Scientific Annals of Computer Science vol. 21, 2011, pp. 5–38

Desynchronizability of (Partial) Synchronous Closed Loop Systems

Harsh BEOHAR[1], Pieter CUIJPERS[1]

Abstract

The task of implementing supervisory controllers is non-trivial, even though there are different theories that allow automatic synthesis of such controllers in the form of automata. One of the reasons for this discord, is the asynchronous interaction between a plant and its controller in implementations, whereas the existing supervisory control theories assume synchronous interaction. As a consequence the implementation suffers from the so-called inexact synchronization problem. To address this issue, we find sufficient conditions under which a synchronous closed loop system is branching bisimilar to its corresponding asynchronous closed loop system. Furthermore, we extend this result to include interaction of plant or supervisor with its environment.

1 Introduction

Supervisory control theory provides an automatic way of synthesizing a supervisor that forces a process to comply to a given requirement. In supervisory control theory terminology:

- the model that is to be controlled is known as the *plant*,
- the model that specifies the requirement is known as the *specification*,
- the model that forces the plant to meet the specification by interacting with it is known as the *supervisor* or the *controller*.

[1]Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.
Email: {H.Beohar, P.J.L.Cuijpers}@tue.nl

- the interaction between a plant and its supervisor is known as *closed-loop behavior*.

The closed loop behavior in supervisory control theory is realized by the synchronous parallel composition operator. Informally, it allows a plant and a supervisor to synchronize on common events when other events can happen independently.

One of the main problems when implementing a supervisor that is synthesized by supervisory control theory, is inexact synchronization [8]. In practical industrial applications, the interaction between a plant and its supervisor is asynchronous, rather than synchronous, while supervisory control theory assumes a composition of plant and supervisor that ascertains *strict* interaction. By strict, we mean that either the plant or the supervisor has to wait for the other party while synchronizing.

Balemi was the first to consider the inexact synchronization problem in the context of supervisory control theory [4]. An *input-output* interpretation was adopted between a plant and its supervisor, and a special delay operator was introduced to model the delay in communication between the plant and its supervisor. One of the main results of [4] was the existence of a supervisor in the presence of delays. However, the authors in [21] argued that Balemi's construction was partially asynchronous in nature because the output actions from a plant can occur asynchronously, while the output actions from a supervisor must occur synchronously. In [21], this requirement was relaxed; necessary and sufficient conditions were provided for the existence of a controller under bounded delay between a plant and its supervisor.

In [9], a synchronous closed loop system (SCLS) was considered to be a specification, with the asynchronous closed loop system (ACLS) as its implementation. The main result of [9], were a number of sufficient conditions under which a SCLS and its corresponding ACLS are failure equivalent [11]. This work was motivated by the so-called "Foam-rubber wrapper" principle [17], borrowed from the field of delay insensitive circuits, which states that "a process is delay insensitive if it is equivalent to the same process connected with buffers". In [9], the foam-rubber wrapper principle was studied in the context of the parallel composition operator and it was shown that an extra condition is required to preserve this principle.

In this paper, we address the issue of inexact synchronization in the process algebra TCP [3], and we determine sufficient conditions under which a SCLS and the corresponding ACLS are branching bisimilar rather than failure equivalent [18]. Our motivation for using such a fine notion of

equivalence, is that we would like our ACLS to have the same modal-logical properties as the SCLS.

We observe the following differences between the related work mentioned above and our current work:

- In this paper, we solve a refinement problem instead of solving a control synthesis problem (*cf.* [4, 21]). Rather than computing a new supervisor under the presence of delays, we assume a given plant and supervisor, synthesized using supervisory control theory, and find sufficient conditions under which such a synthesized supervisor can control the same plant in an asynchronous environment.

- The key differences with the work of [9] are the type of buffers that is used, and the abstraction scheme (to hide the certain interactions in an ACLS). In this paper, we use a bag as the buffer and hide the interactions between a plant and the buffer (see Subsection 1.1 for details).

We extend our previous result [7] to a class of SCLS called partial synchronous closed loop system (PSCLS), whose alphabet contains external actions of either the plant or its supervisor that result in interaction with the external world (environment). This makes it possible to desynchronize a SCLS present in a decentralized or hierarchical architecture (see [20] for the two architectures) by decomposing the given SCLS into a number of PSCLSs and by verifying each of the individual PSCLSs. However, more research is required to ascertain a SCLS in decentralized or hierarchical architecture as desynchronizable when each of the PSCLSs (constituting a SCLS) are desynchronizable.

1.1 Architecture

This paper and the companion paper [7] are the result of a pre-study carried out in [6], where four construction methods were proposed to construct an ACLS from its corresponding SCLS. In this subsection, we introduce the architecture of an ACLS, discuss the reasons for using a bag as a buffer, and describe one of the abstraction schemes that will be used throughout this paper.

An ACLS can be constructed by introducing a buffer between a plant and its supervisor, thus decoupling the interaction between the two. In practice, the buffering mechanism is realized by the interactions of different

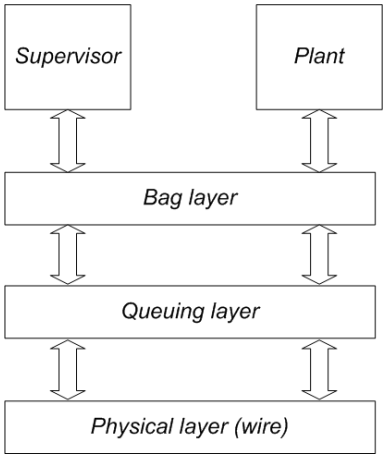


Figure 1: Asynchronous closed loop system in practice.

layers (also known as protocol stack) as shown in Figure 1. Various authors [9, 10, 13] have abstracted from the interaction of different layers by using data structures based on a particular level of abstraction. For example, to model delay insensitive (DI) circuits, which are at a lower level of abstraction (physical layer), wires are used as a buffering mechanism [12]. On the other hand, to model data flow networks, which are at a higher level of abstraction (in comparison to DI circuits), queues are used as a buffering mechanism [10]. Systems with unordered message buffers are considered to be a convenient abstraction of systems with lossy FIFO-message buffers (see [16] and the references therein). Therefore, we use a bag as the buffering mechanism. In practice, such buffers can for example be implemented using the User Datagram Protocol (UDP) [15], which allows an unordered delivery of messages to a receiver.

It is obvious that when introducing the bag as a buffer, the ACLS contains interactions that are not present in the SCLS. In order to be able to compare these two closed loop systems, it is therefore necessary to hide some interactions using a suitable abstraction scheme. We introduce bags between the plant and the supervisor, and distinguish the following alternatives for hiding part of the interaction:

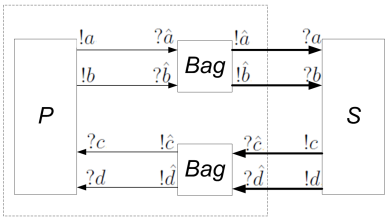
M1. we can hide the interaction between the plant and the bag (see Figure 2(a));

M2. we can hide the interaction between the supervisor and the bag (see

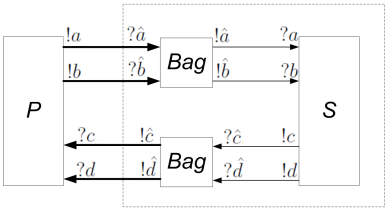
Figure 2(b));

M3. we can hide the communication at the input of both the plant and the supervisor (see Figure 2(c));

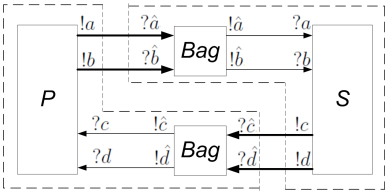
M4. we can hide the communication at the output of both the plant and the supervisor (see Figure 2(d)).



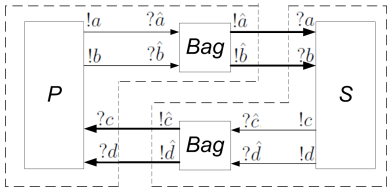
(a) Construction method M1.



(b) Construction method M2.



(c) Construction method M3.



(d) Construction method M4.

Figure 2: Different ways to hide the interactions from an ACLS. The thick lines are used to show the visible interaction and the thin lines are used to show the invisible interaction. The notation $!a$ means 'send action a' and $?a$ means 'receive action a'.

In this paper, we develop the theory for the construction method M1 (see Section 4 for the rationale behind this choice) and leave other construction methods as open for future study. Moreover, the techniques presented in this paper are restricted to reactive systems (so, no termination).

1.2 Outline

The remainder of this paper is organized as follows. In Section 2, we start our exposition by defining the overall background required for this paper.

Section 3 provides a brief introduction to supervisory control theory in a process algebraic way. In Section 4, the abstraction scheme for construction method M1 is defined formally. In Section 5, we give the formal definition of a desynchronizable closed loop system with the conditions that are sufficient for desynchronizability. In Section 6, we extend our results to PSCLSs. In Section 7, we discuss in the context of SCLS whether some of the sufficient conditions can be weakened any further. Finally, the conclusions are presented in Section 8 with some directions for future research.

2 Background

In this section, we define the basic notations and definitions that will be used throughout this article.

Let Act be a set of action names. We use the symbols $a, b, c, \ldots$ to range over the set Act. Then we define the following actions for an action label $a \in Act$,

- $!a$: send action a.
- $?a$: receive action a.
- $⁉a$: communicated action a.

Let A denote the set of all actions defined in the following way:

$$A = \{!a, ?a, ⁉a \mid a \in Act\}.$$

The variables $\alpha, \alpha_1, \alpha_2, \cdots$ are used to denote elements from the set A when the information about the type of action is irrelevant.

Next, we give a brief overview of the syntax and semantics of the process theory TCP [3]. The set of process terms $\mathbb{P}$ is generated by the grammar given in Table 1. We let the symbols p, r, s, q, q', with indices, range over the elements of $\mathbb{P}$, and fix p, r, s for the process terms associated with supervisory control theory. The constant $\mathbf{0}$ is an *inaction* process term. A unary operator $\alpha._-$ for each action $\alpha \in A$, denoting *action* prefix, is introduced in the TCP syntax. Intuitively, the process term $\alpha.q$ performs the action α and then behaves as the process term q. Observe that action prefix allows communicated actions in the syntax of a process term; this is required to model the requirements of a synchronous (asynchronous) closed loop system (see Section 3 for the formal definition of a plant, a supervisor and a

requirement). The binary operator + denotes the *alternative composition or choice* between any two process terms. The *encapsulation* operator $\partial_H(\_)$ blocks the execution of actions, which are the elements of the set $H \subseteq A$; while allowing the execution of actions present in the set $A \setminus H$. The abstraction operator $\tau_I(\_)$ *renames* the actions in the set I to the silent step $\tau \notin A$, and leaves the other actions unchanged.

q	$::=$	$\mathbf{0}$	inaction
	$\mid$	$\alpha.q$	action prefix, where $\alpha \in A$
	$\mid$	$q + q'$	alternative composition
	$\mid$	$q \parallel_\gamma q'$	parallel composition
	$\mid$	$\partial_H(q)$	action encapsulation, where $H \subseteq A$
	$\mid$	$\tau_I(q)$	abstraction (hiding of actions), where $I \subseteq A$
	$\mid$	$X = t$	recursive definition, where $X \in \mathcal{V}$
	$\mid$	$\rho_f(q)$	renaming on process terms, where $f : A \to A$

Table 1: Syntax of TCP [3].

The *synchronous parallel execution* of two process terms $q, q' \in \mathbb{P}$ is denoted as the term $q \parallel_\gamma q'$, where γ is a partial binary function called *communication function*: $\gamma(!a, ?a) = \gamma(?a, !a) = ⁉a$ for any $a \in Act$; γ is undefined otherwise.

Let $\mathcal{V}$ be a set of recursion variables. A recursive equation [3] over the signature of TCP and $\mathcal{V}$ is an equation of the form $X = t$, where X is a recursion variable from $\mathcal{V}$ and t is a term over the signature of TCP in which no other variables than those from $\mathcal{V}$ occur.

Finally, the renaming operator $\rho_f(\_)$ is defined on process terms, where $f : A \to A$ is a function on actions. Note that we use the renaming operator for technical reasons, not for modeling the basic entities in supervisory control theory.

The semantic domain of the process terms is a transition system [3] and the meaning to each process term is given by the so-called structural operational semantic rules [14], given in Table 2.

Definition 1. *Let $A_\tau = A \cup \{\tau\}$. A* transition system *over a set of actions A is a set Q of states, equipped with a transition relation $\to \subseteq Q \times A_\tau \times Q$. The transition system induced by the syntax of TCP is the tuple $(\mathbb{P}, \to)$.*

$$\frac{}{\alpha.p \xrightarrow{\alpha} p} \quad (1) \qquad \frac{p \xrightarrow{\alpha} p'}{\begin{array}{l} p+q \xrightarrow{\alpha} p' \\ q+p \xrightarrow{\alpha} p' \end{array}} \quad (2)$$

$$\frac{p \xrightarrow{\alpha} p'}{\begin{array}{l} p \parallel_\gamma q \xrightarrow{\alpha} p' \parallel_\gamma q \\ q \parallel_\gamma p \xrightarrow{\alpha} q \parallel_\gamma p' \end{array}} \quad (3) \qquad \frac{p \xrightarrow{\alpha} p', q \xrightarrow{\alpha'} q', \gamma(\alpha,\alpha') = \alpha''}{p \parallel_\gamma q \xrightarrow{\alpha''} p' \parallel_\gamma q'} \quad (4)$$

$$\frac{p \xrightarrow{\alpha} p', \alpha \notin H}{\partial_H(p) \xrightarrow{\alpha} \partial_H(p')} \quad (5) \qquad \frac{p \xrightarrow{\alpha} p', \alpha \notin I}{\tau_I(p) \xrightarrow{\alpha} \tau_I(p)} \quad (6) \qquad \frac{p \xrightarrow{\alpha} p', \alpha \in I}{\tau_I(p) \xrightarrow{\tau} \tau_I(p)} \quad (7)$$

$$\frac{t \xrightarrow{\alpha} p, X = t}{X \xrightarrow{\alpha} p} \quad (8) \qquad \frac{p \xrightarrow{\alpha} p', f : A \to A}{\rho_f(p) \xrightarrow{f(\alpha)} \rho_f(p')} \quad (9)$$

Table 2: Operational semantics for a fragment of TCP, where $\alpha \in A \cup \{\tau\}$.

Definition 2. *The* alphabet *of a process term q is a function* $\mathrm{Alph} : \mathbb{P} \to A_\tau$ *that returns the set of atomic actions that it can perform. It is defined as the least solution (w.r.t $\subseteq$) of the following equation:*

$$\begin{aligned} \mathrm{Alph}(q) &= \{\alpha \mid q \xrightarrow{\alpha} q'\} \cup \bigcup_{q \xrightarrow{\alpha} q'} \mathrm{Alph}(q'), \\ \mathrm{Alph}(q) &= \emptyset \quad \textit{if } \nexists q', \alpha.[q \xrightarrow{\alpha} q']. \end{aligned}$$

Note that the alphabet of a process term q is not defined as the alphabet operator of TCP process algebra, since it returns only the visible actions of a process term and here, $\tau \in \mathrm{Alph}(q)$ is possible.

As mentioned in the introduction, we use branching bisimulation to relate a SCLS with its corresponding ACLS in which τ actions are present. The presence of τ actions in an ACLS will become evident in Section 4. The notation $q \xrightarrow{w}\!\!\!\!\!\rightarrow q'$ for some $w \in A_\tau^*$ is inductively defined in the following

way,

$$q \xrightarrow{w}\!\!\!\!\!\rightarrow q' \triangleq q \xrightarrow{\epsilon} q \qquad \text{if } w = \epsilon,$$
$$q \xrightarrow{\alpha.w}\!\!\!\!\!\rightarrow q' \triangleq \exists q''.[q \xrightarrow{\alpha} q'' \wedge q'' \xrightarrow{w}\!\!\!\!\!\rightarrow q'].$$

The function Reach : $\mathbb{P} \to \mathbb{P}$ gives the set of reachable states for a state, and is defined Reach$(q) \triangleq \{q' \mid \exists w.[q \xrightarrow{w}\!\!\!\!\!\rightarrow q']\}$.

Definition 3. *A binary relation $\Phi \subseteq \mathbb{P} \times \mathbb{P}$ is called a* branching bisimulation relation *[3, 18] iff:*

- $\forall q, q_1, q', \alpha \in A.\Big[(q, q') \in \Phi \wedge q \xrightarrow{\alpha} q_1 \Rightarrow \exists q_1', q_2'.\big[q' \xrightarrow{\tau^*}\!\!\!\!\!\rightarrow q_1' \xrightarrow{\alpha} q_2' \wedge (q, q_1') \in \Phi \wedge (q_1, q_2') \in \Phi\big]\Big]$.
- $\forall q, q_1, q'.\Big[(q, q') \in \Phi \wedge q \xrightarrow{\tau} q_1 \Rightarrow (q_1, q') \in \Phi \vee \exists q_1', q_2'.\big[q' \xrightarrow{\tau^*}\!\!\!\!\!\rightarrow q_1' \xrightarrow{\tau} q_2' \wedge (q, q_1') \in \Phi \wedge (q_1, q_2') \in \Phi\big]\Big]$.
- $\forall q, q', q_1', \alpha \in A.\Big[(q, q') \in \Phi \wedge q' \xrightarrow{\alpha} q_1' \Rightarrow \exists q_1, q_2.\big[q \xrightarrow{\tau^*}\!\!\!\!\!\rightarrow q_1 \xrightarrow{\alpha} q_2 \wedge (q_1, q') \in \Phi \wedge (q_2, q_1') \in \Phi\big]\Big]$.
- $\forall q, q', q_1'.\Big[(q, q') \in \Phi \wedge q' \xrightarrow{\tau} q_1' \Rightarrow (q, q_1') \in \Phi \vee \exists q_1, q_2.\big[q \xrightarrow{\tau^*}\!\!\!\!\!\rightarrow q_1 \xrightarrow{\tau} q_2 \wedge (q_1, q') \in \Phi \wedge (q_2, q_1') \in \Phi\big]\Big]$.

Two processes q and q' are said to be branching bisimilar, denoted as $q \leftrightarrows_b q'$, iff there exists a branching bisimulation relation Φ such that $(q, q') \in \Phi$.

Note that the branching bisimulation equivalence relation is not a congruence on $\mathbb{P}$; although, it is congruence for the parallel composition operator $\|_\gamma$. To remedy this defect, a well-known extra condition of rootedness is required (see [3] for details).

3 Supervisory Control Theory

In this section, we give a brief introduction to supervisory control theory and define its fundamental entities in our setup. The basic entities (a plant, or a supervisor, or a requirement) in the supervisory control theory are deterministic automata. Furthermore, the proofs of Theorems 1 and 2 require the fact that a given SCLS is also deterministic process.

Definition 4. *A process $q \in \mathbb{P}$ is called* deterministic *iff*

$$\forall q_1, q_2, q_3, \alpha \in A_\tau. \left[q_1 \in \text{Reach}(q) \wedge q_1 \xrightarrow{\alpha} q_2 \wedge q_1 \xrightarrow{\alpha} q_3 \Rightarrow q_2 \equiv q_3 \right],$$

where the symbol $\equiv$ denotes syntactical equivalence between the process terms.

In supervisory control theory, plants and supervisors are allowed to perform events that are divided into two disjoint subsets: *controllable* events and *uncontrollable* events. The idea behind this partition is that the supervisor can enable or disable controllable events so that the closed loop behavior is equivalent to the requirement. The supervisor can observe but cannot influence uncontrollable events. In this paper, we follow the input-output interpretation [4] between a plant and its supervisor, wherein the uncontrollable events are outputs from a plant to a supervisor and the controllable events are outputs from a supervisor to a plant. Thus, processes that model plants or supervisors must have distinct input and output actions their alphabet. Such processes are called input-output processes.

Definition 5. *The set of input actions for an arbitrary process $q \in \mathbb{P}$ is denoted by* $\text{Alph}^?(q)$ *and is defined as* $\text{Alph}^?(q) = \{a \mid ?a \in \text{Alph}(q)\}$. *Similarly, the set of output actions (denoted by* $\text{Alph}^!(q)$*) is defined as* $\text{Alph}^!(q) = \{a \mid !a \in \text{Alph}(q)\}$. *A process q is* input-output process *iff*

$$\text{Alph}^?(q) \cap \text{Alph}^!(q) = \emptyset \wedge \text{Alph}(q) \cap I = \emptyset \wedge \tau \notin \text{Alph}(q),$$

where $I = \{⁉a \mid a \in Act\}$.

The condition $\text{Alph}(q) \cap I = \emptyset$ ensures that an input-output process does not contain communicated actions in its alphabet. This is because bags are introduced to buffer both input and output events of input-output process $q \in \mathbb{P}$; if communicated actions were allowed in the specification of the process q, the information whether the action $⁉a$ is an input or an output action of the process q would be unknown. Finally, the third condition $\tau \notin \text{Alph}(q)$ ensures that the silent action is not present in the alphabet of both, the plant and the supervisor.

We now define the three basic entities of supervisory control theory in our formalism. A *plant* $p \in \mathbb{P}$ is a deterministic input-output process. Similarly, a *supervisor* $s \in \mathbb{P}$ is a deterministic input-output process. A requirement is a process specifying the legal interaction that should occur while the plant and its supervisor are interacting such that a required task

(for which the supervisor is synthesized) is completed. Thus, a *requirement* is a deterministic process $r \in \mathbb{P}$ such that

$$\mathrm{Alph}(r) \cap H = \emptyset \wedge \tau \notin \mathrm{Alph}(r),$$

where $H = \{!a, ?a \mid a \in Act\}$. This condition ensures that a requirement process only contains communicated actions in its alphabet.

Now, we can state the *control problem* as follows: given a plant p and a requirement r, find a supervisor s such that

$$\partial_H(p \parallel_\gamma s) \leftrightarrows_{\mathrm{b}} r.$$

In this paper, we are not interested in how this supervisor is computed. Instead, we assume that we are provided with a solution to the above equation. The goal of this paper is then to find certain conditions on the given SCLS such that it is branching bisimilar to the corresponding ACLS, which we define in the next section. Note that in supervisory control theory the control problem is based on language equivalence, but branching bisimilarity coincides with language equivalence in the presence of determinism and in the absence of τ actions. However, we use branching bisimulation because the asynchronous closed loop systems as constructed in the next section are always nondeterministic. The cause of nondeterminism is due to the abstraction of interactions between a plant and the bags.

4 From Synchrony to Asynchrony

The aim of this section is to formally define the construction method M1 (see Section 1) when a SCLS is given. Next, we define a multiset and some operations over multisets in order to define a bag as a process in TCP.

A multiset ξ over the set of communicated actions I is a function $\xi : I \to \mathbb{N}$ that returns the corresponding multiplicity of the elements in a multiset. We write the empty multiset as $\varepsilon : I \to 0$, where $\forall \alpha \in I.[\xi(\alpha) = 0]$.

Definition 6. *Let $\xi : I \to \mathbb{N}$ be a multiset over the set I.*

- *The predicate $\in'$ is used to denote an element that* belongs *to a multiset. It is defined as $⁉a \in' \xi \triangleq ⁉a \in I \wedge \xi(⁉a) > 0$.*
- *The operator $\oplus$ is used to denote an* addition *of an element to a multiset. It is defined as $\xi \oplus ⁉a = \xi'$, where $\xi'(⁉a) = \xi(⁉a) + 1$ and $\xi'(\alpha) = \xi(\alpha)$ for all $\alpha \neq ⁉a$.*

- *The operator $\ominus$ is used to denote a* removal *of an element from a multiset. It is defined as $\xi \ominus ⁉a = \xi'$, where $⁉a \in' \xi, \xi'(⁉a) = \xi(⁉a) - 1$ and $\xi'(\alpha) = \xi(\alpha)$ for all $\alpha \neq ⁉a$.*
- *The size of a multiset ξ, notation $|\xi|$, is defined as $|\xi| = \sum_{\alpha \in' \xi} \xi'(\alpha)$.*

Next, we give a process algebraic definition of the construction method M1. Recall that the interactions between a plant and the bags are to be made hidden in the construction method M1; such interactions are represented with an auxiliary set $\hat{I} = \{⁉\hat{a} \mid a \in Act\}$. Similarly, we assume that the sets $\hat{A}, \hat{H}$ are defined.

Definition 7 (Bag). *A* bag process *of size n over a set of action labels $A_1 \subseteq Act$ is defined in the following way:*

$$\begin{aligned} B^n_{A_1}(\varepsilon) &= \sum_{a \in A_1} ?\hat{a}.B^n_{A_1}(\varepsilon \oplus ⁉a) \ , \\ B^n_{A_1}(\xi) &= \sum_{⁉a \in' \xi} !\hat{a}.B^n_{A_1}(\xi \ominus ⁉a) + \sum_{b \in A_1} ?\hat{b}.B^n_{A_1}(\xi \oplus ⁉b) \\ &\quad \textit{if} \quad |\xi| < n, \\ B^n_{A_1}(\xi) &= \sum_{⁉a \in' \xi} !\hat{a}.B^n_{A_1}(\xi \ominus ⁉a) \quad \textit{if} \quad |\xi| = n, \end{aligned}$$

where $\sum$ is the generalised choice that generalises the choice between an arbitrary number of processes [3].

Given a SCLS $\partial_H(p \parallel_\gamma s)$, the corresponding ACLS constructed using the method M1 is denoted by the process term:

$$\tau_{\hat{I}}(\partial_{H \cup \hat{H}}(p \parallel_{\gamma'} B^{m,n}[\varepsilon, \varepsilon] \parallel_{\gamma'} s)) \tag{1}$$

for some $m, n > 0$, where

- The notation $B^{m,n}[\varepsilon, \varepsilon]$ represents two empty interleaving bags and is defined in the following way:

  $$B^{m,n}[\varepsilon, \varepsilon] \triangleq B^m_{A_1}(\varepsilon) \parallel B^n_{A_2}(\varepsilon),$$

  where $A_1 = \text{Alph}^?(p)$, $A_2 = \text{Alph}^!(p)$. The variables $m > 0$ ($n > 0$) denotes the size of bag associated with input (output) actions of the plant p, and the sets A_1 and A_2 denote the set of input and output action labels of the plant p, respectively.

- $\gamma' : \left(A \cup \hat{A}\right) \times \left(A \cup \hat{A}\right) \to \left(A \cup \hat{A}\right)$ is the modified communication function (or the abstraction scheme for method M1) defined in the following way,
  $\gamma'(!a, ?\hat{a}) = \begin{cases} ⁉\hat{a} \text{ if } a \in \text{Alph}^{!}(p) \\ ⁉a \text{ if } a \in \text{Alph}^{!}(s) \end{cases} \quad \gamma'(!\hat{a}, ?a) = \begin{cases} ⁉\hat{a} \text{ if } a \in \text{Alph}^{?}(p) \\ ⁉a \text{ if } a \in \text{Alph}^{?}(s) \end{cases}$.
  Intuitively, the communication function γ' with the operators $\partial_{H \cup \hat{H}}, \tau_{\hat{I}}$ ensures that an ACLS contains only communication actions in its alphabet, and the interactions between a plant and the bags are invisible while the interactions between a supervisor and the bags are visible.

We explain the above construction with the help of Figure 3. Consider that the process p is able to send an output $!a$ and the process s is able to receive the input $?a$. The modified communication function γ' then transforms

- any interaction a between the process p and the bag B^n to $⁉\hat{a}$;
- any interaction a between the bag B^n and the process s to $⁉a$.

Furthermore, the operator $\tau_{\hat{I}}(\_)$ in the ACLS (Equation (1)) renames the interaction $⁉\hat{a}$ to the silent step τ.

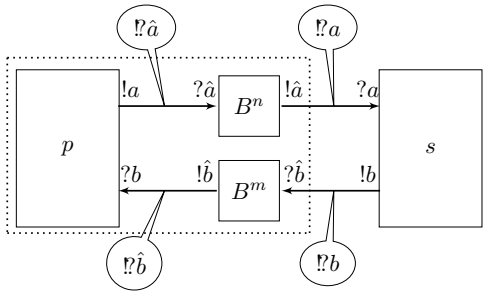


Figure 3: An ACLS constructed using the method M1.

Now consider (Figure 3) that the process s is able to send an output $!b$ and the process p is able to receive the input $?b$. The function γ' then transforms

- any interaction b between the process s and the bag B^m to $⁉b$;
- any interaction b between the bag B^m and the process p to $⁉\hat{b}$.

Furthermore, the operator $\tau_{\hat{I}}(\_)$ in the ACLS (Equation (1)) renames the interaction $⁉\hat{b}$ to the silent step τ.

The rationale behind the choice of M1 is based on the observation that the transition system generated by a supervisor s is isomorphic to the corresponding SCLS $\partial_H(p \parallel_\gamma s)$, modulo the difference in the type of action labels [6]. This is because in the synthesis of supervisors no transitions are introduced that a plant cannot execute. Moreover, the action labels in s will be decorated as either an input action (?) or an output action (!) while in $\partial_H(p \parallel_\gamma s)$ the same label will be decorated as a communicated action (⁉). Formally, this fact is equivalent to

$$\rho_f(s) \leftrightarrows \partial_H(p \parallel_\gamma s)$$

where, $f : A \to A$ is a function that renames an input/output action to a communicated action, i.e., $\forall ?a, !a \in A.[f(!a) = f(?a) = ⁉a]$. As a consequence, the supervisor model remains unaffected in the abstraction scheme M1; while, in the other abstraction schemes (M2, M3 and M4) this fact does not hold. Thus, it is easier to study abstraction scheme M1 than other schemes.

5 Desynchronizable Closed Loop System

In the previous section, we have shown how to construct an ACLS from a given SCLS. In general, the newly constructed ACLS will not be branching bisimilar to the given SCLS (for instance, see Example 1). For this purpose, we find the sufficient conditions under which a SCLS is branching bisimilar to the corresponding ACLS. Such synchronous systems are called desynchronizable closed loop systems.

Definition 8. *Let $\partial_H(p \parallel_\gamma s)$ be a SCLS and let m, n be any two nonzero natural numbers. Then, $\partial_H(p \parallel_\gamma s)$ is said to be desynchronizable with input and output buffers of size n and m (or in short desynchronizable closed loop system), respectively, if*

$$\partial_H(p \parallel_\gamma s) \leftrightarrows_b \tau_{\hat{I}}(\partial_{H \cup \hat{H}}(p \parallel_{\gamma'} B^{m,n}[\varepsilon, \varepsilon] \parallel_{\gamma'} s)) \ .$$

We now present three sufficient conditions for desynchronizability. The objective of these conditions is the following. The conditions given in Definition 9 and Definition 10 prevent an ACLS from getting deadlocked. The condition in Definition 11 ensures that the silent steps introduced by the abstraction scheme are inert with respect to branching bisimulation.

Definition 9. *Let $\partial_H(p \parallel_\gamma s)$ be a SCLS. Then, $\partial_H(p \parallel_\gamma s)$ is called* well posed *if there exists a binary relation $W \subseteq \mathbb{P} \times \mathbb{P}$ such that $(p, s) \in W$ and the following conditions are satisfied:*

- $\forall\, a, p, p', s.\, \Big[(p, s) \in W \wedge p \xrightarrow{!a} p' \Rightarrow \exists s'.\, \Big[s \xrightarrow{?a} s' \wedge (p', s') \in W\Big]\Big]$,
- $\forall\, a, p, s, s'.\, \Big[(p, s) \in W \wedge s \xrightarrow{!a} s' \Rightarrow \exists p'.\, \Big[p \xrightarrow{?a} p' \wedge (p', s') \in W\Big]\Big]$.

In other words, if a plant (supervisor) is able to send an output label $!a$ then the supervisor (plant) is able to receive the input label $?a$.

We now partition the set I of communicated actions into two disjoint non-empty subsets $I_p^?, I_p^!$ with respect to a plant process p as:

- $I_p^? \triangleq \{\text{⁉}a \mid \text{⁉}a \in I \wedge a \in \mathrm{Alph}^?(p)\}$.
- $I_p^! \triangleq \{\text{⁉}a \mid \text{⁉}a \in I \wedge a \in \mathrm{Alph}^!(p)\}$.

Definition 10. *Let $\vec{\mu} \in I_p^{?*}$ and $\vec{\nu} \in I_p^{!*}$ be sequences in $I_p^?$ and $I_p^!$, respectively. A SCLS $\partial_H(p \parallel_\gamma s)$ is said to satisfy the reordering property iff both the following conditions are satisfied,*

1. $\forall p', p_2, s', \partial_H(p_1 \parallel_\gamma s_1) \in \mathrm{Reach}(\partial_H(p \parallel_\gamma s)), \text{⁉}a \in I_p^?.$
   $\big[\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\mu}.\text{⁉}a} \partial_H(p' \parallel_\gamma s') \wedge p_1 \xrightarrow{?a} p_2 \Rightarrow$
   $\exists s_2.[\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\text{⁉}a} \partial_H(p_2 \parallel_\gamma s_2)]\big]$
2. $\forall p', s', s_2, \partial_H(p_1 \parallel_\gamma s_1) \in \mathrm{Reach}(\partial_H(p \parallel_\gamma s)), \text{⁉}a \in I_p^!.$
   $\big[\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\nu}.\text{⁉}a} \partial_H(p' \parallel_\gamma s') \wedge s_1 \xrightarrow{?a} s_2 \Rightarrow$
   $\exists p_2.[\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\text{⁉}a} \partial_H(p_2 \parallel_\gamma s_2)]\big]$.

Informally, the reordering property states that if the receiver (plant or supervisor) is willing to receive an input $?a$ and the sender (supervisor or plant) can perform a sequence of outputs $\vec{\mu}.\text{⁉}a$, then the receiver can receive the message a before receiving the sequence of outputs $\vec{\mu}$ in the SCLS. This is due to the nature of buffers (i.e., bags) that are inserted between the plant and its supervisor.

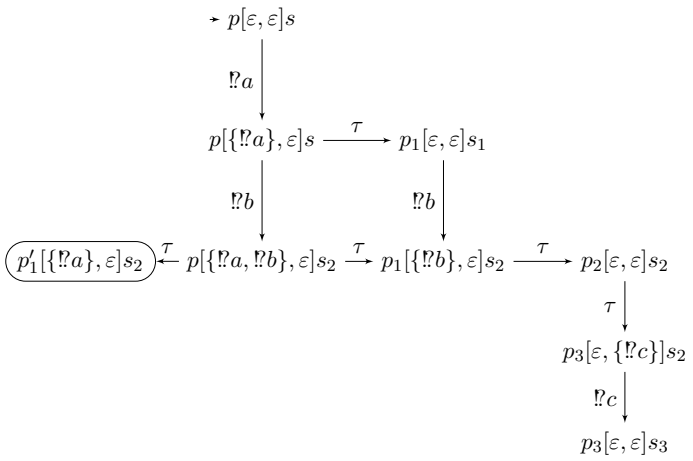


Figure 4: An example showing a deadlock in the ACLS, which is not present in the SCLS. Notation, $p[\mu,\nu]s = \tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p \parallel_{\gamma'} B^{m,n}[\mu,\nu] \parallel_{\gamma'} s))$.

Example 1. *Consider the behaviour of a plant p and its supervisor s specified by the following set of equations:*

$$p = ?a.p_1 + ?b.p_1' \qquad p_1 = ?b.p_2 \quad p_2 = !c.p_3$$
$$s = !a.s_1 \qquad s_1 = !b.s_2 \quad s_2 = ?c.s_3$$

The behaviour of the SCLS $\partial_H(p \parallel_\gamma s)$:

$$\partial_H(p \parallel_\gamma s) = \wr a.\partial_H(p_1 \parallel_\gamma s_1) \quad \partial_H(p_1 \parallel_\gamma s_1) = \wr b.\partial_H(p_2 \parallel_\gamma s_2)$$
$$\partial_H(p_2 \parallel_\gamma s_2) = \wr c.\partial_H(p_3 \parallel_\gamma s_3).$$

The transition system generated by the ACLS $\tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} s))$ is depicted in Figure 4. Note that the interactions in ACLS can delay in the buffers, so the supervisor can perform the output sequence $!a.!b$ without allowing any moves from the plant p, i.e., the reachable state

$$\tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p \parallel_{\gamma'} B^{m,n}[\{\wr a,\wr b\},\varepsilon] \parallel_{\gamma'} s_2)).$$

Due to the equation $p = ?a.p_1 + ?b.p_1'$, the plant can remove the input b from its input bag before the input a and leading the ACLS into a deadlock state (shown as rounded rectangle in Figure 4).

Definition 11. *A process $q \in \mathbb{P}$ is said to satisfy the diamond property iff the following condition holds (see Figure 5)*

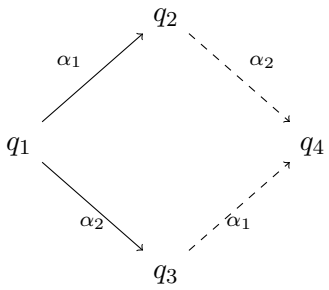


Figure 5: Diamond property.

- $\forall \alpha_1, \alpha_2, q_1, q_2, q_3. \Big[q_1 \in \text{Reach}(q) \wedge q_1 \xrightarrow{\alpha_1} q_2 \wedge q_1 \xrightarrow{\alpha_2} q_3 \wedge \alpha_1 \neq \alpha_2 \Rightarrow \exists q_4. [q_2 \xrightarrow{\alpha_2} q_4 \wedge q_3 \xrightarrow{\alpha_1} q_4] \Big]$.

For a reader familiar with the concepts of *true concurrency* [19], the conditions given in Definitions 4, 10 and 11 are similar to the axioms of asynchronous transition systems. The formulation of these axioms is based on the definition of an independence relation, which is an irreflexive and symmetric relation on the set of actions A. However, the techniques for desynchronizability for such models are not investigated here, although it will be worthwhile to examine this research direction in the future. Note that in our approach we do not need an additional notion of the independence relation.

5.1 Sufficient Condition for Desynchronizability

In this subsection, we first introduce a notion of sequence generated from a multiset. Secondly, Definitions 10 and 11 are lifted from an action to a sequence of actions. Finally, we prove the main theorem stating: If an arbitrary SCLS $\partial_H(p \parallel_\gamma s)$ satisfies the conditions in Definitions 9, 10, and 11 then

$$\forall m, n > 0. \Big[\partial_H(p \parallel_\gamma s) \leftrightarrows_{\mathrm{b}} \tau_{\hat{I}}(\partial_{H \cup \hat{H}}(p \parallel_{\gamma'} B^{m,n}[\varepsilon, \varepsilon] \parallel_{\gamma'} s)) \Big] .$$

We fix the symbols μ, ν for the contents of the bag attached to input and output actions of the plant process p, respectively. Formally, $\forall \alpha \in I_p^!.[\mu(\alpha) = 0]$ and $\forall \alpha \in I_p^?.[\nu(\alpha) = 0]$. For an arbitrary multiset ξ, we write

$\vec{\xi}$ to denote a sequence generated from this multiset. For example, consider a multiset $\xi = \{⁉a, ⁉a, ⁉b, ⁉b\}$. Then a possible sequence $\vec{\xi}$ over the given ξ can be of the form $<⁉a.⁉b.⁉a.⁉b>$. Let $f_i : I^* \to H^*$ be the function defined as $f_i(⁉a.\vec{\xi}) = ?a.f_i(\vec{\xi})$. Similarly, let $f_o : I^* \to H^*$ be the function defined as $f_o(⁉a.\vec{\xi}) = !a.f_o(\vec{\xi})$.

Proposition 1. *1. Given a trace* $\partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\mu}} \partial_H(p_1 \parallel_\gamma s_1)$, *we find using the above function* f_i *and semantics of* $\parallel_\gamma$ *that* $p \xrightarrow{f_i(\vec{\mu})} p_1 \wedge s \xrightarrow{f_o(\vec{\mu})} s_1$.

2. Similarly, given a trace $\partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\nu}} \partial_H(p_1 \parallel_\gamma s_1)$, *we conclude that* $p \xrightarrow{f_o(\vec{\nu})} p_1 \wedge s \xrightarrow{f_i(\vec{\nu})} s_1$.

Lemma 1 is a generalization of Definition 11. It states that if two different states q_1, q_2 are reachable from a state q_0, then there exists a state q_3 reachable from q_1 and q_2 such that, the trace between q_0, q_1 and the trace between q_0, q_2 commute.

Lemma 1 (**Generalized diamond property**). *Let* $\partial_H(p \parallel_\gamma s)$ *be an arbitrary SCLS satisfying the diamond property (Definition 11). If* $\partial_H(p_1 \parallel_\gamma s_1) \in \mathrm{Reach}(\partial_H(p \parallel_\gamma s))$ *and* $\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\xi}} \partial_H(p_2 \parallel_\gamma s_2) \wedge \partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\xi'}} \partial_H(p_3 \parallel_\gamma s_3)$ *then,*

$$\exists p_4, s_4.[\partial_H(p_2 \parallel_\gamma s_2) \xrightarrow{\vec{\xi'}} \partial_H(p_4 \parallel_\gamma s_4) \wedge \partial_H(p_3 \parallel_\gamma s_3) \xrightarrow{\vec{\xi}} \partial_H(p_4 \parallel_\gamma s_4)].$$

Lemma 2 is the result obtained by direct instantiation of the reordering property (Definition 10) and the generalized diamond property (Lemma 1).

Lemma 2. *Let* $\partial_H(p \parallel_\gamma s)$ *be a SCLS satisfying the conditions in Definitions 10 and 11.*

1. Suppose $⁉a \in I_p^? \wedge \partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\mu}.⁉a} \partial_H(p_2 \parallel_\gamma s_2) \wedge p \xrightarrow{?a} p_1$ *then,*

$$\exists s_1. \left[\partial_H(p \parallel_\gamma s) \xrightarrow{⁉a} \partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\mu}} \partial_H(p_2 \parallel_\gamma s_2)\right].$$

2. Suppose $⁉a \in I_p^! \wedge \partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\nu}.⁉a} \partial_H(p_3 \parallel_\gamma s_3) \wedge s \xrightarrow{?a} s_1$ *then,*

$$\exists p_1. \left[\partial_H(p \parallel_\gamma s) \xrightarrow{⁉a} \partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\nu}} \partial_H(p_3 \parallel_\gamma s_3)\right].$$

Proof: *We give the proof of Item 1 only. It is given that* $\partial_H(p \parallel_\gamma s)$ *satisfies the conditions in Definitions 9, 10 and 11 with* $⁉a \in I_p^? \wedge \partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\mu}.⁉a} \partial_H(p_2 \parallel_\gamma s_2) \wedge p \xrightarrow{?a} p_1$. *Then by the reordering property (Definition 10) we get,*

$$\exists s_1.[\partial_H(p \parallel_\gamma s) \xrightarrow{⁉a} \partial_H(p_1 \parallel_\gamma s_1)] \ .$$

By the given transitions $\partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\mu}.⁉a} \partial_H(p_2 \parallel_\gamma s_2)$ *we infer that,*

$$\exists p_1', s_1'.[\partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\mu}} \partial_H(p_1' \parallel_\gamma s_1') \xrightarrow{⁉a} \partial_H(p_2 \parallel_\gamma s_2)] \ .$$

Applying the generalised diamond property (Lemma 1) at the state $\partial_H(p \parallel_\gamma s)$ *we get,*

$$\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\mu}} \partial_H(p_2 \parallel_\gamma s_2) \ .$$

Hence, the desired result is achieved. Likewise, Item 2 can be proved. □

Theorem 1. *Let* $\partial_H(p \parallel_\gamma s)$ *be an arbitrary SCLS satisfying the conditions in Definitions 9, 10 and 11. Then for any* $m, n > 0$ *we have,*

$$\partial_H(p \parallel_\gamma s) \leftrightarrows_b \tau_{\hat{I}}(\partial_{H \cup \hat{H}}(p \parallel_{\gamma'} B^{m,n}[\varepsilon, \varepsilon] \parallel_{\gamma'} s)).$$

Sketch of the proof: *Define a relation* Φ *as follows.*

$$\begin{aligned}
\Phi \triangleq \{(\partial_H(p \parallel_\gamma s), \tau_{\hat{I}}(\partial_{H \cup \hat{H}}(p' \parallel_{\gamma'} B^{m,n}[\mu, \nu] \parallel_{\gamma'} s))) \mid \\
\Big(p' = p \wedge \mu = \varepsilon \wedge \nu = \varepsilon\Big) \bigvee \quad &[C1] \\
\Big(\mu = \varepsilon \wedge \exists s'.\big[\partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\nu}} \partial_H(p' \parallel_\gamma s')\big]\Big) \bigvee \quad &[C2] \\
\Big(\nu = \varepsilon \wedge \exists s'.\big[\partial_H(p' \parallel_\gamma s') \xrightarrow{\vec{\mu}} \partial_H(p \parallel_\gamma s)\big]\Big) \bigvee \quad &[C3] \\
\Big(\exists p'', s', s''.\big[\partial_H(p' \parallel_\gamma s') \xleftarrow{\vec{\nu}} \partial_H(p'' \parallel_\gamma s'') \xrightarrow{\vec{\mu}} \partial_H(p \parallel_\gamma s)\big]\Big) \bigvee \quad &[C4] \\
\Big(\exists p'', s', s''.\big[\partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\nu}} \partial_H(p'' \parallel_\gamma s'') \xleftarrow{\vec{\mu}} \partial_H(p' \parallel_\gamma s')\big]\Big) \ \}. \quad &[C5]
\end{aligned}$$

Note that the above conditions C1, C2, C3, C4 and C5 are independent of n, m. *The proof of the theorem is based on showing that* Φ *is a witnessing branching bisimulation relation. A state* $\partial_H(p \parallel_\gamma s)$ *in a SCLS is related to those states in an ACLS that contain the same supervisor state* s. *The* Φ *relation between two states is indicated by dotted lines in Figure 6. The complete proof uses a lot of case distinction and can be found in [5].*

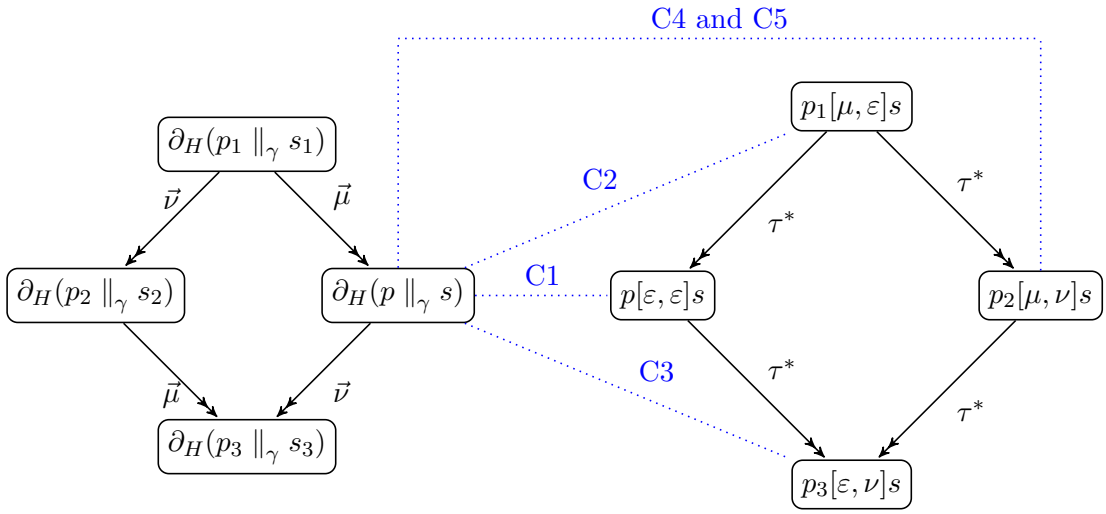


Figure 6: Illustration of the relation Φ, where $p[\mu,\nu]s = \tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p \parallel_{\gamma'} B^{m,n}[\mu,\nu] \parallel_{\gamma'} s))$.

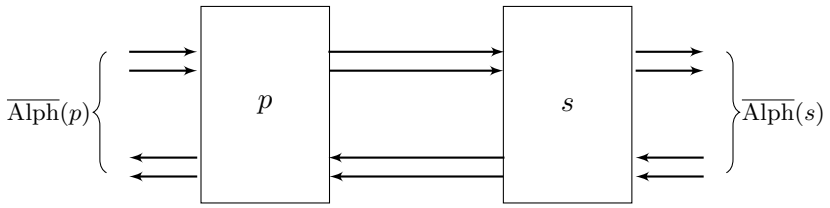


Figure 7: A partial synchronous closed loop system.

6 Desynchronization of Partial Synchronous Closed Loop Systems

In the previous section, we showed that the well posedness, reordering and diamond properties are sufficient conditions under which a SCLS is desynchronizable. In this section, we extend the desynchronizability result to a class of SCLSs whose alphabet not only contains interaction between a plant and its supervisor; but also, external actions of plant or supervisor or both. These external actions will result in interaction with an external environment. Such closed loop systems are called partial synchronous closed loop systems (PSCLSs) and Figure 7 shows the context diagram of a PSCLS.

To carry out similar work as in Section 5, we first modify the definitions of a plant, a supervisor, and a requirement in order to construct a PSCLS. Secondly, we extend the definitions of the aforementioned sufficient conditions in a conservative way. For a plant p (supervisor s), we assume a given set of external actions $\overline{\mathrm{Alph}}(p)$ ($\overline{\mathrm{Alph}}(s)$).

Definition 12. *A plant (supervisor) $p \in \mathbb{P}$ ($s \in \mathbb{P}$) is an input-output and deterministic process such that the set of external actions are nonempty, i.e., $\overline{\mathrm{Alph}}(p) \neq \emptyset$. Define the modified blocking set,*

$$H(p,s) = \{\alpha, \beta \mid \alpha \notin \overline{\mathrm{Alph}}(p) \wedge \beta \notin \overline{\mathrm{Alph}}(s)\}.$$

A requirement $r \in \mathbb{P}$ for a partial SCLS $\partial_{H(p,s)}(p \parallel_\gamma s)$ is a deterministic process such that,

$$\mathrm{Alph}(r) \cap H(p,s) = \emptyset \wedge \tau \notin \mathrm{Alph}(r).$$

Furthermore, the control equation in this new setting is the following.

$$\partial_{H(p,s)}(p \parallel_\gamma s) \leftrightarrows_b r.$$

The above modifications results in the following change of the definition of the communication function γ' which implements the abstraction scheme M1. We write the ACLS as

$$\tau_{\hat{I}}(\partial_{H(P,S)\cup\hat{H}(P,S)}(p \parallel_{\gamma'} B^{m,n}(\varepsilon,\varepsilon) \parallel_{\gamma'} s))$$

(for some $m, n > 0$), constructed from a PSCLS $\partial_{H(p,s)}(p \parallel_\gamma s)$ with,

$$\gamma'(!a, ?\hat{a}) = \begin{cases} ⁉\hat{a} \text{ if } !a \in \mathrm{Alph}^!(p) \wedge !a \in H(p,s) \\ ⁉a \text{ if } !a \in \mathrm{Alph}^!(s) \wedge !a \in H(p,s) \end{cases},$$
$$\gamma'(!\hat{a}, ?a) = \begin{cases} ⁉\hat{a} \text{ if } ?a \in \mathrm{Alph}^?(p) \wedge ?a \in H(p,s) \\ ⁉a \text{ if } ?a \in \mathrm{Alph}^?(s) \wedge ?a \in H(p,s) \end{cases}.$$

Informally, the above definition of communication function γ' not only implements the abstraction scheme M1, but it also *restricts* the communication of external actions of both the plant and the supervisor with the bags.

6.1 Modified Sufficient Conditions

In this section, we extend the well-posedness, the reordering and the diamond property in a conservative way. Furthermore, we introduce an additional property, called fair-noise property, that ensures the presence of external actions of plant is safe. The intuition of these conditions will be explain alongside with their respective formal definitions.

Notation. We write a PSCLS $\partial_{H(P,S)}(p \parallel_\gamma s)$ as $\partial_\diamond(p \parallel_\gamma s)$ and the corresponding partial asynchronous closed loop system as $\widehat{\nabla}(p \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} s)$.

Definition 13. *Let $\partial_\diamond(p \parallel_\gamma s)$ be a PCLS. Then, $\partial_\diamond(p \parallel_\gamma s)$ is called* well posed *if there exists a binary relation $W \subseteq \mathbb{P} \times \mathbb{P}$ such that $(p,s) \in W$ and the following conditions are satisfied:*

1. $\forall p, p', s, a. \Big[(p,s) \in W \wedge p \xrightarrow{!a} p' \wedge\ !a \notin \overline{\mathrm{Alph}}(p) \Rightarrow$
$\exists s'. \big[s \xrightarrow{?a} s' \wedge (p',s') \in W\big]\Big]$,
2. $\forall p, p', s, \alpha. \Big[(p,s) \in W \wedge p \xrightarrow{\alpha} p' \wedge \alpha \in \overline{\mathrm{Alph}}(p) \Rightarrow (p',s) \in W\Big]$,
3. $\forall p, s, s', a. \Big[(p,s) \in W \wedge s \xrightarrow{!a} s' \wedge\ !a \notin \overline{\mathrm{Alph}}(s) \Rightarrow$
$\exists p'. [p \xrightarrow{?a} p' \wedge (p',s') \in W]\Big]$,
4. $\forall p, s, s', \alpha. \Big[(p,s) \in W \wedge s \xrightarrow{\alpha} s' \wedge \alpha \in \overline{\mathrm{Alph}}(s) \Rightarrow (p,s') \in W\Big]$.

The conditions 1, 3 in the above definition are the usual conditions (see Definition 9) of well-posedness property. However, the conditions 2, 4 ensures that an external step $q \xrightarrow{\alpha} q' (\alpha \in \overline{\mathrm{Alph}}(p) \cup \overline{\mathrm{Alph}}(s))$ performed at a receiver's state (either plant's state or supervisor's state, i.e., $q \equiv p$ or $q \equiv s$) do not alter the set of input actions enabled at the state q and thus remains well-posed.

As already mentioned, the above well-posedness property was designed to prevent a PSCLS from getting deadlocked. Unfortunately, the well-posedness property *alone* is not sufficient for this purpose. The following example illustrates this fact.

Example 2. *Let $\alpha \in \overline{\mathrm{Alph}}(s)$ and consider the following equations describing*

the incomplete behaviour of a plant and a supervisor.

$$p = ?a.\mathbf{0} + ?b.p_1 \qquad s = !b.s_1,\ s_1 = \alpha.s_2$$
$$p_1 = ?a.p_2,\ p_2 = !c.p \qquad s_2 = !a.s_3,\ s_3 = ?c.s$$

Figures 8(a) and 8(b) shows the transition system of $\partial_\diamond(p \parallel_\gamma s)$ *and* $\widehat{\nabla}(p \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} s)$*, respectively. Note, that the PSCLS* $\partial_\diamond(p \parallel_\gamma s)$ *is deadlock free; however, the ACLS* $\widehat{\nabla}(p \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} s)$ *contains the following deadlock trace* $<\!\!\text{⁉}b.\alpha.\text{⁉}a.\tau>$*.*

Thus, the following reordering property is designed to eliminate such scenarios.

Definition 14. *Let* $\vec{\mu} \in (I_p^? \cup \overline{\text{Alph}}(s))^*$ *and* $\vec{\nu} \in (I_p^! \cup \overline{\text{Alph}}(p))^*$ *be sequences in* $I_p^?$ *and* $I_p^!$*, respectively. A PSCLS* $\partial_\diamond(p \parallel_\gamma s)$ *is said to satisfy the* reordering *property iff,*

- $\forall p', p_2, s', \partial_\diamond(p_1 \parallel_\gamma s_1) \in \text{Reach}(\partial_\diamond(p \parallel_\gamma s)), \text{⁉}a \in I_p^?.$
  $\Big[\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\mu}.\text{⁉}a}\!\!\!\!\rightarrow\ \partial_\diamond(p' \parallel_\gamma s') \wedge p_1 \xrightarrow{?a} p_2 \Rightarrow$
  $\exists s_2.\big[\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\text{⁉}a} \partial_\diamond(p_2 \parallel_\gamma s_2)\big]\Big]$
- $\forall p', s', s_2, \partial_\diamond(p_1 \parallel_\gamma s_1) \in \text{Reach}(\partial_\diamond(p \parallel_\gamma s)), \text{⁉}a \in I_p^!.$
  $\Big[\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\nu}.\text{⁉}a}\!\!\!\!\rightarrow\ \partial_\diamond(p' \parallel_\gamma s') \wedge s_1 \xrightarrow{?a} s_2 \Rightarrow$
  $\exists p_2.\big[\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\text{⁉}a} \partial_\diamond(p_2 \parallel_\gamma s_2)\big]\Big].$

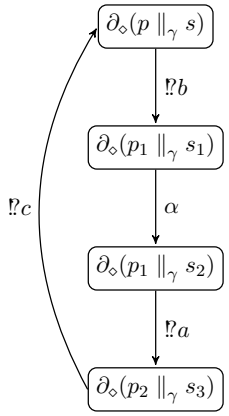
In the new setting, the diamond property (Definition 15) also ensures that the silent steps generated by the abstraction scheme are inert.

Definition 15. *A process* $q \in \mathbb{P}$ *is said to satisfy the diamond property iff the following condition holds:*

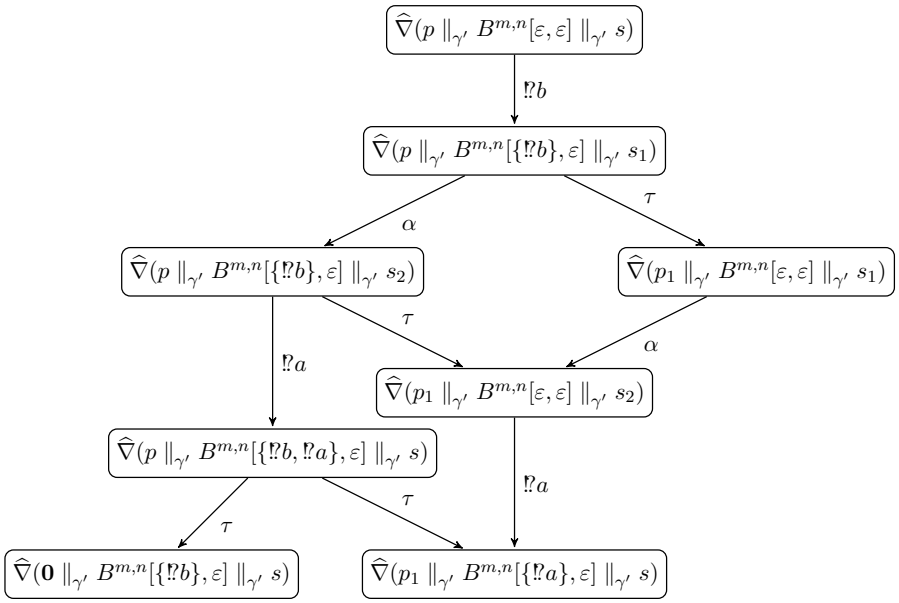
- $\forall \alpha_1, \alpha_2, q_1, q_2, q_3.\Big[q_1 \in \text{Reach}(q) \wedge q_1 \xrightarrow{\alpha_1} q_2 \wedge q_1 \xrightarrow{\alpha_2} q_3 \wedge \alpha_1 \neq \alpha_2 \Rightarrow$
  $\exists q_4.[q_2 \xrightarrow{\alpha_2} q_4 \wedge q_3 \xrightarrow{\alpha_1} q_4]\Big]$.

We have extended the old sufficient conditions in the setting of PSCLSs and expect the desynchronizability result to hold via a branching bisimulation relation $\widehat{\Phi}$ [2] similar to the one used in the proof of Theorem 1.

[2] We introduce a different notation for branching bisimulation in the context of PSCLS because of the presence of external actions of a plant and its supervisor.



(a) Transition system of $\partial_H(p \parallel_\gamma s)$.



(b) Partial transition system of $\widehat{\nabla}(p \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} s)$ showing the deadlock at the state $\widehat{\nabla}(\mathbf{0} \parallel_{\gamma'} B^{m,n}[\{\between b\},\varepsilon] \parallel_{\gamma'} s)$.

Figure 8: Example 2.

The idea behind the design of the relation $\widehat{\Phi}$ should be *to relate a state* $\partial_\diamond(p \parallel_\gamma s)$ *in a PSCLS to those states* $\widehat{\nabla}(p' \parallel_{\gamma'} B^{m,n}[\mu,\nu] \parallel_{\gamma'} s)$ *in an ACLS that contain the same supervisor state* s. This is due to the abstraction scheme used to construct a partial asynchronous closed loop system from a given PSCLS. Unfortunately, there are certain scenarios (explained in the following paragraph) due to the external step made by plant, which cause more behaviour in an ACLS that will not be present in the corresponding PSCLS. However, if a PSCLS contains only the external actions of the supervisor (i.e. no external actions of the plant) in its alphabet then the above modified conditions are sufficient for desynchronizability.

Next we explore the scenarios in which the external step made by a plant in a PSCLS obstructs its desynchronizability.

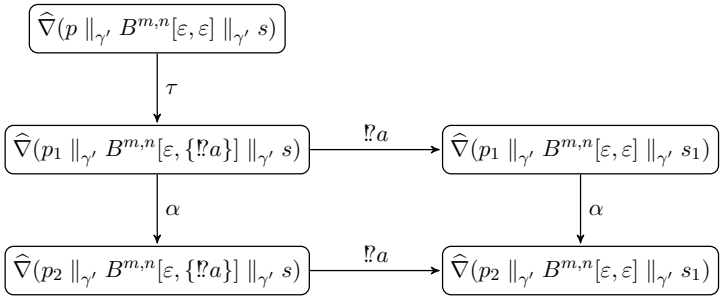


Figure 9: Transition system of $\widehat{\nabla}(p \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} s)$ in the Example 3.

Example 3. *Consider the behaviour of a PSCLS specified by the following equations*

$$\partial_\diamond(p \parallel_\gamma s) = ⁉a.\partial_\diamond(p_1 \parallel_\gamma s_1),\ \partial_\diamond(p_1 \parallel_\gamma s_1) = \alpha.\partial_\diamond(p_2 \parallel_\gamma s_1)$$

where $⁉a \in I_p^!, \alpha \in \overline{\mathrm{Alph}}(p)$. *The transition system generated by the ACLS is shown in Figure 9. Immediately, we observe the trace* $< \tau.\alpha.⁉a >$ *from the state* $\widehat{\nabla}(p \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} s)$ *and thus disallowing the states* $\partial_\diamond(p \parallel_\gamma s)$, $\widehat{\nabla}(p_1 \parallel_{\gamma'} B^{m,n}[\varepsilon,\{⁉a\}] \parallel_{\gamma'} s)$ *to be related by a branching bisimulation relation* $\widehat{\Phi}$. *Moreover, it contradicts our intuition about* $\widehat{\Phi}$ *that "a state* $\partial_\diamond(p \parallel_\gamma s)$ *in a PSCLS to those states in an ACLS that contain the same supervisor state* s*". To rectify this, we require that the state* $\partial_\diamond(p \parallel_\gamma s)$ *must contain the trace* $< \alpha.⁉a >$ *reachable to the state* $\partial_\diamond(p_2 \parallel_\gamma s_1)$. *This leads to another sufficient condition.*

Definition 16. *Let* $⁉a \in I_p^!$ *and* $\alpha \in \overline{\text{Alph}}(p)$. *A plant* p *satisfies the* fair-noise *property in a PSCLS* $\partial_\diamond(p \parallel_\gamma s)$ *iff,*

$$\forall \partial_\diamond(p_1 \parallel_\gamma s_1) \in \text{Reach}(\partial_\diamond(p \parallel_\gamma s)).\Big[\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{⁉a.\alpha}\!\!\!\!\rightarrow \partial_\diamond(p_2 \parallel_\gamma s_2) \Rightarrow \exists p_1'.\big[\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\alpha} \partial_\diamond(p_1' \parallel_\gamma s_1)\big]\Big].$$

Lemma 3. *Let* $\vec{\nu} \in I_p^{!*}$, $\alpha \in \overline{\text{Alph}}(p)$ *and* $\partial_\diamond(p_1 \parallel_\gamma s_1)$ *satisfies the fair-noise property (Definition 16) and the diamond property (Definition 15). If* $\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\nu}.\alpha}\!\!\!\!\rightarrow \partial_\diamond(p_2 \parallel_\gamma s_2)$ *then,*

$$\exists p_1'.[\partial_\diamond(p_1' \parallel_\gamma s_1) \xrightarrow{\alpha.\vec{\nu}}\!\!\!\!\rightarrow \partial_\diamond(p_2 \parallel_\gamma s_2)].$$

Proof: *Straightforward, by induction on* $\vec{\nu}$. □

Proposition 2. *Let* $\vec{\nu} \in I_p^{!*}, \vec{\sigma} \in \overline{\text{Alph}}(p)^*$ *and suppose* $\partial_\diamond(p \parallel_\gamma s)$ *satisfies the fair-noise property (Definition 16) and the diamond property (Definition 15). If* $\partial_\diamond(p \parallel_\gamma s) \xrightarrow{\vec{\nu}.\vec{\sigma}}\!\!\!\!\rightarrow \partial_\diamond(p_1 \parallel_\gamma s_1)$ *then,*

$$\exists p', s'. \left[\partial_\diamond(p \parallel_\gamma s) \xrightarrow{\vec{\sigma}}\!\!\!\!\rightarrow \partial_\diamond(p' \parallel_\gamma s') \xrightarrow{\vec{\nu}}\!\!\!\!\rightarrow \partial_\diamond(p_1 \parallel_\gamma s_1)\right].$$

Proof: *Direct from Lemma 3.* □

Lemma 4. *Let* $\partial_\diamond(p \parallel_\gamma s)$ *be an arbitrary PSCLS satisfying the extended diamond property (Definition 15). If* $\partial_\diamond(p_1 \parallel_\gamma s_1) \in \text{Reach}(\partial_\diamond(p \parallel_\gamma s))$ *and*

$$\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\xi}}\!\!\!\!\rightarrow \partial_\diamond(p_2 \parallel_\gamma s_2) \ \wedge\ \partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\xi'}}\!\!\!\!\rightarrow \partial_\diamond(p_3 \parallel_\gamma s_3) \text{ then,}$$

$$\exists p_4, s_4.[\partial_\diamond(p_2 \parallel_\gamma s_2) \xrightarrow{\vec{\xi'}}\!\!\!\!\rightarrow \partial_\diamond(p_4 \parallel_\gamma s_4) \wedge \partial_\diamond(p_3 \parallel_\gamma s_3) \xrightarrow{\vec{\xi}}\!\!\!\!\rightarrow \partial_\diamond(p_4 \parallel_\gamma s_4)].$$

Lemma 5. *Let* $\partial_\diamond(p_1 \parallel_\gamma s_1)$ *be a PSCLS satisfying the well-posedness (Definition 13), reordering property (Definition 14) and the diamond property (Definition 15). Suppose* $⁉a \in I_p^?$, $\vec{\sigma} \in (\overline{\text{Alph}}(p) \cup \overline{\text{Alph}}(s))^*$ *such that* $\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\sigma}}\!\!\!\!\rightarrow \partial_\diamond(p_2 \parallel_\gamma s_2) \xrightarrow{⁉a} \partial_\diamond(p_3 \parallel_\gamma s_3)$. *Then,*

$$\exists p_3', s_3'.[\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{⁉a} \partial_\diamond(p_3' \parallel_\gamma s_3') \xrightarrow{\vec{\sigma}}\!\!\!\!\rightarrow \partial_\diamond(p_3 \parallel_\gamma s_3)].$$

Proof: *We use structural induction on $\vec{\sigma}$ to prove this result. Without loss of generality, assume that $\vec{\sigma} = \alpha.\vec{\sigma}'$. Consider the transition $\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\alpha} \partial_\diamond(p_1' \parallel_\gamma s_1') \xrightarrow{\vec{\sigma}'} \partial_\diamond(p_2 \parallel_\gamma s_2) \xrightarrow{\overset{\shortmid}{?}a} \partial_\diamond(p_3 \parallel_\gamma s_3)$. By induction hypothesis we have,*

$$\exists p_3', s_3'.[\partial_\diamond(p_1' \parallel_\gamma s_1') \xrightarrow{\overset{\shortmid}{?}a} \partial_\diamond(p_3' \parallel_\gamma s_3') \xrightarrow{\vec{\sigma}'} \partial_\diamond(p_3 \parallel_\gamma s_3)].$$

We identify two cases based on the external action α performed either by the plant (p) or the supervisor (s).

1. *$\alpha \in \overline{\text{Alph}}(p)$. Then by semantics of $\parallel_\gamma$ we know that $s_1 = s_1'$. Furthermore, from the induction hypothesis, $\overset{\shortmid}{?}a \in I_p^?$ and the semantics of $\parallel_\gamma$ we get, $p_1' \xrightarrow{?a} p_3' \wedge s_1' \xrightarrow{!a} s_3'$. By applying well-posedness (Definition 13) at the state $\partial_\diamond(p_1 \parallel_\gamma s_1')$ we get,*

   $$\exists p_1''.[\partial_\diamond(p_1 \parallel_\gamma s_1') \xrightarrow{\overset{\shortmid}{?}a} \partial_\diamond(p_1'' \parallel_\gamma s_3')].$$

   Applying the Lemma 4 we get the desired result,

   $$\partial_\diamond(p_1 \parallel_\gamma s_1') \xrightarrow{\overset{\shortmid}{?}a} \partial_\diamond(p_1'' \parallel_\gamma s_3') \xrightarrow{\alpha} \partial_\diamond(p_3' \parallel_\gamma s_3').$$

2. *$\alpha \in \overline{\text{Alph}}(s)$. Then by semantics of $\parallel_\gamma$ we know that $p_1 = p_1'$. Furthermore, from the induction hypothesis, $\overset{\shortmid}{?}a \in I_p^?$ and the semantics of $\parallel_\gamma$ we get, $p_1' \xrightarrow{?a} p_3' \wedge s_1' \xrightarrow{!a} s_3'$. Using Definition 14 with the transitions $\partial_\diamond(p_1' \parallel_\gamma s_1) \xrightarrow{\alpha.\overset{\shortmid}{?}a} \partial_\diamond(p_3' \parallel_\gamma s_3')$ and $p_1' \xrightarrow{?a} p_3'$ we get,*

   $$\exists p_1'', s_1''.[\partial_\diamond(p_1' \parallel_\gamma s_1) \xrightarrow{\overset{\shortmid}{?}a} \partial_\diamond(p_1'' \parallel_\gamma s_1'')].$$

   Applying the Lemma 4 we get the desired result,

   $$\partial_\diamond(p_1' \parallel_\gamma s_1) \xrightarrow{\overset{\shortmid}{?}a} \partial_\diamond(p_1'' \parallel_\gamma s_1'') \xrightarrow{\alpha} \partial_\diamond(p_3' \parallel_\gamma s_3').$$

□

Lemma 6. *Let $\partial_\diamond(p_1 \parallel_\gamma s_1)$ be a PSCLS satisfying the well-posedness (Definition 13), reordering property (Definition 14) and the diamond property*

(Definition 15). Suppose $?a \in I_p^!$, $\vec{\sigma} \in (\overline{\mathrm{Alph}}(p) \cup \overline{\mathrm{Alph}}(s))^*$ *such that* $\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\sigma}} \!\!\!\!\rightarrow \partial_\diamond(p_2 \parallel_\gamma s_2) \xrightarrow{?a} \partial_\diamond(p_3 \parallel_\gamma s_3)$. *Then,*

$$\exists p_3', s_3'.[\partial_\diamond(p_1 \parallel_\gamma s_1) \xrightarrow{?a} \partial_\diamond(p_3' \parallel_\gamma s_3') \xrightarrow{\vec{\sigma}} \!\!\!\!\rightarrow \partial_\diamond(p_3 \parallel_\gamma s_3)].$$

Proof: *Similar to the proof of Lemma 5.* □

Next we pose the following main result of this section: "If a PSCLS satisfies the condition of Definition 13, 14, 15 and 16 then it is desynchronizable."

Theorem 2. *Let* $\partial_\diamond(p \parallel_\gamma s)$ *be an arbitrary PSCLS satisfying the conditions in Definitions 13, 14, 15 and 16. Then for any* $m, n > 0$ *we have,*

$$\partial_\diamond(p \parallel_\gamma s) \leftrightarrows_b \widehat{\nabla}(P \parallel_{\gamma'} B^{m,n}[\varepsilon, \varepsilon] \parallel_{\gamma'} S).$$

Sketch of the proof: *The proof of this theorem requires lots of case distinction and can be found in [5]. Here, we only give the witnessing branching bisimulation relation* $\widehat{\Phi}$. *Recall the relation* Φ *defined in the proof of Theorem 1 and define a relation* $\widehat{\Phi}$ *as follows,*

$$\widehat{\Phi} \triangleq \Phi \bigcup \Big\{(\partial_\diamond(p \parallel_\gamma s), \widehat{\nabla}(p' \parallel_{\gamma'} B^{m,n}[\mu, \nu] \parallel_{\gamma'} s)) \mid$$

$$\exists s_1, \vec{\sigma} \in \overline{\mathrm{Alph}}(S)^*.\Big[\Big(\partial_\diamond(p \parallel_\gamma s_1), \widehat{\nabla}(p' \parallel_{\gamma'} B^{m,n}[\mu, \nu] \parallel_{\gamma'} s_1)\Big) \in \Phi \wedge$$

$$\partial_\diamond(p \parallel_\gamma s_1) \xrightarrow{\vec{\sigma}} \!\!\!\!\rightarrow \partial_\diamond(p \parallel_\gamma s)\Big] \bigvee \quad \text{(C6)}$$

$$\exists p_1, p_1', s_1, s', \vec{\sigma} \in \overline{\mathrm{Alph}}(P)^*.\Big[\Big(\partial_\diamond(p_1 \parallel_\gamma s), \widehat{\nabla}(p_1' \parallel_{\gamma'} B^{m,n}[\mu, \nu] \parallel_{\gamma'} s)\Big) \in \Phi \wedge$$

$$\partial_\diamond(p_1' \parallel_\gamma s') \xrightarrow{\vec{\sigma}} \!\!\!\!\rightarrow \partial_\diamond(p' \parallel_\gamma s') \wedge$$

$$\partial_\diamond(p_1 \parallel_\gamma s) \xrightarrow{\vec{\sigma}} \!\!\!\!\rightarrow \partial_\diamond(p \parallel_\gamma s)\Big]\Big\}. \quad \text{(C7)}$$

7 Discussion

In this section, we discuss in the context of desynchronizable closed loop system (Section 5), whether the reordering property (Definition 10) or the diamond property (Definition 11) can be further weakened to attain the same result (Theorem 1).

Informally, the reordering property states that if the receiver (plant or supervisor) is willing to receive an input $?a$ and the sender (supervisor

or plant) can perform a sequence of outputs $f_o(\vec{\mu}).!a$, then the receiver can receive the message a before receiving the sequence of outputs $f_o(\vec{\mu})$ in the SCLS[3]. To prevent such a stringent condition, one may design the following *locking* mechanism in a ACLS by allowing *the execution of an output action at a sender's state, whenever the input bag attached to it is empty.*

Example 4. *Consider the behaviour of a plant p and a supervisor s specified by the following set of equations.*

$$p = ?a.p_1,\ p_1 = !b.p_2,\ p_2 = ?c.p$$
$$s = !a.s_1,\ s_1 = ?b.s_2,\ s_2 = !c.s$$

To observe the effect of a locking mechanism, consider the initial state of the ACLS, $\tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} s))$. The supervisor performs the output $!a$ and transforms to the state s_1. Thus, we infer the following transition by the ACLS.

$$\tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} s)) \xrightarrow{\natural a} \tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p \parallel_{\gamma'} B^{m,n}[\{\natural a\},\varepsilon] \parallel_{\gamma'} s_1))$$

Note that the supervisor in the state s_1 is waiting for the input action $?b$, while the plant in the state p can receive the input action $?a$. Thus, the only possible transition at the state $\tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p \parallel_{\gamma'} B^{m,n}[\{\natural a\},\varepsilon] \parallel_{\gamma'} s_1))$ is the plant removing the content from its input bag. A similar phenomenon can be observed in the other states of the ACLS. Moreover, it can be verified that the SCLS $\partial_H(p \parallel_\gamma s)$ is desynchronizable.

From the above example it is clear that a locking mechanism can be implicitly built into a SCLS in order to avoid the reordering property. However, upon inspection it can be concluded that the reordering property is vacuously satisfied in the above example (there is nothing to reorder). This also suggests that the reordering property can be a suitable candidate for the necessary conditions for desynchronizability.

In comparison to the reordering property, the diamond property can be further weakened. We give an example in which a SCLS satisfies well-posedness and reordering properties, and is still desynchronizable.

Example 5. *Consider the behavior of a plant and a supervisor specified by the following set of equations:*

$$\begin{array}{ll} p = ?a.p_1 + ?c.p_2, & s = !a.s_1 + !c.s_2, \\ p_1 = !b.p,\ p_2 = !d.p & s_1 = ?b.s,\ s_2 = ?d.s \end{array}.$$

[3]Recall the definition of the function f_o from Page 22.

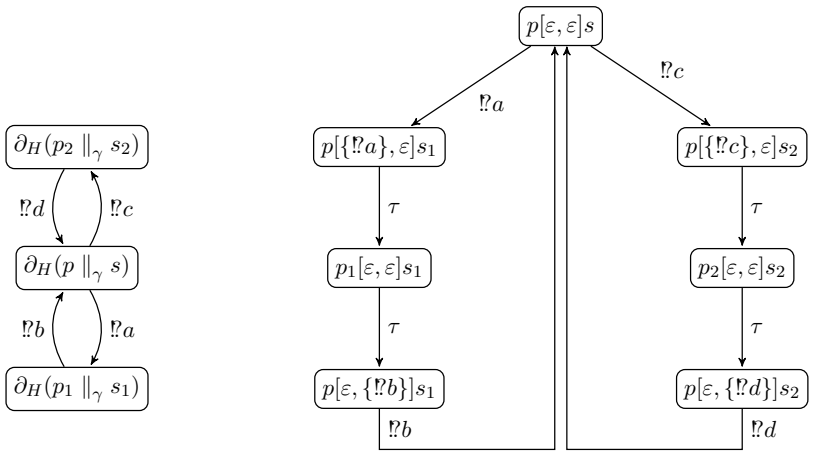


Figure 10: Example 5, $p[\mu,\nu]s = \tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p \parallel_{\gamma'} B^{m,n}[\mu,\nu] \parallel_{\gamma'} s))$.

The transition system of the synchronous and asynchronous closed loop system is depicted in Figure 10. Clearly, the two transition system are branching bisimilar.

Thus, we anticipate that the diamond property (Definition 11) can be further weakened. In particular, if the actions $?a, ?b \in I_p^?$ are enabled at a state q then it may not be necessary for the traces $?a.?b$ and $?b.?a$ to commute. Furthermore, we conjecture that if a SCLS satisfies the well-posedness property (Definition 9), the reordering property (Definition 10), and the weaker form of diamond property, then it is desynchronizable.

8 Conclusions and Future Work

The goal of this paper was to check for desynchronizability of a SCLS without building the corresponding asynchronous system. We presented sufficient conditions for desynchronizability in a process algebraic setting and showed that an asynchronous implementation using bags (of arbitrary size) is a refinement of the SCLS satisfying these conditions. Moreover, we generalized this result for PSCLSs whose alphabets may contain the external actions from the plant and its supervisor in addition to the communicated actions.

The prominent features of our work can be summarized as follows:

- We solve a refinement problem instead of a supervisory control problem, and do not compute a new supervisor in the presence of buffers, as done

in [4, 21]. Our approach is *intended* to be computationally cheaper than the one developed in [4, 21], however this conjecture needs to be verified by analyzing the complexities associated with the conditions presented here. In particular, we conjecture that supervisory control theory always results in SCLSs, which are well-posed (Definition 9), but the other conditions, (Definition 10 and Definition 11), are not likely to be attained so easily.

- We present our conditions for desynchronizability over the components of a SCLS conjointly, in contrast with [9], where the check for the foam rubber wrapper principle on the two components was applied separately. Note the sender domination property from [9] is equivalent to the well posed condition (Definition 9). However, the two approaches are incomparable because in [9] the construction method M3 was studied while in this paper the construction method M1 is studied.

- We use branching bisimulation equivalence instead of the failure equivalence that was adopted in [9]. As a consequence our techniques are applicable to all the weak equivalences in the 'van Glabbeek spectrum' [18] (including failure equivalence). The branching bisimulation is the preferred equivalence in TCP process algebra under the presence of τ action [3]. Furthermore, the conditions (well-posedness and diamond property) given here are similar to the ones mentioned in [9], where sufficient conditions for desynchronizability was given modulo failure equivalence. Thus, we conjecture that achieving desynchronizability for weaker equivalences will not lead to weaker sufficient conditions.

A topic that was not treated in this paper, is whether the conditions we posed are in fact reasonable for industrial applications. This may become clear in the near future, when we study the case studies involved with supervisory control theory in the context of the MULTIFORM project [1] with the language CIF [2]. The authors of CIF are currently developing techniques that will incorporate supervisory control theory and model based engineering into a single framework, thus making it suitable for the design of industrial applications. In particular, the elevator case study and the toy example, which were desynchronizable in [6] using the construction method M1, satisfy our conditions.

The desynchronizability of SCLS present in either decentralized or hierarchical architecture of [20] is not answered completely, although initial results in this directions are presented here by the desynchronization of a

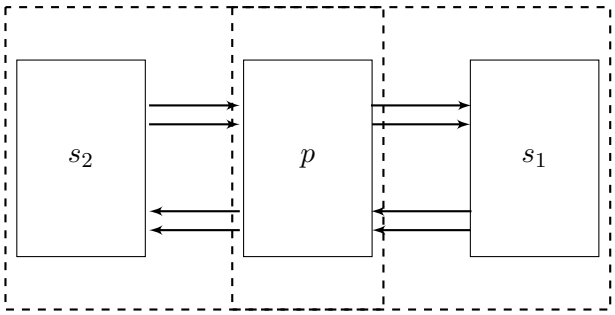


Figure 11: A SCLS in a decentralised architecture [20].

PSCLS (Section 6). Consider the SCLS $\partial_H(p \parallel_\gamma s_1 \parallel_\gamma s_2)$ in a decentralized architecture, which can be further decomposed into two PSCLS's as shown in Figure 11. At this moment, we can only ascertain that these individual PSCLSs are desynchronizable by inspecting the sufficient conditions (Definitions 13, 14, 15, and 16) on them; however, to conclude that the overall desynchronizable closed loop system (i.e. $\partial_H(p \parallel_\gamma s_1 \parallel_\gamma s_2)$) is desynchronizable, more research is required.

Lastly, the research performed in this paper can of course be repeated for different architectures. One might study whether wires or queues can be used instead of bags, or study different abstraction schemes, or try to study the conditions for desynchronizability by focusing on other notions of weak equivalences.

Acknowledgements

The authors would like to thank the reviewers for their valuable critical remarks on the earlier draft of this paper. The authors would also like to thank Jos Baeten, Koos Rooda, Bert van Beek, Rong Su, Jasen Markowski, Damian Nadales, and Mihaly Petreczky for various discussions and clarification about this work.

This work has been performed as part of the "Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems" (MULTIFORM) project, supported by the Seventh Research Framework Programme of the European Commission (Grant agreement number: INFSO-ICT-224249).

References

[1] Integrated multi-formalism tool support for the design of networked embedded control systems : Multiform. http://cms.multiform.bci.tu-dortmund.de/.

[2] D. E. Nadales Agut, D. A. v. Beek, R. R. H. Schiffelers, D. Hendriks, and J. E. Rooda. Abstract syntax and formal semantics of the CIF. Technical report, Eindhoven University of Technology, October 2009.

[3] J. C. M. Baeten, T. Basten, and M. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, 2009.

[4] S. Balemi. *Control of Discrete Event Systems: Theory And Application*. PhD thesis, Swiss Federal Institute of Technology, Automatic Control Laboratory, ETH Zurich, May 1992.

[5] H. Beohar and P. J. L. Cuijpers. Desynchronisability of (partial) synchronous closed loop systems. CS Technical report 10-19, Eindhoven university of technology, 2010. Also, available at http://www.win.tue.nl/~pcuijper/pages/publications.html.

[6] H. Beohar, P. J. L. Cuijpers, and J. C. M. Baeten. Design of asynchronous supervisors. abs/0910.0868, 2009. http://arxiv.org/abs/0910.0868.

[7] H. Beohar and P.J.L. Cuijpers. A theory of desynchronisable closed loop systems. In *Pre-proceedings of Interaction and Concurrency Experience (ICE'10)*, 2010.

[8] M. Fabian and A. Hellgren. PLC-based implementation of supervisory control for discrete event systems. *Proceedings of the 37th IEEE Conference on Decision and Control, 1998*, 3:3305–3310, 1998.

[9] C. Fischer and W. Janssen. Synchronous development of asynchronous systems. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 735–750. Springer-Verlag, 1996.

[10] He Jifeng, M. B. Josephs, and C. A. R. Hoare. A theory of synchrony and asynchrony. In M. Broy and C. B. Jones, editors, *Programming Concepts and Methods*, pages 459–479, 1990.

[11] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International Series in Computer Science, 1985.

[12] H. K. Kapoor and M. B. Josephs. Modelling and verification of delay-insensitive circuits using CCS and the concurrency workbench. volume 89, pages 293–296, 2004.

[13] M. Mousavi, P. Le Guernic, J.-P. Talpin, S. K. Shukla, and T. Basten. Modeling and validating globally asynchronous design in synchronous frameworks. In *Proceedings of the Conference on Design Automation and Test in Europe*, pages 384–389. IEEE Computer Society Press, 2003.

[14] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[15] A. Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2002.

[16] P. Thati and M. Viswanathan. Verification of asynchronous systems with unbounded and unordered message buffers. In *International Workshop on Automated Verification of Infinite State Systems (AVIS)*, 2004.

[17] J.T. Udding. *Classification and Composition of Delay-Insensitive Circuits*. PhD thesis, Eindhoven University of Technology, Eindhoven, 1984.

[18] R. J. v. Glabbeek. *Comparative concurrency semantics and refinement of actions*. Ph.D. thesis, CWI, Amsterdam, 1990.

[19] G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of Logic in Computer Science*, pages 1–148. Oxford University Press, 1995.

[20] W. M. Wonham. Supervisory control of discrete-event systems. Monograph ECE 1636F/1637S, University of Toronto, Dept. of Electrical & Computer Engineering, 2008.

[21] S. Xu and R. Kumar. Asynchronous implementation of synchronous discrete event control. pages 181 –186, May 2008.