufficient to guarantee safety of the protocol if the parameters satisfy $d > e \cdot D > 0$.

$$\text{C} \;:\; \left(\begin{array}{c} \dot{x}_1 \leq e\ \dot{x}_2 \\ \dot{x}_2 \leq e\ \dot{x}_1 \end{array} \right)$$

Note, that we find $C \Leftrightarrow \delta$ if $e < 1$, therefore, we assume $e \geq 1$ in the remainder of this report.

The variable *turn*, is shared by all participating processes, and is used to communicate which process wants to access the resource. The variable loc_i, is used in the process algebraic description of the protocol, to refer to the name of the associated node of the hybrid automaton. Both *turn* and loc_i are discrete variables, and we assume that all solutions to flow-predicates keep these variables constant. The ι actions, are used in the process algebraic description to represent transitions in the hybrid automaton. These actions are assumed not to synchronize. We model this by leaving $(\iota\ \gamma\ \iota)$ and $(\iota\ \gamma$ access) undefined.

$$\text{Idle}_i \;:\; \left(\begin{array}{c|c} turn & \\ loc_i & loc_i = \text{Idle} \end{array} \right) \blacktriangleright \left[\begin{array}{c|l} & turn^- = 0 \\ x_i & x_i^+ = 0 \\ loc_i & loc_i^+ = \text{Request} \end{array} \right] \gg \iota \odot \text{Request}_i$$

$$\text{Request}_i \;:\; \left(\begin{array}{c|l} x_i & loc_i = \text{Request} \\ turn & \dot{x}_i \geq 0 \\ loc_i & x_i \leq D \end{array} \right) \blacktriangleright \left[\begin{array}{c|l} x_i & x_i^+ = 0 \\ loc_i & turn^+ = i \\ turn & loc_i^+ = \text{Check} \end{array} \right] \gg \iota \odot \text{Check}_i$$

$$\text{Check}_i \;:\; \left(\begin{array}{c|l} x_i & \\ turn & loc_i = \text{Check} \\ loc_i & \dot{x}_i \geq 0 \end{array} \right) \blacktriangleright \left(\begin{array}{l} \left[\begin{array}{c|l} & x_i^- \geq d \\ loc_i & turn^- \neq i \\ & loc_i^+ = \text{Idle} \end{array} \right] \gg \iota \odot \text{Idle}_i \\ \oplus \\ \left[\begin{array}{c|l} & x_i^- \geq d \\ loc_i & turn^- = i \\ & loc_i^+ = \text{Access} \end{array} \right] \gg \iota \odot \text{Access}_i \end{array} \right)$$

$$\text{Access}_i \;:\; \left(\begin{array}{c|c} turn & \\ loc_i & loc_i = \text{Access} \end{array} \right) \blacktriangleright \left[\begin{array}{c|l} turn & turn^+ = 0 \\ loc_i & loc_i^+ = \text{Idle} \end{array} \right] \gg \iota \odot \text{Idle}_i$$

Our verification goal, is to show that loc_1 and loc_2 are never set to 'Access' at the same time, during the execution of the protocol. We can show that the protocol is safe, if the parameters satisfy $d > e \cdot D > 0$, and if we have the following initial condition:

$$\text{init} = \left[\begin{array}{l} \mathit{turn}^- = 0 \\ \mathit{loc}_1^- = \text{Idle} \\ \mathit{loc}_2^- = \text{Idle} \end{array} \right] .$$

Formally, we have to prove for the process term

$$S \ : \ \text{init} \gg (\text{Idle}_1 \parallel \text{Idle}_2 \parallel \text{C})$$

that it is safe for the predicate $P_{\text{m}} = (\mathit{loc}_1 = \mathit{loc}_2 = \text{Access})$. I.e. that $S \leftrightarrows \partial_{P_{\text{m}}}(S)$.

Notice, that the initial condition is a global condition, i.e. one condition on all initial processes. The reason for this, is that we are modeling a number of parallel hybrid automata, for which the initial conditions are synchronized. If each process would be initialized separately, according to $\text{init}_i = \left[\mathit{turn}^- = 0 \ \wedge \ \mathit{loc}_i^- = \text{Idle}\right]$, it turns out that a deadlock may occur if one of the processes immediately performs actions leading to the check mode. In that case, one process must delay, while the other cannot because the initial condition is not satisfied.

Next, we show some of the results, of applying the method of safety analysis presented in the previous section, to our variant of Fischer's protocol. Obviously, the definition of S is not a linear process definition. However, using the rewriting techniques described in [9, 26], we can rewrite it into the linear recursive specification given in appendix F, for which we have

$$S \leftrightarrows \text{init} \gg X_{i,i} \ .$$

In the linearization, there are 16 recursion variables involved. There is one variable for each pair of recursion variables of the original two processes. We use the letter X for the variables in the linearization, and label them using pairs of abbreviated location names ((i,i) for (Idle,Idle), (i,r) for (Idle,Request), etc.). As an example, we find the following definition for the variable $X_{c,r}$, which will turn out represent a decision in the protocol.

$$\begin{array}{lcl}
X_{c,r} & : & \left[\ \mathit{true} \ \right] \gg \left(\begin{array}{l|l} \mathit{turn} & \mathit{loc}_1 = \text{Check} \\ \mathit{loc}_1 & \mathit{loc}_2 = \text{Request} \\ \mathit{loc}_2 & 0 \leq \dot{x}_1 \leq e \ \dot{x}_2 \\ x_1 & 0 \leq \dot{x}_2 \leq e \ \dot{x}_1 \\ x_2 & x_2 \leq D \end{array} \right) \rhd X_{c,r} \\
& \oplus & \left[\begin{array}{l|l} x_1 & x_2^+ = 0 \\ \mathit{loc}_1 & \mathit{turn}^+ = 2 \\ \mathit{turn} & \mathit{loc}_2^+ = \text{Check} \end{array} \right] \gg \iota \odot X_{c,c} \oplus \left[\begin{array}{l|l} & x_1^- \geq d \\ \mathit{loc}_2 & \mathit{turn}^- \neq 1 \\ & \mathit{loc}_1^+ = \text{Idle} \end{array} \right] \gg \iota \odot X_{r,i} \\
& \oplus & \left[\begin{array}{l|l} & x_1^- \geq d \\ \mathit{loc}_2 & \mathit{turn}^- = 1 \\ & \mathit{loc}_1^+ = \text{Access} \end{array} \right] \gg \iota \odot X_{r,a}
\end{array}$$

As an example of the calculations involved in the safety analysis of the 16 recursion variables, we show the calculation of the re-initialization clause $d''_{c,r} \sim D(c_{c,r})$.

$$d''_{c,r} \sim D(c_{c,r}) \ = \ \left[\ \mathit{true} \ \right] \sim D \left(\begin{array}{l|l} \mathit{turn} & \mathit{loc}_1 = \text{Check} \\ \mathit{loc}_1 & \mathit{loc}_2 = \text{Request} \\ \mathit{loc}_2 & 0 \leq \dot{x}_1 \leq e \ \dot{x}_2 \\ x_1 & 0 \leq \dot{x}_2 \leq e \ \dot{x}_1 \\ x_2 & x_2 \leq D \end{array} \right) \ = \ D \left(\begin{array}{l|l} \mathit{turn} & \mathit{loc}_1 = \text{Check} \\ \mathit{loc}_1 & \mathit{loc}_2 = \text{Request} \\ \mathit{loc}_2 & 0 \leq \dot{x}_1 \leq e \ \dot{x}_2 \\ x_1 & 0 \leq \dot{x}_2 \leq e \ \dot{x}_1 \\ x_2 & x_2 \leq D \end{array} \right)$$

$$= \left[\begin{array}{l|l} & \mathit{loc}_1^- = \text{Check} \\ & \mathit{loc}_2^- = \text{Request} \\ \begin{array}{l} x_1 \\ x_2 \end{array} & \exists_{t,v_1,v_2} \left\{ \begin{array}{l} x_1^+ = x_1^- + \int_0^t v_1(\tau) d\tau \\ x_2^+ = x_2^- + \int_0^t v_2(\tau) d\tau \\ \forall_\tau \ 0 \leq v_1(\tau) \leq e \ v_2(\tau) \\ \forall_\tau \ 0 \leq v_2(\tau) \leq e \ v_1(\tau) \\ \forall_{t' \leq t} \ x_2^- + \int_0^{t'} v_2(\tau) d\tau \leq D \end{array} \right. \end{array} \right] = \left[\begin{array}{l|l} & \mathit{loc}_1^- = \text{Check} \\ & \mathit{loc}_2^- = \text{Request} \\ & 0 \leq (x_1^+ - x_1^-) \\ x_1 & (x_1^+ - x_1^-) \leq e(x_2^+ - x_2^-) \\ x_2 & 0 \leq (x_2^+ - x_2^-) \\ & (x_2^+ - x_2^-) \leq e(x_1^+ - x_1^-) \\ & x_2^+ \leq D \end{array} \right]$$

Calculation of R_i in this way, is straightforward for Fischer's protocol. Admittedly, this calculation is performed in a rather ad hoc manner. The calculation of $(R_i \sim D(c_i))^!$, and similar terms, turns out to be hard to automate in practice, and more research is needed to find out for which classes of flow-clauses and re-initialization clauses these calculations are at all possible. Some work in this direction is done, for example, in [1], showing results on the calculation of $(R_i \sim D(c_i))^!$ for linear differential equations c_i and polyhedral initial conditions R_i.

For our variant of Fischer's protocol, we ultimately, we find the following R_i, where i is taken over the aforementioned pairs. Those re-initializations that are not mentioned here, can be easily obtained using symmetry of the protocol.

$$\begin{array}{lcl}
R_{i,i} & = & \left[\ \mathit{turn}^- = 0 \ \wedge\ \mathit{loc}_1^- = \text{Idle} \ \wedge\ \mathit{loc}_2^- = \text{Idle}\ \right] \\
R_{r,i} & = & \left[\ \mathit{turn}^- = 0 \ \wedge\ \mathit{loc}_1^- = \text{Request} \ \wedge\ \mathit{loc}_2^- = \text{Idle} \ \wedge\ x_1^- \geq 0\ \right] \\
R_{r,r} & = & \left[\begin{array}{l} \mathit{turn}^- = 0 \ \wedge\ \mathit{loc}_1^- = \text{Request} \ \wedge\ \mathit{loc}_2^- = \text{Request} \ \wedge \\ \left(\begin{array}{l} 0 \leq x_1^- \leq D \\ x_1^- \leq e \cdot x_2^- \leq e \cdot D \end{array}\right) \vee \left(\begin{array}{l} x_2^- \leq e \cdot x_1^- \leq e \cdot D \\ 0 \leq x_2^- \leq D \end{array}\right) \end{array}\right] \\
R_{c,i} & = & \left[\ \mathit{turn}^- = 1 \ \wedge\ \mathit{loc}_1^- = \text{Check} \ \wedge\ \mathit{loc}_2^- = \text{Idle} \ \wedge\ x_1^- \geq 0\ \right] \\
R_{c,r} & = & \left[\ \mathit{turn}^- = 1 \ \wedge\ \mathit{loc}_1^- = \text{Check} \ \wedge\ \mathit{loc}_2^- = \text{Request} \ \wedge\ 0 \leq x_1^- \leq e \cdot x_2^- \leq e \cdot D\ \right] \\
R_{c,c} & = & \left[\begin{array}{l} \mathit{loc}_1^- = \text{Check} \ \wedge\ \mathit{loc}_2^- = \text{Check} \ \wedge \\ \left(\begin{array}{l} \mathit{turn}^- = 1 \ \wedge \\ 0 \leq x_1^- \leq e \cdot x_2^- \\ 0 \leq x_2^- \leq e \cdot x_1^- + D \end{array}\right) \vee \left(\begin{array}{l} \mathit{turn}^- = 2 \ \wedge \\ 0 \leq x_1^- \leq e \cdot x_2^- + D \\ 0 \leq x_2^- \leq e \cdot x_1^- \end{array}\right) \end{array}\right] \\
R_{a,c} & = & \left[\begin{array}{l} \mathit{turn}^- = 1 \ \wedge\ \mathit{loc}_1^- = \text{Access} \ \wedge\ \mathit{loc}_2^- = \text{Check} \ \wedge \\ d \leq x_1^- \leq e \cdot x_2^- \ \wedge\ d \leq x_2^- \leq e \cdot x_1^- + D \end{array}\right] \\
R_{a,i} & = & \left[\ \mathit{turn}^- = 1 \ \wedge\ \mathit{loc}_1^- = \text{Access} \ \wedge\ \mathit{loc}_2^- = \text{Idle}\ \right] \\
R_{a,r} & = & R_{a,a} \ = \ \left[\ \mathit{false}\ \right]
\end{array}$$

Note, that $R_{a,a} = [\ \mathit{false}\ ]$, is crucial to the safety of the protocol. It signifies that the unwanted recursion variable $X_{a,a}$, in which both parties access the resource at the same time, is unreachable. A prerequisite for this, turns out to be that $R_{a,r} = [\ \mathit{false}\ ]$, which in turn follows from the fact that the discrete action from $X_{c,r}$ to $X_{a,r}$, is blocked. In other words, to obtain $R_{a,a} = [\ \mathit{false}\ ]$, we indirectly need that

$$\begin{array}{rcl}
R_{c,r} \sim d'_{c,r,a,r} & = & \left[\begin{array}{l} \mathit{turn}^- = 1 \\ \mathit{loc}_1^- = \text{Check} \\ \mathit{loc}_2^- = \text{Request} \\ 0 \leq x_1^- \leq e \cdot x_2^- \leq e \cdot D \end{array}\right] \sim \left[\begin{array}{c|l} \mathit{loc}_2 & \begin{array}{l} x_1^- \geq d \\ \mathit{turn}^- = 1 \\ \mathit{loc}_1^+ = \text{Access} \end{array} \end{array}\right] \\
 & = & \left[\begin{array}{l} \mathit{turn}^- = 1 \\ \mathit{loc}_1^- = \text{Check} \ \wedge\ \mathit{loc}_1^+ = \text{Access} \\ \mathit{loc}_2^- = \text{Request} \\ d \leq x_1^- \leq e \cdot x_2^- \leq e \cdot D \end{array}\right] = \left[\ \mathit{false}\ \right]
\end{array}$$

This is only true, because of our initial assumption that $d > e \cdot D$.

Lastly, we have to verify safety for the predicate $P_{\mathrm{m}} = (\mathit{loc}_1 = \mathit{loc}_2 = \text{Access})$, for the processes, $\left(R_i \sim d'_j\right) \gg a_j$, with $i \in I$ and $j \in J'(i)$, and $\left(R_i \sim d''_j\right) \gg c_j$, with $i \in I$ and $j \in J''(i)$. This is straightforward to do, and in fact, can be concluded already from the fact that $R_{a,a} = [\ \mathit{false}\ ]$. We may finally conclude that Fischer's protocol is indeed safe, if initially $d > e \cdot D > 0$.

7 Conclusive remarks

In this report, we have shown how the hybrid process algebra HyPA can be used for the analysis of safety properties of hybrid systems. More precisely, we have shown how the question of safety for actions and safety for predicates on model variables can be broken into (hopefully simpler) questions, regarding flow clauses and re-initialization clauses. We have reduced the problem of action and predicate safety of processes to smaller questions that cannot be answered using process algebra, but must be answered using the theories for describing flows and re-initializations.

As an example, we have analyzed a variant of Fischer's protocol, in which the only restriction on the internal clocks is that they are monotone, and that their rates do not differ by more than a given factor. Admittedly, similar analysis of this protocol has been shown before, but not with those specific constraints, that in our opinion give new insights in the way the protocol works. Hybrid process algebra, provides us with the opportunity to model dependence between the continuous variables of parallel processes, which is used in the analysis of Fischer's protocol when we restrict the relative error between clocks.

Our recommendations for future research can roughly be divided into three directions. The first direction is research on the process algebra HyPA itself. In particular, the two phase analysis strategy that is used in this report, in which first a process definition is turned into a linear process definition using one equivalence, and then further analysis is carried out using a weaker equivalence, deserves more attention.

The second direction, is with respect to safety properties. The method we have shown in this report, relies on an iteration that possibly does not terminate, and relies on the ability to analyze flow-clauses and re-initialization clauses. Perhaps, the method can be combined with methods like predicate abstraction [1], in order to guarantee termination and make automated calculation on clauses possible. Even then, calculation on reachable sets (for our method captured in the formulas $(R_i \sim d'_j)^!$ and $(R_i \sim d''_j \sim D(c_j))^!$), is often difficult. For simple linear differential equations, one needs assumptions on the initial conditions in order to be able to perform calculations automatically (see [1]). For more difficult flow-clauses, the calculation becomes impossible all together. In the example we have shown in this report, symbolic reasoning solved the problem for us, for a large part. However, this is only an option in manual analysis, or when process algebraic tools can be coupled to tools like Mathematica, in which symbolic reasoning is possible. An advantage of symbolic reasoning, is that also induction on the indices may be used, to exploit a possible structure in R_i.

It may be interesting to note, that we have recently obtained new results, regarding manual analysis of the variant of Fischer's protocol discussed in this report. By inverse reasoning on the construction of R_i, we have been able to derive the initial conditions for which the protocol is safe, from the algebraic description of the protocol, rather than assuming them a priori. More research in the direction of inverse reasoning on R_i, will hopefully lead to a more general strategy for controlling processes in a safe manner.

The third direction, is to consider other kinds of analysis problems as well. More complicated safety properties, but also fairness properties of systems (a certain desired property is eventually fulfilled) and analysis problems from the field of control theory, like for example stability, come to mind. Process algebraic specification of these analysis problems, would then be the first start. Admittedly, we expect the specification of fairness properties to be more difficult than the specification of safety, because, in general, eventuality of an event is hard to express in process algebra.

Acknowledgements The authors would like to thank George Pappas, Paulo Tabuada, Franjo Ivančič, Insup Lee, Rajeev Alur, Oleg Sokolsky and the other people at PENN's Grasp-lab and hybrid systems group, for the fruitful discussions during Pieter's short but enjoyable visit to their group. Much of the work presented in this report, originated there. Furthermore, we would like to thank Paul van den Bosch, Jan Friso Groote, Maurice Heemels, Aleksandar Juloski, Tim Willemse and our industrial partners, for their valuable feedback during the development of HyPA. Lastly, our gratitude goes to Progress/STW (Grant EES5173), Philips-CFT and Assembleon, for their financial and material support.

References

[1] R. Alur, T. Dang, and F. Ivančić. Reachability analysis of hybrid systems via predicate abstraction. In *Fifth International Workshop on Hybrid Systems: Computation and Control*, volume 2289 of *Lecture Notes in Computer Science*, pages 35–48. Springer-Verlag, 2002.

[2] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Trancts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 1990.

[3] F. Balarin. Approximate reachability analysis of timed automata. In *Proc. Real-Time Systems Symposium '96*, pages 52–61. IEEE, 1996.

[4] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[5] J.A. Bergstra and J.W. Klop. Verification of an alternating bit protocol by means of process algebra. volume 215 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 1986.

[6] J.A. Bergstra and C.A. Middelburg. Process algebra for hybrid systems. Technical Report CSR 03-06, TU/e, Eindhoven, Netherlands, 2003.

[7] M.S. Branicky, V.S. Borkar, and S.K. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):pp. 31–45, January 1998.

[8] E. Brinksma. A tutorial on LOTOS. In Michel Diaz, editor, *Proc. Protocol Specification, Testing and Verification V*, pages 171–194, Amsterdam, Netherlands, 1985.

[9] P.J.L. Cuijpers and M.A. Reniers. Hybrid process algebra. *Journal of Logic and Algebraic Programming.* to appear.

[10] P.J.L. Cuijpers and M.A. Reniers. Hybrid process algebra. Technical Report CSR 03-07, TU/e, Eindhoven, Netherlands, 2003.

[11] P.J.L. Cuijpers, M.A. Reniers, and W.P.M.H. Heemels. Hybrid transition systems. Technical Report CS-Report 02-12, TU/e, Eindhoven, Netherlands, 2002.

[12] J. Davoren and A. Nerode. Logics for hybrid systems. In P. Antsaklis and J.H. van Schuppen, editors, *Proceedings of the IEEE Special Issue on Hybrid Systems: Theory and Applications*, volume 88, pages 985–1010, July 2000.

[13] J.F. Groote and M.A. Reniers. Algebraic process verification. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 17, pages 1151–1208. Elsevier Science B.V., Amsterdam, 2001.

[14] J.F. Groote and J.J. van Wamel. Analysis of three hybrid systems in timed μCRL. *Science of Computer Programming*, 39:215–247, 2001.

[15] E. Haghverdi, P. Tabuada, and G.J. Pappas. Bisimulation relations for dynamical and control systems. In R. Blute and P. Selinger, editors, *Electronic Notes in Theoretical Computer Science*, volume 69. Elsevier, 2003.

[16] T.A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pages 278–292. IEEE Computer Society Press, 1996.

[17] H. Jifeng. From CSP to hybrid systems. In A.W.Roscoe, editor, *A Classical Mind, Essays in Honour of C.A.R. Hoare*, pages 171–189. Prentice-Hall International, 1994.

[18] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.

[19] K.G. Larsen, P. Petterson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *Proc. Real-Time Systems Symposium '95*, pages 76–87. IEEE, 1995.

[20] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata revisited. In M.D. Di Benedetto and A.L. Sangiovanni-Vincentelli, editors, *Proceedings Fourth International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 403–417. Springer-Verlag, 2001.

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

[22] M. Moussavi, M.A. Reniers, and J.F. Groote. Congruence for SOS with data. Technical Report CSR-04-05, Technische Universiteit Eindhoven (TU/e), 2004.

[23] W.C. Rounds and H. Song. The ϕ-calculus: A language for distributed control of reconfigurable embedded systems. In F. Wiedijk, O. Maler, and A. Pnueli, editors, *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003*, volume 2623 of *Lecture Notes in Computer Science*, pages 435–449. Springer-Verlag, 2003.

[24] R.R.H. Schiffelers, D.A. van Beek, K.L. Man, M.A. Reniers, and J.E. Rooda. Formal semantics of hybrid chi. In K.G. Larssen and P. Niebert, editors, *Formal Modelling and Analysis of Timed Systems*, Lecture Notes in Computer Science.

[25] E.D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, volume 6 of *Texts in Applied Mathematics*. Springer-Verlag, 1998.

[26] Y.S. Usenko. *Linearization in μCRL*. PhD thesis, Technische Universiteit Eindhoven (TU/e), 2002.

[27] 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.

A Semantics HyPA

Table 4: Operational semantics of HyPA

$$\frac{}{<\epsilon,\nu>\checkmark}(1) \qquad \frac{}{<a,\nu> \overset{a,\nu}{\mapsto} <\epsilon,\nu>}(2) \qquad \frac{(\nu,\sigma)\models_c c,\ \ dom(\sigma)=[0,t]}{<c,\nu> \overset{\sigma}{\rightsquigarrow} <c,\sigma(t)>}(3)$$

$$\frac{(\nu,\nu')\models_d d,\ <p,\nu'>\checkmark}{<d \gg p,\nu>\checkmark}(4) \qquad \frac{(\nu,\nu')\models_d d,\ <p,\nu'> \overset{l}{\rightarrow} <p',\nu''>}{<d \gg p,\nu> \overset{l}{\rightarrow} <p',\nu''>}(5)$$

Table 5: Operational semantics of HyPA, alternative and sequential composition

$$\frac{<p,\nu>\checkmark}{\begin{array}{c}<p \oplus q,\nu>\checkmark\\ <q \oplus p,\nu>\checkmark\end{array}}(6) \qquad \frac{<p,\nu> \xrightarrow{l} <p',\nu'>}{\begin{array}{c}<p \oplus q,\nu> \xrightarrow{l} <p',\nu'>\\ <q \oplus p,\nu> \xrightarrow{l} <p',\nu'>\end{array}}(7) \qquad \frac{<p,\nu>\checkmark,\ <q,\nu>\checkmark}{<p \odot q,\nu>\checkmark}(8)$$

$$\frac{<p,\nu> \xrightarrow{l} <p',\nu'>}{<p \odot q,\nu> \xrightarrow{l} <p' \odot q,\nu'>}(9) \qquad \frac{<p,\nu>\checkmark,\ <q,\nu> \xrightarrow{l} <q',\nu'>}{<p \odot q,\nu> \xrightarrow{l} <q',\nu'>}(10)$$

Table 6: Operational semantics of HyPA, disrupt

$$\frac{<p,\nu>\checkmark}{\begin{array}{c}<p \blacktriangleright q,\nu>\checkmark\\ <p \triangleright q,\nu>\checkmark\end{array}}(11) \qquad \frac{<p,\nu> \xrightarrow{l} <p',\nu'>}{\begin{array}{c}<p \blacktriangleright q,\nu> \xrightarrow{l} <p' \blacktriangleright q,\nu'>\\ <p \triangleright q,\nu> \xrightarrow{l} <p' \blacktriangleright q,\nu'>\end{array}}(12)$$

$$\frac{<q,\nu>\checkmark}{<p \blacktriangleright q,\nu>\checkmark}(13) \qquad \frac{<q,\nu> \xrightarrow{l} <q',\nu'>}{<p \blacktriangleright q,\nu> \xrightarrow{l} <q',\nu'>}(14)$$

Table 7: Operational semantics of HyPA, parallel composition

$$\frac{<p,\nu>\checkmark, <q,\nu>\checkmark}{\begin{array}{c}<p \,\|\, q,\nu>\checkmark\\ <p \,|\, q,\nu>\checkmark\end{array}}(15) \qquad \frac{<p,\nu> \overset{\sigma}{\rightsquigarrow} <p',\nu'>,\ <q,\nu> \overset{\sigma}{\rightsquigarrow} <q',\nu'>}{\begin{array}{c}<p \,\|\, q,\nu> \overset{\sigma}{\rightsquigarrow} <p' \,\|\, q',\nu'>\\ <p \,|\, q,\nu> \overset{\sigma}{\rightsquigarrow} <p' \,\|\, q',\nu'>\end{array}}(16)$$

$$\frac{<p,\nu> \overset{\sigma}{\rightsquigarrow} <p',\nu'>,\ <q,\nu>\checkmark}{\begin{array}{c}<p \,\|\, q,\nu> \overset{\sigma}{\rightsquigarrow} <p',\nu'>\\ <q \,\|\, p,\nu> \overset{\sigma}{\rightsquigarrow} <p',\nu'>\\ <p \,|\, q,\nu> \overset{\sigma}{\rightsquigarrow} <p',\nu'>\\ <q \,|\, p,\nu> \overset{\sigma}{\rightsquigarrow} <p',\nu'>\end{array}}(17) \qquad \frac{<p,\nu> \overset{a,\nu'}{\mapsto} <p',\nu''>}{\begin{array}{c}<p \,\|\, q,\nu> \overset{a,\nu'}{\mapsto} <p' \,\|\, q,\nu''>\\ <q \,\|\, p,\nu> \overset{a,\nu'}{\mapsto} <q \,\|\, p',\nu''>\\ <p \mathbin{\lfloor\!\lfloor} q,\nu> \overset{a,\nu'}{\mapsto} <p' \,\|\, q,\nu''>\end{array}}(18)$$

$$\frac{<p,\nu> \overset{a,\nu'}{\mapsto} <p',\nu''>,\ <q,\nu> \overset{a',\nu'}{\mapsto} <q',\nu''>,\ a'' = a\,\gamma\,a'}{\begin{array}{c}<p \,\|\, q,\nu> \overset{a'',\nu'}{\mapsto} <p' \,\|\, q',\nu''>\\ <p \,|\, q,\nu> \overset{a'',\nu'}{\mapsto} <p' \,\|\, q',\nu''>\end{array}}(19)$$

Table 8: Operational semantics of HyPA, encapsulation and recursion

$$\frac{<p,\nu> \overset{a,\nu'}{\mapsto} <p',\nu''>,\ a \notin H}{<\partial_H(p),\nu> \overset{a,\nu'}{\mapsto} <\partial_H(p'),\nu''>}(20)$$

$$\frac{<p,\nu> \overset{\sigma}{\rightsquigarrow} <p',\nu'>}{<\partial_H(p),\nu> \overset{\sigma}{\rightsquigarrow} <\partial_H(p'),\nu'>}(21) \qquad \frac{<p,\nu> \checkmark}{<\partial_H(p),\nu> \checkmark}(22)$$

$$\frac{<p,\nu> \checkmark}{<X,\nu> \checkmark}(23)\ \ X \Leftrightarrow p \in E \qquad \frac{<p,\nu> \xrightarrow{l} <p',\nu'>}{<X,\nu> \xrightarrow{l} <p',\nu'>}(24)\ \ X \Leftrightarrow p \in E$$

B Specification of safety: proof

Theorem 12 (Specification of safety for actions) *A process term p is safe for actions in H if and only if $p \Leftrightarrow \partial_H(p)$.*

Proof Suppose that indeed $p \Leftrightarrow \partial_H(p)$. Obviously, $\partial_H(p)$ is safe for actions in H, since all unsafe actions are blocked. Now, suppose $\mathcal{S}$ is a bisimulation relation that witnesses $p \Leftrightarrow \partial_H(p)$. Then every reachable state from p has a related reachable state from $\partial_H(p)$. I.e. for every $(q,\nu') \in \mathcal{R}(p,\nu)$, with $\nu,\nu' \in \mathit{Val}$, we find that there exists $\mu,\mu' \in \mathit{Val}$ and $(q',\nu'') \in \mathcal{R}(\partial_H(p),\nu)$, such that $(q,\nu')\mathcal{S}(q',\nu'')$. Suppose that there is a transition $<q,\nu'> \overset{a,\mu}{\mapsto} <r,\mu'>$, then we may use the fact that $\mathcal{S}$ is a bisimulation relation to conclude that there exists (r',μ'') such that $<q',\nu'> \overset{a,\mu}{\mapsto} <r'',\mu''>$. Finally, from the fact that $\partial_H(p)$ is safe, we conclude that $a \notin H$. So, every transition from reachable states from p is safe, and hence p is safe.

Now, consider the reverse case, which we prove in more detail. Let p be a process term that is safe for actions in H. Then we define the following relation

$$\mathcal{S} = \{(<q,\nu>,<\partial_H(q),\nu>) \mid \exists_{\nu'}\ <q,\nu> \in \mathcal{R}(p,\nu')\},$$

and show that it is a bisimulation relation that witnesses $p \Leftrightarrow \partial_H(p)$.

Obviously, it is a witness relation, since $<p,\nu> \in \mathcal{R}(<p,\nu>)$ for all ν and hence $<p,\nu>\mathcal{S}<\partial_H(p),\nu>$. Now, to show that it is a bisimulation relation, assume that $<x,\nu>\mathcal{S}<\partial_H(x),\nu>$ and consider the following cases:

- $<x,\nu> \checkmark$
  Using the definition of encapsulation we find $<\partial_H(x),\nu> \checkmark$.
- $<\partial_H(x),\nu> \checkmark$
  For this, we need to assume $<x,\nu> \checkmark$, according to the semantics of encapsulation.
- $<x,\nu> \overset{a,\mu}{\mapsto} <x',\mu'>$
  Using the fact that $<x,\nu> \in \mathcal{R}(<p,\xi>)$ for some ξ, and using the fact that p is safe for actions in H, we conclude that $a \notin H$. Then, using the semantics of encapsulation of actions, we conclude $<\partial_H(x),\nu> \overset{a,\mu}{\mapsto} <\partial_H(x'),\mu'>$. with $<x',\mu'>\mathcal{S}<\partial_H(x'),\mu'>$.

- $<\partial_H(x), \nu'> \overset{a,\mu}{\mapsto} <y, \mu'>$
  From the semantics of encapsulation it follows that for $<\partial_H(x), \nu> \overset{a,\mu}{\mapsto} <y, \mu'>$ we need that y is of the form $\partial_H(x')$ and that $<x, \nu> \overset{a,\mu}{\mapsto} <x', \mu'>$. Finally, because $<x', \mu'>$ is reachable from $<x, \mu>$ and hence is reachable from $<p, \xi>$ for some ξ, we may conclude $<x', \mu'>\mathcal{S}<\partial_{x'}(\mu')>$.

- $<x, \nu> \overset{\sigma}{\rightsquigarrow} <x', \mu>$
  Using the semantics of encapsulation of actions, we conclude $<\partial_H(x), \nu> \overset{\sigma}{\rightsquigarrow} <\partial_H(x'), \mu>$ with $<x', \mu'>\mathcal{S}<\partial_H(x'), \mu'>$.

- $<\partial_H(x), \nu'> \overset{\sigma}{\mapsto} <y, \mu'>$
  From the semantics of encapsulation it follows that for $<\partial_H(x), \nu> \overset{\sigma}{\mapsto} <y, \mu'>$ we need that y is of the form $\partial_H(x')$ and that $<x, \nu> \overset{\sigma}{\mapsto} <x', \mu'>$. Finally, because $<x', \mu'>$ is reachable from $<x, \mu>$ and hence is reachable from $<p, \xi>$ for some ξ, we may conclude $<x', \mu'>\mathcal{S}<\partial_{x'}(\mu')>$.

⊠

C Axioms for predicate encapsulation

C.1 $\partial_{P_{\mathrm{m}}}(d \gg \delta) \leftrightarrows \delta$

Trivial.

C.2 $\partial_{P_{\mathrm{m}}}(d \gg \epsilon) \leftrightarrows d \gg \epsilon$

Trivial.

C.3 $\partial_{P_{\mathrm{m}}}(d \gg a) \leftrightarrows (d \sim [\neg P^{-}_{\mathrm{m}}]) \gg a$

The axiom $\partial_{P_{\mathrm{m}}}(d \gg a) \leftrightarrows (d \sim [\neg P^{-}_{\mathrm{m}}]) \gg a$ is, for given $P_{\mathrm{m}} \in \mathcal{P}_{\mathrm{m}}$, $d \in D$ and $a \in \mathcal{A}$, witnessed by the relation $\mathcal{S} = \{(<\partial_{P_{\mathrm{m}}}(d \gg a), \nu>, <(d \sim [\neg P^{-}_{\mathrm{m}}]) \gg a, \nu>) \mid \nu \in \mathit{Val}\} \cup \{(<\partial_{P_{\mathrm{m}}}(\epsilon), \nu>, <\epsilon, \nu>) \mid \nu \in \mathit{Val}\}$. That this is indeed a witnessing relation is obvious, and also that it is robust is straightforward to verify. That it is a bisimulation relation follows from the following cases.

Assume that $<p, \nu>\mathcal{S}<q, \nu>$. From the definition, it follows that either $p = \partial_{P_{\mathrm{m}}}(d \gg a)$ and $q = d \sim [\neg P^{-}_{\mathrm{m}}] \gg a$, or $p = \partial_{P_{\mathrm{m}}}(\epsilon)$ and $q = \epsilon$. Since the proofs for the second case is trivial, we will only discuss the first.

Because actions cannot terminate, nor can they perform flow transitions, we find that $\partial_{P_{\mathrm{m}}}(d \gg a)$ and $(d \sim [\neg P^{-}_{\mathrm{m}}]) \gg a$ can both not terminate or perform flow transitions. We therefore only need to consider the following cases.

- $<\partial_{P_{\mathrm{m}}}(d \gg a), \nu> \overset{a,\mu}{\mapsto} <p', \mu'>$
  For this, according to the semantics of HyPA, we need the assumption that $\mu \models_{\mathrm{m}} \neg P_{\mathrm{m}}$, that $p' = \partial_{P_{\mathrm{m}}}(\epsilon)$ and that $(\nu, \mu) \models_d d$. Hence, $(\nu, \mu) \models_d (d \sim \neg P^{-}_{\mathrm{m}})$, from which we conclude

that $<(d \sim [\neg P_\mathrm{m}^-]) \gg a, \nu> \overset{a,\mu}{\mapsto} <\epsilon, \mu'>$ with $<\partial_{P_\mathrm{m}}(\epsilon), \mu'> \mathcal{S} <\epsilon, \mu'>$.

- $<(d \sim [\neg P_\mathrm{m}^-]) \gg a, \nu> \overset{a,\mu}{\mapsto} <p', \mu'>$
  For this, according to the semantics of HyPA, we need the assumption that there exists ν' such that $(\nu, \nu') \models_d d$ and $(\nu', \mu) \models_d [\neg P_\mathrm{m}^-]$. From this we conclude that $\nu' = \mu$ and hence $\mu \models_\mathrm{m} \neg P_\mathrm{m}$. Finally, we find that $<\partial_{P_\mathrm{m}}(d \gg a), \nu> \overset{a,\mu}{\mapsto} <\partial_{P_\mathrm{m}}(\epsilon), \mu'>$
  with $<\partial_{P_\mathrm{m}}(\epsilon), \mu'> \mathcal{S} <\epsilon, \mu'>$.

C.4 $\partial_{P_\mathrm{m}}(d \gg c) \leftrightarrows d \gg (c \wedge (\neg P_\mathrm{m}))$

The axiom $\partial_{P_\mathrm{m}}(d \gg c) \leftrightarrows d \gg (c \wedge (\neg P_\mathrm{m}))$ is, for given $P_\mathrm{m} \in \mathcal{P}_\mathrm{m}$, $d \in D$ and $c \in C$, witnessed by the relation $\mathcal{S} = \{(<\partial_{P_\mathrm{m}}(d \gg c), \nu>, <d \gg (c \wedge (\neg P_\mathrm{m})), \nu>) \mid \nu \in \mathit{Val}\} \cup \{(<\partial_{P_\mathrm{m}}(c), \nu>, <c \wedge (\neg P_\mathrm{m}), \nu>) \mid \nu \in \mathit{Val}\}$. That this is indeed a witnessing relation is obvious, and also that it is robust is straightforward to verify. That it is a bisimulation relation follows from the following cases.

Assume that $<p, \nu> \mathcal{S} <q, \nu>$. From the definition, it follows that either $p = \partial_{P_\mathrm{m}}(d \gg c)$ and $q = d \gg (c \wedge (\neg P_\mathrm{m}))$, or $p = \partial_{P_\mathrm{m}}(c)$ and $q = c \wedge (\neg P_\mathrm{m})$. Since the proofs for these cases are similar, we will only discuss the first.

Because flow clauses cannot terminate, nor perform actions transitions, $\partial_{P_\mathrm{m}}(d \gg c)$ and $d \gg (c \wedge (\neg P_\mathrm{m}))$ cannot terminate, nor perform action transitions. We, therefore, only need to study the following cases.

- $<\partial_{P_\mathrm{m}}(d \gg c), \nu> \overset{\sigma}{\rightsquigarrow} <p', \mu'>$
  From the semantics of predicate encapsulation, we obtain that for every $\tau \in \mathit{dom}(\sigma)$ we have $\sigma(\tau) \models_\mathrm{m} \neg P_\mathrm{m}$, and hence $\sigma \models_\mathrm{f} \neg P_\mathrm{m}$. Furthermore, for the above transition, we need the assumption that there exists $\nu' \in \mathit{Val}$ with $(\nu, \nu') \models_d d$ and $(\nu', \sigma) \models_c c$. From all this we conclude that $(\nu', \sigma) \models_d [\neg P_\mathrm{m}]$, and hence $<d \gg (c \wedge (\neg P_\mathrm{m})), \nu> \overset{\sigma}{\rightsquigarrow} <(c \wedge (\neg P_\mathrm{m})), \mu'>$. Finally, by definition of $\mathcal{S}$, we have $<\partial_{P_\mathrm{m}}(c), \mu'> \mathcal{S} <(c \wedge (\neg P_\mathrm{m})), \mu'>$.
- $<d \gg (c \wedge (\neg P_\mathrm{m})), \nu> \overset{\sigma}{\rightsquigarrow} <p', \mu'>$
  For this, we need the assumption that there exists $\nu' \in \mathit{Val}$ with $(\nu, \nu') \models_d d$ and $(\nu', \sigma) \models_c c$ and $(\nu', \sigma) \models_c (\neg P_\mathrm{m})$. Also, we need that $\mu' = \sigma(t)$, with $\mathit{dom}(\sigma) = [0, t]$. From all this, we may directly conclude that $<\partial_{P_\mathrm{m}}(d \gg c), \nu> \overset{\sigma}{\rightsquigarrow} <\partial_{P_\mathrm{m}}(c), \mu'>$.
  Finally, by definition of $\mathcal{S}$, we have $<\partial_{P_\mathrm{m}}(c), \mu'> \mathcal{S} <(c \wedge (\neg P_\mathrm{m})), \mu'>$.

C.5 $\partial_{P_\mathrm{m}}(x \oplus y) \leftrightarrows \partial_{P_\mathrm{m}}(x) \oplus \partial_{P_\mathrm{m}}(y)$

Similar to the proof of axiom $\partial_H(x \oplus y) \leftrightarrows \partial_H(x) \oplus \partial_H(y)$ in [9].

C.6 $\partial_{P_\mathrm{m}}(x \odot y) \leftrightarrows \partial_{P_\mathrm{m}}(x) \odot \partial_{P_\mathrm{m}}(y)$

Similar to the proof of axiom $\partial_H(x \odot y) \leftrightarrows \partial_H(x) \odot \partial_H(y)$ in [9].

C.7 $\partial_{P_\mathrm{m}}(x \rhd y) \leftrightarrows \partial_{P_\mathrm{m}}(x) \rhd \partial_{P_\mathrm{m}}(y)$

Similar to the proof of axiom $\partial_H(x \rhd y) \leftrightarrows \partial_H(x) \rhd \partial_H(y)$ in [9].

D Axioms for (initially stateless) bisimilarity

D.1 $d \gg a \odot x \leftrightarroweq d \gg a \odot d^! \gg x$

The axiom $d \gg a \odot x \leftrightarroweq d \gg a \odot d^! \gg x$, for a given re-initialization clause d and action $a \in \mathcal{A}$, is witnessed by the bisimulation relation $\mathcal{S} = \{(<d \gg a \odot x, \nu>, <d \gg a \odot d^! \gg x>) \mid x \in \mathcal{T}(\mathcal{V}_\mathrm{r}),\ \nu \in \mathit{Val}\} \cup \{(<\epsilon \odot x, \nu>, <\epsilon \odot d^! \gg x, \nu>) \mid x \in \mathcal{T}(\mathcal{V}_\mathrm{r}),\ \nu \in \mathit{Val},\ (\nu, \nu) \models_d d^!\} \cup \{(<x, \nu>, <x, \nu>) \mid x \in \mathcal{T}(\mathcal{V}_\mathrm{r}),\ \nu \in \mathit{Val}\}$. Obviously, this is a witnessing relation. We will now prove that it is also a bisimulation relation.

Assume that $<p, \nu>\mathcal{S}<q, \nu'>$. Obviously, $\nu = \nu'$. Furthermore, the case where $p = q$, and the case where $p = \epsilon \odot x$ and $q = \epsilon \odot d^! \gg x$ with $(\nu, \nu) \models_d d^!$, are straightforward. We focus on the case where $p = d \gg a \odot x$ and $q = d \gg a \odot d^! \gg x$.

- $<d \gg a \odot x, \nu> \checkmark$
  This cannot be the case, since (using the semantics of sequential composition and re-initialization) we would need the assumption $<a, \nu> \checkmark$, which is false.
- $<d \gg a \odot d^! \gg x, \nu> \checkmark$
  Similarly, this can also not be the case.
- $<d \gg a \odot x, \nu> \overset{a,\mu}{\mapsto} <x', \mu'>$
  From the semantics of sequential composition and re-initialization it follows that $x' = \epsilon \odot x$ and that $(\nu, \mu') \models_d d$. Hence, we have that $(\mu', \mu') \models_d d^!$, from which we conclude that $(\epsilon \odot x, \mu')\mathcal{S}(\epsilon \odot d^! \gg x, \mu')$. Furthermore, from the semantics it also follows that $<d \gg a \odot d^! \gg x, \nu> \overset{a,\mu}{\mapsto} <\epsilon \odot d^! \gg x, \mu'>$.
- $<d \gg a \odot d^! \gg x, \nu'> \overset{a,\mu}{\mapsto} <x', \mu'>$
  Similar to the previous case.
- $<d \gg a \odot x, \nu> \overset{\sigma}{\leadsto} <x', \mu'>$
  This cannot be the case, since it would require a flow transition from a.
- $<d \gg a \odot d^! \gg x, \nu'> \overset{\sigma}{\leadsto} <x', \mu'>$
  Similarly, this cannot be the case.

D.2 $d \gg c \rhd x \leftrightarroweq d \gg c \rhd (d \sim D(c))^! \gg x$

The axiom $d \gg c \rhd x \leftrightarroweq d \gg c \rhd (d \sim D(c))^! \gg x$, for a given re-initialization clause $d \in D$ and flow clause $c \in C$, is witnessed by the bisimulation relation $\mathcal{S} = \{(<d \gg c \rhd x, \nu>, <d \gg c \rhd (d \sim D(c))^! \gg x, \nu>) \mid x \in \mathcal{T}(\mathcal{V}_\mathrm{r}),\ \nu \in \mathit{Val}\} \cup \{(<c \blacktriangleright x, \nu>, <c \blacktriangleright (d \sim D(c))^! \gg x, \nu>) \mid x \in \mathcal{T}(\mathcal{V}_\mathrm{r}),\ \nu \in \mathit{Val},\ (\nu, \nu) \models_d (d \sim D(c))^!\} \cup \{(<x, \nu>, <x, \nu>) \mid x \in \mathcal{T}(\mathcal{V}_\mathrm{r}),\ \nu \in \mathit{Val}\}$. Obviously, this is a witnessing relation. We will now prove that it is also a bisimulation relation.

In this proof, we need the assumption that the solutions of flow clauses are closed under concatenation. Concretely, this means that if $(\nu, \sigma) \models_c c$ and $(\sigma(t), \sigma') \models_c c$, with $\mathit{dom}(\sigma) = [0, t]$ then $(\nu, \sigma \circ \sigma') \models_c c$, with $\circ$ the concatenation operator on flows. As a result, we find for the associated re-initializations that $D(c) \sim D(c) = D(c)$. In the proof, we also use the axiom on reachability that $(d \sim d')^! = (d^! \sim d')^!$.

Assume that $<p, \nu>\mathcal{S}<q, \nu'>$. Obviously, $\nu = \nu'$. Furthermore, the case where $p = q$ is trivial. We focus firstly on the case where $p = d \gg c \rhd x$ and $q = d \gg c \rhd (d \sim D(c))^! \gg x$.

- $< d \gg c \,\rhd\, x, \nu > \checkmark$
  This cannot be the case, since (using the semantics of left-disrupt and re-initialization) we would need the assumption $< c, \nu > \checkmark$, which is false.

- $< d \gg c \,\rhd\, (d \sim D(c))^! \gg x, \nu > \checkmark$
  Similarly, this can also not be the case.

- $< d \gg c \,\rhd\, x, \nu > \overset{a,\mu}{\mapsto} < x', \mu' >$
  This cannot be the case, since it would require an action transition from c.

- $< d \gg c \,\rhd\, (d \sim D(c))^! \gg x, \nu' > \overset{a,\mu}{\mapsto} < x', \mu' >$
  Similarly, this cannot be the case.

- $< d \gg c \,\rhd\, x, \nu > \overset{\sigma}{\rightsquigarrow} < x', \mu' >$
  From the semantics of the left-disrupt and re-initialization it follows that $x' = c \blacktriangleright x$ and that there exists μ such that $(\nu, \mu) \models_d d$ and $(\mu, \sigma) \models_c c$. Hence, we have that $(\nu, \sigma(t')) \models_d (d \sim D(c))^!$ for every $t' \in [0, t] = dom(\sigma)$. In particular, we have $(\nu, \mu') \models_d (d \sim D(c))^!$, because $\mu' = \sigma(t)$. This leads to the conclusion that $< c \blacktriangleright x, \mu' > \mathcal{S} < c \blacktriangleright (d \sim D(c))^! \gg x, \mu' >$. Furthermore, from the semantics it also follows that $< d \gg c \,\rhd\, (d \sim D(c))^! \gg x, \nu > \overset{\sigma}{\rightsquigarrow} < c \blacktriangleright (d \sim D(c))^! \gg x, \mu' >$.

- $< d \gg c \,\rhd\, (d \sim D(c))^! \gg x, \nu' > \overset{\sigma}{\rightsquigarrow} < x', \mu' >$
  Similar to the previous case.

For the case where $p = c \blacktriangleright x$ and $q = c \blacktriangleright (d \sim D(c))^! \gg x$, and $(\nu, \nu) \models_d (d \sim D(c))^!$ we find:

- $< c \blacktriangleright x, \nu > \checkmark$
  For which, according to the semantics, we need the assumption that $< x, \nu > \checkmark$. From the fact that $(\nu, \nu) \models_d (d \sim D(c)^!)$ we then derive that also $< (d \sim D(c))^! \gg x, \nu > \checkmark$ and hence $< c \blacktriangleright (d \sim D(c))^! \gg x, \nu > \checkmark$.

- $< c \blacktriangleright (d \sim D(c)^!) \gg x, \nu > \checkmark$
  For which, according to the semantics, we need the assumption that $< x, \nu > \checkmark$, and hence $< c \blacktriangleright x, \nu > \checkmark$.

- $< c \blacktriangleright x, \nu > \overset{a,\mu}{\mapsto} < x', \mu' >$
  For which, according to the semantics, we need the assumption that $< x, \nu > \overset{a,\mu}{\mapsto} < x', \mu >$, since c cannot perform any action transitions. From the fact that $(\nu, \nu) \models_d (d \sim D(c))^!$ we then derive that also $< (d \sim D(c))^! \gg x, \nu > \overset{a,\mu}{\mapsto} < x', \mu >$ and ultimately $< c \blacktriangleright (d \sim D(c))^! \gg x, \nu > \overset{a,\mu}{\mapsto} < x', \mu >$. By definition, we have $< x', \mu > \mathcal{S} < x', \mu >$.

- $< c \blacktriangleright (d \sim D(c))^! \gg x, \nu > \overset{a,\mu}{\mapsto} < x', \mu >$
  For which, according to the semantics, we need the assumption that $< x, \nu > \overset{a,\mu}{\mapsto} < x', \mu >$ and $(\nu, \nu) \models_d (d \sim D(c))^!$. Obviously, we may conclude that $< c \blacktriangleright x, \nu > \overset{a,\mu}{\mapsto} < x', \mu >$ and we find $< x', \mu > \mathcal{S} < x', \mu >$.

- $< c \blacktriangleright x, \nu > \overset{\sigma}{\rightsquigarrow} < x', \mu' >$
  According to the semantics, we can distinguish two possible assumptions.

  - $< x, \nu > \overset{\sigma}{\rightsquigarrow} < x', \mu' >$
    From which we conclude, using the assumption $(\nu, \nu) \models_d (d \sim D(c))^!$, that $< c \blacktriangleright (d \sim D(c))^! \gg x, \nu > \overset{\sigma}{\rightsquigarrow} < x', \mu' >$ and we find $< x', \mu > \mathcal{S} < x', \mu >$.

- $< c, \nu > \overset{\sigma}{\rightsquigarrow} < c, \mu' >$ and $x' = c \blacktriangleright x$
  From which we conclude that $(\nu, \sigma) \models_c c$. Hence, using $\mu' = \sigma(t)$, with $dom(\sigma) = [0, t]$, we find $(\nu, \mu') \models_d (d \sim D(c))^! \sim D(c)$. Using the assumption that flow clauses are closed under concatenation of flows, we find that $((d \sim D(c))^! \sim D(c))^! = (d \sim D(c) \sim D(c))^! = (d \sim D(c))^!$, from which we conclude that $(\mu', \mu') \models_d (d \sim D(c))^!$ and hence $(c \blacktriangleright x, \mu')\mathcal{S}(c \blacktriangleright (d \sim D(c))^! \gg x, \mu')$. Finally, from the semantics, and the assumption that $(\nu, \sigma) \models_c c$, we may conclude that $< c \blacktriangleright (d \sim D(c))^! \gg x, \nu > \overset{\sigma}{\rightsquigarrow} < c \blacktriangleright (d \sim D(c))^! \gg x, \mu' >$.

- $< c \blacktriangleright (d \sim D(c))^! \gg x, \nu > \overset{\sigma}{\rightsquigarrow} < x', \mu >$
  According to the semantics, we can distinguish two possible assumptions.
  - $< (d \sim D(c))^! \gg x, \nu > \overset{\sigma}{\rightsquigarrow} < x', \mu' >$
    From which we conclude directly $< x, \nu > \overset{\sigma}{\rightsquigarrow} < x', \mu' >$ and hence $< c \blacktriangleright x, \nu > \overset{\sigma}{\rightsquigarrow} < x', \mu >$, with $< x', \mu > \mathcal{S} < x', \mu >$.
  - $< c, \nu > \overset{\sigma}{\rightsquigarrow} < c, \mu' >$ and $x' = c \blacktriangleright (d \sim D(c))^! \gg x$
    From which we conclude directly that $< c \blacktriangleright x, \nu > \overset{\sigma}{\rightsquigarrow} < c \blacktriangleright x, \mu' >$. Furthermore, using the concatenation closure of flow clauses, as before, we conclude that $(\mu', \mu') \models_d ((d \sim D(c))^! \sim D(c))^! = (d \sim D(c) \sim D(c))^! = (d \sim D(c))^!$, from which we ultimately conclude $< c \blacktriangleright x, \mu' > \mathcal{S} < c \blacktriangleright (d \sim D(c))^! \gg x, \mu' >$.

E RSP for (initially stateless) bisimilarity

When reasoning about recursion, it is often useful to have a principle that claims that a solution of a certain recursive specification exists, and is unique modulo the notion of equivalence that we are interested in. That a solution exists modulo (initially stateless) bisimilarity, follows directly from the operational semantics of HyPA, but it is not immediately clear that that particular solution is the only process term satisfying the recursive equations. In [9], existence and uniqueness of solutions of so-called guarded recursive specifications, was shown for the notion of robust bisimilarity. In this appendix, we adapt the proof of [9] for initially stateless bisimilarity. The changes we need to make are only minor, so that we can restrict ourselves to the outline of the proof here, and refer to [9] for most of the details.

We start out by formalizing what a solution of a recursive specification is.

Definition 9 (Solution) *Let E be a recursive specification. An interpretation $S \in \mathcal{V}_{\mathrm{r}} \to \mathcal{T}(\mathcal{V}_{\mathrm{r}})$ of recursion variables as process terms, is a* solution *of E (denoted $S \models E$) if for every recursive definition $X \Leftrightarrow p \in E$ we have $S(X) \Leftrightarrow S(p)$, where $S(p)$ denotes the process term induced by application of S to the variables of p. In particular, $S(X)$ is called a solution of $X \Leftrightarrow p \in E$.*

The *recursive specification principle* RSP, which is quite standard in process algebra [5], states that so called guarded recursive specifications have at most one solution. For HyPA, guardedness of a recursive specification is defined as follows.

Definition 10 (Guardedness) *A process term p is* guarded *if all occurrences of recursion variables in p, are in the scope of an action prefix $a \odot \_$ or a flow prefix $c \vartriangleright \_$. A recursive specification E is* guarded *if for each recursive definition $X \Leftrightarrow p \in E$, p can be rewritten into a guarded process term using the axiomatization of HyPA.*

This leads to the principle given in table 9.

Table 9: Recursive Specification Principle

$$\frac{S \models E,\ S' \models E,\ E \text{ guarded}}{S(X) \leftrightarrow S'(X)}\ X \in \mathcal{V}_{\mathrm{r}}$$

The proof of this, usually goes via another principle, called the *approximation induction principle* AIP [5], which makes use of a family of projection operators π_n. AIP states that if every finite projection of two processes is bisimilar, then the two processes are bisimilar. For the kind of semantical model we use, AIP is restricted in the sense that one of the compared processes should have bounded non-determinism. This is usually referred to as the *restricted approximation induction principle* AIP$^-$. In this section, we introduce the family of projection operators, and formalize the notion of bounded non-determinism. Then we formally pose the approximation induction principle. After that, we show the existence of a bounded solution for guarded recursive specifications, and prove a projection property for guarded process terms. Finally, this allows us to prove soundness of RSP using AIP$^-$.

Projection has the operational semantics stated in table 10.

Table 10: Operational semantics of projection

$$\frac{<p,\nu>\checkmark}{<\pi_n(p),\nu>\checkmark}, \qquad \frac{<p,\nu> \xrightarrow{l} <p',\nu'>}{<\pi_{n+1}(p),\nu> \xrightarrow{l} <\pi_n(p'),\nu'>}.$$

Based on the formats introduced in [22], we claim that initially stateless bisimilarity is a congruence for projection.

Bounded non-determinism $B(p)$ is defined as follows.

Definition 11 (Bounded non-determinism) *Bounded non-determinism is recursively defined as:*

- *Every state has bounded non-determinism in* 0 *steps.*
- *A state* $<p,\nu>$ *has bounded non-determinism in* $n+1$ *steps, if for every* l *the set* $R = \{<p',\nu'> \mid <p,\nu> \xrightarrow{l} <p',\nu'>\}$ *is finite, and all elements* $<p',\nu'> \in R$ *have bounded non-determinism in* n *steps themselves.*
- *A state* $<p,\nu>$ *has bounded non-determinism (denoted* $B(<p,\nu>)$*) if it has bounded non-determinism for any arbitrary number of steps.*
- *A process term* p *has bounded non-determinism (denoted* $B(p)$*) if for every valuation* $\nu \in Val$ *we find that* $<p,\nu>$ *has bounded non-determinism.*

These definitions allow us to state the restricted approximation induction principle AIP$^-$, given in table 11. Next, we prove that this principle is sound.

Table 11: Restricted approximation induction principle

$$\frac{\forall_n \pi_n(p) \leftrightarrows \pi_n(q) \ \wedge \ B(q)}{p \leftrightarrows q} \quad \mathrm{AIP}^-$$

Theorem 13 *AIP$^-$ is sound for the semantics of HyPA, modulo initially stateless bisimilarity.*

Proof To prove this principle sound, suppose that $\mathcal{R}$ is the union of all bisimulation relations. In particular, it contains the bisimulation relations witnessing $\pi_n(p) \leftrightarrows \pi_n(q)$. Note, that $\mathcal{R}$ is an equivalence relation on states. We now construct the following relation

$$\mathcal{S} = \{(<x,\nu>, <y,\mu>) \mid \forall_n <\pi_n(x),\nu>\mathcal{R}<\pi_n(y),\mu>,\ B(<y,\nu>)\}.$$

The proof that this is indeed a witnessing bisimulation relation, follows the same lines as the proof for robust bisimulation in [9]. ⊠

Before we can use AIP$^-$ to prove RSP, we need to study bounded non-determinism and projections of guarded recursive specifications in more detail. We need to show existence of a bounded non-deterministic solution for each guarded recursive specification, and we need an axiomatization for projection with respect to guarded process terms.

Theorem 14 (Bounded non-determinism) *Each guarded recursive specification E has a bounded non-deterministic solution.*

Proof In [9], this was shown for robust bisimilarity. The same solution is also a solution for the guarded recursive specification E under (initially stateless) bisimilarity. Hence, the theorem also holds for this weaker equivalence. ⊠

Theorem 15 (Guarded projection push) *Define the interpretation $\Pi_n \in \mathcal{V}_\mathrm{r} \rightarrow \mathcal{T}(\mathcal{V}_\mathrm{r})$ as before, and let S be an arbitrary interpretation of recursion variables. Then, we find the following axioms for guarded process terms p:*

$$\pi_0(p) \leftrightarrows \pi_0(S(p)), \qquad \pi_{n+1}(p) \leftrightarrows \pi_{n+1}(\Pi_n(p)).$$

Proof See [9]. ⊠

Corollary 1 *Define the interpretation $\Pi_n \in \mathcal{V}_\mathrm{r} \rightarrow \mathcal{T}(\mathcal{V}_\mathrm{r})$ as before, and let S be an arbitrary interpretation of recursion variables. Then the axioms $\pi_0(p) \leftrightarrows \pi_0(S(p))$, and $\pi_{n+1}(p) \leftrightarrows \pi_{n+1}(\Pi_n(p))$ are sound.*

Now, using corollary 1, the theorem on bounded non-determinism of guarded recursive specifications, and AIP$^-$, it is easy to derive soundness of RSP.

Theorem 16 *The recursive specification principle is sound.*

Proof For convenience assume that $X \leftrightarrow p \in E$ implies that p is already rewritten into a guarded process term. Using the theorem on bounded non-determinism, we know that there exists a solution S of E that has bounded non-determinism, i.e. $B(S(X))$ for every $X \in \mathcal{V}_\mathrm{r}$. Suppose that S' is an arbitrary other solution for E. We will show by induction on n that for every $X \in \mathcal{V}_\mathrm{r}$ we have $\pi_n(S(X)) \leftrightarrow \pi_n(S'(X))$. From that we then may conclude $S(X) \leftrightarrow S'(X)$ using AIP$^-$. Note, that if we have two arbitrary solutions of E, that we may conclude them equal by showing that both are equal to S.

The base case, where $n = 0$, is derived using congruence (derivation rule (4)) and the first part of corollary 1:

$$\pi_0(S(X)) \leftrightarrow \pi_0(S(p)) \leftrightarrow \pi_0(p) \leftrightarrow \pi_0(S'(p)) \leftrightarrow \pi_0(S'(X)).$$

Using the second part of corollary 1, and the induction hypothesis that $\pi_n(S(X)) \leftrightarrow \pi_n(S'(X))$ we find firstly, using congruence again, that $S(\Pi_n(p)) \leftrightarrow S'(\Pi_n(p))$, and using this we derive:

$$\begin{array}{rcl}
\pi_{n+1}(S(X)) & \leftrightarrow & \pi_{n+1}(S(p)) \leftrightarrow S(\pi_{n+1}(p)) \leftrightarrow S(\pi_{n+1}(\Pi_n(p))) \\
& \leftrightarrow & \pi_{n+1}(S(\Pi_n(p))) \leftrightarrow \pi_{n+1}(S'(\Pi_n(p))) \leftrightarrow S'(\pi_{n+1}(\Pi_n(p))) \\
& \leftrightarrow & S'(\pi_{n+1}(p)) \leftrightarrow \pi_{n+1}(S'(p)) \leftrightarrow \pi_{n+1}(S'(X)).
\end{array}$$

⊠

F Linearization of Fischers protocol

In this section, we give the linearization of the process $S\ :\ \mathrm{init} \gg (\mathrm{Idle}_1 \parallel \mathrm{Idle}_2 \parallel \mathrm{C})$. We find that $S \rightleftarrows \mathrm{init} \gg X_{i,i}$ with:

$$\begin{array}{lcl}
X_{i,i} & : & \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ turn & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_1 & loc_1 = \mathrm{Idle} \\ & loc_2 = \mathrm{Idle} \end{array} \right) \rhd X_{i,i} \ \oplus \\
& & \left[\begin{array}{l|l} x_1 & turn^- = 0 \\ loc_1 & x_1^+ = 0 \\ & loc_1^+ = \mathrm{Request} \end{array} \right] \gg \iota \odot X_{r,i} \oplus \left[\begin{array}{l|l} x_2 & turn^- = 0 \\ loc_2 & x_2^+ = 0 \\ & loc_2^+ = \mathrm{Request} \end{array} \right] \gg \iota \odot X_{i,r} \\
X_{i,r} & : & \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ turn & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_1 & loc_1 = \mathrm{Idle} \\ loc_2 & loc_2 = \mathrm{Request} \\ x_2 & \dot{x}_2 \geq 0 \\ & x_2 \leq D \end{array} \right) \rhd X_{i,r} \\
& \oplus & \left[\begin{array}{l|l} x_1 & turn^- = 0 \\ loc_1 & x_i^+ = 0 \\ & loc_1^+ = \mathrm{Request} \end{array} \right] \gg \iota \odot X_{r,r} \oplus \left[\begin{array}{l|l} x_2 & x_2^+ = 0 \\ loc_2 & turn^+ = 2 \\ turn & loc_2^+ = \mathrm{Check} \end{array} \right] \gg \iota \odot X_{i,c}
\end{array}$$

$$X_{i,c} \quad : \quad \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ turn & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_1 & loc_1 = \text{Idle} \\ loc_2 & loc_2 = \text{Check} \\ x_2 & \dot{x}_2 \geq 0 \end{array}\right) \rhd X_{i,c}$$

$$\oplus \left[\begin{array}{l|l} & turn^- = 0 \\ x_1 & x_1^+ = 0 \\ loc_1 & loc_1^+ = \text{Request} \end{array}\right] \gg \iota \odot X_{r,c} \oplus \left[\begin{array}{l|l} & x_2^- \geq d \\ loc_2 & turn^- \neq 2 \\ & loc_2^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{i,i}$$

$$\oplus \left[\begin{array}{l|l} & x_2^- \geq d \\ loc_2 & turn^- = 2 \\ & loc_2^+ = \text{Access} \end{array}\right] \gg \iota \odot X_{i,a}$$

$$X_{i,a} \quad : \quad \left(\begin{array}{l|l} turn & \dot{x}_1 \leq e\ \dot{x}_2 \\ loc_1 & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_2 & loc_1 = \text{Idle} \\ & loc_2 = \text{Access} \end{array}\right) \rhd X_{i,a}$$

$$\oplus \left[\begin{array}{l|l} & turn^- = 0 \\ x_1 & x_i^+ = 0 \\ loc_1 & loc_1^+ = \text{Request} \end{array}\right] \gg \iota \odot X_{r,a} \oplus \left[\begin{array}{l|l} turn & turn^+ = 0 \\ loc_2 & loc_2^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{i,i}$$

$$X_{r,i} \quad : \quad \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ turn & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_1 & loc_1 = \text{Request} \\ loc_2 & loc_2 = \text{Idle} \\ x_1 & \dot{x}_1 \geq 0 \\ & x_1 \leq D \end{array}\right) \rhd X_{r,i}$$

$$\oplus \left[\begin{array}{l|l} x_1 & x_1^+ = 0 \\ loc_1 & turn^+ = 1 \\ turn & loc_1^+ = \text{Check} \end{array}\right] \gg \iota \odot X_{c,i} \oplus \left[\begin{array}{l|l} & turn^- = 0 \\ x_2 & x_2^+ = 0 \\ loc_2 & loc_2^+ = \text{Request} \end{array}\right] \gg \iota \odot X_{r,r}$$

$$X_{r,r} \quad : \quad \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ turn & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_1 & loc_1 = \text{Request} \\ loc_2 & loc_2 = \text{Request} \\ x_1 & \dot{x}_1 \geq 0 \\ x_2 & x_1 \leq D \\ & \dot{x}_2 \geq 0 \\ & x_2 \leq D \end{array}\right) \rhd X_{r,r}$$

$$\oplus \left[\begin{array}{l|l} x_1 & x_1^+ = 0 \\ loc_1 & turn^+ = 1 \\ turn & loc_1^+ = \text{Check} \end{array}\right] \gg \iota \odot X_{r,c} \oplus \left[\begin{array}{l|l} x_2 & x_2^+ = 0 \\ loc_2 & turn^+ = 2 \\ turn & loc_2^+ = \text{Check} \end{array}\right] \gg \iota \odot X_{r,c}$$

$$X_{r,c} \quad : \quad \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ turn & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_1 & loc_1 = \text{Request} \\ loc_2 & loc_2 = \text{Check} \\ x_1 & \dot{x}_1 \geq 0 \\ x_2 & x_1 \leq D \\ & \dot{x}_2 \geq 0 \end{array}\right) \rhd X_{r,c}$$

$$\oplus \left[\begin{array}{l|l} x_1 & x_1^+ = 0 \\ loc_1 & turn^+ = 1 \\ turn & loc_1^+ = \text{Check} \end{array}\right] \gg \iota \odot X_{c,c} \oplus \left[\begin{array}{l|l} & x_2^- \geq d \\ loc_2 & turn^- \neq 2 \\ & loc_2^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{r,i}$$

$$\oplus \left[\begin{array}{l|l} & x_2^- \geq d \\ loc_2 & turn^- = 2 \\ & loc_2^+ = \text{Access} \end{array}\right] \gg \iota \odot X_{r,a}$$

$$X_{r,a} \quad : \quad \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ \mathit{turn} & \dot{x}_2 \leq e\ \dot{x}_1 \\ \mathit{loc}_1 & \mathit{loc}_1 = \text{Request} \\ \mathit{loc}_2 & \mathit{loc}_2 = \text{Access} \\ x_1 & \dot{x}_1 \geq 0 \\ & x_1 \leq D \end{array}\right) \rhd X_{r,a}$$

$$\oplus \left[\begin{array}{l|l} x_1 & x_1^+ = 0 \\ \mathit{loc}_1 & \mathit{turn}^+ = 1 \\ \mathit{turn} & \mathit{loc}_1^+ = \text{Check} \end{array}\right] \gg \iota \odot X_{c,a} \oplus \left[\begin{array}{l|l} \mathit{turn} & \mathit{turn}^+ = 0 \\ \mathit{loc}_2 & \mathit{loc}_2^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{r,i}$$

$$X_{c,i} \quad : \quad \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ \mathit{turn} & \dot{x}_2 \leq e\ \dot{x}_1 \\ \mathit{loc}_1 & \mathit{loc}_1 = \text{Check} \\ \mathit{loc}_2 & \mathit{loc}_2 = \text{Idle} \\ x_1 & \dot{x}_1 \geq 0 \end{array}\right) \rhd X_{c,i}$$

$$\oplus \left[\begin{array}{l|l} & x_1^- \geq d \\ \mathit{loc}_1 & \mathit{turn}^- \neq 1 \\ & \mathit{loc}_1^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{i,i} \oplus \left[\begin{array}{l|l} & x_1^- \geq d \\ \mathit{loc}_1 & \mathit{turn}^- = 1 \\ & \mathit{loc}_1^+ = \text{Access} \end{array}\right] \gg \iota \odot X_{a,i}$$

$$\oplus \left[\begin{array}{l|l} & \mathit{turn}^- = 0 \\ x_2 & x_2^+ = 0 \\ \mathit{loc}_2 & \mathit{loc}_2^+ = \text{Request} \end{array}\right] \gg \iota \odot X_{c,r}$$

$$X_{c,r} \quad : \quad \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ \mathit{turn} & \dot{x}_2 \leq e\ \dot{x}_1 \\ \mathit{loc}_1 & \mathit{loc}_1 = \text{Check} \\ \mathit{loc}_2 & \mathit{loc}_2 = \text{Request} \\ x_1 & \dot{x}_1 \geq 0 \\ x_2 & \dot{x}_2 \geq 0 \\ & x_2 \leq D \end{array}\right) \rhd X_{c,r}$$

$$\oplus \left[\begin{array}{l|l} & x_1^- \geq d \\ \mathit{loc}_1 & \mathit{turn}^- \neq 1 \\ & \mathit{loc}_1^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{i,r} \oplus \left[\begin{array}{l|l} & x_1^- \geq d \\ \mathit{loc}_1 & \mathit{turn}^- = 1 \\ & \mathit{loc}_1^+ = \text{Access} \end{array}\right] \gg \iota \odot X_{a,r}$$

$$\oplus \left[\begin{array}{l|l} x_2 & x_2^+ = 0 \\ \mathit{loc}_2 & \mathit{turn}^+ = 2 \\ \mathit{turn} & \mathit{loc}_2^+ = \text{Check} \end{array}\right] \gg \iota \odot X_{c,c}$$

$$X_{c,c} \quad : \quad \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ \mathit{turn} & \dot{x}_2 \leq e\ \dot{x}_1 \\ \mathit{loc}_1 & \mathit{loc}_1 = \text{Check} \\ \mathit{loc}_2 & \mathit{loc}_2 = \text{Check} \\ x_1 & \dot{x}_1 \geq 0 \\ x_2 & \dot{x}_2 \geq 0 \end{array}\right) \rhd X_{c,c}$$

$$\oplus \left[\begin{array}{l|l} & x_1^- \geq d \\ \mathit{loc}_1 & \mathit{turn}^- \neq 1 \\ & \mathit{loc}_1^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{i,c} \oplus \left[\begin{array}{l|l} & x_1^- \geq d \\ \mathit{loc}_1 & \mathit{turn}^- = 1 \\ & \mathit{loc}_1^+ = \text{Access} \end{array}\right] \gg \iota \odot X_{a,c}$$

$$\oplus \left[\begin{array}{l|l} & x_2^- \geq d \\ \mathit{loc}_2 & \mathit{turn}^- \neq 2 \\ & \mathit{loc}_2^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{c,i} \oplus \left[\begin{array}{l|l} & x_2^- \geq d \\ \mathit{loc}_2 & \mathit{turn}^- = 2 \\ & \mathit{loc}_2^+ = \text{Access} \end{array}\right] \gg \iota \odot X_{c,a}$$

$$X_{c,a} \quad : \quad \left(\begin{array}{l|l} & \dot{x}_1 \leq e\ \dot{x}_2 \\ \mathit{turn} & \dot{x}_2 \leq e\ \dot{x}_1 \\ \mathit{loc}_1 & \mathit{loc}_1 = \text{Check} \\ \mathit{loc}_2 & \mathit{loc}_2 = \text{Access} \\ x_1 & \dot{x}_1 \geq 0 \end{array}\right) \rhd X_{c,a}$$

$$
\begin{aligned}
&\oplus && \left[\begin{array}{c|l} loc_1 & x_1^- \geq d \\ & turn^- \neq 1 \\ & loc_1^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{i,a} \oplus \left[\begin{array}{c|l} loc_1 & x_1^- \geq d \\ & turn^- = 1 \\ & loc_1^+ = \text{Access} \end{array}\right] \gg \iota \odot X_{a,a} \\
&\oplus && \left[\begin{array}{c|l} turn & turn^+ = 0 \\ loc_2 & loc_2^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{c,i} \\
X_{a,i} &: && \left(\begin{array}{c|l} turn & \dot{x}_1 \leq e\ \dot{x}_2 \\ loc_1 & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_2 & loc_1 = \text{Access} \\ & loc_2 = \text{Idle} \end{array}\right) \rhd X_{a,i} \\
&\oplus && \left[\begin{array}{c|l} turn & turn^+ = 0 \\ loc_1 & loc_1^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{i,i} \oplus \left[\begin{array}{c|l} x_2 & turn^- = 0 \\ loc_2 & x_2^+ = 0 \\ & loc_2^+ = \text{Request} \end{array}\right] \gg \iota \odot X_{a,r} \\
X_{a,r} &: && \left(\begin{array}{c|l} turn & \dot{x}_1 \leq e\ \dot{x}_2 \\ loc_1 & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_2 & loc_1 = \text{Access} \\ x_2 & loc_2 = \text{Request} \\ & \dot{x}_2 \geq 0 \\ & x_2 \leq D \end{array}\right) \rhd X_{a,r} \\
&\oplus && \left[\begin{array}{c|l} turn & turn^+ = 0 \\ loc_1 & loc_1^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{i,r} \oplus \left[\begin{array}{c|l} x_2 & x_2^+ = 0 \\ loc_2 & turn^+ = 2 \\ turn & loc_2^+ = \text{Check} \end{array}\right] \gg \iota \odot X_{a,c} \\
X_{a,c} &: && \left(\begin{array}{c|l} turn & \dot{x}_1 \leq e\ \dot{x}_2 \\ loc_1 & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_2 & loc_1 = \text{Access} \\ x_2 & loc_2 = \text{Check} \\ & \dot{x}_2 \geq 0 \end{array}\right) \rhd X_{a,c} \\
&\oplus && \left[\begin{array}{c|l} turn & turn^+ = 0 \\ loc_1 & loc_1^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{i,c} \oplus \left[\begin{array}{c|l} loc_2 & x_2^- \geq d \\ & turn^- \neq 2 \\ & loc_2^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{a,i} \\
&\oplus && \left[\begin{array}{c|l} loc_2 & x_2^- \geq d \\ & turn^- = 2 \\ & loc_2^+ = \text{Access} \end{array}\right] \gg \iota \odot X_{a,a} \\
X_{a,a} &: && \left(\begin{array}{c|l} turn & \dot{x}_1 \leq e\ \dot{x}_2 \\ loc_1 & \dot{x}_2 \leq e\ \dot{x}_1 \\ loc_2 & loc_1 = \text{Access} \\ & loc_2 = \text{Access} \end{array}\right) \rhd X_{a,a} \\
&\oplus && \left[\begin{array}{c|l} turn & turn^+ = 0 \\ loc_1 & loc_1^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{i,a} \oplus \left[\begin{array}{c|l} turn & turn^+ = 0 \\ loc_1 & loc_1^+ = \text{Idle} \end{array}\right] \gg \iota \odot X_{a,i}
\end{aligned}
$$