University of Amsterdam
Programming Research Group

$\alpha\ \beta$

A Formal Axiomatization for Alphabet Reasoning with Parametrized Processes

Henri Korver
Alex Sellink

Report P9703 Februari 1997

University of Amsterdam

Department of Computer Science

Programming Research Group

A formal axiomatization for alphabet reasoning with parametrized processes

Henri Korver

Alex Sellink

Report P9703 Februari 1997

H.P. Korver

ID Research

Groningenweg 6
2803 PV Gouda
The Netherlands

tel. +31 182 546546
e-mail: H.Korver@idr.nl

M.P.A. Sellink

Programming Research Group
Department of Computer Science
University of Amsterdam

Kruislaan 403
NL-1098 SJ Amsterdam
The Netherlands

tel. +31 20 525 7583
e-mail: alex@fwi.uva.nl

Universiteit van Amsterdam, 1997

A Formal Axiomatization for Alphabet Reasoning with Parametrized Processes

Henri Korver[1] and Alex Sellink[2]

[1] ID Research, Groningenweg 6, 2803 PV Gouda, The Netherlands;

[2] University of Amsterdam, Programming Research Group, P.O. Box 41882, 1009 DB Amsterdam, The Netherlands

Abstract

In the process-algebraic verification of systems with three or more components put in parallel, alphabet axioms are considered to be useful. These are rules that exploit the information about the alphabets of the processes involved. The alphabet of a process is the set of actions it can perform. In this paper, we extend μCRL (a formal proof system for ACP$_\tau$ + data) with such axioms. The alphabet axioms that are added to the proof theory are completely formal and therefore highly suited for computer-checked verification. This is new compared to previous papers where the formulation of alphabet axioms relies for a considerable extend on informal data parameters and implicit (infinite) set theory.

1 Introduction

During the proof checking of Milner's Scheduler (see [KS93]), we found out that there was a need for an explicit treatment of the so-called alphabet axioms in a context of *data*, i.e. a setting where actions and processes are parametrized with data values (possibly ranging over infinite domains).[1] Alphabet axioms are rules which use the information about the alphabet of processes. Intuitively, the alphabet of a process is the set of atomic actions it can perform. An example of a (conditional) axiom which uses the information about the alphabet of a process is given by:

$$\alpha(x) \cap I = \varnothing \quad \rightarrow \quad \tau_I(x) = x \qquad \text{(CA4)}.$$

In words the rule CA4 says: if no action from I occurs in the alphabet $\alpha(x)$ of process x, then hiding (renaming into τ) actions of I in x has no effect. This rule is one of the seven conditional alphabet axioms which Baeten, Bergstra and Klop [BBK87] added to ACP$_\tau$ [BK85] for making verifications of systems

[1] In [KS93] (unpublished) we added some alphabet axioms to the theory of μCRL. This extension was sufficient to get the verification of the scheduler done, but incomplete as a structural formalization of alphabet reasoning. Later on, an extended abstract of this paper was published as [KS94]. In that abstract all the work concerning alphabet axioms (static laws) was left out.

with two or more components put in parallel feasible. [2] A more interesting axiom is CA2 (see Table 3), which allows pushing the τ_I operator inside parallel compositions (in appropriate circumstances).

Milner uses similar rules in the verification of the scheduler, although he uses another terminology (see [Mil89]). In particular, he calls the alphabet of a process the *sort* of a process and rules making use of such information *static* laws, as they mostly describe static properties of processes. Because these laws center around the restriction operator (\) in CCS, Milner also calls them *restriction* laws.

Existing formulations of alphabet axioms like the rule CA4 given above, are based on (infinite) set theory. In this approach the question arises what axioms should be adopted for the set operators $\cup$, $\cap$, etc. One option, which is implicitly adopted in various papers on process algebra, is to take the equalities which are true in set theory. This collection is unstructured and too large for our purpose: formalizing and mechanizing process algebra proofs.

In this paper, it is shown that alphabet axioms like CA4 can be formalized within the proof theory of μCRL [GP95, GP93, GP91], which is a formal theory where ACP_τ is combined with abstract data types. The main idea is that *infinite* alphabets can be represented by finite sets of actions that are parametrized by *data* variables. For instance, the singleton set $\{a(n)\}$ (where n is a variable ranging over the natural numbers) is used for representing the infinite alphabet $\{a(0), a(1), a(2), \ldots\}$.

For a complete treatment, the renaming mechanism of μCRL is refined such that also instances of parametrized actions can be encapsulated or hidden. In [KS93, KS94] we find out that the ordinary encapsulation and hiding operators in μCRL were not refined enough to formalize all the alphabet reasoning used in Milner's Scheduler proof.

The new axioms are highly suited for proof checking purposes. In particular, the axioms are used for checking the correctness proof of Milner's Scheduler by computer (see [KS93]).[3] Because the verification of Milner Scheduler is rather technical, especially when it comes to alphabet reasoning, we shall here take a much simpler verification example for illustrating the usefulness of the new axioms. In fact, we will verify some properties about 'bags'. One of the properties verified in this paper is the fact that two connected bags form a bag. This verification is a reformulation of a proof given in [BBK87]. The contribution of the new proof is that it is completely formal and self-contained, i.e. does not rely on auxiliary machinery (meta-theory) such as implicit set theory and α/β-calculus.

The paper is organized as follows. In Section 2, the conventional approach towards alphabet axioms is summarized. Then a new, more explicit, formulation is given in the remaining sections. In Section 4, μCRL is extended with alphabet axioms for *gates*.[4] For being able to simulate infinite alphabets, the encapsulation and hiding operator are refined to actions that are parametrized with data in Section 5. In Section 6, conditional axioms are given for *parametrized* actions. This section is important as it shows how the reasoning with infinite alphabets

[2] For the axioms of ACP_τ, see also [BW90, p.123].

[3] In [KS93] the alphabet axioms appear in a slightly different representation.

[4] Actions of which the data (if any) is omitted. E.g. in the example specification on page 19 the actions are a, d, $a(0)$, $b(0)$, $c(0)$, $a(S(0))$, $b(S(0))$ etcetera, whereas the gates are a, b, c and d

can be represented symbolically within μCRL. In Section 7, some axioms are introduced which combine the power of the axiom systems given in Section 4 and 6. Then, in Section 8, an illustrative example is given where the working of the new axioms is extensively demonstrated. Finally, a short overview of μCRL is given in Appendix 1.

2 Traditional alphabet axioms

In this section, a standard representation of alphabet axioms is summarized. The *alphabet* of a process is the set of atomic actions it can perform. The axioms given in Table 1, taken from [BBK87], define the alphabet function α on closed ACP$_\tau$ terms. In this table, $a \in Act$ (the set of actions, not including τ

AB1	$\alpha(\delta) = \varnothing$	AB5	$\alpha(x+y) = \alpha(x) \cup \alpha(y)$
AB2	$\alpha(\tau) = \varnothing$	AB6	$\alpha(x) = \bigcup_{n \geq 1} \alpha(\pi_n(x))$
AB3	$\alpha(a \cdot x) = \{a\} \cup \alpha(x)$	AB7	$\alpha(\tau_I(x)) = \alpha(x) - I$
AB4	$\alpha(\tau \cdot x) = \alpha(x)$		

Table 1: Alphabet.

and δ) and x, y are arbitrary (ACP$_\tau$) processes. The axioms AB6 and AB7 are used to compute the alphabet of *infinite* processes. [5] Axiom AB6 makes use of a projection operator which is defined in Table 2. In this table, b ranges over δ and Act. This operator cuts off a process after performing n atomic steps. In

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

Table 2: Projection.

[Vaa90] alphabet axioms are also given for the renaming operator ρ. In most practical examples, however, the renaming operator does not occur or is easy to eliminate. This, and the fact that the alpabet axioms for the renaming operator are substantially more complex than the alphabet axioms for encapsulation and hide, made us decide to ignore them in this paper.

In practice it is rather cumbersome to compute alphabets of infinite processes by using axiom AB6. Therefore in [BBK87] the α/β-calculus is developed for

[5] In ACP infinite processes are defined as the solution of a guarded recursive specification (see [BW90]).

finding the alphabet of a process. This is important because in general it is not possible to compute the alphabet of a process in an effective way.

In Table 3, seven conditional axioms taken from [BBK87] are presented[6] which use the alphabets of processes. As usual in process algebra, the binary operator γ denotes the communication function (see e.g. [BW90, Chapter 4]).

CA1	$\gamma(\alpha(x), \alpha(y) \cap H) \subseteq H$	$\rightarrow$	$\partial_H(x \parallel y) = \partial_H(x \parallel \partial_H(y))$
CA2	$\gamma(\alpha(x), \alpha(y) \cap I) \subseteq \varnothing$	$\rightarrow$	$\tau_I(x \parallel y) = \tau_I(x \parallel \tau_I(y))$
CA3	$\alpha(x) \cap H = \varnothing$	$\rightarrow$	$\partial_H(x) = x$
CA4	$\alpha(x) \cap I = \varnothing$	$\rightarrow$	$\tau_I(x) = x$
CA5	$H = H_1 \cup H_2$	$\rightarrow$	$\partial_H(x) = \partial_{H_1} \circ \partial_{H_2}(x)$
CA6	$I = I_1 \cup I_2$	$\rightarrow$	$\tau_I(x) = \tau_{I_1} \circ \tau_{I_2}(x)$
CA7	$H \cap I = \varnothing$	$\rightarrow$	$\tau_I \circ \partial_H(x) = \partial_H \circ \tau_I(x)$

Table 3: Conditional Axioms.

The first axiom in Table 3, says that the encapsulation operator can be pushed inside when the encapsulated actions are not possible vehicles of communication between x and y. The axiom CA2 can be explained in a similar way. The other axioms speak for themselves. The major advantage of the axioms given in Table 3 is that they allow to prove important properties of processes without having to expand processes to their (head) normal form which can be very tiresome in case of parallel compositions with more than two components. For instance, showing that three connected bags form a bag without using such axioms may be very involved (see Section 8).

3 Reformulating the Alphabet Axioms in ACP$_\tau$

If *Act* contains infinitely many actions, the use of axioms AB6 and AB7 would involve infinite set theory which is not easy to mechanize, at least not in the context of μCRL as set out in [CH88]. Moreover, the side conditions of CA1$-$4 use the alphabet operator α, which can not be computed effectively. In many verifications, however, one would like to reason with infinite sets and alphabets. For instance, in Milner's Scheduler proof (see [Mil89] and [KS93]), but also in the verification given in Section 8. The solution here proposed is to use alphabet axioms that built upon gates instead of actions. This enables us to represent infinite sets of actions like $a(0)$, $a(1)$, $a(2), \ldots$ by the single gate a. As an intermediate step, we reformulate (within the setting of ACP$_\tau$) the alphabet axioms CA1$-$4, which make use of the α-operator, by the axioms CAr1$-$4 (CA reformulated). In these latter axioms the α-operator is eliminated. Of course, CAr1$-$4 still involves infinite set theory because $H, H_1, H_2 \subseteq Act$ may be infinite. As said before, this problem is solved by replacing actions by gates.

In Table 4 we use the following conventions:

[6] Note that in CA1$-$2 we used the denotation convention that $F(V, W)$ stands for $\{F(x, y) : x \in V,\ y \in W\}$ if V and W are sets and F is a function on $V \times W$. The same convention in used in Table 5.

CAr1	$\gamma(\overline{H}_1, H - H_2) \subseteq H$	$\rightarrow$	$\partial_H(\partial_{H_1}(x) \parallel \partial_{H_2}(y)) = \partial_H(\partial_{H_1}(x) \parallel \partial_H\partial_{H_2}(y))$
CAr2	$\gamma(\overline{H}_1, H - H_2) = \varnothing$	$\rightarrow$	$\tau_H(\partial_{H_1}(x) \parallel \partial_{H_2}(y)) = \tau_H(\partial_{H_1}(x) \parallel \tau_H\partial_{H_2}(y))$
CAr3	$H \subseteq (H_1 \cap H_2) - \gamma(\overline{H}_1, \overline{H}_2)$	$\rightarrow$	$\partial_H(\partial_{H_1}(x) \parallel \partial_{H_2}(y)) = \partial_{H_1}(x) \parallel \partial_{H_2}(y)$
CAr3′			$\partial_\varnothing(x) = x$
CAr3″	$H_1 \subseteq H_2$	$\rightarrow$	$\tau_{H_1}\partial_{H_2}(x) = \partial_{H_2}(x)$
CAr4	$H \subseteq (H_1 \cap H_2) - \gamma(\overline{H}_1, \overline{H}_2)$	$\rightarrow$	$\tau_H(\tau_{H_1}(x) \parallel \tau_{H_2}(y)) = \tau_{H_1}(x) \parallel \tau_{H_2}(y)$
CAr4′			$\tau_\varnothing(x) = x$
CAr4″	$H_1 \subseteq H_2$	$\rightarrow$	$\partial_{H_1}\tau_{H_2}(x) = \tau_{H_2}(x)$

Table 4: Alternative Alphabet Axioms for CA1–4 in ACP_τ

- The symbols $H, H_1, H_2 \subseteq Act$ stand for (possibly infinite) sets of actions.
- $\overline{H} = Act - H$ is the complement of H.
- For readability we write $F_{H_1}F'_{H_2}(x)$ for $F_{H_1}(F'_{H_2}(x))$ for all $F, F' \in \{\partial, \tau\}$.

Theorem 3.1. *Axioms CAr1–4 are derivable in ACP_τ + CA1–7.*

Proof. That CAr1 is derivable is motivated as follows. By substituting $\partial_{H_1}(x)$ for x and $\partial_{H_2}(y)$ for y in CA1 (see Table 3), we obtain

$$\gamma(\alpha(\partial_{H_1}(x)), \alpha(\partial_{H_2}(y)) \cap H) \subseteq H \;\rightarrow \partial_H(\partial_{H_1}(x) \parallel \partial_{H_2}(y)) = \partial_H(\partial_{H_1}(x) \parallel \partial_H\partial_{H_2}(y)) \tag{1}$$

Obviously, H_1 can not be part of the alphabet of the process $\partial_{H_1}(x)$. In formula: $\alpha(\partial_{H_1}(x)) \subseteq \overline{H_1}$. Similarly we have $\alpha(\partial_{H_2}(y)) \subseteq \overline{H_2}$, and hence also $\gamma(\alpha(\partial_{H_1}(x)), \alpha(\partial_{H_2}(y)) \cap H) \subseteq \gamma(\overline{H_1}, \overline{H_2} \cap H)$. Thus,

$$\partial_H(\partial_{H_1}(x) \parallel \partial_{H_2}(y)) = \partial_H(\partial_{H_1}(x) \parallel \partial_H\partial_{H_2}(y)) \quad \text{if } \gamma(\overline{H_1}, \overline{H_2} \cap H) \subseteq H \tag{2}$$

Then, by using the fact that $\overline{H_2} \cap H$ is equal to the set $H - H_2$, we have transformed CA1 into CAr1. The derivability of the other rules (CAr2–4) can be motivated in a similar way. ❑

The converse of Theorem 3.1 does not hold.

Theorem 3.2. *Axioms CA1–4 are not derivable in ACP_τ + CAr1–4 + CA5–7.*

Proof. For each $H \subseteq Act$ let X_H be the set of processes defined by the following abstract syntax

$$X_H ::= \mathit{Var} \mid X_H + X_H \mid X_H \cdot X_H \mid \tau_{H'}(X_H) \text{ with } H' \cap H = \varnothing$$

where *Var* is the set of process variables.

Next we introduce for each term p and for each $H \subseteq Act$ the boolean value $\{p\}_H$, which indicates whether p contains subexpressions of the form $\partial_H(x)$ with $x \in X_H$ and H non-empty:

$$\begin{array}{rcll} \{p\}_\varnothing & = & \bot & \text{for all terms } p \\ \{x\}_H & = & \bot & \text{for all } x \in \mathit{Var} \\ \{a\}_H & = & \bot & \text{for all } a \in \mathit{Act} \\ \{p \circ q\}_H & = & \{p\}_H \vee \{q\}_H & \text{for all terms } p, q \text{ and for all } \circ \in \{+, \cdot, \|\} \\ \{\tau_{H'}(p)\}_H & = & \{p\}_H & \end{array}$$

$$\{\partial_{H'}(p)\}_H = \begin{cases} \top & \text{if } H = H' \neq \varnothing \text{ and } p \in X_H \\ \{p\}_H & \text{otherwise} \end{cases}$$

Let LHS(φ) and RHS(φ) denote the left-hand-side and the right-hand-side of axiom φ respectively. We claim that

$$\{\mathrm{LHS}(\varphi)\sigma\}_H = \{\mathrm{RHS}(\varphi)\sigma\}_H \quad \text{for all } \varphi \in \mathrm{ACP}_\tau + \mathrm{CAr1{-}4} + \mathrm{CA5{-}7}$$

for all $H \subseteq Act$ and for all substitutions σ. As an illustration we work this out for $\varphi \equiv$ CAr1. Let $H_0 \subseteq Act$ be fixed and let $\sigma \equiv [t_x/x][t_y/y]$ be a substitution. We compute $\{\partial_H(\partial_{H_1}(t_x) \parallel \partial_{H_2}(t_y))\}_{H_0}$ and $\{\partial_H(\partial_{H_1}(t_x) \parallel \partial_H\partial_{H_2}(t_y))\}_{H_0}$. Suppose $\{\partial_{H_1}(t_x)\}_{H_0} = b_x$ and $\{\partial_{H_2}(t_y)\}_{H_0} = b_y$. Now

$$\{\partial_H(\partial_{H_1}(t_x) \parallel \partial_{H_2}(t_y))\}_{H_0} = \{\partial_{H_1}(t_x) \parallel \partial_{H_2}(t_y)\}_{H_0} = b_x \vee b_y$$

since $\partial_{H_1}(t_x) \parallel \partial_{H_2}(t_y) \notin X_{H_0}$. However, we also have that $\partial_{H_2}(t_y) \notin X_{H_0}$ so

$$\{\partial_H\partial_{H_2}(t_y)\}_{H_0} = \{\partial_{H_2}(t_y)\}_{H_0} = b_y.$$

Thus, both LHS(CAr1)σ and RHS(CAr1)σ are equal to $b_x \vee b_y$.

Now if for instance CA3, stating that $\partial_{H_0}(x) = x$ with $x \in \mathit{Var}$ is derivable, say

$$\partial_{H_0}(x) \equiv t_0 = t_1 = \ldots = t_{n-1} = t_n \equiv x$$

then $t_{i-1} = t_i$ are instances of axioms of ACP_τ + CAr1$-$4 + CA5$-$7 for all $i = 1, \ldots, n$. Thus, $\{t_{i-1}\}_{H_0} = \{t_i\}_{H_0}$ for all $i = 1, \ldots, n$ and in particular $\{t_0\}_{H_0} = \{t_n\}_{H_0}$. Contradiction, because $\{t_0\}_{H_0} = \{\partial_{H_0}(x)\}_{H_0} = \top$ and $\{t_n\}_{H_0} = \{x\}_{H_0} = \bot$. We conclude that CA3 is not derivable. Analogously we find that CA1 is not derivable. The non-derivability of CA2 and CA4 is shown by copying the proof with ∂ and τ exchanged. ❑

Note that each closed instance of the non-derivable axioms CA1$-$4 is derivable in ACP_τ + CAr1$-$4 + CA5$-$7. Thus, Theorem 3.2 means that ACP_τ + CAr1$-$4 + CA5$-$7 is a non-ω-complete system. However, considering axioms like $\partial_H(x) = x$, of which every closed instance is derivable, is interesting mainly because it limits the number of reduction steps in practical examples. In other words, ω-completeness is not our first concern.

Also, extending CAr1$-$4 in such a way that CA1$-$4 become derivable is problematic, at least if one wants to have side conditions that do not rely on the α-operator. In particular CA3 and CA4 heavily rely on this α-operator.

Moreover, it is still an open problem whether ACP + CA1$-$7 itself is ω-complete[7] so if we would reformulate the alphabet axioms in such a way that CA1$-$7 were derivable then still we would not have a ω-completeness result.

[7]The question of ω-completeness usually is rather complex (see [Gro90]).

4 Alphabet axioms for gates in μCRL

In ordinary μCRL it is not possible to let the result of encapsulating or hiding an action depend on the data parameter of that action. In other words: the encapsulation operator and the hide operator either rename none or all occurrences of an action. Thus, it is allowed to write expressions of the form $\partial_{\{a\}}(\sum_{j:nat} a(j))$ — which equals δ because all summands $a(j)$ are renamed to δ — but it is not allowed to write expressions of the form $\partial_{<a(i)>}(\sum_{j:nat} a(j))$ — where only one specific summand, namely a_i, is renamed to δ. Expressions of the latter form are used in the proof of Milner's Scheduler (see [KS93]). In more general terms such expressions are interesting because they allow for reasoning with infinite alphabets in a symbolic way. This will be treated in Section 5 and 6.

If one sticks to the ordinary encapsulation and hiding operators (which only rename gates), the alphabet axioms given in Section 3 can be formalized in μCRL simply by replacing the sets of actions H, H_1 and H_2 by sets of gates, as is shown in Table 5.[8] In this table we use the following conventions:

- The symbols $G, G_1, G_2 \subseteq \mathcal{G}$ stand for subsets of the set $\mathcal{G}$ of gates. Note that $\mathcal{G}$ is always a finite set because in μCRL one can only declare a finite set of gates.
- $\overline{G} = \mathcal{G} - G$ is the complement of G.
- For readability we write $F_{G_1} F'_{G_2}(x)$ for all $F_{G_1}(F'_{G_2}(x))$ for $F, F' \in \{\partial, \tau\}$.

Note that the side-conditions in Table 5 only involve finite set theory because a μCRL specification can not contain infinitely many gates. Thus, the side conditions can be computed effectively, i.e. one can write a computer program that checks them. This means that the axioms given in Table 5 are easily translated in a proof tool.

In order prove that adding CAG1−7 to μCRL does not result in an inconsistent system one has to show that each of the CAG equations is valid in the operational semantics of μCRL (see [GP95]). This exercise is rather tedious but straightforward. Since the operational semantics of μCRL is defined completely analogous to the operational semantics of ACP$_\tau$, the consistence of axioms CAr1−4 (Theorem 3.1) and CA5−7 in ACP$_\tau$ make the consistence of the 'corresponding' CAG axioms in μCRL plausible. Formally, however, the consistence of the CAG axioms is not justified by Theorem 3.1 since no results relating consistence of 'corresponding' axioms in μCRL and ACP$_\tau$ exist.

Lemma 4.1. *From the axioms in Table 5 the following two special rules are derivable:*

1. $\partial_G(\partial_{G_1}(x) \parallel y) = \partial_G(\partial_{G_1}(x) \parallel \partial_G(y))$ *if* $\overline{G}_1 \mid G \subseteq G$,
2. $\tau_G(\partial_{G_1}(x) \parallel y) = \tau_G(\partial_{G_1}(x) \parallel \tau_G(y))$ *if* $\overline{G}_1 \mid G = \varnothing$.

Proof. Using CAG1−2 and CAG3′. ❑

[8] These axioms were formulated in collaboration with Jan Friso Groote.

CAG1	$\partial_G(\partial_{G_1}(x) \parallel \partial_{G_2}(y)) = \partial_G(\partial_{G_1}(x) \parallel \partial_G\partial_{G_2}(y))$	if $\overline{G}_1 \mid (G - G_2) \subseteq G$
CAG2	$\tau_G(\partial_{G_1}(x) \parallel \partial_{G_2}(y)) = \tau_G(\partial_{G_1}(x) \parallel \tau_G\partial_{G_2}(y))$	if $\overline{G}_1 \mid (G - G_2) = \varnothing$
CAG3	$\partial_G(\partial_{G_1}(x) \parallel \partial_{G_2}(y)) = \partial_{G_1}(x) \parallel \partial_{G_2}(y)$	if $G \subseteq (G_1 \cap G_2) - (\overline{G}_1 \mid \overline{G}_2)$
CAG3′	$\partial_\varnothing(x) = x$	
CAG3″	$\tau_{G_1}\partial_{G_2}(x) = \partial_{G_2}(x)$	if $G_1 \subseteq G_2$
CAG4	$\tau_G(\tau_{G_1}(x) \parallel \tau_{G_2}(y)) = \tau_{G_1}(x) \parallel \tau_{G_2}(y)$	if $G \subseteq (G_1 \cap G_2) - (\overline{G}_1 \mid \overline{G}_2)$
CAG4′	$\tau_\varnothing(x) = x$	
CAG4″	$\partial_{G_1}\tau_{G_2}(x) = \tau_{G_2}(x)$	if $G_1 \subseteq G_2$
CAG5	$\partial_G(x) = \partial_{G_1}\partial_{G_2}(x)$	if $G = G_1 \cup G_2$
CAG6	$\tau_G(x) = \tau_{G_1}\tau_{G_2}(x)$	if $G = G_1 \cup G_2$
CAG7	$\tau_{G_1}\partial_{G_2}(x) = \partial_{G_2}\tau_{G_1}(x)$	if $G_1 \cap G_2 = \varnothing$

Table 5: Alphabet Axioms for Gates (CAG) in μCRL.

This lemma appears to be very useful in the verification example that is given in Section 8.

Note that for closed terms Axiom CAG3′ is derivable. E.g. if $x \equiv a \cdot a \parallel b$ with $a, b \in Act$, then

$$\partial_\varnothing(a \cdot a \parallel b) \stackrel{\text{D1}\,(3\times)}{=} \partial_\varnothing(\partial_\varnothing(a) \cdot \partial_\varnothing(a) \parallel \partial_\varnothing(b)) \stackrel{\text{D4}}{=} \partial_\varnothing(\partial_\varnothing(a \cdot a) \parallel \partial_\varnothing(b)) \stackrel{\text{CAG3}}{=}$$

$$\partial_\varnothing(a \cdot a) \parallel \partial_\varnothing(b) \stackrel{\text{D4}}{=} \partial_\varnothing(a) \cdot \partial_\varnothing(a) \parallel \partial_\varnothing(b) \stackrel{\text{D1}\,(3\times)}{=} a \cdot a \parallel b$$

More illustrations of the use of the axioms given in Table 5 can be found in Section 8. (Axioms D1 and D4 can be found in Table 11 of the appendix.)

5 Refining encapsulation and hiding in μCRL

In the previous section we gave alphabet axioms for gates in μCRL. These axioms enable us to encapsulate infinitely many actions, whereas the side conditions only use finitary set theory. In order to have the same flexibility as in ACP$_\tau$, however, we should also be able to encapsulate one or more specific elements of a set $\{a(d) : d \in D\}$ of actions. For instance in the proof of Milner's Scheduler (see [KS93]), we want to rename only those occurrences of a in a process p of the form $a(i)$. In other words, we would like to be able to write expressions like $\partial_{<a(i)>}(\sum_{j:nat} a(j))$. (For technical reasons we use the symbols $<\ >$ instead of brackets $\{\ \}$.) This can be done by extending the syntax of μCRL (see Appendix 1.1) as follows:

$$p \quad ::= \quad \ldots \mid \partial_{<a\text{-}list>}(p) \mid \tau_{<a\text{-}list>}(p)$$

$$a\text{-}list \quad ::= \quad a \mid a\text{-}list,\, a$$

$$a \quad ::= \quad n(t_1, \ldots, t_n)$$

where n is a gate and t_i stand for data terms. The new operators are distinguished from the original encapsulation operator and hiding operator (which are only defined on gates) by the symbols '$<$' and '$>$' for notating sets instead of brackets. Note that '$<$' and '$>$' have now become reserved names in the μCRL syntax.

The rules in Table 6 state that the operators introduced above have the desired properties, for example that $\partial_{<a(1)>}(a(1) + a(2)) = a(2)$. In this table, we use the following conventions:

- A is a *finite* subset of Act. Note that Act may be an infinite set, because actions may be parameterized by data domains containing infinitely many terms (as in the example μCRL specification on page 19)
- $\bigwedge\limits_{a \in A}$ is a shorthand for a finite conjunction and $\bigvee\limits_{a \in A}$ for a finite disjunction over actions. We adopt the convention that $\bigwedge\limits_{a \in \varnothing}$ stands for $\neg\mathcal{F}$ and $\bigvee\limits_{a \in \varnothing}$ stands for $\mathcal{F}$, where $\mathcal{F}$ stands for the false μCRL proposition.

The rules in Table 6 resemble the original rules D1−4 and TI1−4, but differ on a delicate detail. In Table 6 the conditions are formulated within the proof rule whereas in the older rules for gates the conditions are given outside the rule at the right-hand side of the table. This is due to the fact that the conditions of the latter rules do not involve actions that are parameterized with data variables. So these conditions are not subject to the internal substitution mechanism of the proof system.

In analogue with the standard μCRL axioms SUM8 and SUM9, we need similar axioms for the new encapsulation and hiding operator such as given in Table 7. In this table $\bar{t}$ stands for a sequence $t_1, \ldots, t_n$ $(n \geq 1)$ of data terms. The usefulness of these axioms (in particular SUM8′) is illustrated in the next proposition.

Proposition 5.1. *In each specification containing the fragment given in Table 8 the following two identities are derivable.*

1. $\partial_{<a(i)>}(\delta \lhd eq(i,j) \rhd a(j)) = \delta \lhd eq(i,j) \rhd a(j)$
2. $\partial_{<a(i)>}(\sum_{j:nat} a(j)) = \sum_{j:nat}(\delta \lhd eq(i,j) \rhd a(j))$.

Proof.

1. (a) If we assume that we have derivation for $eq(i,j) = \mathsf{t}$ then the lemma follows immediately: $\partial_{<a(i)>}(\delta \lhd eq(i,j) \rhd a(j)) = \partial_{<a(i)>}(\delta) \overset{\text{DAD}}{=} \delta = \delta \lhd \mathsf{t} \rhd a(j) = \delta \lhd eq(i,j) \rhd a(j)$.

   (b) If we assume that we have derivation for $eq(i,j) = \mathsf{f}$ then the precondition of DA1 holds, as the following μCRL derivation shows. (We omitted the derivation of $eq(i,i) = \mathsf{t}$ which follows from $eq(0,0) = \mathsf{t}$ and $eq(k,k) = \mathsf{t} \rightarrow eq(S(k), S(k)) = \mathsf{t}$ by induction.)

DAD	$\partial_{<a>}(\delta) = \delta$	
DAT	$\partial_{<a>}(\tau) = \tau$	
DA0	$\partial_A(x) = \partial_{A'}\partial_{<a>}(x)$	if $A = A' \cup \{a\}$
DA1	$\neg(t_i = u_i) \to \partial_{<n(t_1,\ldots,t_n)>} m(u_1,\ldots,u_n) = m(u_1,\ldots,u_n)$	$1 \leq i \leq n$
DA1$'$	$\partial_{<n_1(t_1,\ldots,t_n)>} n_2(u_1,\ldots,u_m) = n_2(u_1,\ldots,u_m)$	if $n_1 \not\equiv n_2$
DA1$''$	$\partial_{<n_1(t_1,\ldots,t_n)>} n_2(u_1,\ldots,u_m) = n_2(u_1,\ldots,u_m)$	if $n \neq m$
DA2	$\partial_{<n(t_1,\ldots,t_n)>} n(t_1,\ldots,t_n) = \delta$	
DA3	$\partial_A(x+y) = \partial_A(x) + \partial_A(y)$	
DA4	$\partial_A(x \cdot y) = \partial_A(x) \cdot \partial_A(y)$	
TAD	$\tau_{<a>}(\delta) = \delta$	
TAT	$\tau_{<a>}(\tau) = \tau$	
TA0	$\tau_A(x) = \tau_{A'}\tau_{<a>}(x)$	if $A = A' \cup \{a\}$
TA1	$\neg(t_i = u_i) \to \tau_{<n(t_1,\ldots,t_n)>} m(u_1,\ldots,u_n) = m(u_1,\ldots,u_n)$	$1 \leq i \leq n$
TA1$'$	$\tau_{<n_1(t_1,\ldots,t_n)>} n_2(u_1,\ldots,u_m) = n_2(u_1,\ldots,u_m)$	if $n_1 \not\equiv n_2$
TA1$''$	$\tau_{<n_1(t_1,\ldots,t_n)>} n_2(u_1,\ldots,u_m) = n_2(u_1,\ldots,u_m)$	if $n \neq m$
TA2	$\tau_{<n(t_1,\ldots,t_n)>} n(t_1,\ldots,t_n) = \tau$	
TA3	$\tau_A(x+y) = \tau_A(x) + \tau_A(y)$	
TA4	$\tau_A(x \cdot y) = \tau_A(x) \cdot \tau_A(y)$	

Table 6: Axioms for encapsulating and hiding parametrized actions.

SUM8$'$	$\partial_{<n(\overline{t})>} \sum_{d:D}(p) = \sum_{d:D} \partial_{<n(\overline{t})>}(p)$	if d does not occur in $\overline{t}$
SUM9$'$	$\tau_{<n(\overline{t})>} \sum_{d:D}(p) = \sum_{d:D} \tau_{<n(\overline{t})>}(p)$	if d does not occur in $\overline{t}$

Table 7: Additional axioms for summation.

$$\dfrac{\dfrac{\dfrac{\dfrac{eq(i,j)=\mathsf{f} \qquad \dfrac{\dfrac{[\neg\neg i=j] \qquad [\neg i=j]}{\mathcal{F}}\,\to\mathrm{E}}{i=j}\,\mathrm{RAA}}{eq(i,i)=\mathsf{f}}\,\mathrm{REPL} \qquad \dfrac{\vdots}{eq(i,i)=\mathsf{t}}\,\mathrm{IND}}{\mathsf{t}=\mathsf{f}}\,\mathrm{REPL} \qquad \dfrac{}{\neg(\mathsf{t}=\mathsf{f})}\,\mathrm{BOOL1}}{\mathcal{F}}\,\to\mathrm{E}}{\neg(i=j)}\,\mathrm{RAA}$$

Now $\partial_{<a(i)>}(\delta \triangleleft eq(i,j) \triangleright a(j)) = \partial_{<a(i)>}(a(j)) \stackrel{\mathrm{DA1}}{=} a(j) = \delta \triangleleft \mathsf{f} \triangleright a(j) =$

sort	nat
func	$0 :\rightarrow nat$
	$S : nat \rightarrow nat$
var	$i, j : nat$
rew	$eq(0,0) = \mathsf{t}$
	$eq(S(0),0) = \mathsf{f}$
	$eq(0,S(0)) = \mathsf{f}$
	$eq(S(i),S(j)) = eq(i,j)$
act	$a : nat$

Table 8: Fragment of a μCRL specification.

$\delta \lhd eq(i,j) \rhd a(j)$.
Now, the desired result is obtained by using that either (a) or (b) holds, i.e.

$$(b = \mathsf{t} \rightarrow \phi) \rightarrow (b = \mathsf{f} \rightarrow \phi) \rightarrow \phi$$

which can easily be proved by induction on b.

2. In the proof below, we use the identity

$$\sum_{j:nat} a(j) = \sum_{j:nat} (\delta \lhd eq(i,j) \rhd a(j)) + a(i) \tag{3}$$

which is an instance of a basic μCRL lemma proven in [GP91]. This lemma expresses that the sum operator denotes the alternative composition of all data instances of a process term.

$$\begin{aligned}
&\partial_{<a(i)>} \textstyle\sum_{j:nat} a(j) \\
&\stackrel{(3)}{=} \quad \partial_{<a(i)>}(\textstyle\sum_{j:nat}(\delta \lhd eq(i,j) \rhd a(j)) + a(i)) \\
&\stackrel{\text{DA3}}{=} \quad \partial_{<a(i)>}(\textstyle\sum_{j:nat}(\delta \lhd eq(i,j) \rhd a(j))) + \partial_{<a(i)>}(a(i)) \\
&\stackrel{\text{DA2}}{=} \quad \partial_{<a(i)>} \textstyle\sum_{j:nat}(\delta \lhd eq(i,j) \rhd a(j)) + \delta \\
&\stackrel{\text{SUM8}'}{=} \quad \textstyle\sum_{j:nat} \partial_{<a(i)>}(\delta \lhd eq(i,j) \rhd a(j)) \\
&\stackrel{5.1.1}{=} \quad \textstyle\sum_{j:nat}(\delta \lhd eq(i,j) \rhd a(j))
\end{aligned}$$

❑

Note that the axioms in Table 7 are formulated with singleton sets for readability. It is obvious that these rules can be generalized to arbitrary sets by repeated application of the axioms DA0 and TA0 given in Table 6. For example, see the lemma below.

Lemma 5.2. *Let $A \subseteq Act$ be a finite set of actions.*

1. $\partial_A \sum_{d:D}(p) = \sum_{d:D} \partial_A(p)$ *if d does not occur in $\bar{t}$ for all $n(\bar{t}) \in A$*
2. $\tau_A \sum_{d:D}(p) = \sum_{d:D} \tau_A(p)$ *if d does not occur in $\bar{t}$ for all $n(\bar{t}) \in A$*

6 Alphabet axioms for parametrized actions

In Section 4, the alphabet axioms were formulated for gates in μCRL. In this section, we formulate such axioms for parametrized actions. These (conditional) axioms are presented in Table 9.

CAA1	$\left(\bigwedge_{n' \in Part^*(n)} \partial_{<n'(\bar{t})>}(x) = x\right) \to \partial_{<n(\bar{t})>}(x \parallel y) = \partial_{<n(\bar{t})>}(x \parallel \partial_{<n(\bar{t})>}(y))$
CAA2	$\left(\bigwedge_{n' \in Part(n)} \partial_{<n'(\bar{t})>}(x) = x\right) \to \tau_{<n(\bar{t})>}(x \parallel y) = \tau_{<n(\bar{t})>}(x \parallel \tau_{<n(\bar{t})>}(y))$
CAA3	$\partial_A(\partial_A(x) \parallel \partial_A(y)) = \partial_A(x) \parallel \partial_A(y)$ if $gate(A) \cap range(\gamma) = \varnothing$
CAA4	$\tau_A(\tau_A(x) \parallel \tau_A(y)) = \tau_A(x) \parallel \tau_A(y)$ if $gate(A) \cap range(\gamma) = \varnothing$
CAA7	$\neg(a = b) \to \tau_{<a>}\partial_{<b>}(x) = \partial_{<b>}\tau_{<a>}(x)$

Table 9: Conditional Axioms for parametrized Actions (CAA).

In Table 9 the following notation is used:

- $a, a_1, \ldots, a_n, b$ range over Act and n, n' range over $\mathcal{G}$.
- $Part(n)$ is the set of all gates that are possible communication partners of gate n. $Part^*(n)$ is the set of all gates n' that can communicate with n such that the resulting gate n'' is not equal to gate n. In formulas:

  $$Part(n) \stackrel{\text{def}}{=} \{n' \mid \exists n'' : \gamma(n, n') = n''\}$$

  $$Part^*(n) \stackrel{\text{def}}{=} \{n' \mid \exists n'' : \gamma(n, n') = n'' \quad \wedge \quad n'' \not\equiv n\}$$

  where γ is the symmetrical closure of the auxiliary pre-communication function $\tilde{\gamma}$, which is defined such that $\tilde{\gamma}(n_1, n_2) = n_3$ if a rule **comm** $n_1 \mid n_2 = n_3$ appears in the μCRL specification in question.

  For example, in the specification of a bag given in Section 8, $Part(s_2) = Part^*(s_2) = \{r_2\}$.

- *range* is the range of the communication function γ. For example, in the specification of a bag given in Section 8, this is $\{c_2, c_3\}$.
- $gate(A)$ is the set of gates that are related to the set of actions A. For instance,
$$gate(< r(i), s(i) >) = \{r, s\}.$$
- $\bar{t}$ stands for a finite list $t_1, \ldots, t_n$.

Due to the presence of data parameters, these axioms can be used for simulating the traditional alphabet axioms with infinite alphabets (see Section 8). Note however that the presence of data parameters forced us to formulate side conditions as premises. For instance if we consider CAG2 versus CAA2 then we see that the side condition $\overline{G_1} \mid (G - G_2) = \varnothing$, which expresses that those gates that occur in the left argument of the parallel composition must not[9] communicate with gates of G that occur in the right argument of the parallel composition, is translated to $\bigwedge_{n' \in Part(n)} \partial_{<n'(\bar{t})>}(x) = x$. This latter expression expresses the same condition, except that we restricted the encapsulation to one gate, namely n. If we instantiate G in CAG2 with $\{n\}$ we find that gates that occur in the left argument of the parallel composition must not communicate with n — i.e. are not in $part(n)$ — provided that n occurs in the right argument of the parallel composition. Thus, the precondition in CAA2 is slightly stronger than needed because the proviso that n should occur in y is ignored. A similar thing can be said about CAA1. For our purposes the CAA-axioms in its present form are strong enough.

Because the side-conditions in Table 9 are effectively computable (one could check them by a computer program) we did not develop a formal proof theory for these functions.

7 Linking CAG with CAA

In Table 10, we present four axioms that form a bridge between the axiom systems CAG and CAA. Recall that $G \subseteq \mathcal{G}$ is a finite set of gates, $A \subseteq Act$ is a finite set of actions, and $gate(A)$ is the set of gates related to the action set A.

CAGA1	$\partial_G(x) = \partial_G \partial_A(x)$	if $gate(A) \subseteq G$
CAGA2	$\tau_G(x) = \tau_G \tau_A(x)$	if $gate(A) \subseteq G$
CAGA3	$\partial_G \partial_A(x) = \partial_A \partial_G(x)$	
CAGA4	$\tau_G \tau_A(x) = \tau_A \tau_G(x)$	

Table 10: Axioms for linking Gates and Actions (CAGA).

[9] Otherwise, a fruitful communication could be lost

8 An example with bags

In this section, the axioms given in the previous sections are used for proving some properties about bags, for instance that two connected bags form a bag. The example illustrates the use of both the CAG and the CAA axioms. A larger example which uses CAA axioms can be found in [KS93].

A bag which reads from a channel i and writes on a channel j is defined as follows:

act $r, s, c : I \times D$
comm $r \mid s = c$
proc $B(i, j : Y) = \sum_{d:D}(r(i,d) \cdot (s(j,d) \parallel B(i,j)))$

where $Y = \{1,2,3\}$ and D represents the type of information that is in the bag.

Part 5 of the theorem below says that two connected bags form a bag. In the last part (6) it is stated that also three connected bags form a bag. The rest of the theorem consists of auxiliary statements.

Theorem 8.1. *For each $y \in Y$ let $H_y = \{r(y,d), s(y,d) : d \in D\}$, $I_y = \{c(y,d) : d \in D\}$, $H = H_2 \cup H_3$ and $I = I_2 \cup I_3$.*

1. $\partial_{H_3}(B(1,2)) = B(1,2)$ *(used in 2 and 6)*
2. $\partial_{<s(3,d)>}(B(1,2)) = B(1,2)$ *(used in 3)*
3. $c(2,d) \cdot s(3,d) \parallel B(2,3) = \partial_{<s(2,d)>}(s(2,d) \parallel B(2,3))$ *(used in 4)*
4. $\partial_{H_2}(B(1,2) \parallel B(2,3)) = \sum_{d:D}(r(1,d) \cdot (c(2,d) \cdot s(3,d) \parallel \partial_{H_2}(B(1,2) \parallel B(2,3))))$ *(used in 5)*
5. $\tau_{I_2}\partial_{H_2}(B(1,2) \parallel B(2,3)) = B(1,3)$ *(used in 6)*
6. $\tau_I \partial_H(B(1,2) \parallel B(2,3) \parallel B(3,4)) = B(1,4)$

Although the main idea of the proof below is borrowed from [BBK87], there are three points of difference. First, the proof given here does not depend on meta-theory whereas the verification in [BBK87] relies on an informal treatment of data parameters (e.g. sum operator) and implicit set theory. Second, our proof given below is ready to be proof-checked conform standard methods (see [CH88, Sel93]). For a computer-checked verification involving a part of the alphabet axioms given in this paper, one is referred to [KS93]. Third, the proof given here is a generalization of the one given in [BBK87] because there the data domain D is assumed to be finite and here it may also be infinite, e.g. allowing D to be the set of natural numbers.

Proof. In the proofs below we use, besides the alphabet axioms presented in the previous sections, several axioms (e.g. SUM8, D1, TI1) from standard μCRL which can be found in Appendix 1. A step in the proof is labelled with SC, which is an acronym for Standard Concurrency, in case one or more of the axioms given in Table 12 of the appendix are applied.

1. $\partial_{H_3}(B(1,2))$
$$\begin{aligned} &= \partial_{H_3}(\textstyle\sum_{d:D}(r(1,d) \cdot (s(2,d) \parallel B(1,2)))) \\ &\overset{\text{SUM8}}{=} \textstyle\sum_{d:D}(\partial_{H_3}(r(1,d) \cdot (s(2,d) \parallel B(1,2)))) \end{aligned}$$

$$\begin{array}{cl}
\stackrel{\text{D1}}{=} & \sum_{d:D}(r(1,d)\cdot\partial_{H_3}(s(2,d)\parallel B(1,2)))\\
\stackrel{\text{D1}}{=} & \sum_{d:D}(r(1,d)\cdot\partial_{H_3}(\partial_{H_3}(s(2,d))\parallel B(1,2)))\\
\stackrel{\text{lemma 4.1}}{=} & \sum_{d:D}(r(1,d)\cdot\partial_{H_3}(\partial_{H_3}(s(2,d))\parallel \partial_{H_3}(B(1,2)))) \quad \text{(see note I)}\\
\stackrel{\text{CAG3}}{=} & \sum_{d:D}(r(1,d)\cdot(\partial_{H_3}(s(2,d))\parallel \partial_{H_3}(B(1,2)))) \qquad \text{(see note II)}\\
\stackrel{\text{D1}}{=} & \sum_{d:D}(r(1,d)\cdot(s(2,d)\parallel \partial_{H_3}(B(1,2))))
\end{array}$$

Thus we see that $\partial_{H_3}(B(1,2))$ is a solution for $B(1,2)$. Therefore, by the Recursive Specification Principle (RSP), we have that $\partial_{H_3}(B(1,2)) = B(1,2)$.

This finishes the proof of Theorem 8.1.1.

Note I: $\overline{H}_3 \mid H_3 = \{r(1,d), r(2,d), s(1,d), s(2,d), c(2,d), c(3,d) : d \in D\} \mid \{r(3,d), s(3,d) : d \in D\} = \varnothing \subseteq H_3$.

Note II: $H_3 \subseteq (H_3 \cap H_3) - (\overline{H}_3 \mid \overline{H}_3) = \{r(3,d), s(3,d) : d \in D\} - \{c(2,d) : d \in D\} = H_3$.

In note (II) we have verified the side-condition of axiom CAG3. Note that all the side-conditions in our proof system are effectively computable. For instance, in the side-condition of Table 5 only finite sets are involved. This means that a proof tool can check these conditions fully automatically. To guide the intuition we have checked the side-conditions (informally) by hand by using very simple finite set theory.

2. $\partial_{<s(3,d)>}(B(1,2))$

$$\begin{array}{cl}
\stackrel{(1)}{=} & \partial_{<s(3,d)>}(\partial_{H_3}(B(1,2)))\\
\stackrel{\text{CAGA3}}{=} & \partial_{H_3}(\partial_{<s(3,d)>}(B(1,2)))\\
\stackrel{\text{CAGA1}}{=} & \partial_{H_3}(B(1,2))\\
\stackrel{(1)}{=} & B(1,2)
\end{array}$$

3. On the one hand we have,

$$\begin{array}{cl}
\multicolumn{2}{l}{c(2,d)s(3,d)\parallel B(2,3)}\\
= & c(2,d)\cdot(s(3,d)\parallel B(2,3)) + \sum_{e:D}(r(2,e)\cdot(c(2,d)s(3,d)\parallel s(3,e)\parallel B(2,3)))\\
\stackrel{\text{SC}}{=} & c(2,d)\cdot(s(3,d)\parallel B(2,3)) + \sum_{e:D}(r(2,e)\cdot(s(3,e)\parallel (c(2,d)s(3,d)\parallel B(2,3))))
\end{array}$$

and on the other hand we have,

$$\begin{array}{cl}
\multicolumn{2}{l}{\partial_{<s(2,d)>}(s(2,d)\parallel B(2,3))}\\
= & c(2,d)\cdot\partial_{<s(2,d)>}(s(3,d)\parallel B(2,3)) +\\
 & \sum_{e:D}(r(2,e)\cdot\partial_{<s(2,d)>}(s(3,e)\parallel s(2,d)\parallel B(2,3)))\\
\stackrel{\text{CAA3}}{=} & c(2,d)\cdot(s(3,d)\parallel B(2,3))\\
+ & \sum_{e:D}(r(2,e)\cdot\partial_{<s(2,d)>}(s(3,e)\parallel s(2,d)\parallel B(2,3))) \text{ (see note I)}\\
\stackrel{\text{CAA1}}{=} & c(2,d)\cdot(s(3,d)\parallel B(2,3))
\end{array}$$

$$
\begin{array}{ll}
+ & \sum_{e:D}(r(2,e)\cdot\partial_{<s(2,d)>}(s(3,e) \parallel \partial_{<s(2,d)>}(s(2,d) \parallel B(2,3)))) \text{ (see note II)} \\
\overset{\text{CAA3}}{=} & c(2,d)\cdot(s(3,d) \parallel B(2,3)) \\
+ & \sum_{e:D}(r(2,e)\cdot(s(3,e) \parallel \partial_{<s(2,d)>}(s(2,d) \parallel B(2,3))) \text{ (see note III)}
\end{array}
$$

Thus we see that both $c(2,d)s(3,d) \parallel B(2,3)$ and $\partial_{<s(2,d)>}(s(2,d) \parallel B(2,3))$ satisfy the guarded equation:

$$X = c(2,d)(s(3,d) \parallel B(2,3)) + \sum_{e:D}(r(2,e)\cdot(s(3,e) \parallel X)).$$

Therefore, by RSP, we have

$$c(2,d)s(3,d) \parallel B(2,3) = \partial_{<s(2,d)>}(s(2,d) \parallel B(2,3)).$$

This finishes the proof of Theorem 8.1.3.

Note I: Using the facts $\partial_{<s(2,d)>}(s(3,d)) = s(3,d)$ and $\partial_{<s(2,d)>}(B(2,3)) = B(2,3)$. The first identity directly follows from DA1 and the other one can be proven analogous to (2).

Note II: Because $\bigwedge_{n'\in Part^*(s)}\partial_{<n'(2,d)>}(s(3,d)) = \partial_{<r(2,d)>}(s(3,d)) = s(3,d)$ we may apply CAA1.

Note III: Since $\neg(2{=}3)$ we can — according to DA1 — replace $s(3,e)$ by $\partial_{<s(2,d)>}(s(3,e))$. Now CAA3 can be applied because $\{s(2,d) : d \in D\} \cap \{c(2,d), c(3,d) : d \in D\} = \varnothing$.

4. $\partial_{H_2}(B(1,2) \parallel B(2,3))$

$$
\begin{array}{ll}
= & \sum_{d:D}(r(1,d)\cdot\partial_{H_2}(s(2,d) \parallel B(1,2) \parallel B(2,3))) \\
\overset{\text{CAGA1, SC}}{=} & \sum_{d:D}(r(1,d)\cdot\partial_{H_2}\partial_{<s(2,d)>}(B(1,2) \parallel (s(2,d) \parallel B(2,3)))) \\
\overset{\text{CAA1}}{=} & \sum_{d:D}(r(1,d)\cdot\partial_{H_2}\partial_{<s(2,d)>}(B(1,2) \parallel \partial_{<s(2,d)>}(s(2,d) \parallel B(2,3)))) \text{ (see note I)} \\
\overset{(3)}{=} & \sum_{d:D}(r(1,d)\cdot\partial_{H_2}(B(1,2) \parallel c(2,d)s(3,d) \parallel B(2,3))) \\
\overset{\text{SC}}{=} & \sum_{d:D}(r(1,d)\cdot\partial_{H_2}(c(2,d)s(3,d) \parallel (B(1,2) \parallel B(2,3)))) \\
\overset{\text{CAG1}}{=} & \sum_{d:D}(r(1,d)\cdot\partial_{H_2}(c(2,d)s(3,d) \parallel \partial_{H_2}(B(1,2) \parallel B(2,3)))) \text{ (see note II)} \\
\overset{\text{CAG3}}{=} & \sum_{d:D}(r(1,d)\cdot(c(2,d)s(3,d) \parallel \partial_{H_2}(B(1,2) \parallel B(2,3)))) \text{ (see note III)}
\end{array}
$$

This finishes the proof of Theorem 8.1.4.

Note I: Using $\bigwedge_{n'\in Part^*(s)}\partial_{<n'(2,d)>}(B(1,2)) = \partial_{<r(2,d)>}(B(1,2)) = B(1,2)$ where the last step goes analogous to (2).

Note II: It is easy to see that ∂_{H_2} is the identity for $c(2,d)s(3,d)$. This together with the fact that $\overline{H}_2 \mid H_2 - \varnothing = \varnothing \subseteq H_2$ implies that CAG1 may be applied.

Note III: Using the fact that ∂_{H_2} is the identity for $c(2,d)s(3,d)$, and $H_2 \subseteq H_2 \cap H_2 - (\overline{H}_2 \mid \overline{H}_2) = H_2 - \varnothing = H_2$.

5. $\tau_{I_2}\partial_{H_2}(B(1,2) \parallel B(2,3))$

$$
\begin{array}{ll}
\overset{(4)}{=} & \tau_{I_2}(\sum_{d:D}(r(1,d)\cdot(c(2,d)s(3,d) \parallel \partial_{H_2}(B(1,2) \parallel B(2,3)))))
\end{array}
$$

$$
\begin{aligned}
&\overset{\text{SUM9,TI1}}{=} && \textstyle\sum_{d:D}(r(1,d)\cdot\tau_{I_2}(c(2,d)s(3,d)\parallel\partial_{H_2}(B(1,2)\parallel B(2,3))))\\
&\overset{\text{CAG2}}{=} && \textstyle\sum_{d:D}(r(1,d)\cdot\tau_{I_2}(\tau_{I_2}(c(2,d)s(3,d))\parallel\tau_{I_2}\partial_{H_2}(B(1,2)\parallel B(2,3))))\\
&\overset{\text{CAG4}}{=} && \textstyle\sum_{d:D}(r(1,d)\cdot(\tau_{I_2}(c(2,d)s(3,d))\parallel\tau_{I_2}\partial_{H_2}(B(1,2)\parallel B(2,3))))\\
&\overset{\text{TAI1}-2}{=} && \textstyle\sum_{d:D}(r(1,d)\cdot(\tau\cdot s(3,d)\parallel\tau_{I_2}\partial_{H_2}(B(1,2)\parallel B(2,3))))\\
&= && \textstyle\sum_{d:D}(r(1,d)\cdot(s(3,d)\parallel\tau_{I_2}\partial_{H_2}(B(1,2)\parallel B(2,3))))\ \text{(see note I)}
\end{aligned}
$$

Therefore the process $\tau_{I_2}\partial_{H_2}(B(1,2)\parallel B(2,3))$ is a solution for the defining equations of $B(1,3)$. By RSP we then have that $B(1,3)=\tau_{I_2}\partial_{H_2}(B(1,2)\parallel B(2,3))$.

This finishes the proof of Theorem 8.1.5.

<u>Note I:</u> $r(1,d)(\tau s(3,d)\parallel x) = r(1,d)\tau s(3,d) \mathbin{\lfloor\!\lfloor} x = r(1,d)s(3,d)\mathbin{\lfloor\!\lfloor} x = r(1,d)(s(3,d)\parallel x)$.

6. $\tau_I\partial_H(B(1,2)\parallel B(2,3)\parallel B(3,4))$

$$
\begin{aligned}
&\overset{\text{CAG5}-6}{=} && \tau_{I_2}\tau_{I_3}\partial_{H_2}\partial_{H_3}(B(1,2)\parallel B(2,3)\parallel B(3,4))\\
&\overset{\text{CAG7}}{=} && \tau_{I_2}\partial_{H_2}\tau_{I_3}\partial_{H_3}(B(1,2)\parallel B(2,3)\parallel B(3,4))\\
&\overset{(1)}{=} && \tau_{I_2}\partial_{H_2}\tau_{I_3}\partial_{H_3}(\partial_{H_3}(B(1,2))\parallel B(2,3)\parallel B(3,4))\\
&\overset{\text{Lemma 4.1}}{=} && \tau_{I_2}\partial_{H_2}\tau_{I_3}\partial_{H_3}(\partial_{H_3}(B(1,2))\parallel\partial_{H_3}(B(2,3)\parallel B(3,4)))\ \text{(see note I)}\\
&\overset{\text{CAG3}}{=} && \tau_{I_2}\partial_{H_2}\tau_{I_3}(\partial_{H_3}(B(1,2))\parallel\partial_{H_3}(B(2,3)\parallel B(3,4)))\text{(see note II)}\\
&\overset{(1)}{=} && \tau_{I_2}\partial_{H_2}\tau_{I_3}(B(1,2)\parallel\partial_{H_3}(B(2,3)\parallel B(3,4)))\\
&= && \tau_{I_2}\partial_{H_2}\tau_{I_3}(\tau_{I_3}(B(1,2))\parallel\partial_{H_3}(B(2,3)\parallel B(3,4)))\ \text{(see note III)}\\
&\overset{\text{Lemma 4.1}}{=} && \tau_{I_2}\partial_{H_2}\tau_{I_3}(\tau_{I_3}(B(1,2))\parallel\tau_{I_3}\partial_{H_3}(B(2,3)\parallel B(3,4)))\ \text{(see note IV)}\\
&\overset{\text{CAG3}''}{=} && \tau_{I_2}\partial_{H_2}\tau_{I_3}(\tau_{I_3}(B(1,2))\parallel\tau_{I_3}\tau_{H_3}\partial_{H_3}(B(2,3)\parallel B(3,4)))\\
&\overset{\text{CAG6}}{=} && \tau_{I_2}\partial_{H_2}\tau_{I_3}(\tau_{I_3}(B(1,2))\parallel\tau_{I_3\cup H_3}\partial_{H_3}(B(2,3)\parallel B(3,4)))\\
&\overset{\text{CAG4}}{=} && \tau_{I_2}\partial_{H_2}(\tau_{I_3}(B(1,2))\parallel\tau_{I_3}\partial_{H_3}(B(2,3)\parallel B(3,4)))\ \text{(see note V)}\\
&= && \tau_{I_2}\partial_{H_2}(B(1,2)\parallel\tau_{I_3}\partial_{H_3}(B(2,3)\parallel B(3,4)))\qquad\text{(see note III)}\\
&\overset{(5)}{=} && \tau_{I_2}\partial_{H_2}(B(1,2)\parallel B(2,4))\\
&\overset{(5)}{=} && B(1,4)
\end{aligned}
$$

This finishes the proof of Theorem 8.1.6.

<u>Note I:</u> $\overline{H_3}\mid H_3=\{r(1,d),r(2,d),s(1,d),s(2,d),c(2,d),c(3,d):d\in D\}\mid\{r(3,d),s(3,d):d\in D\}=\varnothing\subseteq H_3$.
<u>Note II:</u> $H_3\subseteq(H_3\cap H_3)-(\overline{H_3}\mid\overline{H_3})=\{r(3,d),s(3,d):d\in D\}-\{c(2,d):d\in D\}=H_3$.
<u>Note III:</u> Using that $\tau_{I_3}(B(1,2))=B(1,2)$; the proof of this identity goes analogous to (1).
<u>Note IV:</u> $\overline{I_3}\mid I_3=\{r(1,d),r(2,d),r(3,d),s(1,d),s(2,d),s(3,d),s(4,d),c(1,d),c(2,d):d\in D\}\mid\{c(3,d):d\in D\}=\varnothing$.
<u>Note V:</u> $I_3\subseteq I_3\cap(I_3\cup H_3)-(\overline{I_3}\mid\overline{I_3\cup H_3})=I_3-\{c(1,d),c(2,d):d\in D\}=I_3$.

❑

9 Concluding Remarks

We have shown how to bring the alphabet axioms (or static laws) as used in ACP and CCS down to a completely formal level in μCRL. An important implication of this work is that the new axioms are highly suited for computer-checked verification. For instance, in [KS93] we have used these axioms for proof checking Milner's Scheduler. Moreover, we have obtained a completely formal and self-contained version of the 'bag' verification by Baeten, Bergstra and Klop [BBK87].

At the moment, some interesting related work is going on. In particular, in [Kor94] it is shown that Theorem 8.1.5 can be proved in μCRL without alphabet axioms at the price of adding an extended version of RSP [BW90, GP93] called CL-RSP introduced by Groote and Bezem (see [BG94]).[10] These kind of results may indicate that alphabet axioms are theoretically superfluous in the context of CL-RSP. But of course alphabet axioms remain useful and interesting in their own right. For example, these axioms appear to be very natural in proving identities like the one stated in Theorem 8.1.6. A more serious example showing the importance of the use of alphabet axioms is Milner's Scheduler [Mil89, KS93]. It has to be investigated whether the correctness of this protocol can be proved in an elegant way without alphabet axioms, using CL-RSP instead.

Acknowledgements

We would like to thank Jan Bergstra, Jan Friso Groote, Kees Middelburg, Alban Ponse, Jan Springintveld and Frits Vaandrager for feedback and suggestions.

1 An overview of the proof theory for μCRL

See [GP95, GP93, GP91] for more details on μCRL.

1.1 The syntax of μCRL.

First, we assume the existence of a set $\mathcal{N}$ of *names* that are used to denote sorts, variables, functions, processes and gates of actions. The names in $\mathcal{N}$ are words over an alphabet not containing

$$\perp \; + \; \| \; \underline{\|} \; | \; \lhd \; \rhd \; \cdot \; \delta \; \tau \; \partial \; \rho \; \sum \; \surd \; \times \; \rightarrow \; := \;) \; (\; \} \; \{ \; , \; \texttt{space} \; \texttt{newline} \; \texttt{t} \; \texttt{f}$$

The space and the newline serve as separators between names and are used for the layout of specifications. The other symbols have special functions. Moreover, $\mathcal{N}$ does not contain the reserved keywords **sort**, **proc**, **var**, **act**, **func**, **comm**, **rew** and **from**.

Data types are specified as the standard abstract data types [EM85], using sorts, functions and axioms. Sorts are declared using the keyword **sort** and functions are declared using the keyword **func**. Axioms are declared using the keyword **rew**, referring to the possibility to use rewriting technology for

[10] RSP and CL are acronyms for Recursive Specification Principle and Convergent Linear, respectively.

evaluation of abstract data types. The variables that are used in the axioms must be declared directly before the axioms. Their scope only extends to the next single **rew** declaration.

The following example shows how natural numbers with a zero, a successor and addition can be declared. The sort **Bool** with 0-ary function symbols t and f are obligatory in a μCRL specification.

Example 1.1. (μCRL specification)

$$
\begin{array}{ll}
\textbf{sort} & Nat, \mathbf{Bool} \\
\textbf{func} & \mathsf{t}, \mathsf{f} : \to \mathbf{Bool} \\
 & 0 : \to Nat \\
 & S : Nat \to Nat \\
 & add : Nat \times Nat \to Nat \\
\textbf{var} & x, y : Nat \\
\textbf{rew} & add(x, 0) = x \\
 & add(x, S(y)) = S(add(x, y)) \\
\textbf{act} & a, b, c : Nat \\
 & a, d \\
\textbf{comm} & a \,|\, b = c \\
\textbf{proc} & p(x : Nat) = a(x) \cdot p(S(x)) \\
 & q = a + b(0) \parallel d
\end{array}
$$

□

Processes may contain actions representing elementary activities that can be performed. These actions must be explicitly declared using the keyword **act**. Actions may be parameterized by data. In the following lines an action declaration is displayed.

Here parameterless actions a, b, c and actions a, d depending on natural numbers are declared. Note that overloading is allowed, as long as this cannot lead to confusion (see [GP95, GP93, GP91] for details). In this case a and $a(0)$ are different actions.

In μCRL parallel processes communicate via synchronization of actions. A communication specification, declared using the keyword **comm**, prescribes which actions may synchronize on the level of the gates related to the actions. In the example each action $a(n)$ can communicate with $b(n')$ to $c(n)$ provided n and n' denote the same data element of sort Nat.

Definition 1.2. *An expression p is called a* process term *iff p has the following syntax:*

$$
\begin{array}{rll}
p & ::= & (p+p) \mid (p \cdot p) \mid (p \parallel p) \mid (p \mathbin{\lfloor\!\lfloor} p) \mid (p \,|\, p\,) \mid (p \triangleleft t \triangleright p) \\
 & \mid & \sum_{d:D}(p) \mid \partial_H(p) \mid \tau_I(p) \mid \rho_R(p) \mid \delta \mid \tau \mid n \mid n(t_1, ..., t_m).
\end{array}
$$

where the n, n_i, n_i' are names, the t, t_i stand for data terms, d is a variable and D denotes a sort name. Furthermore, H and I stand for a finite set of names $\{n_1, ..., n_m\}$, and R stands for a finite set of renamings $\{n_1 \to n_1', ..., n_m \to n_m'\}$.

Most operators stem from ACP [BW90]. Only the conditional construct $p \triangleleft t \triangleright p$ is taken from [HHJ$^+$87] (see also [BB92]). In process terms we omit brackets according to the convention that $\cdot$ binds strongest, the conditional construct binds stronger than the parallel operators which in turn bind stronger than $+$. When clear from the context we also omit the dot in sequential compositions.

1.1.1 Axioms of ACP

Table 11 lists the axioms of ACP in μCRL. In this tables x, y are process variables and p, q are process terms in which the variable d may occur. The letters $t_1, \ldots, t_n$ and $u_1, \ldots, u_m$ stand for data terms, and $\bar{t}$ for a sequence of data terms where ϵ is the empty sequence. The symbols a, b represent δ, τ or range over (declared) actions $n(\bar{t})$, where $n(\bar{t})$ represents n if $\bar{t} = \epsilon$. $\tilde{\gamma}$ is the pre-communication function such that $\tilde{\gamma}(n_1, n_2) = n_3$ if a rule **comm** $n_1 \,|\, n_2 = n_3$ appears in the μCRL specification. Otherwise $\tilde{\gamma}(n_1, n_2) = \delta$. γ is the symmetrical closure of $\tilde{\gamma}$.

A1	$x + y = y + x$			
A2	$x + (y + z) = (x + y) + z$	CF	$n(\bar{t}) \,|\, m(\bar{t}) = \begin{cases} \gamma(n,m)(\bar{t}) & \text{if } \gamma(n,m) \text{ defined} \\ \delta & \text{otherwise} \end{cases}$	
A3	$x + x = x$			
A4	$(x + y) \cdot z = x \cdot z + y \cdot z$			
A5	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$			
A6	$x + \delta = x$			
A7	$\delta \cdot x = \delta$	CD1	$\delta \,|\, x = \delta$	
		CD2	$x \,|\, \delta = \delta$	
CM1	$x \parallel y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x + x \,|\, y$	CT1	$\tau \,|\, x = \delta$	
CM2	$a \mathbin{\lfloor\!\lfloor} x = a \cdot x$	CT2	$x \,|\, \tau = \delta$	
CM3	$a \cdot x \mathbin{\lfloor\!\lfloor} y = a \cdot (x \parallel y)$			
CM4	$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$	DD	$\partial_H(\delta) = \delta$	
CM5	$a \cdot x \,|\, b = (a \,|\, b) \cdot x$	DT	$\partial_H(\tau) = \tau$	
CM6	$a \,|\, b \cdot x = (a \,|\, b) \cdot x$	D1	$\partial_H(n(\bar{t})) = n(\bar{t})$	if $n \notin H$
CM7	$a \cdot x \,|\, b \cdot y = (a \,|\, b) \cdot (x \parallel y)$	D2	$\partial_H(n(\bar{t})) = \delta$	if $n \in H$
CM8	$(x + y) \,|\, z = x \,|\, z + y \,|\, z$	D3	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	
CM9	$x \,|\, (y + z) = x \,|\, y + x \,|\, z$	D4	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$	

Table 11: The axioms of ACP in μCRL.

1.1.2 Axioms of Standard Concurrency

In Table 12 some axioms for the merge operators, known as the Standard Concurrency laws (see [BW90]) are presented. These axioms are derivable for closed process terms without recursion.
Using the axioms of standard concurrency we can derive the following identities.

Lemma 1.3.

SC1	$(x \mathbin{\|\!\lfloor} y) \mathbin{\|\!\lfloor} z = x \mathbin{\|\!\lfloor} (y \parallel z)$	SC4	$(x \mid y) \mid z = x \mid (y \mid z)$
SC2	$x \parallel \delta = x\delta$	SC5	$x \mid (y \mathbin{\|\!\lfloor} z) = (x \mid y) \mathbin{\|\!\lfloor} z$
SC3	$x \mid y = y \mid x$	HA	$x \mid (y \mid z) = \delta$

Table 12: Axioms of Standard Concurrency.

1. $x \parallel y = y \parallel x$,
2. $x \parallel (y \parallel z) = (x \parallel y) \parallel z$.

1.1.3 Axioms for the hiding operator

For abstraction in μCRL we present the axioms in Table 13. Here $I \subseteq \mathcal{N}$ is the set of gate names to be abstracted from.

TI1	$\tau_I(n(\bar{t})) = n(\bar{t})$	if $n \notin I$		
TI2	$\tau_I(n(\bar{t})) = \tau$	if $n \in I$		
TI3	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$		TID	$\tau_I(\delta) = \delta$
TI4	$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$		TIT	$\tau_I(\tau) = \tau$

Table 13: Axioms for hiding.

1.1.4 Axioms for the renaming operator

For renaming in μCRL we present the axioms in Table 14. Here $R \subseteq \mathcal{N} \times \mathcal{N}$ is the set rename pairs.

REN1	$\rho_R(n(\bar{t})) = n(\bar{t})$	if $n \notin \pi_1(R)$
REN2	$\rho_R(n(\bar{t})) = n'(\bar{t})$	if $\langle n, n' \rangle \in R$
REN3	$\rho_R(x + y) = \rho_R(x) + \rho_R(y)$	
REN4	$\rho_R(x \cdot y) = \rho_R(x) \cdot \rho_R(y)$	

Table 14: Axioms for renaming.

1.1.5 Axioms for the sum operator

One of the key features of the proof theory for μCRL is that it supports axioms for the sum operator (see Table 15) in order for being able to treat summation over (infinite) data in an explicit way.

SUM1	$\sum_{d:D}(p) = p$	if d not free in p
SUM2	$\sum_{d:D}(p) = \sum_{e:D}(p[e/d])$	if e not free in p
SUM3	$\sum_{d:D}(p) = \sum_{d:D}(p) + p$	
SUM4	$\sum_{d:D}(p_1 + p_2) = \sum_{d:D}(p_1) + \sum_{d:D}(p_2)$	
SUM5	$\sum_{d:D}(p_1 \cdot p_2) = \sum_{d:D}(p_1) \cdot p_2$	if d not free in p_2
SUM6	$\sum_{d:D}(p_1 \mathbin{\|\!\lfloor} p_2) = \sum_{d:D}(p_1) \mathbin{\|\!\lfloor} p_2$	if d not free in p_2
SUM7	$\sum_{d:D}(p_1 \mid p_2) = \sum_{d:D}(p_1) \mid p_2$	if d not free in p_2
SUM8	$\sum_{d:D}(\partial_H(p)) = \partial_H(\sum_{d:D}(p))$	
SUM9	$\sum_{d:D}(\tau_I(p)) = \tau_I(\sum_{d:D}(p))$	
SUM10	$\sum_{d:D}(\rho_R(p)) = \rho_R(\sum_{d:D}(p))$	
SUM11	$\dfrac{\begin{array}{c}\mathcal{D}\\ p_1 = p_2\end{array}}{\sum_{d:D}(p_1) = \sum_{d:D}(p_2)}$	provided d not free in the assumptions of $\mathcal{D}$

Table 15: Axioms for the sum operator.

1.1.6 Axioms for the conditional construct

The axioms for the *conditional* construct $p \lhd c \rhd q$, are given in Table 16. Axiom COND1 expresses that process $p \lhd t \rhd q$ behaves as p if the data term t evaluates to true (t) and Cond2 expresses that it behaves as q if t evaluates to false (f).

COND1	$x \lhd \mathsf{t} \rhd y$	$=$	x
COND2	$x \lhd \mathsf{f} \rhd y$	$=$	y

Table 16: Axioms for the conditional construct.

1.1.7 The rule RSP

In our verification the we used the rule RSP (Recursive Specification Principle). Informally, it says that each guarded recursive specification has at most one solution. For more information one is referred to [BW90, GP93, GP91].

1.1.8 τ-law

In our verification we only needed one simple τ-law which is given below.

B1	$x\tau$	$= x$

Table 17: A simple τ-law.

References

[BB92] J.C.M. Baeten and J.A. Bergstra. Process algebra with signals and conditions. In M. Broy, editor, *Programming and Mathematical Methods, Proceedings Summer School Marktoberdorf 1990*, pages 273–323. Springer-Verlag, 1992. NATO ASI Series F88.

[BBG95] M.A. Bezem, R. Bol, and J.F. Groote. Formalizing process algebraic verifications in the calculus of constructions. Computing Science Report 95-02, Eindhoven University of Technology, 1995.

[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, *Formal Description of Programming Concepts – III, Proceedings of the 3^{th} IFIP WG 2.2 working conference,* Ebberup 1986, pages 53–75, Amsterdam, 1987. North-Holland.

[BG94] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In B. Jonsson and J. Parrow, editors, *Proceedings of the 5^{th} conference on Theories of Concurrency, CONCUR '94,* Uppsala, Sweden, August 1994, volume 836 of *Lecture Notes in Computer Science*, pages 401–416. Springer-Verlag, 1994.

[BK85] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

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

[CH88] T. Coquand and G.P. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of algebraic specifications I*, volume 6 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, 1985.

[Gro90] J.F. Groote. A new strategy for proving ω-completeness with applications in process algebra. In *Proceedings of the 1^{st} Conference on Theories of Concurrency, CONCUR '90,* Amsterdam, the Netherlands, August 1990, volume 458 of *Lecture Notes in Computer Science*, pages 314–331, 1990.

[GP91] J.F. Groote and A. Ponse. Proof theory for μCRL. Report CS-R9138, CWI, Amsterdam, August 1991.

[GP93] J.F. Groote and A. Ponse. Proof theory for μCRL: A language for processes with data. In D.J. Andrews, J.F. Groote, and C.A. Middelburg, editors, *Proceedings of the International Workshop on Semantics of Specification Languages*, Workshops in Computer Science, pages 231–250. Springer-Verlag, 1993.

[GP95] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes,* Utrecht, 1994, Workshops in Computing, pages 26–62. Springer-Verlag, 1995.

[GvdP93] J.F. Groote and J. van de Pol. A bounded retransmission protocol for large data packets. Logic Group Preprint Series 100, Dept. of Philosophy, Utrecht University, October 1993.

[HHJ+87] C.A.R. Hoare, I.J. Hayes, He Jifeng, C.C. Morgan, A.W. Roscoe, J.W. Sanders, I.H. Sorensen, J.M. Spivey, and B.A. Sufrin. Laws of programming. *Communications of the ACM*, 30(8):672–686, August 1987.

[Kor94] H.P. Korver. *Protocol verification in μCRL.* PhD thesis, University of Amsterdam, June 1994.

[KS93] H. Korver and J. Springintveld. A computer-checked verification of Milner's scheduler. Report CS-R9371, CWI, Amsterdam, December 1993. Full version.

[KS94] H. Korver and J. Springintveld. A computer-checked verification of Milner's scheduler. In M. Hagiya and J. Mitchell, editors, *Proceedings of the International Symposium on Theoretical Aspects of Computer Software (TACS'94),* Sendai, Japan, volume 789 of *Lecture Notes in Computer Science.* Springer-Verlag, 1994. Extended abstract.

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

[Sel93] M.P.A. Sellink. Verifying process algebra proofs in type theory. In D.J. Andrews, J.F. Groote, and C.A. Middelburg, editors, *Proceedings of the International Workshop on Seman tics of Specification Languages,* Utrecht, The Netherlands, pages 315–339. Workshops in Computer Science, Springer-Verlag, 1993.

[Vaa90] F.W. Vaandrager. *Algebraic Techniques for Concurrency and their Application.* PhD thesis, University of Amsterdam, February 1990.

Technical Reports of the Programming Research Group

Note: These reports can be obtained using the technical reports overview on our WWW site (`http://www.fwi.uva.nl/research/prog/reports/`) or by anonymous ftp to `ftp.fwi.uva.nl`, directory `pub/programming-research/reports/`.

[P9703] H.P. Korver and M.P.A. Sellink. *A Formal Axiomatization for Alphabet Reasoning with Parametrized Processes.*

[P9702] M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. *Reengineering COBOL Software Implies Specification of the Underlying Dialects.*

[P9701] E. Visser. *Polymorphic Syntax Definition.*

[P9618] M.G.J. van den Brand, P. Klint, and C. verhoef. *Re-engineering needs Generic Programming Language Technology.*

[P9617] P.I. Manuel. *ANSI Cobol III in SDF + an ASF Definition of a Y2K Tool.*

[P9616] P.H. Rodenburg. *A Complete System of Four-valued Logic.*

[P9615] S.P. Luttik and P.H. Rodenburg. *Transformations of Reduction Systems.*

[P9614] M.G.J. van den Brand, P. Klint, and C. Verhoef. *Core Technologies for System Renovation.*

[P9613] L. Moonen. *Data Flow Analysis for Reverse Engineering.*

[P9612] J.A. Hillebrand. *Transforming an ASF+SDF Specification into a ToolBus Application.*

[P9611] M.P.A. Sellink. *On the conservativity of Leibniz Equality.*

[P9610] T.B. Dinesh and S.M. Üsküdarlı. *Specifying input and output of visual languages.*

[P9609] T.B. Dinesh and S.M. Üsküdarlı. *The VAS formalism in VASE.*

[P9608] J.A. Hillebrand. *A small language for the specification of Grid Protocols.*

[P9607] J.J. Brunekreef. *A transformation tool for pure Prolog programs: the algebraic specification.*

[P9606] E. Visser. *Solving type equations in multi-level specifications (preliminary version).*

[P9605] P.R. D'Argenio and C. Verhoef. *A general conservative extension theorem in process algebras with inequalities.*

[P9602b] J.A. Bergstra and M.P.A. Sellink. *Sequential data algebra primitives (revised version of P9602).*

[P9604] E. Visser. *Multi-level specifications.*

[P9603] M.G.J. van den Brand, P. Klint, and C. Verhoef. *Reverse engineering and system renovation: an annotated bibliography.*

[P9602] J.A. Bergstra and M.P.A. Sellink. *Sequential data algebra primitives.*

[P9601] P.A. Olivier. *Embedded system simulation: testdriving the ToolBus.*

[P9512] J.J. Brunekreef. *TransLog, an interactive tool for transformation of logic programs.*

[P9511] J.A. Bergstra, J.A. Hillebrand, and A. Ponse. *Grid protocols based on synchronous communication: specification and correctness.*

[P9510] P.H. Rodenburg. *Termination and confluence in infinitary term rewriting.*

[P9509] J.A. Bergstra and Gh. Stefanescu. *Network algebra with demonic relation operators.*

[P9508] J.A. Bergstra, C.A. Middelburg, and Gh. Stefanescu. *Network algebra for synchronous and asynchronous dataflow.*

[P9507] E. Visser. *A case study in optimizing parsing schemata by disambiguation filters.*

[P9506] M.G.J. van den Brand and E. Visser. *Generation of formatters for context-free languages.*

[P9505] J.M.T. Romijn. *Automatic analysis of term rewriting systems: proving properties of term rewriting systems derived from* ASF+SDF *specifications.*

[P9504] M.G.J. van den Brand, A. van Deursen, T.B. Dinesh, J.F.Th. Kamperman, and E. Visser (editors). ASF+SDF*'95: a workshop on Generating Tools from Algebraic Specifications, May 11&12, 1995, CWI Amsterdam.*

[P9503] J.A. Bergstra and A. Ponse. *Frame-based process logics.*

[P9208c] J.C.M. Baeten and J.A. Bergstra. *Discrete time process algebra (revised version of P9208b).*

[P9502] J.A. Bergstra and P. Klint. *The discrete time ToolBus.*

[P9501] J.A. Hillebrand and H.P. Korver. *A well-formedness checker for* μ*CRL.*

[P9426] P. Klint and E. Visser. *Using filters for the disambiguation of context-free grammars.*

[P9425] B. Diertens and A. Ponse. *New features in PSF II: iteration and nesting.*

[P9424] M.A. Bezem and A. Ponse. *Two finite specifications of a queue.*

[P9423] J.J. van Wamel. *Process algebra with language matching.*