Process Algebra with Language Matching

Jos van Wamel *

University of Amsterdam, Department of Mathematics and Computer Science

Kruislaan 403, 1098 SJ Amsterdam, The Netherlands.

E-mail: `vanw@fwi.uva.nl`

Abstract

An axiom system $\mathrm{ACP}^{\tau}_{\mathrm{LM}}$ is presented as a variant of the process algebra ACP (Algebra of Communicating Processes). The acronym $\mathrm{ACP}^{\tau}_{\mathrm{LM}}$ stands for ACP with abstraction, extended with operators and axioms for *language matching*. Language matching is a technique based on trace information for labelling and cutting off process terms that do not match some given trace (or set of traces). It is shown that in combination with the axioms for *action alphabets* interesting results are derivable, the most important of which is the Redundancy Theorem 3.3.6, which roughly states that if no *trace labels* occur in the expression $\partial_H(p_l \parallel q)$, where p_l is a labelled version of some process p, then it holds that $\partial_H(p_l \parallel q) = \partial_H(p \parallel q)$. It is shown that under certain natural conditions a similar result holds when abstraction is applied to p_l and p, respectively.

As an example the Concurrent Alternating Bit Protocol (CABP) is verified. The CABP is a simple communication protocol, which can be recursively specified over the signature of ACP^{τ}, and it can be regarded as a slightly more 'sophisticated' variant of the well-known Alternating Bit Protocol. The verification is carried out using ACP^{τ} with language matching, extended with the abstraction rule CFAR^{b}, and the *conditional alphabet axioms*. The verification is split in two parts so that it is 'modular', and during the verification some basic knowledge about the expected behaviour of the system is used, in order to effectively apply language matching.

1991 Mathematics Subject Classification: 68Q10, 68Q40, 68Q45, 68Q60.
1987 CR Categories: C.2.2, D.1.3, F.1.2, F.4.1.
Key Words & Phrases: Algebraic Computation, Communication Protocols, Process Algebra, Protocol Verification.
Note: The example in this paper is a revised version of [Wam92], which also appeared in the proceedings of the workshop on protocol verification in Mierlo, The Netherlands, 1992.

1 Introduction

The specification and verification of concurrent, communicating processes, such as communication protocols, can often be effectively performed in some algebraic formalism. Here we adhere to the Algebra of Communicating Processes (ACP), and the specific axiom system

*Partly supported by the Foundation for Computer Science in the Netherlands (SION) and by the Netherlands Organisation for Scientific Research (NWO).

we use as our core system is ACP^τ, see e.g. [BK84, BW90], extended with some auxiliary machinery, such as the abstraction rule CFAR^b (Cluster Fair Abstraction Rule), the fixed point rule RSP (Recursive Specification Principle), and the alphabet axioms. The model for this system is that of *rooted branching bisimulation*, where any process specified over the signature of ACP^τ can be represented as a graph.

It often happens in process specification and verification in ACP that a process p has 'redundancies in a context $\partial_H(\_ \parallel q)$'. This means that p has certain subterms (or from a semantic point of view: behaviour) that are (is) eliminated in the context $\partial_H(\_ \parallel q)$, due to encapsulation ∂_H. Of course, redundancy can simply be a result of 'over-specification', i.e. when subterms are specified that are always encapsulated, and which do not communicate with subterms of q. We illustrate redundancy of summands in a given context with the following nice example, borrowed from [Vaa90b].

Example 1.0.1. One of the famous coffee machine specifications from process algebra is given by

$$CM = 30c' \cdot (\mathit{coffee}' + \mathit{tea}') \cdot \mathit{hum} \cdot CM.$$

A typical, very thirsty Dutch user of such a machine is specified by

$$DU = (\mathit{coffee} + 30c \cdot \mathit{coffee}) \cdot \mathit{talk} \cdot DU.$$

Let the communications be defined by $\mathit{coffee} \mid \mathit{coffee}' = \mathit{coffee}^\circ$, $\mathit{tea} \mid \mathit{tea}' = \mathit{tea}^\circ$, and $30c \mid 30c' = 30c^\circ$, and the encapsulation set by $H = \{\mathit{coffee}, \mathit{coffee}', \mathit{tea}, \mathit{tea}', 30c, 30c'\}$.

The interaction between the Dutch user and the coffee machine is described by

$$\partial_H(DU \parallel CM) = 30c^\circ \cdot \mathit{coffee}^\circ \cdot (\mathit{talk} \parallel \mathit{hum}) \cdot \partial_H(DU \parallel CM).$$

It is easy to see that the thrift of the Dutch user makes no sense here, i.e., the first alternative, trying to get the coffee for free, is redundant in the context $\partial_H(\_ \parallel CM)$. If we define a normal thirsty user as

$$TU = 30c \cdot \mathit{coffee} \cdot \mathit{talk} \cdot TU$$

we easily find (using RSP) that $\partial_H(DU \parallel CM) = \partial_H(TU \parallel CM)$. $\square$

In the above example, redundancy is the result of 'over-specification'. Redundancy may also occur in modular specifications and verifications, where parts of a specification are studied separate from their contexts. In general, redundant terms will not have a simple structure. Therefore, a reason for finding possible redundancies at an early stage of a verification is that further analysis of redundant terms is not necessary. Particularly for the 'expansion' of process expressions to linear equations this may be useful.

It is usually the case that some information concerning the behaviour of a process in its context is available in the form of *trace* behaviour. Basic statements of the form "the action a is supposed to happen immediately after b" can be regarded as useful trace information. Our goal is to develop a method for finding and labelling process terms that are possibly redundant in a given context, given some information about the expected behaviour of the total system in the form of a collection of process traces. We refer to our technique as *language matching*. Language matching is a technique for labelling, and cutting off, process terms that do not match some given *language*, or set of traces. Language matching is axiomatised, and we study the interaction of language matching with the operators from ACP^τ. As an example we verify the Concurrent Alternating Bit Protocol as studied earlier in [GV89, KM90, Wam92].

The Concurrent Alternating Bit Protocol is a more complicated variant of the well-known Alternating Bit Protocol (ABP). The ABP is a simple communication protocol and there are many verifications in the literature, because it is often used as *the* test case, either for some algebraic formalism or for some tool for the analysis or verification of concurrent systems. For instance, the ABP is verified in the formalism ACP in [BW90] and in [Vaa90a]. Verifications in related formalisms can be found in e.g. [Mil89]. Such a verification can be made quite easily by straightforward calculations, although it is recently demonstrated how a *really* formal verification leads to a terrific amount of calculations (see [Kam93]). However, the number of states of the ABP is limited, and even without any abstraction from internal actions, the process graph is easy to draw. This is not the case with the CABP, since it suffers from a modest *state space explosion*, which is a combinatory increase of the number of process terms when a parallel composition is being linearised. This state space explosion is due to a lot of internal activity, caused by continuous streams of messages, which are necessary for recovering from error situations. Our goal now is to verify the protocol equationally, and to control this 'explosion' with the help of language matching.

Related work. There are certain parallels between our work and other, previous publications on process operators for ACP, based on trace information. As language matching is particularly interesting in a setting with redundant process behaviour, there are links with [Vaa90b], where a technique is presented for detecting 'redundancy in a context'. Language matching is, in fact, also a technique for doing this, although it differs essentially in one respect. Language matching can detect possible redundancies of a process p in a context $\partial_H(\_ \parallel q)$, using the term $\partial_H(p_l \parallel q)$, where p_l is some labelled, reduced version of p, instead of $\partial_H(p \parallel q)$.

We also refer to [BB88], where operators for *localisation* and *restriction* are studied. Although they do not satisfy the definition for *state operators* in [BB88], our operators for language matching are quite similar. Both the operators for language matching and state operators have some global state, a 'memory', as a parameter. For an interesting application of state operators see [BPW94].

As for the CABP, one can imagine that channels in a protocol can contain more than one datum. Contrary to the ABP, this can happen indeed in the CABP. In [GV89] (and [Gla90]) a specification and verification of a version of the CABP is given with channels that behave as FIFO queues with unbounded capacity that can lose data. The possibility of communicating checksum errors at the receiver is not included, and an extra operator, the *chaining* operator, is added to ACP to make a short and elegant verification of this version of the CABP possible ('chaining' is a technique to easily connect two communicating processes via a data transfer link). Another difference is that no non-determinism in the channels is specified. A second, totally different approach to the protocol was taken in [Hil94], where an analysis of a specification in a timed variant of ACP^τ is made. For timed versions of ACP the reader is referred to e.g. [BB91, BB92].

The version of the protocol verified in this paper originates from [KM90]. An executable version in the specification language PSF (PSF contains a computer readable version of ACP^τ, see e.g. [MV90, MV93]) can be found in [Wam93]. In [KM90] the correctness is proved in *weak bisimulation* semantics. There the protocol is split in two modules and certain properties of the process graphs of the modules are studied. We adopt the idea of modularity as presented in that paper and algebraically verify the correctness of the same specification.

Acknowledgements. Thanks to Joris Hillebrand for Figure 1, and thanks to Jan Bergstra, Willem Jan Fokkink, Henri Korver, Alban Ponse, and Chris Verhoef for helpful comments.

Contents

2 Process algebra

In this section we introduce the axiom system ACP$^\tau$, the Algebra of Communicating Processes with silent step, and all additional rules and axioms that are needed for understanding the main results in this paper.

2.1 ACP$^\tau$

Atomic actions are the building blocks of processes. Therefore, axiom systems in process algebra have a set of atomic actions A, with typical elements $a, b, c, \ldots$, as a parameter. For process variables we use $x, y, z, \ldots$, and for closed process terms we use $p, q, r, \ldots$. An important subsystem of ACP$^\tau$ is BPA$_\delta^\tau$, Basic Process Algebra with deadlock δ and silent step τ. The operators of BPA$_\delta^\tau$ are alternative composition $+$ and sequential composition $\cdot$.

In ACP$^\tau$, communication is modelled by a function $\gamma : A \cup \{\delta, \tau\} \times A \cup \{\delta, \tau\} \longrightarrow A \cup \{\delta\}$, which is commutative and associative. By definition all communications with δ or τ involved are equal to δ. If $\gamma(a, b) = \delta$ then a and b cannot communicate, and if $\gamma(a, b) = c$, then c is the atomic action resulting from the communication between a and b. Concurrency is

constants:	a	for any atomic action $a \in A$
	δ	deadlock
	τ	silent step
unary operators:	∂_H	encapsulation, for any $H \subseteq A$
	τ_I	abstraction, for any $I \subseteq A$
binary operators:	$+$	alternative composition
	$\cdot$	sequential composition
	$\parallel$	parallel composition or merge
	$\mathbin{\|\!\lfloor}$	left merge
	$\mid$	communication merge

Table 1: The signature $\Sigma(\text{ACP}^\tau)$.

described by three operators: the *merge* $\parallel$, the *left merge* $\mathbin{\|\!\lfloor}$ and the *communication merge* $\mid$. The process $p \parallel q$ symbolises the parallel execution of p and q. It 'starts' with an action of either p or q, or with a communication, or *synchronisation*, between p and q. $p \mathbin{\|\!\lfloor} q$ is as $p \parallel q$, but the first action that is performed comes from p. $p \mid q$ is as $p \parallel q$, but the first action is a communication between p and q. Encapsulation operators ∂_H block atomic actions in H by renaming them to δ (note that by axiom A6 δ vanishes in $+$ contexts). Abstraction operators τ_I rename actions in I to τ (note that by axioms B1 and B2 τ vanishes in certain contexts with $\cdot$). Encapsulation is used to enforce communication between processes, and abstraction is used to *hide* certain actions of a process. The signature $\Sigma(\text{ACP}^\tau)$ is summarised in Table 1, and the axioms of ACP$^\tau$ are given in Table 2. The binding of $\cdot$ is taken stronger than that of the other operators, the binding of $+$ is taken weaker than that of the other operators.

The axiom system $\{\text{A1–A7}, \text{B1}, \text{B2}\}$ is referred to as BPA$_\delta^\tau$. The signature $\Sigma(\text{BPA}_\delta^\tau)$ is as $\Sigma(\text{ACP}^\tau)$, but without the renaming operators ∂_H and τ_I, and without the operators for

(A1)	$x + y = y + x$	(CF)	$a \mid b = \gamma(a, b)$
(A2)	$x + (y + z) = (x + y) + z$		
(A3)	$x + x = x$		
(A4)	$(x + y) \cdot z = x \cdot z + y \cdot z$	(CM1)	$x \parallel y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x + x \mid y$
(A5)	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	(CM2)	$a \mathbin{\lfloor\!\lfloor} x = a \cdot x$
(A6)	$x + \delta = x$	(CM3)	$a \cdot x \mathbin{\lfloor\!\lfloor} y = a \cdot (x \parallel y)$
(A7)	$\delta \cdot x = \delta$	(CM4)	$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$
		(CM5)	$a \cdot x \mid b = (a \mid b) \cdot x$
(D1)	$\partial_H(a) = a$ if $a \notin H$	(CM6)	$a \mid b \cdot x = (a \mid b) \cdot x$
(D2)	$\partial_H(a) = \delta$ if $a \in H$	(CM7)	$a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$
(D3)	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	(CM8)	$(x + y) \mid z = x \mid z + y \mid z$
(D4)	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$	(CM9)	$x \mid (y + z) = x \mid y + x \mid z$
(TI1)	$\tau_I(a) = a$ if $a \notin I$		
(TI2)	$\tau_I(a) = \delta$ if $a \in I$		
(TI3)	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	(B1)	$x \cdot \tau = x$
(TI4)	$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$	(B2)	$x \cdot (\tau \cdot (y + z) + y) = x \cdot (y + z)$

Table 2: The axioms of ACP^τ, where $a, b \in A \cup \{\delta, \tau\}$ and $H, I \subseteq A$.

parallelism.

Definition 2.1.1. Let $a \in A$. *Basic terms* over $\Sigma(\mathrm{BPA}_\delta^\tau)$ are terms generated by the following BNF grammar:

$$p ::= a \mid \delta \mid \tau \mid p + p \mid a \cdot p \mid \tau \cdot p.$$

□

Without proof we state some important standard results.

Theorem 2.1.2 (*Elimination*)**.** *If p is a closed term over $\Sigma(\mathrm{ACP}^\tau)$ then there is a basic term q over $\Sigma(\mathrm{BPA}_\delta^\tau)$ such that $\mathrm{ACP}^\tau \vdash p = q$.*

Theorem 2.1.3 (*Conservativity*)**.** *ACP^τ is a conservative extension of BPA_δ^τ, i.e. for all closed terms p, q over $\Sigma(\mathrm{BPA}_\delta^\tau)$ we have: $\mathrm{ACP}^\tau \vdash p = q$ iff $\mathrm{BPA}_\delta^\tau \vdash p = q$.*

2.2 Transition systems and bisimulation semantics

The behaviour of processes is often modelled with *transition rules*, providing an operational semantics as advocated in [Plo81]. With every closed term a *labelled transition system* can be associated, where *bisimulations* are the most commonly used equivalence relations (see e.g. [Par81]).

Definition 2.2.1. A *labelled transition system* $\mathcal{A}$ is a tuple $\langle S_\mathcal{A}, L_\mathcal{A}, \longrightarrow_\mathcal{A}, s_\mathcal{A}\rangle$, where

- $S_{\mathcal{A}}$ is a set of *states*,
- $L_{\mathcal{A}}$ is a set of *labels*,
- $\longrightarrow_{\mathcal{A}}$ is a *transition relation*,
- $s_{\mathcal{A}} \in S_{\mathcal{A}}$ is the *initial state*.

$\square$

Here we consider the set of closed terms over $\Sigma(\mathrm{ACP}^{\tau})$ as the set of states $S_{\mathcal{A}}$, and the set $A \cup \{\tau\}$ as the set of labels $L_{\mathcal{A}}$. The transition relation contains relations

$$\begin{aligned} .\overset{\cdot}{\longrightarrow}. &\subseteq S_{\mathcal{A}} \times L_{\mathcal{A}} \times S_{\mathcal{A}}, \\ .\overset{\cdot}{\longrightarrow}\surd &\subseteq S_{\mathcal{A}} \times L_{\mathcal{A}}, \end{aligned}$$

where $\surd$ (tick) denotes successful termination.

Transition rules are given in a deduction format, which makes it possible to derive transitions for closed process terms. In the tables 3 and 4 transition rules for the operators of $\mathrm{BPA}_{\delta}^{\tau}$ and ACP^{τ} are given.

$a \in A \cup \{\tau\}$	$\dfrac{}{a \xrightarrow{a} \surd}$			
$+$	$\dfrac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'}$	$\dfrac{x \xrightarrow{a} \surd}{x + y \xrightarrow{a} \surd}$	$\dfrac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$	$\dfrac{y \xrightarrow{a} \surd}{x + y \xrightarrow{a} \surd}$
$\cdot$	$\dfrac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y}$	$\dfrac{x \xrightarrow{a} \surd}{x \cdot y \xrightarrow{a} y}$		

Table 3: Transition rules for $\mathrm{BPA}_{\delta}^{\tau}$, where $a \in A \cup \{\tau\}$.

The well-known equivalence relation between transition systems, related with closed terms over $\Sigma(\mathrm{ACP}^{\tau})$ is rooted branching bisimulation (see e.g. [GW91]). So transition systems are studied modulo rooted branching bisimulation.

Let the relation $\Longrightarrow$ be defined by

- $p \Longrightarrow p$,
- if $\exists p''.\ p \xrightarrow{\tau} p'' \Longrightarrow p'$ then $p \Longrightarrow p'$.

The predicate $\surd(p)$ holds iff all execution paths going from p consist of only τ-steps:

- $\surd(\tau)$,

$\mathbin{\|\!\lfloor}$	$\dfrac{x \xrightarrow{a} x'}{x \mathbin{\|\!\lfloor} y \xrightarrow{a} x' \parallel y}$ $\quad \dfrac{x \xrightarrow{a} \surd}{x \mathbin{\|\!\lfloor} y \xrightarrow{a} y}$	
$\parallel$	$\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y}$ $\quad \dfrac{x \xrightarrow{a} \surd}{x \parallel y \xrightarrow{a} y}$	$\dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}$ $\quad \dfrac{y \xrightarrow{a} \surd}{x \parallel y \xrightarrow{a} x}$
$\parallel$ & $\mid$	$\dfrac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y' \quad \gamma(a,b)=c}{x \parallel y \xrightarrow{c} x' \parallel y' \qquad x \mid y \xrightarrow{c} x' \parallel y'}$	$\dfrac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} y' \quad \gamma(a,b)=c}{x \parallel y \xrightarrow{c} y' \qquad x \mid y \xrightarrow{c} y'}$
	$\dfrac{x \xrightarrow{a} x' \quad y \xrightarrow{b} \surd \quad \gamma(a,b)=c}{x \parallel y \xrightarrow{c} x' \qquad x \mid y \xrightarrow{c} x'}$	$\dfrac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} \surd \quad \gamma(a,b)=c}{x \parallel y \xrightarrow{c} \surd \qquad x \mid y \xrightarrow{c} \surd}$
∂_H	$\dfrac{x \xrightarrow{a} x' \quad a \notin H}{\partial_H(x) \xrightarrow{a} \partial_H(x')}$	$\dfrac{x \xrightarrow{a} \surd \quad a \notin H}{\partial_H(x) \xrightarrow{a} \surd}$
τ_I	$\dfrac{x \xrightarrow{a} x' \quad a \notin I}{\tau_I(x) \xrightarrow{a} \tau_I(x')}$	$\dfrac{x \xrightarrow{a} \surd \quad a \notin I}{\tau_I(x) \xrightarrow{a} \surd}$
	$\dfrac{x \xrightarrow{a} x' \quad a \in I}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}$	$\dfrac{x \xrightarrow{a} \surd \quad a \in I}{\tau_I(x) \xrightarrow{\tau} \surd}$

Table 4: Additional transition rules for ACP^τ, where $a, b, c \in A \cup \{\tau\}$ and $H, I \subseteq A$.

- if $\surd(p) \wedge \surd(q)$ then $\surd(p+q)$,
- if $\surd(p) \wedge \surd(q)$ then $\surd(p \cdot q)$.

Definition 2.2.2. Let $a \in A$. A binary, symmetric relation $\mathcal{R} \subseteq \Sigma(\mathrm{ACP}^\tau) \times \Sigma(\mathrm{ACP}^\tau)$ is a *branching bisimulation* iff for all $p, q \in \Sigma(\mathrm{ACP}^\tau)$ $\mathcal{R}$ satisfies

1. if $p\mathcal{R}q$ & $p \xrightarrow{\tau} \surd$ then $\exists s.\ q \Longrightarrow s \xrightarrow{\tau} \surd$ & $p\mathcal{R}s$,
2. if $p\mathcal{R}q$ & $p \xrightarrow{\tau} p'$ then either
   - $p'\mathcal{R}q$, or
   - $\exists q', s.\ q \Longrightarrow s \xrightarrow{\tau} q'$ & $p\mathcal{R}s$ & $p'\mathcal{R}q'$,
3. if $p\mathcal{R}q$ & $p \xrightarrow{a} \surd$ or $p\mathcal{R}q$ & $p \xrightarrow{a} p'$ & $\surd(p')$ then either
   - $\exists s.\ q \Longrightarrow s \xrightarrow{a} \surd$ & $p\mathcal{R}s$, or
   - $\exists q', s.\ q \Longrightarrow s \xrightarrow{a} q'$ & $p\mathcal{R}s$ & $\surd(q')$,
4. if $p\mathcal{R}q$ & $p \xrightarrow{a} p'$ & $\neg\surd(p')$ then $\exists q', s.\ q \Longrightarrow s \xrightarrow{a} q'$ & $p\mathcal{R}s$ & $p'\mathcal{R}q'$.

□

If there is a branching bisimulation which contains the pair (p, q) we call p and q branching bisimilar, notation $p \leftrightarrow_b q$.

The following definition introduces a slightly finer equivalence, by adding some extra restrictions, the so-called *root condition*, to the previous one. This implies that, in addition, all initial steps of two processes must match.

Definition 2.2.3. In order to avoid notational confusion with the previous definition we take $\alpha \in A \cup \{\tau\}$. A binary, symmetric relation $\mathcal{R} \subseteq \Sigma(\mathrm{ACP}^\tau) \times \Sigma(\mathrm{ACP}^\tau)$ is a *rooted branching bisimulation* iff for all $p, q \in \Sigma(\mathrm{ACP}^\tau)$ $\mathcal{R}$ satisfies

1. if $p\mathcal{R}q$ & $p \xrightarrow{\alpha} \surd$ or $p\mathcal{R}q$ & $p \xrightarrow{\alpha} p'$ & $\surd(p')$ then either
   - $q \xrightarrow{\alpha} \surd$, or
   - $\exists q'.\ q \xrightarrow{\alpha} q'$ & $\surd(q')$,
2. if $p\mathcal{R}q$ & $p \xrightarrow{\alpha} p'$ & $\neg\surd(p')$ then $\exists q'.\ q \xrightarrow{\alpha} q'$ & $p' \leftrightarrow_b q'$.

□

If there is a rooted branching bisimulation which contains the pair (p, q) we call p and q rooted branching bisimilar, notation $p \leftrightarrow_{rb} q$. Note that the root condition excludes for instance $(\tau \cdot a + a)\mathcal{R}(\tau \cdot a)$, which does hold in branching bisimulation and *rooted weak (or τ-) bisimulation* semantics (see e.g. [Mil89, BW90]).

We have the following standard results.

Lemma 2.2.4 (*Congruence*). *The relation* $\underline{\leftrightarrow}_{rb}$ *between closed terms over* $\Sigma(\mathrm{ACP}^\tau)$ *is a congruence w.r.t. the operators of* ACP^τ.

Theorem 2.2.5 (*Soundness and Completeness*). *Let* p, q *be closed terms over* $\Sigma(\mathrm{ACP}^\tau)$. *If* $\mathrm{ACP}^\tau \vdash p = q$ *then* $p \underline{\leftrightarrow}_{rb} q$, *and vice versa.*

2.3 Recursion

We introduce processes defined by recursive equations.

Definition 2.3.1. A *recursive specification* $E = \{x = t_x \mid x \in V_E\}$ *over* the signature $\Sigma(\mathrm{ACP}^\tau)$ is a set of equations where V_E is a set of variables and t_x a term over $\Sigma(\mathrm{ACP}^\tau)$, such that the variables in t_x are also in V_E. □

A *solution* of a recursive specification $E = \{x = t_x \mid x \in V_E\}$ is an interpretation of the variables in V_E as processes, such that the equations of E are satisfied. For instance, the recursive specification $\{x = x\}$ has any process as a solution for x, and $\{x = a \cdot x\}$ has the infinite process "a^ω" as a solution for x. The following syntactical restriction on recursive specifications turns out to enforce unique solutions (modulo rooted branching bisimilarity).

Definition 2.3.2. Let t be a term over the signature $\Sigma(\mathrm{ACP}^\tau)$, and $E = \{x = t_x \mid x \in V_E\}$ a recursive specification over $\Sigma(\mathrm{ACP}^\tau)$.

- An occurrence of a variable x in t is *guarded* if t has a subterm of the form $a \cdot M$ with $a \in A$, where x occurs in M, and there are no occurrences of abstraction operators τ_I in M.
- The specification E is *syntactically guarded* iff all occurrences of variables in the terms t_x are guarded.
- The specification E is *guarded* iff there is a syntactically guarded specification $E' = \{x = t'_x \mid x \in V_E\}$ over $\Sigma(\mathrm{ACP}^\tau)$ such that $\mathrm{ACP}^\tau \vdash t_x = t'_x$ for all t_x.

□

Now the signature $\Sigma(\mathrm{ACP}^\tau)_{\mathrm{REC}}$, containing representations of recursively defined processes, is defined as follows.

Definition 2.3.3. The signature $\Sigma(\mathrm{ACP}^\tau)_{\mathrm{REC}}$ is obtained by extending the signature $\Sigma(\mathrm{ACP}^\tau)$ in the following way: for each guarded specification $E = \{x = t_x \mid x \in V_E\}$ over $\Sigma(\mathrm{ACP}^\tau)$ a set of constants $\{<x \mid E>\mid x \in V_E\}$ is added, where the construct $<x \mid E>$ denotes the x-component of a solution of E. □

Some more notations: let $E = \{x = t_x \mid x \in V_E\}$ be a guarded specification over $\Sigma(\mathrm{ACP}^\tau)$, and t some term over $\Sigma(\mathrm{ACP}^\tau)_{\mathrm{REC}}$. Then $<t \mid E>$ denotes the term in which each occurrence of a variable $x \in V_E$ in t is replaced by $<x \mid E>$, e.g., the expression $<a \cdot a \cdot x \mid \{x = a \cdot x\}>$ denotes the term $a \cdot a \cdot <x \mid \{x = a \cdot x\}>$.

For the constants of the form $<x \mid E>$ there are two axioms in Table 5. In these axioms the letter E ranges over guarded specifications. The axiom REC states that the constant

$<x \,|\, E>$ is a solution for the x-component of E, so it expresses that each guarded recursive system has *at least* one solution for each of its (bound) variables. The conditional rule RSP (Recursive Specification Principle) expresses that E has *at most* one solution for each of its variables: whenever one can find processes p_x satisfying the equations of E, notation $E(\vec{p_x})$, then $p_x = <x \,|\, E>$.

(REC)	$<x \,|\, E> = <t_x \,|\, E>$	if $x = t_x \in E$
(RSP)	$\dfrac{E(\vec{p_x})}{p_x = <x \,|\, E>}$	if $x \in V_E$

Table 5: Axioms for guarded recursive specifications.

Finally, a convenient notation is to abbreviate $<x \,|\, E>$ for $x \in V_E$ by X once E is fixed, and to represent E only by its REC instances. We write W_E for the set $\{<x \,|\, E> \mid x \in V_E\}$. The following example shows all notations concerning recursively specified processes, and illustrates the use of REC and RSP.

Example 2.3.4. Consider the guarded recursive specifications $E = \{x = a \cdot x\}$ and $E' = \{y = a \cdot y \cdot b\}$ over $\Sigma(\mathrm{ACP}^\tau)$. So by the convention just introduced, we write $X = a \cdot X$ and $Y = a \cdot Y \cdot b$. With REC and RSP one can prove

$$\mathrm{ACP}^\tau + \mathrm{REC} + \mathrm{RSP} \vdash X = Y$$

in the following way. First note that $X \cdot b = a \cdot X \cdot b$ by REC, so $E(X \cdot b)$ is derivable. Application of RSP yields

$$(i) \quad X \cdot b = X.$$

So $X \cdot b \overset{\mathrm{REC}}{=} a \cdot X \cdot b \overset{(i)}{=} a \cdot X \cdot b \cdot b$, and hence $E'(X \cdot b)$ is derivable. A second application of RSP yields $X \cdot b = Y$. Combining this with (i) gives the desired result. $\square$

The general transition rule by which processes defined by guarded recursive specifications are associated with transitions systems is given in Table 6. The specification $E = \{x = t_x \mid x \in V_E\}$ denotes a guarded recursive specification over the signature $\Sigma(\mathrm{ACP}^\tau)$.

$$\frac{<t_x \,|\, E> \xrightarrow{a} x'}{<x \,|\, E> \xrightarrow{a} x'} \quad \text{if } x = t_x \in E$$

Table 6: Transition rule for guarded recursive specifications, where $a \in A \cup \{\tau\}$.

2.4 Additional axioms and rules for recursively specified processes

There is an additional rule in process algebra that is often used: AIP$^-$. This rule states that if all finite *projections* of two process are equal, and if they do not display *unbounded determinism* they are equal. The term $\pi_n(p)$ represents the projection of process p up to depth n, see Table 7 for the axioms defining projection π_n. For example if $p = a \cdot (b \cdot c + b)$

(PR1)	$\pi_n(a)$	$= a$
(PR2)	$\pi_1(a \cdot x)$	$= a$
(PR3)	$\pi_{n+1}(a \cdot x)$	$= a \cdot \pi_n(x)$
(PR4)	$\pi_n(x + y)$	$= \pi_n(x) + \pi_n(y)$
(PRT1)	$\pi_n(\tau)$	$= \tau$
(PRT2)	$\pi_n(\tau \cdot x)$	$= \tau \cdot \pi_n(x)$

Table 7: The axioms PR for projection, where $a \in A \cup \{\delta\}$ and $n \geq 1$.

then $\pi_2(p) = a \cdot (b + b) = a \cdot b$. A process p displays bounded non-determinism if for each sequence of atomic actions σ the set $\{q \mid p \overset{\sigma}{\twoheadrightarrow} q\}$ is finite ($\twoheadrightarrow$ denotes the transitive closure of $\rightarrow$). Note that guardedly specified processes display bounded non-determinism. AIP$^-$ is therefore often used for motivating the soundness of certain equations, derivable for closed terms, as axioms. For instance, for ACP$^\tau$ there are the axioms for *Standard Concurrency* (SC), e.g. $p \parallel q = q \parallel p$ and $p \parallel (q \parallel r) = (p \parallel q) \parallel r$. In the sequel we also adopt certain formulas that are provable for closed terms as axioms, so that they can be used in a setting with recursion. Soundness is guaranteed by AIP$^-$. It also should be mentioned that AIP$^-$ implies RSP. For details concerning these rules we refer to the standard literature.

Finally, we introduce the Cluster Fair Abstraction Rule, which makes it possible to abstract from cycles of internal actions. This rule was originally conceived in [Vaa86].

Definition 2.4.1. Let E be a guarded specification over $\Sigma(\mathrm{ACP}^\tau)$, and $I \subseteq A$. A set $C \subseteq W_E$ is called a *cluster of* I *in* E if the following condition is satisfied. For all $X \in C$ there are $i_1, \ldots, i_m \in I \cup \{\tau\}$, $X_1, \ldots, X_m \in C$, and $Y_1, \ldots, Y_n \in W_E \setminus C$, with $m \geq 1, n \geq 0$, such that the equation for X is of the form

$$X = \Sigma_{1 \leq k \leq m}\ i_k \cdot X_k + \Sigma_{1 \leq j \leq n}\ Y_j.$$

The terms Y_j are called the *exits* of the cluster. The cluster is called *conservative* if every exit is accessible from every $X \in C$ in the cluster by doing a finite number of steps from $I \cup \{\tau\}$. □

Definition 2.4.2. The Cluster Fair Abstraction Rule is the following rule:

$$\frac{\begin{array}{c} E \text{ is a guarded specification, } I \subseteq A \\ X \in C \text{, where } C \text{ is a finite conservative cluster of } I \text{ in } E \\ U \text{ is the set of exits from } C \end{array}}{\tau \cdot \tau_I(X) = \tau \cdot \Sigma_{Y \in U}\ \tau_I(Y)} \quad \mathrm{CFAR}^b$$

□

The following set of axioms and rules is often used for verifications.

Theorem 2.4.3 (*Soundness*). $\mathrm{ACP}^\tau + \mathrm{SC} + \mathrm{REC} + \mathrm{RSP} + \mathrm{CFAR}^b$ *is sound w.r.t. rooted branching bisimulation.*

Of course, the definitions for recursion, conservative clusters, and consequently CFAR^b, are easily adapted to other axiom systems than ACP^τ.

2.5 Axioms for action alphabets

In the following sections we reason intensively on the *action alphabets* of processes. We take the defining axioms from [BBK87, BW90], listed in Table 8. The axioms AB6 and AB7 are

(AB1)	$\alpha(\nu)$	$= \emptyset$	for $\nu \in \{\delta, \tau\}$
(AB2)	$\alpha(a)$	$= \{a\}$	
(AB3)	$\alpha(\tau \cdot x)$	$= \alpha(x)$	
(AB4)	$\alpha(a \cdot x)$	$= \{a\} \cup \alpha(x)$	
(AB5)	$\alpha(x + y)$	$= \alpha(x) \cup \alpha(y)$	
(AB6)	$\alpha(x)$	$= \cup_{n \geq 1} \alpha(\pi_n(x))$	
(AB7)	$\alpha(\tau_I(x))$	$= \alpha(x) \setminus I$	

Table 8: The axioms AB for action alphabets, where $a \in A$ and $I \subseteq A$.

often necessary for handling recursively specified processes.

Table 9 contains the *conditional alphabet axioms*. They can be derived for closed terms over $\Sigma(\mathrm{ACP}^\tau)$.

(CA1)	$\alpha(x) \mid (\alpha(y) \cap H) \subseteq H$	$\Rightarrow$	$\partial_H(x \parallel y) = \partial_H(x \parallel \partial_H(y))$
(CA2)	$\alpha(x) \mid (\alpha(y) \cap I) = \emptyset$	$\Rightarrow$	$\tau_I(x \parallel y) = \tau_I(x \parallel \tau_I(y))$
(CA3)	$\alpha(x) \cap H = \emptyset$	$\Rightarrow$	$\partial_H(x) = x$
(CA4)	$\alpha(x) \cap I = \emptyset$	$\Rightarrow$	$\tau_I(x) = x$

Table 9: The conditional alphabet axioms CA, where $H, I \subseteq A$.

3 Language matching

In this section we introduce operators for *language matching*, and study their characteristic algebraic properties.

3.1 Basic definitions for traces and languages

We first give some simple definitions concerning *traces*, or *strings* of atomic actions.

Definition 3.1.1. Let A be a set of atomic actions. The set of traces $A^\star$ is inductively defined as the set of strings over A, i.e., if $\star : A \times A^\star \to A^\star$ then if $a \in A$ & $\sigma \in A^\star$ then $a \star \sigma \in A^\star$. We also have $\lambda \in A^\star$, where λ is the empty trace. According to convention, $a \star \sigma$ is written as $a\sigma$. □

Definition 3.1.2. Let $a \in A$ and $\sigma \in A^\star$. The set *acts* of actions in a trace is defined by $acts(\lambda) \overset{\text{def}}{=} \emptyset$ and $acts(a\sigma) \overset{\text{def}}{=} \{a\} \cup acts(\sigma)$. □

Definition 3.1.3. Let $A^\star$ be the set of strings over a set A of atomic actions, with an empty element λ, and let $a \in A$, and $\sigma, \rho \in A^\star$. Trace concatenation $\star : A^\star \times A^\star \to A^\star$ is defined by $\lambda \star \sigma \overset{\text{def}}{=} \sigma$, and $(a\rho) \star \sigma \overset{\text{def}}{=} a(\rho \star \sigma)$. According to convention, we write $\sigma\rho$ for $\sigma \star \rho$. □

Note that $\sigma\lambda = \sigma$ for all $\sigma \in A^\star$. The following definition involves arbitrary sets of traces, or sets of strings over some set of actions A, also known as *languages* over $A^\star$.

Definition 3.1.4. For $a \in A$, $\sigma \in A^\star$, and $L \subseteq A^\star$ we define the functions $acts : 2^{A^\star} \to A^\star$, $\star : A^\star \times 2^{A^\star} \to 2^{A^\star}$, and $\partial L/\partial a : 2^{A^\star} \to 2^{A^\star}$ as follows:

$$
\begin{array}{lcl}
acts(L) & \overset{\text{def}}{=} & \bigcup_{\rho \in L} \; acts(\rho), \\
\sigma \star L & \overset{\text{def}}{=} & \{\sigma\rho \mid \rho \in L\}, \\
\partial L/\partial a & \overset{\text{def}}{=} & \{\rho \mid a\rho \in L\}.
\end{array}
$$

We write σL for $\sigma \star L$. □

In a setting with recursion infinite behaviour of processes can be modelled, so that we may need the possibility to reason about traces of arbitrary length. Therefore we introduce an operator for iteration on languages.

Definition 3.1.5. Let $L \subseteq A^\star$, where $A^\star$ is the set of strings over a set of actions A. The *closure* or *Kleene star* of a language L, denoted by L^*, is inductively defined by $\{\lambda\} \in L^*$, and if $\sigma \in L$ & $\rho \in L^*$ then $\sigma\rho \in L^*$. □

From now on we use the symbol Z for denoting a language over some set of traces $A^\star$, possibly containing the operator *.

3.2 An operator for language matching

We use the operator Δ_Z for symbolising language matching with language Z. The notation Δ_Z is chosen in accordance with the notation for the operators ∇_Z for *restriction* from [BB88]. The axioms for language matching with *trace label* $r \in A$ are given in Table 10. A process trace is cut off and labelled with r if it does not match a given language. So if an action is not 'expected', the subsequent term is replaced by r (see axiom LM4). The action r behaves in all respects as a 'normal' atomic action, and may occur anywhere in process terms. The operators Δ_Z are linear in the sense of [Ver92].

$$
\begin{array}{lll}
\text{(LM1)} & \Delta_Z(\nu) = \nu \qquad \text{for } \nu \in \{\delta, \tau\} \\
\text{(LM2)} & \Delta_Z(x+y) = \Delta_Z(x) + \Delta_Z(y) \\
\text{(LM3)} & \Delta_Z(\tau \cdot x) = \tau \cdot \Delta_Z(x) \\
\text{(LM4)} & \Delta_Z(a \cdot x) = \begin{cases} a \cdot \Delta_Z(x) & \text{if } a \notin acts(Z) \\ a \cdot \Delta_{\partial Z/\partial a}(x) & \text{if } \partial Z/\partial a \neq \emptyset \\ a \cdot r & \text{otherwise} \end{cases}
\end{array}
$$

Table 10: The axioms LM for language matching with trace label $r \in A$, where $a \in A$ and $Z \subseteq A^\star$.

Since $a = a \cdot \tau$, language matching applied to atomic actions can be treated with axiom LM4. We see for instance that $\Delta_{\{ba\lambda\}}(a) = \Delta_{\{ba\lambda\}}(a \cdot \tau) = a \cdot r$. Below, we give a detailed example of language matching.

Example 3.2.1. Let $a, b, c \in A$. If $p = a \cdot (b \cdot c + b) + b \cdot (a + c)$ and $Z = \{ab\lambda\}$ we have that

$$
\begin{array}{lcl}
\Delta_Z(p) & = & \Delta_{\{ab\lambda\}}(a \cdot (b \cdot c + b) + b \cdot (a + c)) \\
& \stackrel{\text{LM2}}{=} & \Delta_{\{ab\lambda\}}(a \cdot (b \cdot c + b)) + \Delta_{\{ab\lambda\}}(b \cdot (a + c)) \\
& \stackrel{\text{LM4}}{=} & a \cdot \Delta_{\{b\lambda\}}(b \cdot c + b) + b \cdot r \\
& \stackrel{\text{LM2}}{=} & a \cdot (\Delta_{\{b\lambda\}}(b \cdot c) + \Delta_{\{b\lambda\}}(b)) + b \cdot r \\
& \stackrel{\text{LM4,B1}}{=} & a \cdot (b \cdot \Delta_{\{\lambda\}}(c \cdot \tau) + \Delta_{\{b\lambda\}}(b \cdot \tau)) + b \cdot r \\
& \stackrel{\text{LM4}}{=} & a \cdot (b \cdot c \cdot \Delta_{\{\lambda\}}(\tau) + b \cdot \Delta_{\{\lambda\}}(\tau)) + b \cdot r \\
& \stackrel{\text{LM1,B1}}{=} & a \cdot (b \cdot c + b) + b \cdot r.
\end{array}
$$

□

We refer to the axiom system ACP^τ extended with the axioms LM for language matching as ACP^τ_{LM}, which of course also includes the definitions concerning traces. With $\Sigma(ACP^\tau_{LM})$ the signature of ACP^τ_{LM} is denoted.

Identities between closed terms over $\Sigma(ACP^\tau_{LM})$ can often be proved by structural induction. The Elimination Theorem below implies that we then only have to consider basic terms over $\Sigma(BPA^\tau_\delta)$. Here and in the following subsection we find several proofs by simultaneous induction on the structure of process terms and languages. For example, if we want to prove a formula for certain fixed p and $Z \subseteq A^\star$, we can assume for induction hypothesis that it holds for all p' and $Z' \subseteq A^\star$, where $|p'| < |p|$. With $|p|$ we denote some syntactic notion of size, say, the number of symbols occurring in p.

In Table 11 transition rules for the Δ_Z operators are given. We have the following standard results.

$$\frac{x \xrightarrow{\tau} x'}{\Delta_Z(x) \xrightarrow{\tau} \Delta_Z(x')} \qquad \frac{x \xrightarrow{\tau} \surd}{\Delta_Z(x) \xrightarrow{\tau} \surd}$$

$$\frac{x \xrightarrow{a} x' \quad a \notin acts(Z)}{\Delta_Z(x) \xrightarrow{a} \Delta_Z(x')} \qquad \frac{x \xrightarrow{a} \surd \quad a \notin acts(Z)}{\Delta_Z(x) \xrightarrow{a} \surd}$$

$$\frac{x \xrightarrow{a} x' \quad \partial Z/\partial a \neq \emptyset}{\Delta_Z(x) \xrightarrow{a} \Delta_{\partial Z/\partial a}(x')} \qquad \frac{x \xrightarrow{a} \surd \quad \partial Z/\partial a \neq \emptyset}{\Delta_Z(x) \xrightarrow{a} \surd}$$

$$\frac{x \xrightarrow{a} x' \quad a \in acts(Z) \quad \partial Z/\partial a = \emptyset}{\Delta_Z(x) \xrightarrow{a} r} \qquad \frac{x \xrightarrow{a} \surd \quad a \in acts(Z) \quad \partial Z/\partial a = \emptyset}{\Delta_Z(x) \xrightarrow{a} r}$$

Table 11: Transition rules for language matching, where $a, r \in A$ and $Z \subseteq A^\star$.

Theorem 3.2.2 (*Elimination*). *If p is a closed term over $\Sigma(\mathrm{ACP}^\tau_{\mathrm{LM}})$ then there is a closed term q over $\Sigma(\mathrm{ACP}^\tau)$, such that $\mathrm{ACP}^\tau_{\mathrm{LM}} \vdash p = q$.*

Proof. See Appendix. $\square$

Theorem 3.2.3 (*Conservativity*). *$\mathrm{ACP}^\tau_{\mathrm{LM}}$ is a conservative extension of ACP^τ, i.e. for all closed terms p, q over $\Sigma(\mathrm{ACP}^\tau)$ we have: $\mathrm{ACP}^\tau_{\mathrm{LM}} \vdash p = q$ iff $\mathrm{ACP}^\tau \vdash p = q$.*

Proof. The transition rules for language matching in Table 11 form an *operationally conservative* extension of the rules for ACP^τ. According to a general result in [Ver93], $\mathrm{ACP}^\tau_{\mathrm{LM}}$ is a conservative extension of ACP^τ. $\square$

Lemma 3.2.4 (*Congruence*). *The relation $\leftrightarrow_{rb}$ between closed terms over $\Sigma(\mathrm{ACP}^\tau_{\mathrm{LM}})$ is a congruence w.r.t. the operators of $\mathrm{ACP}^\tau_{\mathrm{LM}}$.*

Proof. See Appendix. $\square$

Theorem 3.2.5 (*Soundness*). *Let p, q be closed terms over $\Sigma(\mathrm{ACP}^\tau_{\mathrm{LM}})$. If $\mathrm{ACP}^\tau_{\mathrm{LM}} \vdash p = q$ then $p \leftrightarrow_{rb} q$.*

Proof. The relation $\leftrightarrow_{rb}$ between the closed terms over $\Sigma(\mathrm{ACP}^\tau_{\mathrm{LM}})$ is a congruence, and therefore respects the inference rules for equality $(=)$. So soundness only has to be shown for the axioms LM, which is straightforward. $\square$

Theorem 3.2.6 (*Completeness*). *Let p, q be closed terms over $\Sigma(\mathrm{ACP}^{\tau}_{\mathrm{LM}})$. If $p \leftrightarrow_{rb} q$ then $\mathrm{ACP}^{\tau}_{\mathrm{LM}} \vdash p = q$.*

Proof. The transition rules for language matching in Table 11 form an operationally conservative extension of the rules for ACP^{τ}. Moreover we have the Elimination Theorem 3.2.2 and the completeness of ACP^{τ} w.r.t. $\leftrightarrow_{rb}$. According to a result in [Ver93] we then also have completeness of $\mathrm{ACP}^{\tau}_{\mathrm{LM}}$. $\square$

3.3 Some properties of language matching

Some special and useful properties of the operators Δ_Z for language matching are studied. We use the symbol $\circ$ for function composition.

Lemma 3.3.1. *Let $V, Z \subseteq A^{\star}$. For all closed terms p over $\Sigma(\mathrm{ACP}^{\tau}_{\mathrm{LM}})$ we have:*

1. $\Delta_{\emptyset}(p) = p$,
2. $\Delta_Z \circ \Delta_Z(p) = \Delta_Z(p)$,
3. $\Delta_V \circ \Delta_Z(p) = \Delta_Z \circ \Delta_V(p)$.

Proof. By simultaneous induction on the structure of p, V and Z. $\square$

We go in a few steps to our main result.

Lemma 3.3.2. *Let $Z \subseteq A^{\star}$, $H \subseteq A$. For all closed terms p over $\Sigma(\mathrm{ACP}^{\tau}_{\mathrm{LM}})$ we have:*

$$\textit{if} \quad r \notin H \quad \textit{and} \quad r \notin \alpha \circ \partial_H \circ \Delta_Z(p) \quad \textit{then} \quad \partial_H \circ \Delta_Z(p) = \partial_H(p).$$

Proof. See Appendix. $\square$

Lemma 3.3.3. *Let $Z \subseteq A^{\star}$, $H \subseteq A$. For all closed terms p, q over $\Sigma(\mathrm{ACP}^{\tau}_{\mathrm{LM}})$ we have:*

$$\textit{if} \quad r \notin H \quad \textit{and} \quad r \notin \alpha \circ \partial_H(\Delta_Z(p) \parallel q) \quad \textit{then} \quad \partial_H(\Delta_Z(p) \parallel q) = \partial_H(p \parallel q).$$

Proof. See Appendix. $\square$

Example 3.3.4. Let $a, a', a^{\circ}, b, b', b^{\circ}, c \in A$, $H = \{a, a', b, b'\}$, $\gamma(a, a') = a^{\circ}$, and $\gamma(b, b') = b^{\circ}$. If $p = a \cdot (b \cdot c + b) + b \cdot (a + c)$, $q = a' \cdot b'$, and $Z = \{ab\lambda\}$ we have that

$$\Delta_Z(p) \overset{3.2.1}{=} a \cdot (b \cdot c + b) + b \cdot r.$$

It is easy to prove that $\partial_H(\Delta_Z(p) \parallel q) = a^{\circ} \cdot (b^{\circ} \cdot c + b^{\circ})$, so according to Lemma 3.3.3 we may conclude that $\partial_H(\Delta_Z(p) \parallel q) = \partial_H(p \parallel q)$. We see that the summand $b \cdot (a + c)$ of p is redundant in the context $\partial_H(\_ \parallel q)$. $\square$

Below, operators for abstraction are incorporated.

Lemma 3.3.5. *Let $Z \subseteq A^{\star}$, $I \subseteq A$. For all closed terms p over $\Sigma(\mathrm{ACP}^{\tau}_{\mathrm{LM}})$ we have:*

$$\textit{if} \quad r \notin I \quad \textit{and} \quad \textit{acts}(Z) \cap I = \emptyset \quad \textit{then} \quad \tau_I \circ \Delta_Z(p) = \Delta_Z \circ \tau_I(p).$$

Proof. See Appendix. □

Theorem 3.3.6 (*Redundancy***).** *Let $Z \subseteq A^\star$, $H, I \subseteq A$. For all closed terms p, q over $\Sigma(\mathrm{ACP}^\tau_{\mathrm{LM}})$ we have:*

$$\text{(RED)} \quad \textit{if } r \notin H \cup I \quad \textit{and} \quad \textit{acts}(Z) \cap I = \emptyset \quad \textit{and} \quad r \notin \alpha \circ \partial_H(\tau_I \circ \Delta_Z(p) \parallel q) \quad \textit{then}$$

$$\partial_H(\tau_I \circ \Delta_Z(p) \parallel q) = \partial_H(\tau_I(p) \parallel q).$$

Proof. Easy, use the two preceding lemmas. □

The first condition in the Redundancy Theorem can always be satisfied by choosing the trace label r properly. The third condition is crucial: if r does not occur in the action alphabet of a system with language matching applied to one of its components, we immediately have a specification of the total system without operators for language matching. Note that Lemma 3.3.3 is an instance of the Redundancy Theorem (take $I = \emptyset$).

We study the interaction between language matching and the generalised alternative composition Σ more closely.

Lemma 3.3.7. *Let D be a finite set of data, $\{\sigma(e) \mid e \in D\} \subseteq A^\star$. If $a(d) \notin \textit{acts}(\sigma(e))$ for all $e \neq d$, and $\partial\{\sigma(e) \mid e \in D\}/\partial a(d) \neq \emptyset$ for all $d \in D$, then:*

$$\Delta_{\{\sigma(e)|e\in D\}^*} \circ \Sigma_{d\in D}\ a(d) \cdot x = \Sigma_{d \in D}\ a(d) \cdot \Delta_{\partial\sigma(d)\{\sigma(e)|e\in D\}^*/\partial a(d)}(x).$$

Proof. We have for all $d \neq e$ that $a(d) \notin \textit{acts}(\sigma(e))$, so $\partial\sigma(e)\{\sigma(f) \mid f \in D\}^*/\partial a(d) = \emptyset$. Since in general $\mathcal{L}^* = \bigcup_{\sigma\in\mathcal{L}}\ \sigma\mathcal{L}^* \cup \{\lambda\}$, and $\partial\{\sigma(e) \mid e \in D\}/\partial a(d) \neq \emptyset$ it must be that

$$\begin{array}{lll} (i)\ \ \partial\{\sigma(e) \mid e \in D\}^*/\partial a(d) & = & \bigcup_{e\in D}\ \partial\sigma(e)\{\sigma(f) \mid f \in D\}^*/\partial a(d) \\ & = & \partial\sigma(d)\{\sigma(e) \mid e \in D\}^*/\partial a(d). \end{array}$$

Now the proof goes as follows:

$$\begin{array}{lll} \Delta_{\{\sigma(e)|e\in D\}^*} \circ \Sigma_{d\in D}\ a(d) \cdot x & \overset{\mathrm{LM2}}{=} & \Sigma_{d\in D}\ \Delta_{\{\sigma(e)|e\in D\}^*}(a(d) \cdot x) \\ & \overset{\mathrm{LM5}}{=} & \Sigma_{d\in D}\ a(d) \cdot \Delta_{\partial\{\sigma(e)|e\in D\}^*/\partial a(d)}(x) \\ & \overset{i}{=} & \Sigma_{d\in D}\ a(d) \cdot \Delta_{\partial\sigma(d)\{\sigma(e)|e\in D\}^*/\partial a(d)}(x). \end{array}$$

Application of LM2 is allowed since D is finite. □

Remark 3.3.8. Let the expression $tr(p)$ denote the *prefix closed* set of traces of p (see [BB88, BW90]). The relation between the tr operator and the operators for language matching is maybe best illustrated by the following interesting result. It can be proved that

$$\textit{if } r \notin \alpha(p) \quad \textit{then} \quad tr \circ \tau_{\{r\}} \circ \Delta_Z(p) \subseteq tr(p).$$

From the viewpoint of trace semantics, a process with language matching applied to it can execute only a part of its original behaviour. If the language to be matched becomes larger, the process behaviour generally becomes more restricted. □

4 A verification of the concurrent alternating bit protocol

4.1 Description and specification of the CABP

4.1.1 Description

The Concurrent Alternating Bit Protocol can be graphically represented as in Figure 1, where the various communication ports and processes are depicted.

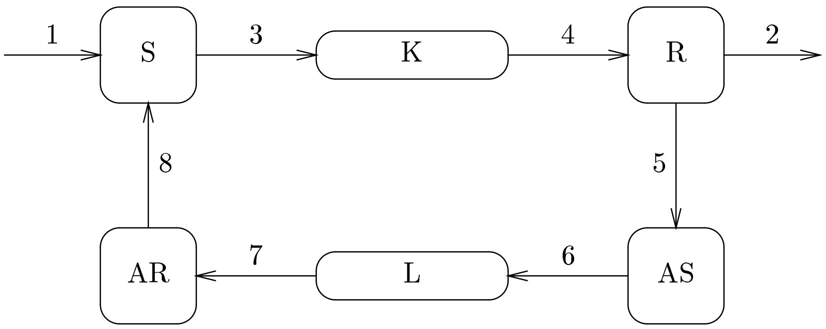

Figure 1: Ports and processes of the CABP.

The following data are used in the specification of the CABP:

- D : arbitrary finite set of data elements.
- $Bit = \{0, 1\}$: set of bits used for internal control, and with an 'inverse' function $inv : Bit \to Bit$.
- $PS = \{1, 2, 3, 4, 5, 6, 7, 8\}$: set of numbers of ports through which data is sent or read. The input port has number 1 and the output port has number 2.
- $DS = D \cup (D \times Bit) \cup Bit \cup \{ce, ac, ae\}$: set of all transferable data in the protocol. The special constants ce, ac and ae are used to denote a checksum error introduced by channel K, an acknowledgement, and an acknowledgement error introduced by channel L, respectively.

The sending part is split in two components. One component, the *sender* S, does the reading of data elements $d \in D$ via the input port, and it sends a continuous stream of *frames* ($\in D \times Bit$) into channel K. A frame is a simple structure composed of a data element d and a control bit $b \in Bit$. The other component, the *acknowledgement receiver* AR, handles incoming acknowledgements. If the acknowledgement receiver receives the expected control bit from channel L, an acknowledgement ac is sent to the sender, the sender reads a new data element at port 1, and sends it in a frame with the control bit inverted into channel K. Again in a continuous stream. The receiving part is also split in two components. One component, the *receiver* R, receives frames from channel K and takes care of the output.

When the expected frame arrives, the data element is sent via the output, port 2, and an acknowledgement is sent to the other component, the *acknowledgement sender AS*. The acknowledgement sender fires a continuous stream of bits into channel L. As soon as it receives an acknowledgement from the receiver the bit is inverted. To make the protocol work correctly all components except the channels are initialised with a certain value of the control bit b.

The sending of continuous streams of data through the channels is a method to overcome the possible loss of data, because the channels are supposed to be 'lossy'. A disadvantage for the verification is that these (independent) streams cause a modest 'state space explosion'. In the case that the channels are replaced by queues of infinite capacity this even leads to an infinite number of states. This is the essential difference between the CABP and the ABP.

4.1.2 Specification

Below, the protocol is specified over $\Sigma(\mathrm{ACP}^\tau)_{\mathrm{REC}}$ in a PSF-like fashion.
Atomic actions:
Let $p \in PS \setminus \{1,2\}$ and $dat \in DS$. We define the following atomic actions:

i, j : internal actions of the channels
$r_p(dat)$: read dat at port p
$s_p(dat)$: send dat at port p
$c_p(dat)$: communicate dat at port p
$r_1(dat)$: read dat at input port
$s_2(dat)$: send dat at output port

Communications:

$$s_p(dat) \mid r_p(dat) = c_p(dat) \quad \text{where} \quad p \in PS \setminus \{1,2\} \quad \text{and} \quad dat \in DS$$

Sets:

$$H = \{s_p(dat), r_p(dat) \mid p \in PS \setminus \{1,2\}, dat \in DS\}$$
$$I = \{c_p(dat) \mid p \in PS \setminus \{1,2\}, dat \in DS\} \cup \{i, j\}$$

Processes:

$$\begin{aligned}
S &= RM(0) \\
RM(b) &= \Sigma_{d \in D}\ r_1(d) \cdot SF(d,b) \\
SF(d,b) &= s_3(d,b) \cdot SF(d,b) + r_8(ac) \cdot RM(inv(b))
\end{aligned}$$

$$\begin{aligned}
K &= \Sigma_{d \in D, b \in Bit}\ r_3(d,b) \cdot K(d,b) \\
K(d,b) &= (i \cdot s_4(d,b) + i \cdot s_4(ce) + i) \cdot K
\end{aligned}$$

$$\begin{aligned}
R &= RF(0) \\
RF(b) &= \Sigma_{d \in D}\ r_4(d,b) \cdot s_2(d) \cdot s_5(ac) \cdot RF(inv(b)) + \\
&\quad\ (\Sigma_{d \in D}\ r_4(d, inv(b)) + r_4(ce)) \cdot RF(b)
\end{aligned}$$

$$\begin{aligned}
AS &= AS(1) \\
AS(b) &= r_5(ac) \cdot AS(inv(b)) + s_6(b) \cdot AS(b)
\end{aligned}$$

$L = \Sigma_{b \in Bit}\ r_6(b) \cdot L(b)$
$L(b) = (j \cdot s_7(b) + j \cdot s_7(ae) + j) \cdot L$

$AR = AR(0)$
$AR(b) = (r_7(ae) + r_7(inv(b))) \cdot AR(b) + r_7(b) \cdot s_8(ac) \cdot AR(inv(b))$

$Cabp = \partial_H(S \parallel K \parallel R \parallel AS \parallel L \parallel AR)$

Externally, the protocol has to behave as a one element buffer. Therefore a verification of the CABP is a proof of the following theorem.

Theorem 4.1.1 (*Requirement*). *It holds that*
$\mathrm{ACP}^{\tau}_{\mathrm{LM}} + \mathrm{SC} + \mathrm{REC} + \mathrm{RSP} + \mathrm{CFAR}^{b} + \mathrm{AB} + \mathrm{CA} + \mathrm{RED} \vdash$

$$\boxed{\tau \cdot \tau_I(Cabp) = \tau \cdot \Sigma_{d \in D}\ r_1(d) \cdot s_2(d) \cdot \tau_I(Cabp)}$$

The τ-actions indicate that the protocol can start with some sequence of internal actions.
The remainder of this paper is dedicated to a proof of the above theorem.

4.1.3 Modular specification

As announced in the introduction, the specification of the external behaviour of the CABP is split in two modules. We follow the approach of [KM90]. One module (P) consists of the components S, K and R, and a second module (Q) consists of the components AS, L and AR. The idea behind this division is quite simple. In the CABP there are two constant, independent streams of messages. Both cause a lot of internal activity. One stream, the stream of frames, flows from S to R. This stream is contained in module P. The other stream, the stream of acknowledgements from AS to AR, is contained in module Q.
Modules:

$H_P = \{r_p(dat), s_p(dat) \mid p \in \{3,4\}, dat \in DS\}$
$I_P = \{c_p(dat) \mid p \in \{3,4\}, dat \in DS\} \cup \{i\}$
$P = \partial_{H_P}(S \parallel K \parallel R)$

$H_Q = \{r_p(dat), s_p(dat) \mid p \in \{6,7\}, dat \in DS\}$
$I_Q = \{c_p(dat) \mid p \in \{6,7\}, dat \in DS\} \cup \{j\}$
$Q = \partial_{H_Q}(AS \parallel L \parallel AR)$

We can now give a modular specification of the external behaviour of the CABP. The sets H and I are defined in Section 4.1.2.

Lemma 4.1.2. *A modular specification of the external behaviour of the CABP is*

$$\tau_I(Cabp) = \tau_I \circ \partial_H(\tau_{I_P}(P) \parallel \tau_{I_Q}(Q)).$$

Proof. Straightforward; use the conditional alphabet axioms CA, see also [KM90]. □

Some simple manipulations show that we also have

$$\tau \cdot \tau_I(Cabp) = \tau \cdot \tau_I \circ \partial_H(\tau_{I_P}(P) \parallel \tau \cdot \tau_{I_Q}(Q)).$$

4.2 Some remarks on the verification

The specifications of the modules P and Q can be linearised by a simple *head-tail expansion* strategy to equational specifications E_P and E_Q respectively. The right hand sides of the equations of these specifications are then in head normal form. When the specifications E_P and E_Q are put in parallel and expanded, an equational specification for the CABP can be derived that consists of an enormous number of equations in head normal form. It can be shown that in total at least a few hundred equations have to be derived this way, for both possible values of the control bit b.

Therefore, it is desirable to reduce the specifications E_P and E_Q as much as possible before they are interleaved. Of course we have the τ-axioms B1 and B2, and the Cluster Fair Abstraction Rule at our disposal for abstracting from internal activity. Another significant reduction, which can already be obtained *before* abstraction is applied, is made with the technique of language matching.

Both the modules P and Q have some expected behaviour, e.g. certain sequences of atomic actions are not supposed to happen when the root processes P and Q are put in parallel. However, there is always the possibility that the modules themselves display redundant behaviour, which is only removed when they are put in parallel, and encapsulation is applied. This is why we use language matching. If language matching is applied, all possibly redundant terms in the specifications for P and Q are labelled and not expanded any further. The Redundancy Theorem roughly tells us that if no trace labels occur in the parallel composition $\partial_H(P_l \parallel Q_l)$, where P_l and Q_l are labelled versions of $\tau_{I_P}(P)$ and $\tau_{I_Q}(Q)$, then $\partial_H(P_l \parallel Q_l) = \partial_H(\tau_{I_P}(P) \parallel \tau_{I_Q}(Q))$. We already have that the trace label r satisfies $r \notin H \cup I$.

In the verification we use the following strategy:

1. A linear, guarded specification E_P with root process $P_0(0)$ is given. This process is equal to a labelled version of the root process P of module P (Lemma 4.3.1).

2. Abstraction is applied, and with the axioms of ACP$^\tau$, together with CFAR^b, a reduced specification $E_{P_{red}}$ is obtained. $E_{P_{red}}$ has $\tau_{I_P}(P_0(0))$ as root process.

3. As 1, for module Q (Lemma 4.4.1).

4. As 2, for module Q. We find that the root of $E_{Q_{red}}$ is in a cluster, therefore it is preceded by τ: $\tau \cdot \tau_{I_Q}(Q_0(0))$. (This is a direct consequence of CFAR^b; we work in rooted branching bisimulation semantics.)

5. A linear, guarded specification can be obtained for $\tau_I \circ \partial_H(\tau_{I_P}(P_0(0)) \parallel \tau_{I_Q}(Q_0(0)))$. This specification is equal to a given specification E_{Cabp} with root process $C_0(0)$. With the help of the Redundancy Theorem 3.3.6 it is proved that $\tau \cdot C_0(0) = \tau \cdot \tau_I(Cabp)$ (Lemma 4.5.1).

6. Finally E_{Cabp} is reduced, and it is proved that $\tau \cdot C_0(0)$, and thus $\tau \cdot \tau_I(Cabp)$, satisfies the requirement in Theorem 4.1.1.

4.3 Linearisation and reduction of module P

4.3.1 An equational specification of module P

The specification of module P can be rewritten to a corresponding, linear equational specification E_P by applying a head-tail expansion strategy. Language matching is applied, since in the context $\partial_H(\_ \parallel \tau_{I_Q}(Q))$ certain atomic actions are not expected to happen after certain other actions: there is an expected infinite sequence $(\Sigma_{d\in D}\ r_1(d)\cdot s_2(d)\cdot c_5(ac)\cdot c_8(ac))^\omega$ of visible actions. For instance, we do not expect $c_8(ac)$ to happen immediately before $s_2(d)$. Therefore, we label those summands in the specification for module P that are in traces that do not match the language containing the traces

$$\sigma_P(d) \stackrel{\text{def}}{=} r_1(d)s_2(d)s_5(ac)r_8(ac)\lambda,$$

where $d \in D$, with r. If our intuitions are correct then in the verification of the CABP the guards in front of the processes r are encapsulated and removed.

Consider the following guarded specification E_P.

$$
\begin{aligned}
E_P{=}\{\ & P_0(b) = \Sigma_{d\in D}\ r_1(d)\cdot P_1(d,b),\\
& P_1(d,b) = r_8(ac)\cdot r + c_3(d,b)\cdot P_2(d,b),\\
& P_2(d,b) = r_8(ac)\cdot r + i\cdot P_1(d,b) + i\cdot P_3(d,b) + i\cdot P_4(d,b),\\
& P_3(d,b) = r_8(ac)\cdot r + c_4(d,b)\cdot P_5(d,b),\\
& P_4(d,b) = r_8(ac)\cdot r + c_4(ce)\cdot P_1(d,b),\\
& P_5(d,b) = r_8(ac)\cdot r + s_2(d)\cdot P_6(d,b) + c_3(d,b)\cdot P_7(d,b),\\
& P_6(d,b) = r_8(ac)\cdot r + s_5(ac)\cdot P_8(d,b) + c_3(d,b)\cdot P_9(d,b),\\
& P_7(d,b) = r_8(ac)\cdot r + s_2(d)\cdot P_9(d,b) + i\cdot P_5(d,b) + i\cdot P_{10}(d,b) + i\cdot P_{11}(d,b),\\
& P_8(d,b) = r_8(ac)\cdot P_0(inv(b)) + c_3(d,b)\cdot P_{12}(d,b),\\
& P_9(d,b) = r_8(ac)\cdot r + s_5(ac)\cdot P_{12}(d,b) + i\cdot P_6(d,b) + i\cdot P_{13}(d,b) + i\cdot P_{14}(d,b),\\
& P_{10}(d,b) = r_8(ac)\cdot r + s_2(d)\cdot P_{13}(d,b),\\
& P_{11}(d,b) = r_8(ac)\cdot r + s_2(d)\cdot P_{14}(d,b),\\
& P_{12}(d,b) = r_8(ac)\cdot P_{15}(d,b) + i\cdot P_8(d,b) + i\cdot P_{16}(d,b) + i\cdot P_{17}(d,b),\\
& P_{13}(d,b) = r_8(ac)\cdot r + s_5(ac)\cdot P_{16}(d,b),\\
& P_{14}(d,b) = r_8(ac)\cdot r + s_5(ac)\cdot P_{17}(d,b),\\
& P_{15}(d,b) = \Sigma_{e\in D}\ r_1(e)\cdot P_{18}(d,e,b) + i\cdot P_0(inv(b)) + i\cdot P_{19}(d,b) + i\cdot P_{20}(b),\\
& P_{16}(d,b) = r_8(ac)\cdot P_{19}(d,b) + c_4(d,b)\cdot P_8(d,b),\\
& P_{17}(d,b) = r_8(ac)\cdot P_{20}(b) + c_4(ce)\cdot P_8(d,b),\\
& P_{18}(d,e,b) = r_8(ac)\cdot r + i\cdot P_{21}(d,e,b) + i\cdot P_{22}(e,b) + i\cdot P_1(e,inv(b)),\\
& P_{19}(d,b) = \Sigma_{e\in D}\ r_1(e)\cdot P_{21}(d,e,b) + c_4(d,b)\cdot P_0(inv(b)),\\
& P_{20}(b) = \Sigma_{d\in D}\ r_1(d)\cdot P_{22}(d,b) + c_4(ce)\cdot P_0(inv(b)),\\
& P_{21}(d,e,b) = r_8(ac)\cdot r + c_4(d,b)\cdot P_1(e,inv(b)),\\
& P_{22}(d,b) = r_8(ac)\cdot r + c_4(ce)\cdot P_1(d,inv(b))\}
\end{aligned}
$$

Lemma 4.3.1. *The root process P of the specification of module P, matched with the language $\{\sigma_P(e) \mid e\in D\}^*$, with trace label r, is equal to the root process of E_P:*

$$\Delta_{\{\sigma_P(e)|e\in D\}^*}(P) = P_0(0).$$

Proof. Expand P with the axioms of ACP$^\tau$ and apply language matching $\Delta_{\{\sigma_P(e)|e\in D\}^*}$. The result follows easily using RSP; it has to be verified that $\Delta_{\{\sigma_P(e)|e\in D\}^*}(P)$ and $P_0(0)$ satisfy the same system of recursion equations. (Under the list of process terms below, we show the first steps of the expansion as an example.) □

Not only $P_0(0)$, but every state of E_P is equivalent to a state of the labelled specification for module P. Below, the processes or states that occur in the equational specification of module P with trace labels are given (for those who feel like checking the proof).

Notations:

$$
\begin{aligned}
[\,\_\,] &\stackrel{\text{def}}{=} \partial_{H_P}(\,\_\,)\\
\Delta_0 &\stackrel{\text{def}}{=} \Delta_{\{\sigma_P(e)|e\in D\}^*}\\
\Delta_{1_d} &\stackrel{\text{def}}{=} \Delta_{s_2(d)s_5(ac)r_8(ac)\{\sigma_P(e)|e\in D\}^*}\\
\Delta_2 &\stackrel{\text{def}}{=} \Delta_{s_5(ac)r_8(ac)\{\sigma_P(e)|e\in D\}^*}\\
\Delta_3 &\stackrel{\text{def}}{=} \Delta_{r_8(ac)\{\sigma_P(e)|e\in D\}^*}
\end{aligned}
$$

$$
\begin{aligned}
&\Delta_0 \circ [RM(b) \parallel K \parallel RF(b)] = P_0(b)\\
&\Delta_{1_d} \circ [SF(d,b) \parallel K \parallel RF(b)] = P_1(d,b)\\
&\Delta_{1_d} \circ [SF(d,b) \parallel K(d,b) \parallel RF(b)] = P_2(d,b)\\
&\Delta_{1_d} \circ [SF(d,b) \parallel s_4(d,b)\cdot K \parallel RF(b)] = P_3(d,b)\\
&\Delta_{1_d} \circ [SF(d,b) \parallel s_4(ce)\cdot K \parallel RF(b)] = P_4(d,b)\\
&\Delta_{1_d} \circ [SF(d,b) \parallel K \parallel s_2(d)\cdot s_5(ac)\cdot RF(inv(b))] = P_5(d,b)\\
&\Delta_2 \circ [SF(d,b) \parallel K \parallel s_5(ac)\cdot RF(inv(b))] = P_6(d,b)\\
&\Delta_{1_d} \circ [SF(d,b) \parallel K(d,b) \parallel s_2(d)\cdot s_5(ac)\cdot RF(inv(b))] = P_7(d,b)\\
&\Delta_3 \circ [SF(d,b) \parallel K \parallel RF(inv(b))] = P_8(d,b)\\
&\Delta_2 \circ [SF(d,b) \parallel K(d,b) \parallel s_5(ac)\cdot RF(inv(b))] = P_9(d,b)\\
&\Delta_{1_d} \circ [SF(d,b) \parallel s_4(d,b)\cdot K \parallel s_2(d)\cdot s_5(ac)\cdot RF(inv(b))] = P_{10}(d,b)\\
&\Delta_{1_d} \circ [SF(d,b) \parallel s_4(ce)\cdot K \parallel s_2(d)\cdot s_5(ac)\cdot RF(inv(b))] = P_{11}(d,b)\\
&\Delta_3 \circ [SF(d,b) \parallel K(d,b) \parallel RF(inv(b))] = P_{12}(d,b)\\
&\Delta_2 \circ [SF(d,b) \parallel s_4(d,b)\cdot K \parallel s_5(ac)\cdot RF(inv(b))] = P_{13}(d,b)\\
&\Delta_2 \circ [SF(d,b) \parallel s_4(ce)\cdot K \parallel s_5(ac)\cdot RF(inv(b))] = P_{14}(d,b)\\
&\Delta_0 \circ [RM(inv(b)) \parallel K(d,b) \parallel RF(inv(b))] = P_{15}(d,b)\\
&\Delta_3 \circ [SF(d,b) \parallel s_4(d,b)\cdot K \parallel RF(inv(b))] = P_{16}(d,b)\\
&\Delta_3 \circ [SF(d,b) \parallel s_4(ce)\cdot K \parallel RF(inv(b))] = P_{17}(d,b)\\
&\Delta_{1_e} \circ [SF(e,inv(b)) \parallel K(d,b) \parallel RF(inv(b))] = P_{18}(d,e,b)\\
&\Delta_0 \circ [RM(inv(b)) \parallel s_4(d,b)\cdot K \parallel RF(inv(b))] = P_{19}(d,b)\\
&\Delta_0 \circ [RM(inv(b)) \parallel s_4(ce)\cdot K \parallel RF(inv(b))] = P_{20}(b)\\
&\Delta_{1_e} \circ [SF(e,inv(b)) \parallel s_4(d,b)\cdot K \parallel RF(inv(b))] = P_{21}(d,e,b)\\
&\Delta_{1_d} \circ [SF(d,inv(b)) \parallel s_4(ce)\cdot K \parallel RF(inv(b))] = P_{22}(d,b)
\end{aligned}
$$

Below we derive the first two equations of E_P. The interaction of language matching with the sum construct is a crucial step in the expansion.

$$\Delta_{\{\sigma_P(e)|e\in D\}^*}(P) = \Delta_{\{\sigma_P(e)|e\in D\}^*} \circ [S \parallel K \parallel R] = \Delta_{\{\sigma_P(e)|e\in D\}^*} \circ [RM(0) \parallel K \parallel RF(0)]$$

$$\begin{array}{cl}
\overset{\text{Exp.}}{=} & \Delta_{\{\sigma_P(e)|e\in D\}^*} \circ \Sigma_{d\in D}\, r_1(d) \cdot [SF(d,0) \parallel K \parallel RF(0)] \\
\overset{3.3.7}{=} & \Sigma_{d\in D}\, \Delta_{\sigma_P(d)\{\sigma_P(e)|e\in D\}^*}(r_1(d) \cdot [SF(d,0) \parallel K \parallel RF(0)]) \\
\overset{\text{LM4}}{=} & \Sigma_{d\in D}\, r_1(d) \cdot \Delta_{s_2(d)s_5(ac)r_8(ac)\{\sigma_P(e)|e\in D\}^*} \circ [SF(d,0) \parallel K \parallel RF(0)] \\
= & \Sigma_{d\in D}\, r_1(d) \cdot \Delta_{1_d} \circ [SF(d,0) \parallel K \parallel RF(0)]
\end{array}$$

$$\Delta_{1_d} \circ [SF(d,0) \parallel K \parallel RF(0)]$$

$$\begin{array}{cl}
\overset{\text{Exp.}}{=} & \Delta_{1_d}(r_8(ac) \cdot [RM(1) \parallel K \parallel RF(0)] + c_3(d,b) \cdot [SF(d,0) \parallel K(d,0) \parallel RF(0)]) \\
\overset{\text{LM2}}{=} & \Delta_{1_d}(r_8(ac) \cdot [RM(1) \parallel K \parallel RF(0)]) + \Delta_{1_d}(c_3(d,b) \cdot [SF(d,0) \parallel K(d,0) \parallel RF(0)]) \\
\overset{\text{LM4}}{=} & r_8(ac) \cdot r + c_3(d,b) \cdot \Delta_{1_d} \circ [SF(d,0) \parallel K(d,0) \parallel RF(0)]
\end{array}$$

4.3.2 Reduction of module P by abstraction

All equations in E_P hold for both values of b, so the total labelled equational specification of P consists of 46 recursive equations. The next step in the verification is abstraction from all internal actions of module P, and reduction with the help of the τ-axioms of ACP$^\tau$ and CFAR^b. In E_P there are 4 conservative clusters, and as the root of a cluster any cluster variable can be chosen. As a consequence, all variables of a cluster satisfy the same recursion equation. The right-hand side of a cluster equation contains only exits, and no variables that are part of this cluster.

Cluster 1:

$$\begin{array}{l}
\tau \cdot \tau_{I_P}(P_1(d,b)) = \tau \cdot \tau_{I_P}(P_2(d,b)) = \tau \cdot \tau_{I_P}(P_4(d,b)) \\
= \tau \cdot (r_8(ac) \cdot r + \tau \cdot \tau_{I_P}(P_3(d,b)))
\end{array}$$

Cluster 2:

$$\begin{array}{l}
\tau \cdot \tau_{I_P}(P_5(d,b)) = \tau \cdot \tau_{I_P}(P_7(d,b)) \\
= \tau \cdot (r_8(ac) \cdot r + s_2(d) \cdot \tau_{I_P}(P_6(d,b)) + s_2(d) \cdot \tau_{I_P}(P_9(d,b)) + \tau \cdot \tau_{I_P}(P_{10}(d,b)) + \\
\qquad \tau \cdot \tau_{I_P}(P_{11}(d,b)))
\end{array}$$

Cluster 3:

$$\begin{array}{l}
\tau \cdot \tau_{I_P}(P_6(d,b)) = \tau \cdot \tau_{I_P}(P_9(d,b)) \\
= \tau \cdot (r_8(ac) \cdot r + s_5(ac) \cdot \tau_{I_P}(P_8(d,b)) + s_5(ac) \cdot \tau_{I_P}(P_{12}(d,b)) + \tau \cdot \tau_{I_P}(P_{13}(d,b)) + \\
\qquad \tau \cdot \tau_{I_P}(P_{14}(d,b)))
\end{array}$$

Cluster 4:

$$\begin{array}{l}
\tau \cdot \tau_{I_P}(P_8(d,b)) = \tau \cdot \tau_{I_P}(P_{12}(d,b)) = \tau \cdot \tau_{I_P}(P_{16}(d,b)) = \tau \cdot \tau_{I_P}(P_{17}(d,b)) \\
= \tau \cdot (r_8(ac) \cdot \tau_{I_P}(P_0(inv(b))) + r_8(ac) \cdot \tau_{I_P}(P_{15}(d,b)) + r_8(ac) \cdot \tau_{I_P}(P_{19}(d,b)) + \\
\qquad r_8(ac) \cdot \tau_{I_P}(P_{20}(b)))
\end{array}$$

Since $\tau \cdot \tau_{I_P}(P_{16}(d,b)) = \tau \cdot \tau_{I_P}(P_{17}(d,b))$, we can easily derive from E_P that $\tau_{I_P}(P_{13}(d,b)) = \tau_{I_P}(P_{14}(d,b))$, and therefore also $\tau_{I_P}(P_{10}(d,b)) = \tau_{I_P}(P_{11}(d,b))$.

$\tau_{I_P}(P_6(d,b))$ and $\tau_{I_P}(P_9(d,b))$ are both in cluster 3.

We remove $\tau_{I_P}(P_9(d,b))$ and $\tau_{I_P}(P_{11}(d,b))$ from the exits of cluster 2:

$$\begin{aligned}
&\tau \cdot \tau_{I_P}(P_5(d,b)) = \tau \cdot \tau_{I_P}(P_7(d,b)) \\
&= \tau \cdot (r_8(ac) \cdot r + s_2(d) \cdot \tau_{I_P}(P_6(d,b)) + \tau \cdot \tau_{I_P}(P_{10}(d,b)))
\end{aligned}$$

$\tau_{I_P}(P_8(d,b))$ and $\tau_{I_P}(P_{12}(d,b))$ are both in cluster 4.
We remove $\tau_{I_P}(P_{12}(d,b))$ and $\tau_{I_P}(P_{14}(d,b))$ from the exits of cluster 3:

$$\begin{aligned}
&\tau \cdot \tau_{I_P}(P_6(d,b)) = \tau \cdot \tau_{I_P}(P_9(d,b)) \\
&= \tau \cdot (r_8(ac) \cdot r + s_5(ac) \cdot \tau_{I_P}(P_8(d,b)) + \tau \cdot \tau_{I_P}(P_{13}(d,b)))
\end{aligned}$$

Even more processes in E_P can be identified: $\tau_{I_P}(P_{21}(d,e,b)) = \tau_{I_P}(P_{22}(e,b))$, and therefore also $\tau_{I_P}(P_{19}(d,b)) = \tau_{I_P}(P_{20}(b))$.
Now we easily find that

$$\begin{aligned}
&\tau \cdot \tau_{I_P}(P_{18}(d,e,b)) = \tau \cdot (r_8(ac) \cdot r + \tau \cdot \tau_{I_P}(P_{22}(e,b)) + \tau \cdot \tau_{I_P}(P_1(e, inv(b)))) \\
&= \tau \cdot (\tau \cdot \tau_{I_P}(P_{22}(e,b)) + \tau_{I_P}(P_{22}(e,b))) \\
&\overset{\text{B2}}{=} \tau \cdot \tau_{I_P}(P_{22}(e,b)) = \tau \cdot \tau_{I_P}(P_{21}(d,e,b))
\end{aligned}$$

Likewise, we also find that $\tau \cdot \tau_{I_P}(P_{15}(d,b)) = \tau \cdot \tau_{I_P}(P_{20}(b))$, so

$$\begin{aligned}
&\tau \cdot \tau_{I_P}(P_{15}(d,b)) = \tau \cdot \tau_{I_P}(P_{19}(d,b)) = \tau \cdot \tau_{I_P}(P_{20}(b)) \\
&= \tau \cdot (\Sigma_{d \in D}\ r_1(d) \cdot \tau_{I_P}(P_{22}(d,b)) + \tau \cdot \tau_{I_P}(P_0(inv(b))))
\end{aligned}$$

We remove $\tau_{I_P}(P_{19}(d,b))$ and $\tau_{I_P}(P_{20}(b))$ from the exits of cluster 4:

$$\begin{aligned}
&\tau \cdot \tau_{I_P}(P_8(d,b)) = \tau \cdot \tau_{I_P}(P_{12}(d,b)) = \tau \cdot \tau_{I_P}(P_{16}(d,b)) = \tau \cdot \tau_{I_P}(P_{17}(d,b)) \\
&= \tau \cdot (r_8(ac) \cdot \tau_{I_P}(P_0(inv(b))) + r_8(ac) \cdot \tau_{I_P}(P_{15}(d,b)))
\end{aligned}$$

We continue with the reduction from cluster 1:

$$\begin{aligned}
&\tau \cdot \tau_{I_P}(P_1(d,b)) = \tau \cdot \tau_{I_P}(P_2(d,b)) = \tau \cdot \tau_{I_P}(P_4(d,b)) \\
&= \tau \cdot (r_8(ac) \cdot r + \tau \cdot \tau_{I_P}(P_3(d,b))) \\
&= \tau \cdot (r_8(ac) \cdot r + \tau \cdot (r_8(ac) \cdot r + \tau \cdot \tau_{I_P}(P_5(d,b)))) \\
&\overset{\text{B2}}{=} \tau \cdot (r_8(ac) \cdot r + \tau \cdot \tau_{I_P}(P_5(d,b))) = \tau \cdot \tau_{I_P}(P_3(d,b))
\end{aligned}$$

Cluster 3:

$$\begin{aligned}
&\tau \cdot \tau_{I_P}(P_6(d,b)) = \tau \cdot \tau_{I_P}(P_9(d,b)) \\
&= \tau \cdot (r_8(ac) \cdot r + s_5(ac) \cdot \tau_{I_P}(P_8(d,b)) + \tau \cdot \tau_{I_P}(P_{13}(d,b))) \\
&= \tau \cdot (r_8(ac) \cdot r + s_5(ac) \cdot \tau_{I_P}(P_8(d,b)) + \tau \cdot (r_8(ac) \cdot r + s_5(ac) \cdot \tau_{I_P}(P_{16}(d,b)))) \\
&= \tau \cdot (r_8(ac) \cdot r + s_5(ac) \cdot \tau_{I_P}(P_8(d,b)) + \tau \cdot (r_8(ac) \cdot r + s_5(ac) \cdot \tau_{I_P}(P_8(d,b)))) \\
&\overset{\text{B2}}{=} \tau \cdot (r_8(ac) \cdot r + s_5(ac) \cdot \tau_{I_P}(P_8(d,b)))) = \tau \cdot \tau_{I_P}(P_{13}(d,b)) = \tau \cdot \tau_{I_P}(P_{14}(d,b))
\end{aligned}$$

Cluster 2:

$$\begin{aligned}
&\tau \cdot \tau_{I_P}(P_5(d,b)) = \tau \cdot \tau_{I_P}(P_7(d,b)) \\
&= \tau \cdot (r_8(ac) \cdot r + s_2(d) \cdot \tau_{I_P}(P_6(d,b)) + \tau \cdot \tau_{I_P}(P_{10}(d,b))) \\
&= \tau \cdot (\tau_{I_P}(P_{10}(d,b)) + \tau \cdot \tau_{I_P}(P_{10}(d,b))) \overset{\text{B2}}{=} \tau \cdot \tau_{I_P}(P_{10}(d,b)) = \tau \cdot \tau_{I_P}(P_{11}(d,b))
\end{aligned}$$

Finally, we see that the equations for the clusters 1 and 2 can be combined:

$$
\begin{aligned}
&\tau \cdot \tau_{I_P}(P_1(d,b)) = \tau \cdot \tau_{I_P}(P_2(d,b)) = \tau \cdot \tau_{I_P}(P_3(d,b) = \tau \cdot \tau_{I_P}(P_4(d,b)) \\
&= \tau \cdot (r_8(ac) \cdot r + \tau \cdot \tau_{I_P}(P_5(d,b))) \\
&= \tau \cdot (r_8(ac) \cdot r + \tau \cdot \tau_{I_P}(P_{10}(d,b))) \\
&= \tau \cdot (r_8(ac) \cdot r + \tau \cdot (r_8(ac) \cdot r + s_2(d) \cdot \tau_{I_P}(P_{13}(d,b)))) \\
&\stackrel{\text{B2}}{=} \tau \cdot (r_8(ac) \cdot r + s_2(d) \cdot \tau_{I_P}(P_{13}(d,b))) \\
&= \tau \cdot \tau_{I_P}(P_{10}(d,b)) = \tau \cdot \tau_{I_P}(P_{11}(d,b)) = \tau \cdot \tau_{I_P}(P_5(d,b)) = \tau \cdot \tau_{I_P}(P_7(d,b))
\end{aligned}
$$

Now we can formulate a reduced specification $E_{P_{red}}$ for module P with 12 equations. Only 6 equations have to be written down. Nearly all cluster variables are removed by substitution from E_P.

$$
\begin{aligned}
E_{P_{red}} = \{ \; &\tau_{I_P}(P_0(b)) = \Sigma_{d\in D}\; r_1(d) \cdot \tau_{I_P}(P_1(d,b)), \\
&\tau \cdot \tau_{I_P}(P_1(d,b)) = \tau \cdot (r_8(ac) \cdot r + s_2(d) \cdot \tau_{I_P}(P_{13}(d,b))), \\
&\tau_{I_P}(P_{13}(d,b)) = r_8(ac) \cdot r + s_5(ac) \cdot \tau_{I_P}(P_8(d,b)), \\
&\tau \cdot \tau_{I_P}(P_8(d,b)) = \tau \cdot (r_8(ac) \cdot \tau_{I_P}(P_0(inv(b))) + r_8(ac) \cdot \tau_{I_P}(P_{15}(d,b))), \\
&\tau \cdot \tau_{I_P}(P_{15}(d,b)) = \tau \cdot (\Sigma_{d\in D}\; r_1(d) \cdot \tau_{I_P}(P_{22}(d,b)) + \tau \cdot \tau_{I_P}(P_0(inv(b)))), \\
&\tau_{I_P}(P_{22}(d,b)) = r_8(ac) \cdot r + \tau \cdot \tau_{I_P}(P_1(d, inv(b)))\}
\end{aligned}
$$

4.4 Linearisation and reduction of module Q

4.4.1 An equational specification of module Q

The specification of module Q can be rewritten to a linear equational specification E_Q by applying a head-tail expansion strategy, just as we did with module P in the previous subsection.

Again we apply language matching, since in the context $\partial_H(\tau_{I_P}(P) \parallel \_)$ certain atomic actions are not expected to happen after certain other actions. There is of course again the expected infinite sequence $(\Sigma_{d\in D}\; r_1(d) \cdot s_2(d) \cdot c_5(ac) \cdot c_8(ac))^\omega$ of visible actions. So we label those summands in the specification for module Q that are in traces that do not match the language containing

$$\sigma_Q \stackrel{\text{def}}{=} r_5(ac)s_8(ac)\lambda$$

with r.

Consider the following guarded specification:

$$
\begin{aligned}
E_Q = \{ \; &Q_0(b) = r_5(ac) \cdot Q_1(b) + c_6(inv(b)) \cdot Q_2(b), \\
&Q_1(b) = r_5(ac) \cdot r + c_6(b) \cdot Q_3(b), \\
&Q_2(b) = r_5(ac) \cdot Q_4(b) + j \cdot Q_0(b) + j \cdot Q_5(b) + j \cdot Q_6(b), \\
&Q_3(b) = r_5(ac) \cdot r + j \cdot Q_1(b) + j \cdot Q_7(b) + j \cdot Q_8(b), \\
&Q_4(b) = r_5(ac) \cdot r + j \cdot Q_1(b) + j \cdot Q_8(b) + j \cdot Q_9(b), \\
&Q_5(b) = r_5(ac) \cdot Q_9(b) + c_7(inv(b)) \cdot Q_0(b), \\
&Q_6(b) = r_5(ac) \cdot Q_8(b) + c_7(ae) \cdot Q_0(b), \\
&Q_7(b) = r_5(ac) \cdot r + c_7(b) \cdot Q_{10}(b), \\
&Q_8(b) = r_5(ac) \cdot r + c_7(ae) \cdot Q_1(b), \\
&Q_9(b) = r_5(ac) \cdot r + c_7(inv(b)) \cdot Q_1(b), \\
&Q_{10}(b) = r_5(ac) \cdot r + c_6(b) \cdot Q_{11}(b) + s_8(ac) \cdot Q_0(inv(b)), \\
&Q_{11}(b) = r_5(ac) \cdot r + j \cdot Q_{10}(b) + j \cdot Q_{12}(b) + j \cdot Q_{13}(b) + s_8(ac) \cdot Q_2(inv(b)),
\end{aligned}
$$

$$Q_{12}(b) = r_5(ac) \cdot r + s_8(ac) \cdot Q_5(inv(b)),$$
$$Q_{13}(b) = r_5(ac) \cdot r + s_8(ac) \cdot Q_6(inv(b))\}$$

Lemma 4.4.1. *The root process Q of the specification of module Q, matched with the language $\{\sigma_Q\}^*$, and with trace label r, is equal to the root process of E_Q:*

$$\Delta_{\{\sigma_Q\}^*}(Q) = Q_0(0).$$

Proof. Expand Q with the axioms of ACP$^\tau$ and apply language matching $\Delta_{\{\sigma_Q\}^*}$. The result follows easily using RSP. □

Every state of E_Q is equivalent to a state of the labelled specification of module Q.
Notation:

$$[\_] \stackrel{\text{def}}{=} \partial_{H_Q}(\_)$$

$$\begin{aligned}
&\Delta_{\{\sigma_Q\}^*} \circ [AS(inv(b)) \parallel L \parallel AR(b)] = Q_0(b)\\
&\Delta_{s_8(ac)\{\sigma_Q\}^*} \circ [AS(b) \parallel L \parallel AR(b)] = Q_1(b)\\
&\Delta_{\{\sigma_Q\}^*} \circ [AS(inv(b)) \parallel L(inv(b)) \parallel AR(b)] = Q_2(b)\\
&\Delta_{s_8(ac)\{\sigma_Q\}^*} \circ [AS(b) \parallel L(b) \parallel AR(b)] = Q_3(b)\\
&\Delta_{s_8(ac)\{\sigma_Q\}^*} \circ [AS(b) \parallel L(inv(b)) \parallel AR(b)] = Q_4(b)\\
&\Delta_{\{\sigma_Q\}^*} \circ [AS(inv(b)) \parallel s_7(inv(b)) \cdot L \parallel AR(b)] = Q_5(b)\\
&\Delta_{\{\sigma_Q\}^*} \circ [AS(inv(b)) \parallel s_7(ae) \cdot L \parallel AR(b)] = Q_6(b)\\
&\Delta_{s_8(ac)\{\sigma_Q\}^*} \circ [AS(b) \parallel s_7(b) \cdot L \parallel AR(b)] = Q_7(b)\\
&\Delta_{s_8(ac)\{\sigma_Q\}^*} \circ [AS(b) \parallel s_7(ae) \cdot L \parallel AR(b)] = Q_8(b)\\
&\Delta_{s_8(ac)\{\sigma_Q\}^*} \circ [AS(b) \parallel s_7(inv(b)) \cdot L \parallel AR(b)] = Q_9(b)\\
&\Delta_{s_8(ac)\{\sigma_Q\}^*} \circ [AS(b) \parallel L \parallel s_8(ac) \cdot AR(inv(b))] = Q_{10}(b)\\
&\Delta_{s_8(ac)\{\sigma_Q\}^*} \circ [AS(b) \parallel L(b) \parallel s_8(ac) \cdot AR(inv(b))] = Q_{11}(b)\\
&\Delta_{s_8(ac)\{\sigma_Q\}^*} \circ [AS(b) \parallel s_7(b) \cdot L \parallel s_8(ac) \cdot AR(inv(b))] = Q_{12}(b)\\
&\Delta_{s_8(ac)\{\sigma_Q\}^*} \circ [AS(b) \parallel s_7(ae) \cdot L \parallel s_8(ac) \cdot AR(inv(b))] = Q_{13}(b)
\end{aligned}$$

4.4.2 Reduction of module Q by abstraction

The labelled equational specification of module Q now consists of 28 recursive equations. Because of the binary variable b, only 14 of them are necessary to specify module Q. The next step is to apply abstraction and to reduce this specification with the help of the τ-axioms of ACP$^\tau$ and CFAR^b. In E_Q there are 3 conservative clusters for each value of b.
Cluster 1:

$$\begin{aligned}
&\tau \cdot \tau_{I_Q}(Q_0(b)) = \tau \cdot \tau_{I_Q}(Q_2(b)) = \tau \cdot \tau_{I_Q}(Q_5(b)) = \tau \cdot \tau_{I_Q}(Q_6(b))\\
&= \tau \cdot (r_5(ac) \cdot \tau_{I_Q}(Q_1(b)) + r_5(ac) \cdot \tau_{I_Q}(Q_4(b)) + r_5(ac) \cdot \tau_{I_Q}(Q_8(b)) +\\
&\qquad r_5(ac) \cdot \tau_{I_Q}(Q_9(b)))
\end{aligned}$$

Cluster 2:

$$\tau \cdot \tau_{I_Q}(Q_1(b)) = \tau \cdot \tau_{I_Q}(Q_3(b)) = \tau \cdot \tau_{I_Q}(Q_8(b)) \ = \tau \cdot (r_5(ac) \cdot r + \tau \cdot \tau_{I_Q}(Q_7(b)))$$

Cluster 3:

$$
\begin{aligned}
&\tau \cdot \tau_{I_Q}(Q_{10}(b)) = \tau \cdot \tau_{I_Q}(Q_{11}(b)) \\
&= \tau \cdot (r_5(ac) \cdot r + s_8(ac) \cdot \tau_{I_Q}(Q_0(inv(b))) + s_8(ac) \cdot \tau_{I_Q}(Q_2(inv(b))) + \\
&\qquad \tau \cdot \tau_{I_Q}(Q_{12}(b)) + \tau \cdot \tau_{I_Q}(Q_{13}(b)))
\end{aligned}
$$

Because $\tau_{I_Q}(Q_1(b))$ and $\tau_{I_Q}(Q_8(b))$ are both in cluster 2 we remove $\tau_{I_Q}(Q_8(b))$ from the equation for cluster 1:

$$
\begin{aligned}
&\tau \cdot \tau_{I_Q}(Q_0(b)) = \tau \cdot \tau_{I_Q}(Q_2(b)) = \tau \cdot \tau_{I_Q}(Q_5(b)) = \tau \cdot \tau_{I_Q}(Q_6(b)) \\
&= \tau \cdot (r_5(ac) \cdot \tau_{I_Q}(Q_1(b)) + r_5(ac) \cdot \tau_{I_Q}(Q_4(b)) + r_5(ac) \cdot \tau_{I_Q}(Q_9(b)))
\end{aligned}
$$

From E_Q and the equation for cluster 2 it follows that

$$
\begin{aligned}
&\tau \cdot \tau_{I_Q}(Q_9(b)) = \tau \cdot (r_5(ac) \cdot r + \tau \cdot \tau_{I_Q}(Q_1(b))) \\
&= \tau \cdot (r_5(ac) \cdot r + \tau \cdot (r_5(ac) \cdot r + \tau \cdot \tau_{I_Q}(Q_7(b)))) \\
&\overset{\text{B2}}{=} \tau \cdot (r_5(ac) \cdot r + \tau \cdot \tau_{I_Q}(Q_7(b))) \\
&= \tau \cdot \tau_{I_Q}(Q_1(b)) = \tau \cdot \tau_{I_Q}(Q_3(b)) = \tau \cdot \tau_{I_Q}(Q_8(b))
\end{aligned}
$$

From E_Q and the previous identity it follows in a similar way that

$$
\begin{aligned}
&\tau \cdot \tau_{I_Q}(Q_4(b)) = \tau \cdot (r_5(ac) \cdot r + \tau \cdot \tau_{I_Q}(Q_1(b))) \\
&= \tau \cdot \tau_{I_Q}(Q_1(b)) = \tau \cdot \tau_{I_Q}(Q_3(b)) = \tau \cdot \tau_{I_Q}(Q_8(b)) = \tau \cdot \tau_{I_Q}(Q_9(b))
\end{aligned}
$$

We also have

$$
\tau \cdot \tau_{I_Q}(Q_0(inv(b))) = \tau \cdot \tau_{I_Q}(Q_2(inv(b))) = \tau \cdot \tau_{I_Q}(Q_5(inv(b))) = \tau \cdot \tau_{I_Q}(Q_6(inv(b))).
$$

We see that $\tau_{I_Q}(Q_{12}(b)) = \tau_{I_Q}(Q_{13}(b))$. Now we can reduce even further: the equation for cluster 3 satisfies

$$
\begin{aligned}
&\tau \cdot \tau_{I_Q}(Q_{10}(b)) = \tau \cdot \tau_{I_Q}(Q_{11}(b)) \\
&= \tau \cdot (r_5(ac) \cdot r + s_8(ac) \cdot \tau_{I_Q}(Q_0(inv(b))) + \tau \cdot \tau_{I_Q}(Q_{12}(b))) \\
&= \tau \cdot (r_5(ac) \cdot r + s_8(ac) \cdot \tau_{I_Q}(Q_0(inv(b))) + \tau \cdot (r_5(ac) \cdot r + s_8(ac) \cdot \tau_{I_Q}(Q_0(inv(b))))) \\
&\overset{\text{B2}}{=} \tau \cdot (r_5(ac) \cdot r + s_8(ac) \cdot \tau_{I_Q}(Q_0(inv(b)))) = \tau \cdot \tau_{I_Q}(Q_{12}(b)) = \tau \cdot \tau_{I_Q}(Q_{13}(b))
\end{aligned}
$$

Some last calculations combine cluster 2 with cluster 3:

$$
\begin{aligned}
&\tau \cdot \tau_{I_Q}(Q_7(b)) = \tau \cdot (r_5(ac) \cdot r + \tau \cdot \tau_{I_Q}(Q_{10}(b))) \\
&= \tau \cdot (r_5(ac) \cdot r + \tau \cdot (r_5(ac) \cdot r + s_8(ac) \cdot \tau_{I_Q}(Q_0(inv(b))))) \\
&\overset{\text{B2}}{=} \tau \cdot (r_5(ac) \cdot r + s_8(ac) \cdot \tau_{I_Q}(Q_0(inv(b)))) \\
&= \tau \cdot \tau_{I_Q}(Q_{10}(b)) = \tau \cdot \tau_{I_Q}(Q_{11}(b)) = \tau \cdot \tau_{I_Q}(Q_{12}(b)) = \tau \cdot \tau_{I_Q}(Q_{13}(b))
\end{aligned}
$$

and therefore

$$
\begin{aligned}
&\tau \cdot \tau_{I_Q}(Q_1(b)) = \tau \cdot \tau_{I_Q}(Q_3(b)) = \tau \cdot \tau_{I_Q}(Q_4(b)) = \tau \cdot \tau_{I_Q}(Q_8(b)) = \tau \cdot \tau_{I_Q}(Q_9(b)) \\
&= \tau \cdot (r_5(ac) \cdot r + \tau \cdot \tau_{I_Q}(Q_7(b))) \\
&= \tau \cdot (r_5(ac) \cdot r + \tau \cdot (r_5(ac) \cdot r + s_8(ac) \cdot \tau_{I_Q}(Q_0(inv(b))))) \\
&\overset{\text{B2}}{=} \tau \cdot (r_5(ac) \cdot r + s_8(ac) \cdot \tau_{I_Q}(Q_0(inv(b)))) \\
&= \tau \cdot \tau_{I_Q}(Q_7(b)) = \tau \cdot \tau_{I_Q}(Q_{10}(b)) = \tau \cdot \tau_{I_Q}(Q_{11}(b)) = \tau \cdot \tau_{I_Q}(Q_{12}(b)) = \tau \cdot \tau_{I_Q}(Q_{13}(b))
\end{aligned}
$$

So finally we obtain a reduced specification for module Q with only 4 equations:

$$
\begin{aligned}
E_{Q_{red}} = \{ &\tau \cdot \tau_{I_Q}(Q_0(b)) = \tau \cdot r_5(ac) \cdot \tau_{I_Q}(Q_1(b)), \\
&\tau \cdot \tau_{I_Q}(Q_1(b)) = \tau \cdot (r_5(ac) \cdot r + s_8(ac) \cdot \tau_{I_Q}(Q_0(inv(b))))\}
\end{aligned}
$$

4.5 Verification of the CABP

4.5.1 Linearisation of the modular protocol specification

The specifications of the modules P and Q are sufficiently reduced to compute an equational specification of the CABP. For clarity we do not yet rename the actions $c_5(ac)$ and $c_8(ac)$ from the abstraction set I.

Consider the following guarded specification:

$$
\begin{aligned}
E_{Cabp} = \{\ & C_0(b) = \Sigma_{d\in D}\ r_1(d)\cdot C_1(d,b) + \tau\cdot C_2(b),\\
& C_1(d,b) = s_2(d)\cdot C_3(d,b) + \tau\cdot C_4(d,b),\\
& C_2(b) = \Sigma_{d\in D}\ r_1(d)\cdot C_4(d,b),\\
& C_3(d,b) = \tau\cdot C_5(d,b),\\
& C_4(d,b) = s_2(d)\cdot C_5(d,b),\\
& C_5(d,b) = \tau_I(c_5(ac))\cdot C_6(d,b),\\
& C_6(d,b) = \tau_I(c_8(ac))\cdot C_2(d,inv(b)) + \tau_I(c_8(ac))\cdot C_7(d,b),\\
& C_7(d,b) = \Sigma_{d\in D}\ r_1(d)\cdot C_8(d,b) + \tau\cdot C_2(d,inv(b)),\\
& C_8(d,b) = \tau\cdot C_4(d,inv(b))\}
\end{aligned}
$$

Lemma 4.5.1. *The root process of E_{Cabp} is related to the external behaviour of the CABP as follows:*

$$\tau\cdot C_0(0) = \tau\cdot\tau_I(Cabp).$$

Proof. Using head-tail expansion, it can be proved with the axioms of ACP$^\tau$, via RSP, that

$$C_0(0) = \tau_I\circ\partial_H(\tau_{I_P}(P_0(0))\ \|\ \tau\cdot\tau_{I_Q}(Q_0(0))).$$

According to the lemmas 4.3.1 and 4.4.1 we then have

$$C_0(0) = \tau_I\circ\partial_H(\tau_{I_P}\circ\Delta_{\{\sigma_P(e)|e\in D\}^*}(P)\ \|\ \tau\cdot\tau_{I_Q}\circ\Delta_{\{\sigma_Q\}^*}(Q)).$$

It is easy to verify that all the clauses in the Redundancy Theorem 3.3.6 are satisfied, so

$$C_0(0) = \tau_I\circ\partial_H(\tau_{I_P}(P)\ \|\ \tau\cdot\tau_{I_Q}(Q)).$$

Finally, with Lemma 4.1.2 we find the desired result. □

Note that two phases of the linearisation were done in one step: expansion and abstraction. The states in E_{Cabp} correspond with substates of $\tau_I\circ\partial_H(\tau_{I_P}(P_0(0))\ \|\ \tau\cdot\tau_{I_Q}(Q_0(0)))$ according to the list below. Although formally not completely correct, we also write P_i', Q_i' for the 'body' of the corresponding processes. For example, if we have the equation $\tau\cdot\tau_I(P_i) = \tau\cdot B$ from $E_{P_{red}}$, we also write P_i' for B.

Notations:

$$
\begin{aligned}
&[\,\_\,] \stackrel{\text{def}}{=} \tau_I\circ\partial_H(\,\_\,)\\
&\tau_{I_P}(P_i) \stackrel{\text{def}}{=} P_i'\\
&\tau_{I_Q}(Q_i) \stackrel{\text{def}}{=} Q_i'
\end{aligned}
$$

$[P'_0(b) \parallel \tau \cdot Q'_0(b)] = C_0(b)$
$[P'_1(d,b) \parallel \tau \cdot Q'_0(b)] = C_1(d,b)$
$[P'_0(b) \parallel Q'_0(b)] = C_2(b)$
$[P'_{13}(d,b) \parallel \tau \cdot Q'_0(b)] = C_3(d,b)$
$[P'_1(d,b) \parallel Q'_0(b)] = C_4(d,b)$
$[P'_{13}(d,b) \parallel Q'_0(b)] = C_5(d,b)$
$[P'_8(d,b) \parallel Q'_1(b)] = C_6(d,b)$
$[P'_{15}(d,b) \parallel Q'_0(inv(b))] = C_7(d,b)$
$[P'_{22}(d,b) \parallel Q'_0(inv(b))] = C_8(d,b)$

4.5.2 The last step in the verification

The last part of the verification is the reduction of E_{Cabp} such that it is proved that the CABP is a correct communication protocol. Reduction is done by applying the axioms of ACP$^\tau$:

$$
\begin{aligned}
&C_3(d,b) = \tau \cdot C_5(d,b) = \tau \cdot C_6(d,b)\\
&C_4(d,b) = s_2(d) \cdot C_5(d,b) = s_2(d) \cdot C_6(d,b)\\
&\tau \cdot C_1(d,b) = \tau \cdot (s_2(d) \cdot C_3(d,b) + \tau \cdot C_4(d,b))\\
&= \tau \cdot (s_2(d) \cdot C_6(d,b) + \tau \cdot s_2(d) \cdot C_6(d,b))\\
&= \tau \cdot s_2(d) \cdot C_6(d,b)\\
&C_8(d,b) = \tau \cdot C_4(d, inv(b)) = \tau \cdot s_2(d) \cdot C_6(d, inv(b))\\
&C_2(b) = \Sigma_{d \in D}\ r_1(d) \cdot C_4(d,b) = \Sigma_{d \in D}\ r_1(d) \cdot s_2(d) \cdot C_6(d,b)\\
&\tau \cdot C_7(d,b) = \tau \cdot (\Sigma_{d \in D}\ r_1(d) \cdot C_8(d,b) + \tau \cdot C_2(d, inv(b)))\\
&= \tau \cdot (\Sigma_{d \in D}\ r_1(d) \cdot s_2(d) \cdot C_6(d, inv(b)) + \tau \cdot \Sigma_{d \in D}\ r_1(d) \cdot s_2(d) \cdot C_6(d, inv(b)))\\
&= \tau \cdot \Sigma_{d \in D}\ r_1(d) \cdot s_2(d) \cdot C_6(d, inv(b))\\
&C_6(d,b) = \tau \cdot C_2(d, inv(b)) + \tau \cdot C_7(d,b)\\
&= \tau \cdot \Sigma_{d \in D}\ r_1(d) \cdot s_2(d) \cdot C_6(d, inv(b))\\
&\tau \cdot C_0(b) = \tau \cdot (\Sigma_{d \in D}\ r_1(d) \cdot C_1(d,b) + \tau \cdot C_2(b))\\
&= \tau \cdot (\Sigma_{d \in D}\ r_1(d) \cdot s_2(d) \cdot C_6(d,b) + \tau \cdot \Sigma_{d \in D}\ r_1(d) \cdot s_2(d) \cdot C_6(d,b))\\
&= \tau \cdot \Sigma_{d \in D}\ r_1(d) \cdot s_2(d) \cdot C_6(d,b)
\end{aligned}
$$

Using the Recursive Specification Principle it follows that $\tau \cdot C_6(d,b) = \tau \cdot C_6(d, inv(b))$, and also that $\tau \cdot C_0(d,b) = \tau \cdot C_6(d,b)$. So

$$\tau \cdot C_0(b) = \tau \cdot \Sigma_{d \in D}\ r_1(d) \cdot s_2(d) \cdot C_0(b).$$

Finally, if we use Lemma 4.5.1 we find that the CABP satisfies the correctness requirement in Theorem 4.1.1.

5 Concluding remarks

We presented a technique, language matching, for labelling process terms, based on trace information. We provided a modular verification of the Concurrent Alternating Bit Protocol, which showed that language matching can be particularly useful for suppressing the effects of redundancies in a context $\partial_H(\_ \parallel x)$. Earlier studies of this protocol had already demonstrated that redundancies are exactly that what makes this protocol so hard to verify ([GV89, KM90, Wam92]). Our verification is relatively short and straightforward. We think that it is reasonable to expect that language matching can be a good help for the modular, algebraic verification of other communication protocols as well.

References

[BB88] J.C.M. Baeten and J.A. Bergstra. Global renaming operators in concrete process algebra. *Information and Computation 78*, pages 205–245, 1988.

[BB91] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Journal of Formal Aspects of Computing Science*, 3(2):142–188, 1991.

[BB92] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. In W.R. Cleaveland, editor, *Proceedings CONCUR 92, Stony Brook*, pages 401–420. LNCS 630, Springer-Verlag, 1992.

[BBK87] J.C.M. Baeten, J.A. Bergstra and J.W. Klop. Conditional axioms and α/β–calculus in process algebra. In M. Wirsing, editor, *Proceedings IFIP Conference on Formal Description of Programming Concepts III, Ebberup*, pages 77–103. North-Holland, Amsterdam, 1987.

[BK84] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.

[BPW94] J.A. Bergstra, A. Ponse and J.J. van Wamel. Process algebra with backtracking. In J.W. de Bakker, W.-P. de Roever and G. Rozenberg, editors, *Proceedings of the REX School/Symposium "A Decade of Concurrency; Reflections and Perspectives"*, pages 46–91. LNCS 803, Springer-Verlag, 1994.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[Bae90] J.C.M. Baeten, editor. *Applications of Process Algebra.* Cambridge Tracts in Theoretical Computer Science 17. Cambridge University Press, 1990.

[GV89] R.J. van Glabbeek and F.W. Vaandrager. Modular specifications in process algebra – with curious queues. In M. Wirsing and J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, pages 465–506. LNCS 394, Springer-Verlag, 1989.

[GW91] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. Technical Report CS-R9120, CWI, Amsterdam, 1991.

[Gla90] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions.* Ph.D. thesis, Department of Mathematics and Computer Science, Free University, Amsterdam, 1990.

[Hil94] J.A. Hillebrand. The ABP and the CABP – a comparison of performances in real time process algebra. Report P9211b, University of Amsterdam, 1994. To appear in A. Ponse, C. Verhoef and S.F.M. van Vlijmen, editors, *Proceedings of ACP94.* Workshops in Computing, Springer-Verlag.

[KM90] C.P.J. Koymans and J.C. Mulder. A modular approach to protocol verification using process algebra. In [Bae90], pages 261–306.

[Kam93] G. Kamsteeg. A formal verification of the alternating bit protocol in μCRL. Technical Report 93-37, Leiden University, 1993.

[MV90] S. Mauw and G.J. Veltink. A process specification formalism. Fundamenta Informaticae XIII, pages 85–139, 1990.

[MV93] S. Mauw and G.J. Veltink, editors. *Algebraic Specification of Communication Protocols.* Cambridge Tracts in Theoretical Computer Science 36, Cambridge University Press, 1993.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5^{th} *GI Conference*, pages 167–183. LNCS 104, Springer-Verlag, 1981.

[Plo81] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Vaa86] F.W. Vaandrager. Verification of two communication protocols by means of process algebra. Technical Report CS-R8606, CWI, Amsterdam, 1986.

[Vaa90a] F.W. Vaandrager. Two simple protocols. In [Bae90], pages 23–44.

[Vaa90b] F.W. Vaandrager. Some observations on redundancy in a context. In [Bae90], pages 237–260.

[Ver92] C. Verhoef. *Linear unary operators in process algebra.* Ph.D. thesis, Department of Mathematics and Computer Science, University of Amsterdam, 1992.

[Ver93] C. Verhoef. A general conservative extension theorem in process algebra. Report CSN 93/38, Eindhoven University of Technology, 1993.

[Wam92] J.J. van Wamel. An algebraic verification of the concurrent alternating bit protocol. Report P9205, University of Amsterdam, 1992.

[Wam93] J.J. van Wamel. Simple protocols. In [MV93], pages 47–70.

Appendix

Proof of the Elimination Theorem 3.2.2. By induction on the structure of p. We first prove that BPA_δ^τ extended with the axioms for language matching is a conservative extension of BPA_δ^τ. Let $\mathrm{BPA}_{\delta,\mathrm{LM}}^\tau \stackrel{\mathrm{def}}{=} \mathrm{BPA}_\delta^\tau + \{\mathrm{LM1},\ldots,\mathrm{LM4}\}$. The term p is of the form a, δ, τ, $q+s$, $a\cdot q$, $\tau\cdot q$, or $\Delta_Z(q)$, where $a\in A$, $Z\subseteq A^\star$.

<u>Case 1.</u> If p is of the form a, δ, τ, $q+s$, $a\cdot q$, or $\tau\cdot q$ the proof is simple. For instance, take $p\equiv q+s$. By induction we may assume that there are closed terms q', s' over $\Sigma(\mathrm{BPA}_\delta^\tau)$ such that $\mathrm{BPA}_{\delta,\mathrm{LM}}^\tau \vdash q=q'$ and $s=s'$. It follows immediately that if $p' \stackrel{\mathrm{def}}{=} q'+s'$ then $\mathrm{BPA}_{\delta,\mathrm{LM}}^\tau \vdash p=p'$.

<u>Case 2.</u> Let $p\equiv \Delta_Z(p')$, with p' a closed term over $\Sigma(\mathrm{BPA}_\delta^\tau)$, and $Z\subseteq A^\star$. We apply simultaneous induction on the structure of p' and Z.

- If $p'\equiv\delta$ or $p'\equiv\tau$ we can apply axiom LM1.
- If $p'\equiv a$ then $p'=a\cdot\tau$, so that axiom LM4 can be applied.
- If $p'\equiv q+s$ then $\Delta_Z(p')=\Delta_Z(q+s)\stackrel{\mathrm{LM2}}{=}\Delta_Z(q)+\Delta_Z(s)\stackrel{\mathrm{i.h.}}{=}q'+s'$, where q' and s' are some closed terms over $\Sigma(\mathrm{BPA}_\delta^\tau)$.
- If $p'\equiv\tau\cdot q$ then $\Delta_Z(p')=\Delta_Z(\tau\cdot q)\stackrel{\mathrm{LM3}}{=}\tau\cdot\Delta_Z(q)\stackrel{\mathrm{i.h.}}{=}\tau\cdot q'$, where q' is some closed term over $\Sigma(\mathrm{BPA}_\delta^\tau)$.
- If $p'\equiv a\cdot q$ then

$$\Delta_Z(p')\equiv\Delta_Z(a\cdot q)\stackrel{\mathrm{LM4}}{=}\begin{cases} a\cdot\Delta_Z(q) \stackrel{\mathrm{i.h.}}{=} a\cdot q' & \text{if } a\notin \mathit{acts}(Z)\\ a\cdot\Delta_{\partial Z/\partial a}(q) \stackrel{\mathrm{i.h.}}{=} a\cdot q'' & \text{if } \partial Z/\partial a\neq\emptyset\\ a\cdot r & \text{otherwise,}\end{cases}$$

  where q' and q'' are some closed terms over $\Sigma(\mathrm{BPA}_\delta^\tau)$.

<u>Case 3.</u> If p' is a closed term over $\Sigma(\mathrm{BPA}_{\delta,\mathrm{LM}}^\tau)$ then if $p\equiv\Delta_Z(p')$, then by induction we may assume that there is a closed term p'' over $\Sigma(\mathrm{BPA}_\delta^\tau)$ such that $\mathrm{BPA}_{\delta,\mathrm{LM}}^\tau\vdash p'=p''$. Using Case 2 we find that p is provably equal to some closed term over $\Sigma(\mathrm{BPA}_\delta^\tau)$.

Finally, since ACP^τ is also a conservative extension of BPA_δ^τ, we can prove by induction on the number of occurrences of process operators from $\{\|, \mathbin{\|\!\_}, |, \partial_H, \tau_I, \Delta_Z\}$ that for every closed term p over $\Sigma(\mathrm{ACP}_{\mathrm{LM}}^\tau)$ there is a closed term p' over $\Sigma(\mathrm{BPA}_\delta^\tau)$, and thus over $\Sigma(\mathrm{ACP}^\tau)$. $\square$

Proof of the Congruence Theorem 3.2.4. The relation $\leftrightarroweq_{rb}$ is a congruence w.r.t. the operators from $\Sigma(\mathrm{ACP}^\tau)$, so it is left to prove that $\leftrightarroweq_{rb}$ is a congruence w.r.t. the operators Δ_Z, i.e., if $p\leftrightarroweq_{rb} q$ then $\Delta_Z(p)\leftrightarroweq_{rb}\Delta_Z(q)$, where $Z\subseteq A^\star$.

Since the definition of $\leftrightarroweq_{rb}$ depends on $\leftrightarroweq_b$, we first prove that $\leftrightarroweq_b$ is a congruence. Let $p\leftrightarroweq_b q$. We prove $\Delta_Z(p)\leftrightarroweq_b\Delta_Z(q)$ by induction on the longest trace of $p\cdot q$. For length $=2$ both p and q must be sums of atomic actions, possibly plus τ, some of which may be followed by δ. So p and q always terminate or block in one step. We therefore further only note that for length ≤ 2 the proof is easy. For length >2 all possible transitions of $\Delta_Z(p)$ have to be considered, and all possible transitions of $\Delta_Z(q)$ have to be derived, given that $p \leftrightarrow_b q$. We elaborate only three cases:

Case 1. $\Delta_Z(p) \xrightarrow{\tau} \surd$. It follows from the rules in Table 11 that $p \xrightarrow{\tau} \surd$, so it must be the case that $\exists s.\ q \Longrightarrow s \xrightarrow{\tau} \surd\ \&\ p \leftrightarrow_b s$ (see clause 1 in Definition 2.2.2). From Table 11 it follows that $\exists s.\ \Delta_Z(q) \Longrightarrow \Delta_Z(s) \xrightarrow{\tau} \surd$, and by induction $\Delta_Z(p) \leftrightarrow_b \Delta_Z(s)$.

Case 2. $\Delta_Z(p) \xrightarrow{\tau} p''$. It must be that $\exists p'.\ p'' \equiv \Delta_Z(p')\ \&\ p \xrightarrow{\tau} p'$, and either $p' \leftrightarrow_b q$, or $\exists q', s.\ q \Longrightarrow s \xrightarrow{\tau} q'\ \&\ p \leftrightarrow_b s\ \&\ p' \leftrightarrow_b q'$ (see clause 2 in Definition 2.2.2). In the former case it follows by induction that $\Delta_Z(p') \leftrightarrow_b \Delta_Z(q)$, and in the latter case we have $\exists q', s.\ \Delta_Z(q) \Longrightarrow \Delta_Z(s) \xrightarrow{\tau} \Delta_Z(q')$, and by induction $\Delta_Z(p) \leftrightarrow_b \Delta_Z(s)\ \&\ \Delta_Z(p') \leftrightarrow_b \Delta_Z(q')$.

Case 3. $\Delta_Z(p) \xrightarrow{a} p''\ \&\ a \in acts(Z)\ \&\ \partial Z/\partial a = \emptyset$. It follows from Table 11 that $p'' \equiv r$ and that either $p \xrightarrow{a} \surd$ or $\exists p'.\ p \xrightarrow{a} p'$.

Case 3.1. $p \xrightarrow{a} \surd$. It follows from Definition 2.2.2 that either $\exists s.\ q \Longrightarrow s \xrightarrow{a} \surd\ \&\ p \leftrightarrow_b s$, or $\exists q', s.\ q \Longrightarrow s \xrightarrow{a} q'\ \&\ p \leftrightarrow_b s\ \&\ \surd(q')$. Using the rules in Table 11 we find in both cases that $\exists s.\ \Delta_Z(q) \Longrightarrow \Delta_Z(s) \xrightarrow{a} r$, and by induction $\Delta_Z(p) \leftrightarrow_b \Delta_Z(s)$.

Case 3.2. Quite similar to Case 3.1.

When the other cases are also studied, it is found that all possible transitions of $\Delta_Z(p)$ are matched by (possibly empty) sequences of transitions from $\Delta_Z(q)$, exactly in accordance with the definition of branching bisimulation. By symmetry we conclude that

$$(i)\quad \textit{if}\ \ p \leftrightarrow_b q\ \ \textit{then}\ \ \ \Delta_Z(p) \leftrightarrow_b \Delta_Z(q).$$

Finally, the congruence properties of Δ_Z w.r.t. the root condition in Definition 2.2.3 have to be verified. This can be done in a similar way as above. Only instead of induction result (i) has to be used. □

Proof of Lemma 3.3.2.

$$\textit{if}\quad r \notin H \quad \textit{and} \quad r \notin \alpha \circ \partial_H \circ \Delta_Z(p) \quad \textit{then} \quad \partial_H \circ \Delta_Z(p) = \partial_H(p).$$

By simultaneous induction on the structure of p and Z. Let $r \notin H$ and $a \in A$.

- $p \equiv \delta$ or $p \equiv \tau$. Trivial.
- $p \equiv a$. Just as $p \equiv a \cdot q$.
- $p \equiv q + s$. If $r \notin \alpha \circ \partial_H \circ \Delta_Z(p)$ then clearly $r \notin \alpha \circ \partial_H \circ \Delta_Z(q)$ and $r \notin \alpha \circ \partial_H \circ \Delta_Z(s)$. We derive
$$\begin{array}{rcl} \partial_H \circ \Delta_Z(p) & = & \partial_H \circ \Delta_Z(q+s) \\ & \overset{\text{LM2}}{=} & \partial_H \circ \Delta_Z(q) + \partial_H \circ \Delta_Z(s) \\ & \overset{\text{i.h.}}{=} & \partial_H(q) + \partial_H(s) \\ & = & \partial_H(q+s) = \partial_H(p). \end{array}$$
- $p \equiv \tau \cdot q$. Very similar to the previous case. Use axiom LM3.

- $p \equiv a \cdot q$. If $a \in H$ the result follows easily using axiom LM4, so we assume $a \notin H$.

$$\partial_H \circ \Delta_Z(p) \equiv \partial_H \circ \Delta_Z(a \cdot q) \stackrel{\text{LM4}}{=} \begin{cases} a \cdot \partial_H \circ \Delta_Z(q) & \text{if } a \notin acts(Z) \quad (i) \\ a \cdot \partial_H \circ \Delta_{\partial Z/\partial a}(q) & \text{if } \partial Z/\partial a \neq \emptyset \quad (ii) \\ a \cdot \partial_H(r) & \text{otherwise} \quad (iii) \end{cases}$$

(i) $a \notin H$ and $r \notin \alpha \circ \partial_H \circ \Delta_Z(p)$ imply $r \notin \alpha \circ \partial_H \circ \Delta_Z(q)$, so we may apply the i.h.:

$$a \cdot \partial_H \circ \Delta_Z(q) \stackrel{\text{i.h}}{=} a \cdot \partial_H(q) = \partial_H(a \cdot q).$$

(ii) Just as (i).

(iii) $r \notin H$ so $a \cdot \partial_H(r) = a \cdot r$. Now if $r \notin \alpha \circ \partial_H \circ \Delta_Z(p)$ we have a contradiction, so it must be the case that $a \notin acts(Z)$ or $\partial Z/\partial a \neq \emptyset$.

We conclude that if $r \notin H$ and $r \notin \alpha \circ \partial_H \circ \Delta_Z(p)$ then $\partial_H \circ \Delta_Z(a \cdot q) = \partial_H(a \cdot q)$. This finishes the last clause of the proof.

□

Proof of Lemma 3.3.3.

$$\textit{if } r \notin H \textit{ and } r \notin \alpha \circ \partial_H(\Delta_Z(p) \parallel q) \textit{ then } \partial_H(\Delta_Z(p) \parallel q) = \partial_H(p \parallel q).$$

By simultaneous induction on the structure of p, q and Z. Due to the properties of $\parallel$ we cannot consider the different shapes of p and q separately, as in the previous proof. Therefore we prove this lemma for more general forms. In [BW90] it is shown that all closed terms over $\Sigma(\text{ACP}^\tau)$ are either of the form τ or $\Sigma_{j \in J}\, a_j \cdot t_j$, where $a_j \in A \cup \{\tau\}$. The *sum convention* defines $\Sigma_{j \in \emptyset}\, x \stackrel{\text{def}}{=} \delta$. Let $r \notin H$.

Case 1. If $p \equiv \tau$ or $p \equiv \delta$ the result follows trivially, so we assume that $p \equiv \Sigma_{j \in J}\, a_j \cdot p_j$, where J is some non-empty index set.

Case 2. Let $q \equiv \tau$. For all closed t we have $t \parallel \tau = t + \tau \cdot t$ (standard). Now if we have that $r \notin \alpha \circ \partial_H(\Delta_Z(p) \parallel q)$ then clearly $r \notin \alpha \circ \partial_H \circ \Delta_Z(p)$, so we may use the previous lemma. We derive

$$\begin{aligned} \partial_H(\Delta_Z(p) \parallel q) &= \partial_H(\Delta_Z(p) \parallel \tau) \\ &= \partial_H \circ \Delta_Z(p) + \tau \cdot \partial_H \circ \Delta_Z(p) \\ &\stackrel{3.3.2}{=} \partial_H(p) + \tau \cdot \partial_H(p) \\ &= \partial_H(p \parallel \tau) = \partial_H(p \parallel q). \end{aligned}$$

Case 3. Let $q \equiv \delta$. For all closed t we have $t \parallel \delta = t \cdot \delta$ (standard). This part of the proof is quite similar to Case 2.

Case 4. Let $q \equiv \Sigma_{k \in K}\, b_k \cdot q_k$, where K is some non-empty index set and $b_k \in A \cup \{\tau\}$. Using axiom LM2 we find

$$\begin{aligned} \partial_H(\Delta_Z(p) \parallel q) = {} & \Sigma_{k \in K}\, \partial_H(b_k) \cdot \partial_H(\Delta_Z(p) \parallel q_k) + \Sigma_{j \in J}\, \partial_H(\Delta_Z(a_j \cdot p_j) \mathbin{\lfloor\!\lfloor} q) + \\ & \Sigma_{j \in J} \Sigma_{k \in K}\, \partial_H(\Delta_Z(a_j \cdot p_j) \mid b_k \cdot q_k). \end{aligned}$$

We study the three summands of $\partial_H(\Delta_Z(p) \parallel q)$ separately.

Case 4.1. If $r \notin \alpha \circ \partial_H(\Delta_Z(p) \parallel q)$ then we have $r \notin \alpha \circ \partial_H(\Delta_Z(p) \parallel q_k)$ for all k for which $b_k \notin H$. So we may directly apply the i.h. to the first summand, and thus

$$(I) \quad \textit{if } r \notin H \textit{ and } r \notin \alpha \circ \partial_H(\Delta_Z(p) \parallel q) \textit{ then}$$

$$\Sigma_{k \in K}\ \partial_H(b_k) \cdot \partial_H(\Delta_Z(p) \parallel q_k) = \Sigma_{k \in K}\ \partial_H(b_k) \cdot \partial_H(p \parallel q_k).$$

Case 4.2. We easily find for all $j \in J$:

$$\partial_H(\Delta_Z(a_j \cdot p_j) \mathbin{\|\!\_} q) \stackrel{\mathrm{LM4}}{=} \begin{cases} \delta & \text{if } a_j \in H & (i) \\ a_j \cdot \partial_H(\Delta_Z(p_j) \parallel q) & \text{if } a_j \notin H \text{ and } a_j \notin acts(Z) & (ii) \\ a_j \cdot \partial_H(\Delta_{\partial Z/\partial a_j}(p_j) \parallel q) & \text{if } a_j \notin H \text{ and } \partial Z/\partial a_j \neq \emptyset & (iii) \\ a_j \cdot \partial_H(r \parallel q) & \text{otherwise} & (iv) \end{cases}$$

(i) $\partial_H(\Delta_Z(a_j \cdot p_j) \mathbin{\|\!\_} q) = \partial_H(a_j) \cdot \partial_H(p_j \parallel q)$.
(ii) If $r \notin \alpha \circ \partial_H(\Delta_Z(p) \parallel q)$ then $r \notin \alpha \circ \partial_H(\Delta_Z(p_j) \parallel q)$, so we may apply the i.h.:

$$\partial_H(\Delta_Z(a_j \cdot p_j) \mathbin{\|\!\_} q) = a_j \cdot \partial_H(\Delta_Z(p_j) \parallel q) \stackrel{\text{i.h.}}{=} a_j \cdot \partial_H(p_j \parallel q) = \partial_H(a_j) \cdot \partial_H(p_j \parallel q).$$

(iii) Similar to (ii).
(iv) Since $r \notin H$ it is always true that $r \in \alpha \circ \partial_H(r \parallel q)$, so we have a contradiction; it cannot be the case that $\partial_H(\Delta_Z(a_j \cdot p_j) \mathbin{\|\!\_} q) = a_j \cdot \partial_H(r \parallel q)$.

From (i)–(iv) we conclude

$$(II) \quad \textit{if } r \notin H \textit{ and } r \notin \alpha \circ \partial_H(\Delta_Z(p) \parallel q) \textit{ then}$$

$$\Sigma_{j \in J}\ \partial_H(\Delta_Z(a_j \cdot p_j) \mathbin{\|\!\_} q) = \Sigma_{j \in J}\ \partial_H(a_j) \cdot \partial_H(p_j \parallel q).$$

Case 4.3. The third, and last summand is treated in a similar way as the second (Case 4.2). It is found that

$$(III) \quad \textit{if } r \notin H \textit{ and } r \notin \alpha \circ \partial_H(\Delta_Z(p) \parallel q) \textit{ then}$$

$$\Sigma_{j \in J}\Sigma_{k \in K}\ \partial_H(\Delta_Z(a_j \cdot p_j) \mid b_k \cdot q_k) = \Sigma_{j \in J}\Sigma_{k \in K}\ \partial_H(a_j \mid b_k) \cdot \partial_H(p_j \parallel q_k).$$

Finally, if we combine (I), (II), and (III) we see that we can reconstruct $\partial_H(p \parallel q)$, so we are ready with Case 4, and therefore with the proof. □

Proof of Lemma 3.3.5.

$$\textit{if} \quad r \notin I \quad \textit{and} \quad acts(Z) \cap I = \emptyset \quad \textit{then} \quad \tau_I \circ \Delta_Z(p) = \Delta_Z \circ \tau_I(p).$$

By simultaneous induction on the structure of p and Z. We suppose throughout the proof that $r \notin I$ and $acts(Z) \cap I = \emptyset$. Let $a \in A$.

- $p \equiv \delta$ or $p \equiv \tau$. Trivial.

- $p \equiv a$. Just as $p \equiv a \cdot q$.

- $p \equiv q + s$.

$$\begin{array}{rcl} \tau_I \circ \Delta_Z(p) & = & \tau_I \circ \Delta_Z(q+s) \\ & \stackrel{\text{LM2}}{=} & \tau_I \circ \Delta_Z(q) + \tau_I \circ \Delta_Z(s) \\ & \stackrel{\text{i.h.}}{=} & \Delta_Z \circ \tau_I(q) + \Delta_Z \circ \tau_I(s) \\ & \stackrel{\text{LM2}}{=} & \Delta_Z \circ \tau_I(q+s) = \Delta_Z \circ \tau_I(p). \end{array}$$

- $p \equiv \tau \cdot q$. Very similar to the previous case. Use axiom LM3.

- $p \equiv a \cdot q$.

$$\tau_I \circ \Delta_Z(p) \equiv \tau_I \circ \Delta_Z(a \cdot q) \stackrel{\text{LM4}}{=} \begin{cases} \tau_I(a) \cdot \tau_I \circ \Delta_Z(q) & \text{if } a \notin acts(Z) \quad (i) \\ \tau_I(a) \cdot \tau_I \circ \Delta_{\partial Z/\partial a}(q) & \text{if } \partial Z/\partial a \neq \emptyset \quad (ii) \\ \tau_I(a \cdot r) & \text{otherwise} \quad (iii) \end{cases}$$

(i) If $a \in I$ then

$$\begin{array}{rcl} \tau_I(a) \cdot \tau_I \circ \Delta_Z(q) & = & \tau \cdot \tau_I \circ \Delta_Z(q) \\ & \stackrel{\text{i.h.}}{=} & \tau \cdot \Delta_Z \circ \tau_I(q) \\ & \stackrel{\text{LM3}}{=} & \Delta_Z(\tau \cdot \tau_I(q)) \\ & = & \Delta_Z \circ \tau_I(a \cdot q). \end{array}$$

If $a \notin I$ then

$$\begin{array}{rcl} \tau_I(a) \cdot \tau_I \circ \Delta_Z(q) & = & a \cdot \tau_I \circ \Delta_Z(q) \\ & \stackrel{\text{i.h.}}{=} & a \cdot \Delta_Z \circ \tau_I(q) \\ & \stackrel{\text{LM4}}{=} & \Delta_Z(a \cdot \tau_I(q)) \\ & = & \Delta_Z \circ \tau_I(a \cdot q). \end{array}$$

(ii) We have $\partial Z/\partial a \neq \emptyset$, so since $acts(Z) \cap I = \emptyset$ we have $a \notin I$. The rest of the proof is similar to (i).

(iii) We have $a \in acts(Z)$, and thus $a \notin I$. Moreover since $r \notin I$ we find

$$\tau_I(a \cdot r) = a \cdot r \stackrel{\text{LM4}}{=} \Delta_Z(a \cdot \tau_I(q)) = \Delta_Z \circ \tau_I(a \cdot q).$$

This finishes the last clause of the proof.

□