A Formal Axiomatization for Alphabet Reasoning with Parametrized Processes

Henri Korver

Department of Philosophy, Utrecht University

Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

korver@phil.ruu.nl

Abstract

In the process-algebraic verification of systems with three or more components put in parallel, alphabet axioms are considered to be very 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 + 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_τ for making verifications of systems with two or more components put in parallel feasible. 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 ($\backslash$) 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

[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) is left out.

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 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]).[2] 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 3, μCRL is extended with alphabet axioms for *unparametrized* [3] actions. For being able to simulate infinite alphabets, the encapsulation and hiding operator are refined to actions that are parametrized with data in Section 4. In Section 5, conditional axioms are given for *parametrized* actions. This section is important as it shows how the reasoning with infinite alphabets can be represented symbolically within μCRL. In Section 6, some axioms are introduced which combine the power of the axiom systems given in Section 3 and 5. Then, in Section 7, 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 A.

2 Classical 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 terms. In this table, $a \in Act$ (the enclosing set of atomic actions, not including τ and δ) and x, y are arbitrary (ACP) processes. The axioms AB6 and AB7 are used to calculate the alphabet of *infinite* processes.[4] 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 practice it is rather cumbersome to calculate alphabets of infinite processes by using axiom AB6. Therefore in [BBK87] the α/β-calculus is developed for 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 which use the alphabets of processes. 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

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

[3] Actions that do not have data arguments. E.g. *send* is an unparametrized action (gate) and *send*(1) is an parametrized action having the natural number 1 as argument.

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

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

Table 1: Alphabet.

PR1	$\pi_n(b) = b$
PR2	$\pi_1(bx) = b$
PR3	$\pi_{n+1}(bx) = b \cdot \pi_n(x)$
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 2: Projection.

CA1	$\alpha(x) \mid (\alpha(y) \cap H) \subseteq H$	$\rightarrow$	$\partial_H(x \parallel y) = \partial_H(x \parallel \partial_H(y))$
CA2	$\alpha(x) \mid (\alpha(y) \cap I) \subseteq \varnothing$	$\rightarrow$	$\tau_I(x \parallel y) = \tau_I(x \parallel \tau_I(y))$
CA3	$\alpha(x) \cap H = \varnothing$	$\rightarrow$	$\partial_H(x) = x$
CA4	$\alpha(x) \cap I = \varnothing$	$\rightarrow$	$\tau_I(x) = x$
CA5	$H = J \cup K$	$\rightarrow$	$\partial_H(x) = \partial_J \circ \partial_K(x)$
CA6	$I = J \cup K$	$\rightarrow$	$\tau_I(x) = \tau_J \circ \tau_K(x)$
CA7	$H \cap I = \varnothing$	$\rightarrow$	$\tau_I \circ \partial_H(x) = \partial_H \circ \tau_I(x)$

Table 3: Conditional Axioms.

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 7).

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, Sel93]. However in many verifications one would like to reason with infinite alphabets. For instance, in Milner's Scheduler proof (see [Mil89] and [KS93]), but also in the verification given in Section 7.

In the remainder of the paper, the conditional (alphabet) axioms given in this section are reformulated within the proof theory of μCRL such that they can be easily mechanized.

3 Alphabet axioms for gates in μCRL

Now I shall formulate the alphabet axioms given above in ordinary μCRL, where the the encapsulation and hiding operators can only rename unparametrized actions (gates) into δ (or τ). In other words, these operators either rename none or all occurrences of an action. For example, in ordinary μCRL it is allowed to write expressions of the form $\partial_{\{a\}}(\sum_{j:nat} a(j))$ (which equals δ) but not of the form $\partial_{<a(i)>}(\sum_{j:nat} a(j))$. Expressions of the last 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 treated in Section 4 and 5.

If one sticks to the ordinary encapsulation and hiding operators (which only rename gates), the alphabet axioms given in Section 2 can be formalized in μCRL as is shown in Table 4.[5] In this table we have the following conventions:

- The symbols $G, G_1, G_2 \subseteq \mathcal{G}'$ stand for sets of unparametrized actions (gates), where $\mathcal{G}'$ is the enclosing set of gates. Note that $\mathcal{G}'$ is always a finite set because in μCRL one can only declare a finite set of gates.

- $G_1 \,|\, G_2 = \{\gamma(n_1, n_2) : n_1 \in G_1, n_2 \in G_2\}$ denotes the set of all possible communication results between actions from G_1 and G_2. Here γ is the communication function. 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** $n_1 \,|\, n_2 = n_3$ appears in the μCRL specification in question.

- $\overline{G} = \mathcal{G}' - G$ is the complement of G.

- For readability we write $\partial_{G_1}\partial_{G_2}(x)$ for $\partial_{G_1}(\partial_{G_2}(x))$.

Note that the axioms in Table 4 do not rely on 'real' set theory as the conditions at the right-hand side only involve finite sets.

The first four axioms CAG1−4 look a bit different compared to CA1−4. This is due to the fact that we eliminated any explicit reference to the set theoretic α function.

Theorem 3.1. *Axioms CAG1-7 hold.*

Proof. That CAG1 is a sound axiom is motivated as follows. By replacing x by $\partial_{G_1}(x)$, y by $\partial_{G_2}(y)$, and H by G in CA1 (see Table 3), we obtain

$$\alpha(\partial_{G_1}(x)) \,|\, (\alpha(\partial_{G_2}(y)) \cap G) \subseteq G \;\rightarrow\; \partial_G(\partial_{G_1}(x) \parallel \partial_{G_2}(y)) = \partial_G(\partial_{G_1}(x) \parallel \partial_G\partial_{G_2}(y)) \quad (1)$$

[5] 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 4: Alphabet Axioms for Gates (CAG) in μCRL.

It is obvious that the actions of G_1 can not be part of the alphabet of the process $\partial_{G_1}(x)$. In formula: $\alpha(\partial_{G_1}(x)) \subseteq \overline{G_1}$. For the same reasons, we have $\alpha(\partial_{G_2}(y)) \subseteq \overline{G_2}$. With this in mind, we can rewrite (1) into the form:

$$\partial_G(\partial_{G_1}(x) \parallel \partial_{G_2}(y)) = \partial_G(\partial_{G_1}(x) \parallel \partial_G\partial_{G_2}(y)) \quad \text{if } \overline{G_1} \mid (\overline{G_2} \cap G) \subseteq G \tag{2}$$

Then, by using the fact that $\overline{G_2} \cap G$ is equal to the set $G - G_2$, we have transformed CA1 into CAG1. The soundness of the other rules (CAG2-7) can be motivated in a similar way. □

Lemma 3.2. *From the axioms in Table 4 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′. □

This lemma appears to very useful in the verification example that is given in Section 7.

Clearly, the axioms given in Table 4 have a more restrictive (syntactical) form than the original axioms given in Table 3. Now, the question arises whether the new axioms are strong enough to prove all the identities that can be proven by the conventional axioms? Up till now, the new axioms appeared to be general enough in practice, e.g. Milner's Scheduler and the bag example given in Section 7. Especially this bag verification was a good test because this example makes extensive use of alphabet reasoning. But a complete answer to the question given above remains open.

Note that in Table 4 the conditions are placed at the right-hand side such that they are not part of the proof system any more. This is harmless because in the new conditions only gates are involved. Because gates do not contain process or data variables, they are not subject to the internal substitution and derivation rules of the proof system. The advantage is that in contrast with Table 3, the new rules in Table 4 have a pure equational format. Notice that the conditions at the right-hand side are just a short-hand for denoting a finite set of equations.

As an illustration of the usage of axiom CAG3, we prove Proposition 3.3 given below.

act $a, b, c : nat$
comm $a|b = c$
proc $P = a \cdot a \cdot a$
$Q = b \cdot b \cdot b$

Proposition 3.3. $\partial_\varnothing(P \parallel Q) = P \parallel Q$

Proof. As an illustration, we prove this fact without using CAG3′.

$$\partial_\varnothing(P \parallel Q) \overset{\text{D1}\,(6\times)}{=} \partial_\varnothing(\partial_\varnothing(P) \parallel \partial_\varnothing(Q)) \overset{\text{CAG3}}{=} \partial_\varnothing(P) \parallel \partial_\varnothing(Q) \overset{\text{D1}\,(6\times)}{=} P \parallel Q.$$

Axiom D1 can be found in Table 9 of the appendix. □

More illustrations of the working of the axioms given in Table 4 can be found in Section 7.

4 Refining encapsulation and hiding in μCRL

In classical μCRL (see [GP95, GP93, GP91]) either none or all occurrences of an action in a process p can be renamed, e.g. $\partial_{\{a\}}(\sum_{j:nat} a(j)) = \delta$. But sometimes as 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 A.1) as follows:

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

$$a\text{-}list ::= a \mid a\text{-}list, a$$
$$a ::= n(t_1, \ldots, t_n)$$

where n is an action name (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 5 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 have the following conventions:

- A is a *finite* subset of $\mathcal{G}$, the enclosing set of parametrized actions. Note that $\mathcal{G}$ may be an infinite set, e.g. the μCRL declaration

  sort nat
  func $0 :\rightarrow nat$
  $S : nat \rightarrow nat$
  act $a : nat$
  proc ...

  already gives rise to an infinite alphabet of parametrized actions

  $$\{a(0), a(S(0)), a(S(S(0))), \ldots\}.$$

- The letter b ranges over δ, τ and actions of the form $n(t_1, \ldots, t_n)$ from $\mathcal{G}$. The letter a also ranges over $\mathcal{G}$ but not over δ and τ.
- $\bigwedge_{a \in A}$ is a shorthand for a finite conjunction and $\bigvee_{a \in A}$ for a finite disjunction over parametrized actions. We adopt the convention that $\bigwedge_{a \in \varnothing}$ stands for $\neg \mathcal{F}$ and $\bigvee_{a \in \varnothing}$ stands for $\mathcal{F}$, where $\mathcal{F}$ stands for the false μCRL proposition.

The rules in Table 5 resemble the original rules D1−4 and TI1−4, but differ on a delicate detail. In Table 5 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 parametrized with data variables. So these conditions are not subject to the internal substitution mechanism of the proof system.

DA1	$(\bigwedge_{a \in A} \neg\, a = b) \rightarrow \partial_A(b) = b$	TA1	$(\bigwedge_{a \in A} \neg\, a = b) \rightarrow \tau_A(b) = b$
DA2	$(\bigvee_{a \in A} a = b) \rightarrow \partial_A(b) = \delta$	TA2	$(\bigvee_{a \in A} a = b) \rightarrow \tau_A(b) = \tau$
DA3	$\partial_A(x + y) = \partial_A(x) + \partial_A(y)$	TA3	$\tau_A(x + y) = \tau_A(x) + \tau_A(y)$
DA4	$\partial_A(x \cdot y) = \partial_A(x) \cdot \partial_A(y)$	TA4	$\tau_A(x \cdot y) = \tau_A(x) \cdot \tau_A(y)$

Table 5: Axioms for encapsulating and hiding parametrized actions.

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 6. In this table $\overline{t}$ 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.

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	$a : nat$

Proposition 4.1. From the specification above 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. This identity can be easily proven by the basic μCRL lemma (see [GP91]):

   $$(\phi \to \psi) \wedge (\neg\phi \to \psi) \to \psi$$

   where ϕ, ψ are arbitrary property-formulas over μCRL expressions. It expresses that μCRL supports the excluded middle principle.

   Let ψ be the identity to be proven, and let $\phi \equiv i = j$. Then by checking the two cases below, we know that the desired identity (ψ) holds.

   (a) If $i = j$ (ϕ) then $\partial_{<a(i)>}(\delta \lhd eq(i,j) \rhd a(j)) = \partial_{<a(i)>}(\delta) \stackrel{\text{DA1}}{=} \delta = \delta \lhd \mathsf{t} \rhd a(j) = \delta \lhd eq(i,j) \rhd a(j)$.

   (b) If $i \neq j$ ($\neg\phi$) then $\partial_{<a(i)>}(\delta \lhd eq(i,j) \rhd a(j)) = \partial_{<a(i)>}(a(j)) \stackrel{\text{DA1}}{=} a(j) = \delta \lhd \mathsf{f} \rhd a(j) = \delta \lhd eq(i,j) \rhd a(j)$.

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.

   $\partial_{<a(i)>} \sum_{j:nat} a(j)$

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

   $\stackrel{\text{DA3}}{=}$ $\partial_{<a(i)>}(\sum_{j:nat}(\delta \lhd eq(i,j) \rhd a(j))) + \partial_{<a(i)>}(a(i))$

   $\stackrel{\text{DA2}}{=}$ $\partial_{<a(i)>} \sum_{j:nat}(\delta \lhd eq(i,j) \rhd a(j)) + \delta$

   $\stackrel{\text{SUM8}'}{=}$ $\sum_{j:nat} \partial_{<a(i)>}(\delta \lhd eq(i,j) \rhd a(j))$

   $\stackrel{4.1.1}{=}$ $\sum_{j:nat}(\delta \lhd eq(i,j) \rhd a(j))$

□

Note that the axioms in Table 6 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 CAA5 and CAA6 given in Table 7. For example, see the lemma below.

Lemma 4.2.

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

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 6: Additional axioms for summation.

CAA1	$(\bigwedge_{n'\in Part^*(n)}\partial_{<n'(\overline{t})>}(x)=x) \rightarrow \partial_{<n(\overline{t})>}(x \parallel y) = \partial_{<n(\overline{t})>}(x \parallel \partial_{<n(\overline{t})>}(y))$	
CAA2	$(\bigwedge_{n'\in Part(n)}\partial_{<n'(\overline{t})>}(x)=x) \rightarrow \tau_{<n(\overline{t})>}(x \parallel y) = \tau_{<n(\overline{t})>}(x \parallel \tau_{<n(\overline{t})>}(y))$	
CAA3	$\partial_A(\partial_A(x) \parallel \partial_A(y)) = \partial_A(x) \parallel \partial_A(y)$	if $label(A) \cap range(\gamma) = \varnothing$
CAA3′	$\partial_{\varnothing_A}(x) = x$	
CAA4	$\tau_A(\tau_A(x) \parallel \tau_A(y)) = \tau_A(x) \parallel \tau_A(y)$	if $label(A) \cap range(\gamma) = \varnothing$
CAA4′	$\tau_{\varnothing_A}(x) = x$	
CAA5	$\partial_{<a_1,\ldots,a_n>}(x) = \partial_{<a_1,\ldots,a_{n-1}>}\partial_{<a_n>}(x)$	
CAA6	$\tau_{<a_1,\ldots,a_n>}(x) = \tau_{<a_1,\ldots,a_{n-1}>}\tau_{<a_n>}(x)$	
CAA7	$\neg(a=b) \rightarrow \tau_{<a>}\partial_{<b>}(x) = \partial_{<b>}\tau_{<a>}(x)$	

Table 7: Conditional Axioms for parametrized Actions (CAA).

5 Alphabet axioms for parametrized actions

In Section 3, the alphabet axioms were formulated for gates in μCRL. In this section, we formulate such axioms for parametrized actions. Seven (conditional) axioms are presented in Table 7.

Due to the presence of data parameters, these axioms can be used for simulating the classical alphabet axioms with infinite alphabets (see Section 7).

In Table 7 the following notation is used:

- $a, a_1, \ldots, a_n, b$ range over $\mathcal{G}$ (parametrized actions), and n, n' range over $\mathcal{G}'$ (gates).
- $Part(n)$ 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. More precisely,

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

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

  where γ is the communication function. 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 7, $Part(s_2) = Part^*(s_2) = \{r_2\}$.
- *range* is the range of the communication function γ. For example, in the specification of a bag given in Section 7, this is $\{c_2, c_3\}$.
- $label(A)$ is the set of labels that occur in the set of actions A. For instance,

  $$label(< r(i), s(i) >) = \{r, s\}.$$
- $\overline{t}$ stands for a finite list $t_1, \ldots, t_n$.

The rule CAA2 expresses that the operator $\tau_{<n(\overline{t})>}$ can be pushed inside if there are *no* communication partners of action $n(\overline{t})$ in x. Otherwise, a fruitful communication could be lost. More precisely, if the equation $\partial_{<n'(\overline{t})>}(x) = x$ can be derived, we know that the action $n'(\overline{t})$ is not in the alphabet of x. Then, in case the formula $\bigwedge_{n' \in Part(n)} \partial_{<n'(\overline{t})>}(x) = x$ holds, we know that none of the communication partners of $n(\overline{t})$ occurs in the alphabet of x, and $\partial_{<n(\overline{t})>}$ may be pushed safely inside. CAA1 can be explained in a similar way with the subtlety that there may be a communication partner of $n(\overline{t})$ in x as long as the label of the resulting action is equal to n. (A similar subtlety already appeared in the conditions of the rules CA1 and CA2 in Table 3.) The other axioms speak for themselves.

For readability, some axioms in Table 7 are formulated with singleton sets. It is obvious that these axioms can easily be generalized to arbitrary sets by using CAA5 and CAA6.

6 Linking CAG with CAA

In Table 8, 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, $A \subseteq \mathcal{G}$ is a finite set of parametrized actions, and $label(A)$ is the set of labels occurring in the action set A.

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

Table 8: Axioms for linking Gates and Actions (CAGA).

7 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. A bag which reads from a channel i and writes on a channel j ($i \neq j$) is defined as follows:

act $r_i, s_j, c_i : D$
comm $r_n \mid s_n = c_n$
proc $B_{ij} = \sum_{d:D}(r_i(d) \cdot (s_j(d) \parallel B_{ij}))$

where $i \in \{1,2,3\}, j \in \{2,3,4\}, n \in \{2,3\}$ and $i \neq j$.

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 7.1. *Let* $H_n = \{r_n, s_n\}$, $I_n = \{c_n\}$, $n \in \{2,3\}$, $H = H_2 \cup H_3$ *and* $I = I_2 \cup I_3$.

1. $\partial_{H_3}(B_{12}) = B_{12}$ *(used in the proof of 2, 6)*
2. $\partial_{<s_3(d)>}(B_{12}) = B_{12}$ *(used in the proof of 3)*
3. $c_2(d) \cdot s_3(d) \parallel B_{23} = \partial_{<s_2(d)>}(s_2(d) \parallel B_{23})$ *(used in the proof of 4)*
4. $\partial_{H_2}(B_{12} \parallel B_{23}) = \sum_{d:D}(r_1(d) \cdot (c_2(d) \cdot s_3(d) \parallel \partial_{H_2}(B_{12} \parallel B_{23})))$ *(used in the proof of 5)*
5. $\tau_{I_2}\partial_{H_2}(B_{12} \parallel B_{23}) = B_{13}$ *(used in the proof of 6)*
6. $\tau_I\partial_H(B_{12} \parallel B_{23} \parallel B_{34}) = B_{14}$

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, the 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 A. 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 10 of the appendix are applied.

1. $\partial_{H_3}(B_{12})$

$$\begin{array}{ll}
= & \partial_{H_3}(\sum_{d:D}(r_1(d)\cdot(s_2(d)\parallel B_{12}))) \\
\overset{\text{SUM8}}{=} & \sum_{d:D}(\partial_{H_3}(r_1(d)\cdot(s_2(d)\parallel B_{12}))) \\
\overset{\text{D1}}{=} & \sum_{d:D}(r_1(d)\cdot\partial_{H_3}(s_2(d)\parallel B_{12})) \\
\overset{\text{D1}}{=} & \sum_{d:D}(r_1(d)\cdot\partial_{H_3}(\partial_{H_3}(s_2(d))\parallel B_{12})) \\
\overset{\text{Lemma 3.2}}{=} & \sum_{d:D}(r_1(d)\cdot\partial_{H_3}(\partial_{H_3}(s_2(d))\parallel \partial_{H_3}(B_{12}))) \qquad \text{(see note I)} \\
\overset{\text{CAG3}}{=} & \sum_{d:D}(r_1(d)\cdot(\partial_{H_3}(s_2(d))\parallel \partial_{H_3}(B_{12}))) \qquad \text{(see note II)} \\
\overset{\text{D1}}{=} & \sum_{d:D}(r_1(d)\cdot(s_2(d)\parallel \partial_{H_3}(B_{12})))
\end{array}$$

Thus we see that $\partial_{H_3}(B_{12})$ is a solution for B_{12}. Therefore, by the Recursive Specification Principle (RSP), we have that $\partial_{H_3}(B_{12}) = B_{12}$.

This finishes the proof of Theorem 7.1.1.

Note I: $\overline{H}_3 \,|\, H_3 = \{r_1, r_2, s_1, s_2, c_2, c_3\} \,|\, \{r_3, s_3\} = \varnothing \subseteq H_3$.

Note II: $H_3 \subseteq (H_3 \cap H_3) - (\overline{H}_3 \,|\, \overline{H}_3) = \{r_3, s_3\} - \{c_2\} = H_3$.

Note that all the calculations in notes (I) and (II) only involve *finite* sets.

2. $\partial_{<s_3(d)>}(B_{12})$

$$\begin{array}{ll}
\overset{(1)}{=} & \partial_{<s_3(d)>}(\partial_{H_3}(B_{12})) \\
\overset{\text{CAGA3}}{=} & \partial_{H_3}(\partial_{<s_3(d)>}(B_{12})) \\
\overset{\text{CAGA1}}{=} & \partial_{H_3}(B_{12}) \\
\overset{(1)}{=} & B_{12}
\end{array}$$

3. On the one hand we have,

$$\begin{array}{ll}
c_2(d)s_3(d)\parallel B_{23} & \\
= & c_2(d)\cdot(s_3(d)\parallel B_{23}) + \sum_{e:D}(r_2(e)\cdot(c_2(d)s_3(d)\parallel s_3(e)\parallel B_{23})) \\
\overset{\text{SC}}{=} & c_2(d)\cdot(s_3(d)\parallel B_{23}) + \sum_{e:D}(r_2(e)\cdot(s_3(e)\parallel(c_2(d)s_3(d)\parallel B_{23})))
\end{array}$$

and on the other hand we have,

$\partial_{<s_2(d)>}(s_2(d)\parallel B_{23})$

$$\begin{array}{ll}
= & c_2(d)\cdot\partial_{<s_2(d)>}(s_3(d)\parallel B_{23}) + \sum_{e:D}(r_2(e)\cdot\partial_{<s_2(d)>}(s_3(e)\parallel s_2(d)\parallel B_{23}) \\
\overset{\text{CAA3}}{=} & c_2(d)\cdot(s_3(d)\parallel B_{23}) \\
+ & \sum_{e:D}(r_2(e)\cdot\partial_{<s_2(d)>}(s_3(e)\parallel s_2(d)\parallel B_{23}) \qquad \text{(see note I)} \\
\overset{\text{CAA1}}{=} & c_2(d)\cdot(s_3(d)\parallel B_{23}) \\
+ & \sum_{e:D}(r_2(e)\cdot\partial_{<s_2(d)>}(s_3(e)\parallel\partial_{<s_2(d)>}(s_2(d)\parallel B_{23})) \qquad \text{(see note II)} \\
\overset{\text{CAA3}}{=} & c_2(d)\cdot(s_3(d)\parallel B_{23}) \\
+ & \sum_{e:D}(r_2(e)\cdot(s_3(e)\parallel\partial_{<s_2(d)>}(s_2(d)\parallel B_{23}))
\end{array}$$

Thus we see that both $c_2(d)s_3(d) \parallel B_{23}$ and $\partial_{<s_2(d)>}(s_2(d) \parallel B_{23})$ satisfy the guarded equation:

$$X = c_2(d)(s_3(d) \parallel B_{23}) + \sum_{e:D}(r_2(e) \cdot (s_3(e) \parallel X)).$$

Therefore, by RSP, we have

$$c_2(d)s_3(d) \parallel B_{23} = \partial_{<s_2(d)>}(s_2(d) \parallel B_{23}).$$

This finishes the proof of Theorem 7.1.3.

Note I: Using the facts $\partial_{<s_2(d)>}(s_3(d)) = s_3(d)$ and $\partial_{<s_2(d)>}(B_{23}) = B_{23}$. The first identity directly follows from DA1 and the other one can be proven analogous to (2).

Note II: Because $\bigwedge_{n' \in Part^*(s_2)} \partial_{<n'(d)>}(s(d)) = \partial_{<r_2>}(s_3(d)) = s_3(d)$ we may apply CAA1.

4. $\partial_{H_2}(B_{12} \parallel B_{23})$

$$\begin{array}{ll}
= & \sum_{d:D}(r_1(d) \cdot \partial_{H_2}(s_2(d) \parallel B_{12} \parallel B_{23})) \\
\overset{\text{CAG1, SC}}{=} & \sum_{d:D}(r_1(d) \cdot \partial_{H_2}\partial_{<s_2(d)>}(B_{12} \parallel (s_2(d) \parallel B_{23}))) \\
\overset{\text{CAA1}}{=} & \sum_{d:D}(r_1(d) \cdot \partial_{H_2}\partial_{<s_2(d)>}(B_{12} \parallel \partial_{<s_2(d)>}(s_2(d) \parallel B_{23}))) \qquad \text{(see note A)} \\
\overset{(3)}{=} & \sum_{d:D}(r_1(d) \cdot \partial_{H_2}(B_{12} \parallel c_2(d)s_3(d) \parallel B_{23})) \\
\overset{\text{SC}}{=} & \sum_{d:D}(r_1(d) \cdot \partial_{H_2}(c_2(d)s_3(d) \parallel (B_{12} \parallel B_{23}))) \\
\overset{\text{CAG1}}{=} & \sum_{d:D}(r_1(d) \cdot \partial_{H_2}(c_2(d)s_3(d) \parallel \partial_{H_2}(B_{12} \parallel B_{23}))) \qquad \text{(see note B)} \\
\overset{\text{CAG3}}{=} & \sum_{d:D}(r_1(d) \cdot (c_2(d)s_3(d) \parallel \partial_{H_2}(B_{12} \parallel B_{23}))) \qquad \text{(see note C)}
\end{array}$$

This finishes the proof of Theorem 7.1.4.

Note A: Using $\bigwedge_{n' \in Part^*(s_2)} \partial_{<n'(d)>}(B_{12}) = \partial_{<r_2(d)>}(B_{12}) = B_{12}$ where the last step goes analogous to (2).

Note B: It is easy to see that ∂_{H_2} is the identity for $c_2(d)s_3(d)$. This together with the fact that $\overline{H}_2 \,|\, H_2 - \varnothing = \varnothing \subseteq H_2$ implies that CAG1 may be applied.

Note C: Using the fact that ∂_{H_2} is the identity for $c_2(d)s_3(d)$, and $H_2 \subseteq H_2 \cap H_2 - (\overline{H}_2 \,|\, \overline{H}_2) = H_2 - \varnothing = H_2$.

5. $\tau_{I_2}\partial_{H_2}(B_{12} \parallel B_{23})$

$$\begin{array}{ll}
\overset{(4)}{=} & \tau_{I_2}(\sum_{d:D}(r_1(d) \cdot (c_2(d)s_3(d) \parallel \partial_{H_2}(B_{12} \parallel B_{23})))) \\
\overset{\text{SUM9, TI1}}{=} & \sum_{d:D}(r_1(d) \cdot \tau_{I_2}(c_2(d)s_3(d) \parallel \partial_{H_2}(B_{12} \parallel B_{23}))) \\
\overset{\text{CAG1}}{=} & \sum_{d:D}(r_1(d) \cdot \tau_{I_2}(\tau_{I_2}(c_2(d)s_3(d)) \parallel \tau_{I_2}\partial_{H_2}(B_{12} \parallel B_{23}))) \\
\overset{\text{CAG3}}{=} & \sum_{d:D}(r_1(d) \cdot (\tau_{I_2}(c_2(d)s_3(d)) \parallel \tau_{I_2}\partial_{H_2}(B_{12} \parallel B_{23}))) \\
\overset{\text{TAI1-2}}{=} & \sum_{d:D}(r_1(d) \cdot (\tau \cdot s_3(d) \parallel \tau_{I_2}\partial_{H_2}(B_{12} \parallel B_{23}))) \\
= & \sum_{d:D}(r_1(d) \cdot (s_3(d) \parallel \tau_{I_2}\partial_{H_2}(B_{12} \parallel B_{23}))) \qquad \text{(see note I)}
\end{array}$$

Therefore the process $\tau_{I_2}\partial_{H_2}(B_{12} \parallel B_{23})$ is a solution for the defining equations of B_{13}. By RSP we then have that $B_{13} = \tau_{I_2}\partial_{H_2}(B_{12} \parallel B_{23})$.

This finishes the proof of Theorem 7.1.5.

Note I: $r_1(d)(\tau s_3(d) \parallel x) = r_1(d)\tau s_3(d) \between x = r_1(d)s_3(d) \between x = r_1(d)(s_3(d) \parallel x)$.

6. $\tau_I\partial_H(B_{12} \parallel B_{23} \parallel B_{34})$

$$
\begin{array}{lll}
\overset{\text{CAG5}-6}{=} & \tau_{I_2}\tau_{I_3}\partial_{H_2}\partial_{H_3}(B_{12} \parallel B_{23} \parallel B_{34}) & \\
\overset{\text{CAG7}}{=} & \tau_{I_2}\partial_{H_2}\tau_{I_3}\partial_{H_3}(B_{12} \parallel B_{23} \parallel B_{34}) & \\
\overset{(1)}{=} & \tau_{I_2}\partial_{H_2}\tau_{I_3}\partial_{H_3}(\partial_{H_3}(B_{12}) \parallel B_{23} \parallel B_{34}) & \\
\overset{\text{Lemma 3.2}}{=} & \tau_{I_2}\partial_{H_2}\tau_{I_3}\partial_{H_3}(\partial_{H_3}(B_{12}) \parallel \partial_{H_3}(B_{23} \parallel B_{34})) & \text{(see note I)} \\
\overset{\text{CAG3}}{=} & \tau_{I_2}\partial_{H_2}\tau_{I_3}(\partial_{H_3}(B_{12}) \parallel \partial_{H_3}(B_{23} \parallel B_{34})) & \text{(see note V)} \\
\overset{(1)}{=} & \tau_{I_2}\partial_{H_2}\tau_{I_3}(B_{12} \parallel \partial_{H_3}(B_{23} \parallel B_{34})) & \\
= & \tau_{I_2}\partial_{H_2}\tau_{I_3}(\tau_{I_3}(B_{12}) \parallel \partial_{H_3}(B_{23} \parallel B_{34})) & \text{(see note II)} \\
\overset{\text{Lemma 3.2}}{=} & \tau_{I_2}\partial_{H_2}\tau_{I_3}(\tau_{I_3}(B_{12}) \parallel \tau_{I_3}\partial_{H_3}(B_{23} \parallel B_{34})) & \text{(see note III)} \\
\overset{\text{CAG3}''}{=} & \tau_{I_2}\partial_{H_2}\tau_{I_3}(\tau_{I_3}(B_{12}) \parallel \tau_{I_3}\tau_{H_3}\partial_{H_3}(B_{23} \parallel B_{34})) & \\
\overset{\text{CAG6}}{=} & \tau_{I_2}\partial_{H_2}\tau_{I_3}(\tau_{I_3}(B_{12}) \parallel \tau_{I_3\cup H_3}\partial_{H_3}(B_{23} \parallel B_{34})) & \\
\overset{\text{CAG4}}{=} & \tau_{I_2}\partial_{H_2}(\tau_{I_3}(B_{12}) \parallel \tau_{I_3}\partial_{H_3}(B_{23} \parallel B_{34})) & \text{(see note IV)} \\
= & \tau_{I_2}\partial_{H_2}(B_{12} \parallel \tau_{I_3}\partial_{H_3}(B_{23} \parallel B_{34})) & \text{(see note II)} \\
\overset{(5)}{=} & \tau_{I_2}\partial_{H_2}(B_{12} \parallel B_{24}) & \\
\overset{(5)}{=} & B_{14} &
\end{array}
$$

This finishes the proof of Theorem 7.1.6.

Note I: $\overline{H_3} \mid H_3 = \{r_1, r_2, s_1, s_2, c_2, c_3\} \mid \{r_3, s_3\} = \varnothing \subseteq H_3$.
Note II: Using that $\tau_{I_3}(B_{12}) = B_{12}$; the proof of this identity goes analogous to (1).
Note III: $\overline{I_3} \mid I_3 = \{r_1, r_2, r_3, s_1, s_2, s_3, s_4, c_1, c_2\} \mid \{c_3\} = \varnothing$.
Note IV: $I_3 \subseteq I_3 \cap (I_3 \cup H_3) - (\overline{I_3} \mid \overline{I_3 \cup H_3}) = I_3 - \{c_1, c_2\} = I_3$.
Note V: $H_3 \subseteq (H_3 \cap H_3) - (\overline{H_3} \mid \overline{H_3}) = \{r_3, s_3\} - \{c_2\} = H_3$.

□

8 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 7.1.5 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]).[6] 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 7.1.6. 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.

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

Acknowledgements

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

A An overview of the proof theory for μCRL

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

A.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 labels of actions. The names in $\mathcal{N}$ are words over an alphabet not containing

$$\bot, +, \|, \between, |, \lhd, \rhd, \cdot, \delta, \tau, \partial, \rho, \sum, \surd, \times, \rightarrow, :, =,), (, \}, \{, \text{,}, \texttt{space}, \texttt{newline}, \mathsf{t}, \mathsf{f}.$$

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 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.

As an example we define the Booleans. The Booleans must be included in each μCRL specification.

sort **Bool**
func $\mathsf{t}, \mathsf{f} :\rightarrow$ **Bool**

The following example shows how natural numbers with a zero, a successor, addition and multiplication can be declared.

Example A.1.

sort Nat
func $0 :\rightarrow Nat$
$S : Nat \rightarrow Nat$
$add, times : Nat \times Nat \rightarrow Nat$
var $x, y : Nat$
rew $add(x, 0) = x$
$add(x, S(y)) = S(add(x, y))$
$times(x, 0) = 0$
$times(x, S(y)) = add(times(x, y), x)$

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.

act a, b, c
$a, d : Nat$

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 labels of actions. For instance, in

comm $in \mid out = com$

each action $in(t_1, ..., t_k)$ can communicate with $out(t'_1, ..., t'_m)$ to $com(t_1, ..., t_k)$ provided $k = m$ and t_i and t'_i denote the same data element for $i = 1, ..., k$.

Processes are declared using the keyword **proc**. An example is

proc $counter(x : Nat) = p$
$buffer = q$

In the first line a counter is declared. It is a process with one parameter x of sort Nat. The parameter x may be used in the process term p that specifies its behaviour. In the second line a parameterless process *buffer* is declared. Its behaviour is given by the process term q.

Definition A.2. *(Process terms).* An expression p is called a *process term* iff p has the following syntax:

$$\begin{aligned} p \quad ::= \quad & (p + p) \mid (p \cdot p) \mid (p \parallel p) \mid (p \between p) \mid (p \mid p) \mid (p \lhd t \rhd p) \\ \mid \quad & \sum_{d:D}(p) \mid \partial_H(p) \mid \tau_I(p) \\ \mid \quad & \rho_R(p) \mid \delta \mid \tau \mid n \mid n(t_1, ..., t_m). \end{aligned}$$

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 $\{n_1 \to n'_1, ..., n_m \to n'_m\}$.

End Definition A.2.

Most operators stem from ACP [BW90]. Only the conditional construct $p \lhd t \rhd p$ 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.

A.1.1 Axioms of ACP

Table 9 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 $\overline{t}$ for a sequence of data terms where ϵ is the empty sequence. The symbols a, b represent δ, τ or range over (declared) actions $n(\overline{t})$, where $n(\overline{t})$ represents n if $\overline{t} = \epsilon$. $\tilde{\gamma}$ is the pre-communication function such that $\tilde{\gamma}(n_1, n_2) = n_3$ if a rule **comm** $n_1 \mid n_2 = n_3$ appears in the μCRL specification. Otherwise $\tilde{\gamma}(n_1, n_2) = \delta$. γ is the symmetrical closure of $\tilde{\gamma}$.

A.1.2 Axioms of Standard Concurrency

In Table 10 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 A.3.

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

A1	$x + y = y + x$	CF	$n(\overline{t}) \mid m(\overline{t}) = \begin{cases} \gamma(n,m)(\overline{t}) & \text{if } \gamma(n,m) \text{ defined} \\ \delta & \text{otherwise} \end{cases}$
A2	$x + (y + z) = (x + y) + z$		
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	$\delta \mid x = \delta$
		CD2	$x \mid \delta = \delta$
CM1	$x \parallel y = x \between y + y \between x + x \mid y$	CT1	$\tau \mid x = \delta$
CM2	$a \between x = a \cdot x$	CT2	$x \mid \tau = \delta$
CM3	$a \cdot x \between y = a \cdot (x \parallel y)$		
CM4	$(x + y) \between z = x \between z + y \between z$	DD	$\partial_H(\delta) = \delta$
CM5	$a \cdot x \mid b = (a \mid b) \cdot x$	DT	$\partial_H(\tau) = \tau$
CM6	$a \mid b \cdot x = (a \mid b) \cdot x$	D1	$\partial_H(n(\overline{t})) = n(\overline{t})$ if $n \notin H$
CM7	$a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$	D2	$\partial_H(n(\overline{t})) = \delta$ if $n \in H$
CM8	$(x + y) \mid z = x \mid z + y \mid z$	D3	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
CM9	$x \mid (y + z) = x \mid y + x \mid z$	D4	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$

Table 9: The axioms of ACP in μCRL.

$(x \between y) \between z = x \between (y \parallel z)$	$(x \mid y) \mid z = x \mid (y \mid z)$
$x \parallel \delta = x\delta$	$x \mid (y \between z) = (x \mid y) \between z$
$x \mid y = y \mid x$	$x \mid (y \mid z) = \delta$ Handshaking

Table 10: Axioms of Standard Concurrency (SC).

A.1.3 Axioms for the hiding operator

For abstraction in μCRL we present the axioms in Table 11. Here $I \subseteq \mathcal{N}$ is the set of gate names to be abstracted from.

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

Table 11: Axioms for abstraction.

A.1.4 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 12) 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	$\sum_{d:D}(p_1 \between p_2) = \sum_{d:D}(p_1) \between p_2$	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	$\dfrac{\begin{matrix}\mathcal{D} \\ p_1 = p_2\end{matrix}}{\sum_{d:D}(p_1) = \sum_{d:D}(p_2)}$	provided d not free in the assumptions of $\mathcal{D}$

Table 12: Axioms for the sum operator.

A.1.5 Axioms for the conditional construct

The axioms for the *conditional* construct $p \lhd c \rhd q$, are given in Table 13. 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).

A.1.6 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].

Cond1	$x \lhd \mathsf{t} \rhd y$	$=$	x
Cond2	$x \lhd \mathsf{f} \rhd y$	$=$	y

Table 13: Axioms for the conditional construct.

A.1.7 τ-law

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

B1	$x\,\tau$	$=$	x

Table 14: 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 1991*, 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.

[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.

[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 Semantics of Specification Languages,* Utrecht, The Netherlands, pages 315–339. Workshops in Computer Science, Springer-Verlag, 1993.