A Correctness Proof of the Bakery Protocol in μCRL

Jan Friso Groote

Department of Philosophy, Utrecht University

Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

email: JanFriso.Groote@phil.ruu.nl

Henri Korver

Department of Software Technology, CWI

P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

email: Henri.Korver@cwi.nl

Abstract

A specification of the bakery protocol is given in μCRL. We provide a simple correctness criterion for the protocol. Then the protocol is proven correct using a proof system that has been developed for μCRL. The proof primarily consists of algebraic manipulations based on specifications of abstract datatypes and elementary rules and axioms from process algebra.

Key Words & Phrases: Bakery Protocol, Protocol Verification, Process Algebra, Abstract Datatypes, μCRL.
1985 Mathematics Subject Classification: 68B10.
1987 CR Categories: D.2.4, D.4.5, F.3.1.
Note: The authors are partly supported by the European Communities under RACE project no. 1046, Specification and Programming Environment for Communication Software (SPECS). This document does not necessarily reflect the view of the SPECS project.

1 Motivation

The main goal of this paper is to show that μCRL, or in more general terms process algebra with abstract datatypes, offers a framework for verifications of processes that goes beyond those found in both its constituents. Process algebra in its basic form does not include processes that are parametrised with data: parameterised sums, conditionals, parametrised actions, etc., and very importantly induction over these parameters. In our verification of the bakery protocol, such an induction is an essential step.

Our work structurally differs from the more conventional 'assertional' verification techniques ([1, 2, 4]). These are mainly based on data and do not often allow for algebraic reductions of processes. In particular, simple and elegant correctness identities such as given in section 2 cannot be formulated.

There are two other points that deserve mention. First, the proof system of μCRL has been defined in such way that it allows for automatic proof checking ([13]). This is important, as a minor mistake in a program or a protocol may have disastrous impacts. And actually, we have so often detected 'oversights' in our calculations that we may expect that also the proof in this paper is not completely flawless. The only way to systematically increase the correctness of proofs is by having these automatically checked using a computer tool. This of course does not decrease the value of this paper, because finding a proof remains the essential step in a verification.

The other point is about the proof in this paper. Although the proof was not easy to find, due to the large number of possible proof strategies, the resulting proof follows a reasonable and straightforward

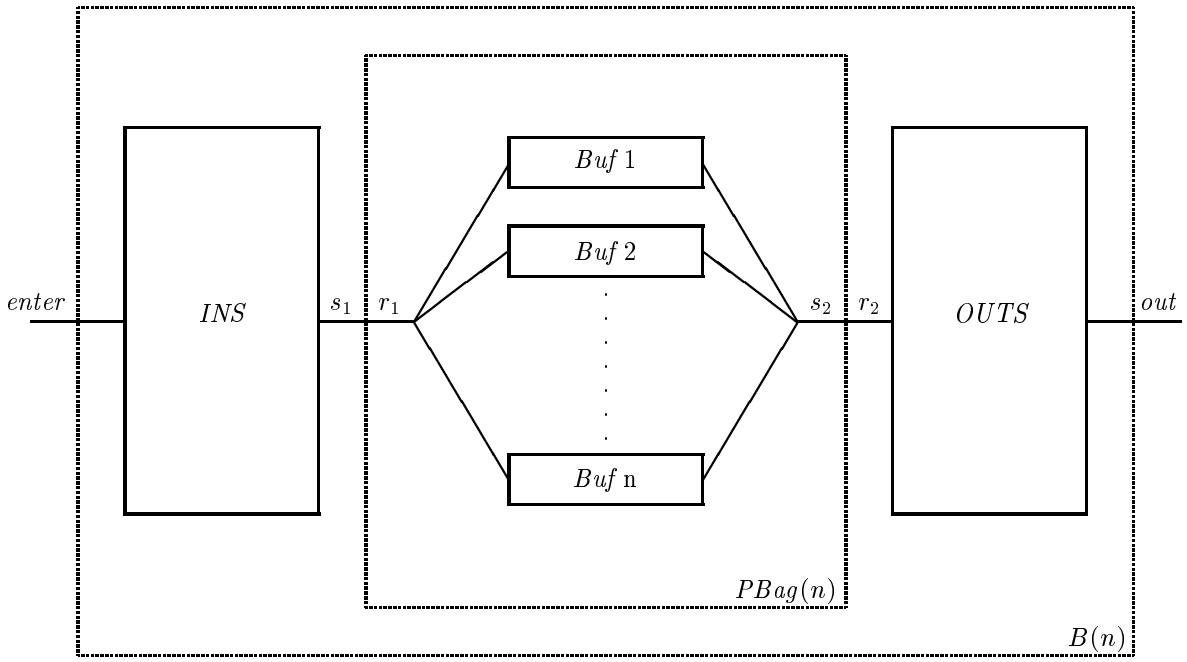

Figure 1: The bakery protocol.

line of thought. This is promising, because we think that if we get more skill and experience in doing calculations such as given in the paper, most communication protocols can be verified in μCRL by a fixed selection of standard strategies.

2 The specification of the bakery protocol

We describe a simple system that captures the well-known bakery protocol [10] and prove its correctness in the proof system for μCRL [8, 9]. We assume that the reader is familiar with μCRL which is a straightforward combination of process algebra [3] and abstract datatypes [6]. A summary of the proof system is given in appendix A.

The bakery protocol derives its name from the well-known situation in a busy bakery where customers pick a number when entering the shop in order to guarantee that they are served in proper order. The system basically consists of n 1-place buffers (see *Buf* in figure 1), that may each contain a customer waiting to be served. Before waiting, each customer picks a sequence number (which are distributed modulo n) indicating when it is his turn. This is modeled by the in-sequencer *INS* in figure 1. A customer is served when his number matches that of the baker, modeled by the out-sequencer *OUTS* in figure 1. The system is supposed to work on a first come first served basis, i.e. it should behave like a queue. We take the existence of basic datatypes, which are in this case booleans and natural numbers, for granted. These datatypes are specified in appendix B. We also need modulo calculations, e.g. for specifying the in-sequencer *INS* and out-sequencer *OUTS*. For this purpose $+_n$ is introduced, which is addition modulo n. Its specification can also be found in appendix B.

The customers are supposed to be given by a (non empty) sort D. Sort D contains a bottom element $d_\perp$ for denoting undefined data elements. We assume that D is equipped with an equality function $eq : D \times D \to \mathbf{Bool}$ that has the obvious property $eq(d, e) = T \leftrightarrow d = e$ where $d, e \in D$. Furthermore,

we have a sort $queue_d$ that consists of queues of customers (see appendix B for its specification). In order to attach a number to a customer the datatype $Pair$ is introduced together with a pairing ($\langle \_, \_ \rangle$) and an equality function (eq). We do not completely obey the syntax of μCRL to increase readability, e.g. by using infix notation and omitting $\cdot$ for sequential composition.

$$
\begin{array}{ll}
\textbf{sort} & Pair \\
\textbf{func} & \langle\,,\rangle : D \times nat \to Pair \\
 & eq : Pair \times Pair \to \textbf{Bool} \\
\textbf{var} & d, e : D \\
 & n, m : nat \\
\textbf{rew} & eq(\langle d, n\rangle, \langle e, m\rangle) = eq(d, e) \text{ and } eq(n, m)
\end{array}
$$

We also need queues which can contain pairs. Therefore, the datatype $queue_p$ is introduced in appendix B.

A buffer process that can contain a customer with a ticket is straightforwardly specified as follows.

$$
\begin{array}{ll}
\textbf{act} & r_1, s_2 : Pair \\
\textbf{proc} & Buf = \Sigma(p : Pair, r_1(p) \cdot s_2(p))Buf
\end{array}
$$

A customer with a ticket, modeled by the pair $\langle d, i\rangle$, can enter the buffer at gate r_1 and leave it at gate s_2.

By putting n of these buffers in parallel, we model that n customers can wait in the shop. As this is the behaviour of a bag which is essentially described by processes, we call this process $PBag$, derived from 'Process bag'.

$$
\textbf{proc} \quad PBag(n : nat) = \delta \lhd eq(n, 0) \rhd (Buf \parallel PBag(n-1))
$$

Note the way in which $PBag$ has been recursively defined, e.g. $PBag(1) = \delta \parallel Buf$ which exactly corresponds to our intuition because $\delta \parallel Buf = Buf$. [1]

The process $INS(n, i)$ assigns a successive number modulo n to each customer. The number i represents the first number that is assigned. A customer enters at the entrance of the bakery, represented by the action $enter(d)$. With a number he walks into the shop, which is modelled by $s_1(\langle d, i\rangle)$. The fact that he directly enters a place in a buffer is modelled by a communication between s_1 and r_1. The process $OUTS(n, i)$ selects the customer to be served. In this case i represents the first number that will be served. Entering $OUTS$ is modelled via the r_2 gate that must communicate with gate s_2. After being served the customer leaves the counter via out.

$$
\begin{array}{ll}
\textbf{act} & enter, out : D \\
 & s_1, r_2, c_1, c_2 : Pair \\
\textbf{proc} & INS(n : nat, i : nat) = \Sigma(d : D, enter(d)\, s_1(\langle d, i\rangle))\, INS(n, i +_n 1) \\
 & OUTS(n : nat, i : nat) = \Sigma(d : D, r_2(\langle d, i\rangle)\, out(d))\, OUTS(n, i +_n 1)
\end{array}
$$

The whole bakery $B(n)$ is given by:

$$
\begin{array}{ll}
\textbf{comm} & r_1 \mid s_1 = c_1,\ r_2 \mid s_2 = c_2 \\
\textbf{proc} & B(n : nat) = \tau(\{c_1, c_2\}, \partial(\{r_1, s_1, r_2, s_2\}, INS(n, 0) \parallel PBag(n) \parallel OUTS(n, 0)))
\end{array}
$$

3 The correctness criterion for the bakery protocol

The bakery protocol $B(n)$ is supposed to work as a bounded queue of size $n + 2$; there can be n customers waiting in the buffers, one can be busy obtaining a number and one can already be selected

[1] In general, the identity $\delta \parallel x = x$ does not hold, e.g. consider the counter example $\delta \parallel a = a\delta \neq a$. However, the identity $\delta \parallel Buf = Buf$ does hold because Buf is a non-terminating process.

to be served. The 'standard' specification of a process $Q(n)$ modelling a queue of size n containing elements of D is:

$$\begin{aligned}\textbf{proc}\quad & Q(n:nat, b:queue_d) = \\ & \quad \Sigma(d:D, enter(d)\cdot Q(n, in(d,b))) \lhd size(b) < n \rhd \delta \; + \\ & \quad out(toe(b))\cdot Q(n, untoe(b)) \lhd size(b) > 0 \rhd \delta \\ & \\ & Q(n:nat) = Q(n, \varnothing_d)\end{aligned}$$

With this specification of a queue the correctness of the bakery protocol is stated as follows.

$$\boxed{n > 0 \rightarrow B(n) = Q(n+2)}$$

The condition $n > 0$ is necessary to guarantee that there is at least one buffer place for a customer to wait. Otherwise, as is easy to see, no customer can reach the counter.

4 Basic lemmas for μCRL

In this section, we present a number of elementary lemmas that are used in the verification of the bakery. These lemmas are interesting in their own right as it is very likely that they are necessary in almost every verification in μCRL.

In this section, we assume that the reader is familiar with the following conventions about open terms and variables. The letters $d, e, \ldots$ stand for *data* variables. The symbols $t, t_1, t_2, \ldots$ stand for *open* data terms. Open data terms are data terms that may contain data variables such as d, e. The symbols $b, b_1, b_2, \ldots$ stand for data variables of sort **Bool** and the symbols $c, c_1, c_2, \ldots$ stand for open data terms of sort **Bool**. The letters $x, y, z, \ldots$ stand for *process* variables. The symbols $p, p_1, p_2, \ldots$ stand for *open* process terms. Open process terms are terms that may contain process variables such as x, y and (open) data terms like t, t_1, t_2. Note that in this setting open terms are more general than variables because an open term can also be a variable by definition.

The following lemma shows that for applying an induction on a boolean variable b, one only has to check the cases $b = T$ and $b = F$.

Lemma 4.1. *(Specialised induction rule for* **Bool***).*

$$(p = q)[T/b] \wedge (p = q)[F/b] \rightarrow p = q$$

Proof. The rule above is an instance of the induction rule IND($\overline{C}$) by setting $\overline{C} = \{T, F\}$. The instantiation is straightforward because the constructor set $\{T, F\}$ of sort **Bool** only consists of constants (see lemma B.1). □

The following conditional identities often occur in μCRL verifications.

Lemma 4.2.

1. $x \lhd b \rhd x = x$,
2. $x + x \lhd b \rhd \delta = x$,
3. $(x + y) \lhd b \rhd z = x \lhd b \rhd z + y \lhd b \rhd z$,
4. $(x + y) \lhd b \rhd y = x \lhd b \rhd \delta + y$,
5. $(b = T \rightarrow b' = F) \rightarrow x \lhd b \rhd y = (x + z \lhd b' \rhd \delta) \lhd b \rhd y$

6. $x \cdot y \lhd b \rhd \delta = (x \lhd b \rhd \delta) \cdot (y \lhd b \rhd z)$,
7. $a \cdot (x \lhd b \rhd y) = a\,x \lhd b \rhd a\,y$,
8. $(x \lhd b \rhd \delta) \between y = (x \between y) \lhd b \rhd \delta$,
9. $x \mid (y \lhd b \rhd \delta) = (x \mid y) \lhd b \rhd \delta$,
10. $p[e/d] = p \lhd eq(d,e) \rhd \delta + p[e/d]$, *provided that* $eq(d,e) \rightarrow d = e$,
11. $(a(d) \cdot x \mid b(e) \cdot y) \lhd c \rhd \delta = ((a(d) \cdot x \mid b(e) \cdot y) \lhd c \rhd \delta) \lhd eq(d,e) \rhd \delta$, *provided that* $d = e \rightarrow eq(d,e)$.

Proof.

1–10. Easy with axioms C1, C2 and lemma 4.1.

11. By C1,C2,CF2′ and lemma 4.1.

□

The following lemma presents a rule which is derived from the SUM axioms. This rule appears to be a powerful tool to eliminate sum expressions in μCRL calculations.

Lemma 4.3. *(Sum Elimination).Assume that there is an equality function eq such that* $eq(d,e) = T \leftrightarrow d = e$. *Then*

$$\Sigma(d : D, p \lhd eq(d,e) \rhd \delta) = p[e/d].$$

Proof. Consider the following obvious identity

$$\Sigma(d : D, p \lhd eq(d,e) \rhd \delta) \stackrel{\text{SUM3}}{=} \Sigma(d : D, p \lhd eq(d,e) \rhd \delta) + p \lhd eq(d,e) \rhd \delta \tag{1}$$

By applying the substitution $[e/d]$ to equation 1 we obtain

$$\Sigma(d : D, p \lhd eq(d,e) \rhd \delta) = \Sigma(d : D, p \lhd eq(d,e) \rhd \delta) + p[e/d] \lhd eq(e,e) \rhd \delta \tag{2}$$

Note that the substitution does not effect the sum construct because substitutions do not change bound variables by definition (see appendix A). By applying axiom C1 to the second summand on the right hand side of equation 2, we get

$$\Sigma(d : D, p \lhd eq(d,e) \rhd \delta) = \Sigma(d : D, p \lhd eq(d,e) \rhd \delta) + p[e/d] \tag{3}$$

Now, we know that $p[e/d]$ is a semantical summand of the left-hand side of equation 3, notated as $p[e/d] \subseteq \Sigma(d : D, p \lhd eq(d,e) \rhd \delta)$. The following derivation

$$\begin{array}{lll} p[e/d] & \stackrel{\text{SUM1}}{=} & \Sigma(d : D, p[e/d]) \\ & \stackrel{4.2.10}{=} & \Sigma(d : D, p \lhd eq(d,e) \rhd \delta + p[e/d]) \\ & \stackrel{\text{SUM1,SUM4}}{=} & \Sigma(d : D, p \lhd eq(d,e) \rhd \delta) + p[e/d] \end{array}$$

shows that we have a summand inclusion $\Sigma(d : D, p \lhd eq(d,e) \rhd \delta) \subseteq p[e/d]$ in the other direction which establishes the proof. [2] □

[2] Here, we use the summand trick, $p \subseteq q \wedge q \subseteq p \rightarrow p = q$, for deriving the desired equational identity. More information on summands in process algebra can be found in [3] (pages 19 and 21).

In the next lemma, we generalise axiom CM3 with a conditional construct and a sum operator.

Lemma 4.4. *(Left Merge with SUM and COND). We assume that the variable d does not occur free in term q.*

1. $\Sigma(d:D, a(d)\cdot p) \between q = \Sigma(d:D, a(d)\cdot(p \parallel q))$
2. $\Sigma(d:D, a(d)\cdot p \lhd c \rhd \delta) \between q = \Sigma(d:D, a(d)\cdot(p \parallel q) \lhd c \rhd \delta)$

Proof.

1. By the following calculation

$$\begin{array}{ll} & \Sigma(d:D, a(d)\cdot p) \between q \\ \overset{\text{SUM6}}{=} & \Sigma(d:D, a(d)\cdot p \between q) \\ \overset{\text{CM3}}{=} & \Sigma(d:D, a(d)\cdot(p \parallel q)) \end{array}$$

2. By the following calculation

$$\begin{array}{ll} & \Sigma(d:D, a(d)\cdot p \lhd c \rhd \delta) \between q \\ \overset{\text{SUM6}}{=} & \Sigma(d:D, a(d)\cdot p \lhd c \rhd \delta \between q) \\ \overset{4.2.8}{=} & \Sigma(d:D, a(d)\cdot p \between q \lhd c \rhd \delta) \\ \overset{\text{CM3}}{=} & \Sigma(d:D, a(d)\cdot(p \parallel q) \lhd c \rhd \delta) \end{array}$$

□

There are two remarks about the lemma above. At first, note that we can avoid the restriction that d is not allowed to occur free in q by renaming it with axiom SUM2. So, the restriction is just a formality and no generality is lost. Secondly, note that in stating properties about sum expressions as in 4.4.2, we often use an open data term c instead of an ordinary data variable b to be as general as possible (see appendix B).

Next, we generalise axiom CM7 with a conditional construct and a sum operator.

Lemma 4.5. *(Communication with SUM and COND). Assume there is an equality function eq such that $eq(d,e) = T \leftrightarrow d = e$, and variable $d:D$ does not occur free in terms q and c_2. d' does not occur free in p and c_1.*

1. $\Sigma(d:D, a(d)\cdot p) \mid b(e)\cdot q = (a(e) \mid b(e))\cdot(p[e/d] \parallel q)$
2. $\Sigma(d:D, a(d)\cdot p \lhd c \rhd \delta) \mid b(e)\cdot q = (a(e) \mid b(e))\cdot(p[e/d] \parallel q) \lhd c[e/d] \rhd \delta$
3. $\Sigma(d:D, a(d)\cdot p \lhd c \rhd \delta) \mid \Sigma(d':D', b(t)\cdot q) = \Sigma(d':D', (a(t) \mid b(t))\cdot(p[t/d] \parallel q) \lhd c[t/d] \rhd \delta)$
4. $\Sigma(d:D, a(d)\cdot p \lhd c_1 \rhd \delta) \mid \Sigma(d':D', b(t)\cdot q \lhd c_2 \rhd \delta) =$
   $\Sigma(d':D', (a(t) \mid b(t))\cdot(p[t/d] \parallel q) \lhd c_1[t/d] \text{ and } c_2 \rhd \delta)$

Proof.

1. By the following calculation

$$\begin{array}{ll} & \Sigma(d:D, a(d)\cdot p) \mid b(e)\cdot q \\ \overset{\text{SUM7}}{=} & \Sigma(d:D, a(d)\cdot p \mid b(e)\cdot q) \\ \overset{4.2.11,\ \text{SUM11}}{=} & \Sigma(d:D, (a(d)\cdot p \mid b(e)\cdot q) \lhd eq(d,e) \rhd \delta) \\ \overset{\text{SumEl.}}{=} & (a(e)\cdot p[e/d]) \mid b(e)\cdot q) \\ \overset{\text{CM7}}{=} & (a(e) \mid b(e))\cdot(p[e/d]) \parallel q) \end{array}$$

Note that the assumption $eq(d,e) = T \leftrightarrow d = e$ is needed for application of lemmas 4.2.11 and 4.3 (Sum Elimination) in the calculation above.

2. By the following calculation

$$\begin{array}{ll} & \Sigma(d:D, a(d)\cdot p \lhd c \rhd \delta) \mid b(e)\cdot q \\ \overset{\text{SUM7}}{=} & \Sigma(d:D, (a(d)\cdot p \lhd c \rhd \delta) \mid b(e)\cdot q) \\ \overset{4.2.9}{=} & \Sigma(d:D, (a(d)\cdot p \mid b(e)\cdot q) \lhd c \rhd \delta) \\ \overset{4.2.11}{=} & \Sigma(d:D, ((a(d)\cdot p \mid b(e)\cdot q) \lhd c \rhd \delta) \lhd eq(d,e) \rhd \delta) \\ \overset{\text{SumEl.}}{=} & (a(e)\cdot p[e/d]) \mid b(e)\cdot q) \lhd c[e/d] \rhd \delta \\ \overset{\text{CM7}}{=} & (a(e) \mid b(e))\cdot (p[e/d]) \parallel q) \lhd c[e/d] \rhd \delta \end{array}$$

3. By the following calculation

$$\begin{array}{ll} & \Sigma(d:D, a(d)\cdot p \lhd c \rhd \delta) \mid \Sigma(d':D', b(t)\cdot q) \\ \overset{\text{SC3,SUM7}}{=} & \Sigma(d':D', \Sigma(d:D, a(d)\cdot p \lhd c \rhd \delta) \mid (b(t)\cdot q)) \\ \overset{4.5.2}{=} & \Sigma(d':D', (a(t) \mid b(t))\cdot (p[t/d] \parallel q) \lhd c[t/d] \rhd \delta) \end{array}$$

4. In the same way as the proof of 4.5.3.

□

Note the curious role of the function $eq(d,e)$ which is used as a catalyst in the proof of lemma 4.5. There are some interesting questions about such use of an equality function. Is eq really needed in proofs such as given above? And can every datatype specification in a μCRL specification be (consistently) extended with an equality function eq such that it satisfies the property $eq(d,e) = T \leftrightarrow d = e$? At last, is it possible to introduce such eq functions by the proof system itself?

5 τ-laws for μCRL

The proof system as presented in [9] is considered as a kernel and does not yet contain axioms for τ. In this section, we extend the proof theory with axioms for τ as we need these axioms in the verification of the bakery protocol. In appendix A, we have revised the original proof system in order to make it compatible with the τ-axioms. Below, we summarise the differences between the original and the adapted proof system.

- In contrast with the original version, the a and b in table 6 do not range over τ. The τ-laws of observation equivalence (see below) may cause inconsistencies when a and b range over τ.
- The standard axiom, SC4: $(x \mid y) \between z = x \mid (y \between z)$, in the original version is replaced here by the weaker axiom $(x \mid ay) \between z = x \mid (ay \between z)$. The former causes inconsistencies when adding all the τ-laws of observation equivalence to the proof system.
- The original RSP rule does not work in a setting of τ-axioms as τ cannot occur as a guard. Therefore, it has slightly been adapted (see definition A.4).

We discuss two approaches for adding τ-laws to the proof system.

T1	$x\,\tau$	$=$	x
T2	$\tau\,x$	$=$	$\tau\,x + x$
T3	$a\,(\tau\,x + y)$	$=$	$a\,(\tau\,x + y) + a\,x$

Table 1: τ-laws for observation equivalence.

5.1 Observation equivalence

One can add the τ-laws of table 1 taken from MILNER [11] to the proof system. These axioms correspond to the well-known *observation equivalence semantics* for τ-actions.

These axioms can be added to the proof system under the restriction that the a and b in table 6 do not range over τ. Otherwise, we are able to derive inconsistent identities (see [3], page 165). The axioms in table 2 model the interaction between τ and the other operators (see [3]).

D0	$\delta_H(\tau)$	$=$	τ
TI0	$\tau_I(\tau)$	$=$	τ
TM1	$\tau \mathbin{\underline{\parallel}} x$	$=$	τx
TM2	$\tau x \mathbin{\underline{\parallel}} y$	$=$	$\tau(x \parallel y)$
TC1	$\tau \mid x$	$=$	δ
TC2	$x \mid \tau$	$=$	δ
TC3	$\tau x \mid y$	$=$	$x \mid y$
TC4	$x \mid \tau y$	$=$	$x \mid y$

Table 2: Completing τ-laws for observation equivalence.

5.2 Branching bisimulation

Another approach, is to extend the proof system with the axioms of table 3 taken from VAN GLABBEEK AND WEIJLAND [7].

B1	$x\,\tau$	$=$	x
B2	$z\,(\tau\,(x + y) + x)$	$=$	$z\,(x + y)$

Table 3: τ-laws for branching bisimulation.

Now, in contrast with observation semantics, the a and b in table 6 do not cause inconsistencies while ranging over τ and we directly obtain a complete axiomatisation for branching bisimulation wrt. the process part.

5.3 About the τ-laws in the bakery proof

In the bakery proof, we only use axioms T1 and T2. It appears that in our proof these axioms work out easier than purely restricting ourselves to the application of B1 and B2 only. However, it is very likely that with a bit of extra complexity the proof can be done completely within branching bisimulation semantics.

6 The correctness proof of the bakery protocol

In this section we prove that the bakery protocol $B(n)$ indeed satisfies the criterion as stated in section 3. The proof transforms the process style description in two steps into a data style description. First, we show that $PBag(n)$ behaves as a 'standard' bounded bag which is usually described by a datatype bag. Then we show that this bounded bag combined with the in-sequencer INS and the out-sequencer $OUTS$ is equal to the bounded bag described above.

Part I: $PBag(n)$ is a bounded bag

We show that $PBag(n)$ behaves like a bounded bag of size n. First we specify the standard behaviour of a bounded bag, where the behaviour mainly relies on the datatype $queue_p$; the D in $DBag$ refers to 'Data'. Although the datatype $queue_p$ is actually a queue, it has been extended with functions $test$ and rem (remove) which makes it possible to use it as a bag.

$$\begin{aligned}&\textbf{proc } DBag(n : nat, b : queue_p) = \\ &\qquad \Sigma(p : Pair, r_1(p) \cdot DBag(n, in(p, b))) \lhd size(b) < n \rhd \delta \; + \\ &\qquad \Sigma(p : Pair, s_2(p) \cdot DBag(n, rem(p, b)) \lhd test(p, b) \rhd \delta)\end{aligned}$$

$$DBag(n : nat) = DBag(n, \varnothing_p)$$

The following process forms an intermediate between a bag described in process style and a bag described in data style. The description tells us, given an element b of type $queue_p$, what the corresponding process describing a bag in process style looks like. In the next theorem we relate (rather straightforwardly) $PBag(n, b)$ with $DBag(n, b)$ (see 6.1.5) and via these we show that $DBag(n) = PBag(n)$ (see 6.1.7) showing that $PBag(n)$ is a bounded bag.

$$\begin{aligned}&\textbf{proc } PBag(n : nat, b : queue_p) = \\ &\qquad \delta \lhd eq(n, 0) \rhd \\ &\qquad\quad ((Buf \parallel PBag(P(n), b)) \\ &\qquad\quad \lhd empty(b) \rhd \\ &\qquad\quad (s_2(hd(b)) \cdot Buf \parallel PBag(P(n), tl(b))))\end{aligned}$$

In the specification above, we use a function hd that extracts the head element of the queue and a function tl that returns the tail of the queue (these functions are specified in appendix B).

Theorem 6.1. *For all $p, p' : Pair$, $b : queue_p$ and $n : nat$ it holds that:*

1. $s_2(p) \cdot Buf \parallel PBag(n, b) = PBag(n + 1, in(p, b))$,
2. $PBag(n + 2, in(p, in(p', b))) = PBag(n + 2, in(p', in(p, b)))$,
3. $size(b) \leq n \rightarrow Buf \parallel PBag(n, b) = PBag(n + 1, b)$,
4. $\neg b = \varnothing_p \rightarrow$
   $\Sigma(p : Pair, s_2(p)\, PBag(n, rem(p, b)) \lhd test(p, b) \rhd \delta) =$
   $\Sigma(p : Pair, s_2(p)\, PBag(n, rem(p, b)) \lhd test(p, tl(b)) \rhd \delta) + s_2(hd(b))\, PBag(n, tl(b))$,
5. $size(b) \leq n \rightarrow DBag(n, b) = PBag(n, b)$,
6. $PBag(n) = PBag(n, \varnothing_p)$,
7. $DBag(n) = PBag(n)$.

Proof. The proof below is rather straightforward, although a large number of details need to be checked. The most complicated is the proof of case 5 in which an application of RSP within an application of induction appears.

1. This fact follows directly:

$$\begin{array}{ll} & PBag(n+1, in(p,b)) \\ = & s_2(hd(in(p,b))) \cdot Buf \parallel PBag(P(n+1), tl(in(p,b))) \\ = & s_2(p) \cdot Buf \parallel PBag(n,b) \end{array}$$

2. $$\begin{array}{ll} & PBag(n+2, in(p, in(p',b))) \\ \overset{6.1.1}{=} & s_2(p) \cdot Buf \parallel PBag(n+1, in(p',b)) \\ \overset{6.1.1}{=} & s_2(p) \cdot Buf \parallel s_2(p') \cdot Buf \parallel PBag(n,b) \\ \overset{A.3.1}{=} & s_2(p') \cdot Buf \parallel s_2(p) \cdot Buf \parallel PBag(n,b) \\ \overset{6.1.1}{=} & s_2(p') \cdot Buf \parallel PBag(n+1, in(p,b)) \\ \overset{6.1.1}{=} & PBag(n+2, in(p', in(p,b))) \end{array}$$

3. This statement is shown with induction on n.

   - For $n = 0$ we have:

$$\begin{array}{ll} & PBag(0+1,b) \\ = & (Buf \parallel PBag(0,b)) \\ & \lhd empty(b) \rhd \\ & (s_2(hd(b)) \cdot Buf \parallel PBag(0, tl(b))) \\ \overset{B.10.5}{=} & Buf \parallel PBag(0,b) \end{array}$$

   As $size(b) \leq 0$ we know that $b = \varnothing_p$.

   - If $n = S(m)$ then

$$\begin{array}{ll} & PBag(n+1,b) \\ = & (Buf \parallel PBag(P(n+1),b)) \\ & \lhd empty(b) \rhd \\ & (s_2(hd(b)) \cdot Buf \parallel PBag(P(n+1), tl(b))) \\ & \\ = & ((Buf \parallel PBag(m+1,b)) \\ & \lhd empty(b) \rhd \\ & (s_2(hd(b)) \cdot Buf \parallel PBag(m+1, tl(b)))) \\ & \\ \overset{I.H.}{=} & ((Buf \parallel PBag(m+1,b)) \\ & \lhd empty(b) \rhd \\ & (s_2(hd(b)) \cdot Buf \parallel Buf \parallel PBag(m, tl(b)))) \\ & \\ \overset{6.1.1,\ B.10.3}{=} & (Buf \parallel PBag(m+1,b)) \\ & \lhd empty(b) \rhd \\ & (Buf \parallel PBag(m+1,b)) \\ & \\ \overset{4.2.1}{=} & Buf \parallel PBag(n,b) \end{array}$$

4. This fact follows using a straightforward calculation:

$$
\begin{array}{ll}
\Sigma(p : Pair, s_2(p) PBag(n, rem(p,b))) \triangleleft test(p,b) \triangleright \delta & \\
\overset{\text{B.10.7}}{=} & \Sigma(p : Pair, s_2(p)\, PBag(n, rem(p,b)) \triangleleft test(p, tl(p)) \text{ or } eq(p, hd(p)) \triangleright \delta) \\
\overset{\text{B.2.2}}{=} & \Sigma(p : Pair, s_2(p)\, PBag(n, rem(p,b)) \triangleleft test(p, tl(p)) \triangleright \delta) \\
+ & \Sigma(p : Pair, s_2(p)\, PBag(n, rem(p,b)) \triangleleft eq(p, hd(b)) \triangleright \delta) \\
\overset{\text{Sum Elimination}}{=} & \Sigma(p : Pair, s_2(p)\, PBag(n, rem(p,b)) \triangleleft test(p, tl(p)) \triangleright \delta) \\
+ & s_2(hd(b))\, PBag(n, rem(hd(b), b)) \\
\overset{\text{B.10.4}}{=} & \Sigma(p : Pair, s_2(p)\, PBag(n, rem(p,b)) \triangleleft test(p, tl(b)) \triangleright \delta) \\
+ & s_2(hd(b))\, PBag(n, tl(b))
\end{array}
$$

5. This case is shown with induction on n:

   - If $n = 0$ then

$$
\begin{array}{ll}
DBag(0,b) & \\
= & \Sigma(p : Pair, r_1(p)\, DBag(0, in(p,b))) \triangleleft size(b) < 0 \triangleright \delta\ + \\
 & \Sigma(p : Pair, s_2(p)\, DBag(0, rem(p,b)) \triangleleft test(p,b) \triangleright \delta) \\
\overset{\text{B.10.5}}{=} & \delta \\
= & PBag(0,b)
\end{array}
$$

   - In case $n = S(m)$ we show that $PBag(n,b)$ is a solution for $DBag(n,b)$. Hence, using RSP $PBag(n,b) = DBag(n,b)$.

$$
\begin{array}{ll}
PBag(n,b) & \\
= & (Buf \parallel PBag(P(n), b)) \\
 & \triangleleft empty(b) \triangleright \\
 & (s_2(hd(b))\, Buf \parallel PBag(P(n), tl(b)))) \\
 & \\
\overset{\text{I.H.}}{=} & (Buf \parallel DBag(m,b)) \\
 & \triangleleft empty(b) \triangleright \\
 & (s_2(hd(b))\, Buf \parallel DBag(m, tl(b))) \\
 & \\
\overset{\text{4.4, 4.2.5}}{=} & (\Sigma(p : Pair, r_1(p)\, (s_2(p)\, Buf \parallel DBag(m,b))) \\
+ & \Sigma(p : Pair, r_1(p)\, (Buf \parallel DBag(m, in(p,b)))) \triangleleft size(b) < m \triangleright \delta \\
+ & \Sigma(p : Pair, s_2(p)\, (Buf \parallel DBag(m, rem(p,b))) \triangleleft test(p,b) \triangleright \delta)) \\
 & \triangleleft empty(b) \triangleright \\
 & (s_2(hd(b))\, (Buf \parallel DBag(m, tl(b))) \\
+ & \Sigma(p : Pair, r_1(p)\, (s_2(hd(b))\, Buf \parallel DBag(m, in(p, tl(b))))) \triangleleft size(tl(b)) < m \triangleright \delta \\
+ & \Sigma(p : Pair, s_2(p)\, (s_2(hd(b))\, Buf \parallel DBag(m, rem(p, tl(b)))) \triangleleft test(p, tl(b)) \triangleright \delta)) \\
 & \\
\overset{\text{I.H.}}{=} & (\Sigma(p : Pair, r_1(p)\, (s_2(p)\, Buf \parallel PBag(m,b))) \\
+ & \Sigma(p : Pair, r_1(p)\, (Buf \parallel PBag(m, in(p,b)))) \triangleleft size(b) < m \triangleright \delta \\
+ & \Sigma(p : Pair, s_2(p)\, (Buf \parallel PBag(m, rem(p,b))) \triangleleft test(p,b) \triangleright \delta)) \\
 & \triangleleft empty(b) \triangleright \\
 & (s_2(hd(b))\, (Buf \parallel PBag(m, tl(b))) \\
+ & \Sigma(p : Pair, r_1(p)\, (s_2(hd(b))\, Buf \parallel PBag(m, in(p, tl(b))))) \triangleleft size(tl(b)) < m \triangleright \delta \\
+ & \Sigma(p : Pair, s_2(p)\, (s_2(hd(b))\, Buf \parallel PBag(m, rem(p, tl(b)))) \triangleleft test(p, tl(b)) \triangleright \delta)) \\
 & \\
\overset{\text{6.1.1, 6.1.3}}{=} & (\Sigma(p : Pair, r_1(p)\, PBag(n, in(p,b))) \\
+ & \Sigma(p : Pair, r_1(p)\, PBag(n, in(p,b))) \triangleleft size(b) < m \triangleright \delta \\
+ & \Sigma(p : Pair, s_2(p) PBag(n, rem(p,b)) \triangleleft test(p,b) \triangleright \delta)) \\
 & \triangleleft empty(b) \triangleright
\end{array}
$$

$$
\begin{array}{ll}
 & (s_2(hd(b))PBag(n, tl(b)) \\
+ & \quad \Sigma(p : Pair, r_1(p)\, PBag(n, in(hd(b), in(p, tl(b))))) \triangleleft size(tl(b)) < m \triangleright \delta \\
+ & \quad \Sigma(p : Pair, s_2(p)\, PBag(n, in(hd(b), rem(p, tl(b))))) \triangleleft test(p, tl(b)) \triangleright \delta))
\end{array}
$$

$$
\begin{array}{ll}
\overset{4.2.2,\ 6.1.2, B.10.4}{=} & (\Sigma(p : Pair, r_1(p)\, PBag(n, in(p, b))) \\
+ & \quad \Sigma(p : Pair, s_2(p)\, PBag(n, rem(p, b)) \triangleleft test(p, b) \triangleright \delta)) \\
 & \triangleleft empty(b) \triangleright \\
 & \quad (s_2(hd(b))\, PBag(n, tl(b)) \\
+ & \quad \Sigma(p : Pair, s_2(p)\, PBag(n, rem(p, b)) \triangleleft test(p, tl(b)) \triangleright \delta) \\
 & \quad \Sigma(p : Pair, r_1(p)\, PBag(n, in(p, b))) \triangleleft size(tl(b)) < m \triangleright \delta)
\end{array}
$$

$$
\begin{array}{ll}
\overset{6.1.4}{=} & (\Sigma(p : Pair, r_1(p)\, PBag(n, in(p, b)) \triangleleft size(b) < n \triangleright \delta) \\
+ & \quad \Sigma(p : Pair, s_2(p)\, PBag(n, rem(p, b))) \triangleleft test(p, b) \triangleright \delta) \\
 & \triangleleft empty(b) \triangleright \\
 & \quad (\Sigma(p : Pair, s_2(p)\, PBag(n, rem(p, b)) \triangleleft test(p, b) \triangleright \delta) \\
+ & \quad \Sigma(p : Pair, r_1(p)\, PBag(n, in(p, b))) \triangleleft size(b) < n \triangleright \delta)
\end{array}
$$

$$
\begin{array}{ll}
\overset{4.2.1}{=} & \Sigma(p : Pair, r_1(p)\, PBag(n, in(p, b)) \triangleleft size(b) < n \triangleright \delta) \\
+ & \quad \Sigma(p : Pair, s_2(p)\, PBag(n, rem(p, b)) \triangleleft test(p, b) \triangleright \delta)
\end{array}
$$

6. We prove this fact with induction on n.

   - For $n = 0$ we find:

     $$PBag(0) = \delta = PBag(0, \varnothing_p).$$

   - For $n = S(m)$ we find:

     $$PBag(n) = (Buf \parallel PBag(m)) \overset{\text{I.H.}}{=} Buf \parallel PBag(m, \varnothing_p) \overset{6.1.3}{=} PBag(n, \varnothing_p).$$

     The application of 6.1.3 above is correct because of B.10.6.

7. This follows easily via:

   $$DBag(n) = DBag(n, \varnothing_p) \overset{6.1.5}{=} PBag(n, \varnothing_p) \overset{6.1.6}{=} PBag(n).$$

   The application of 6.1.5 above is correct because of B.10.6.

□

Part II: $B(n)$ is a queue

The second part of showing the bakery protocol correct consists of proving $INS(i, n)$ and $OUTS(i, n)$ in combination with the $DBag(n)$ equal to a bounded queue.

The essential observation in our proof is to distinguish the following four situations:

0 No customer is busy getting a ticket and no customer is being served by the baker.

1 No customer is busy getting a ticket but there is a customer being served by the baker.

2 A customer is busy getting a ticket and no customer is being served by the baker.

3 A customer is busy getting a ticket and another customer is being served by the baker.

In order to calculate with the four categories above we need to make these explicit as processes.

Imagine the ideal situation towards which we are working, namely a queue b of customers. How are these customers distributed over the bakery in situation 3? The customer that entered the queue first is being served by the baker. So, $toe(b)$ is in $OUTS$, ready to leave the bakery. The person that entered the queue last is still picking a number. So $hd(b)$ is in INS. All other persons in the queue are waiting with a ticket in $PBag$. They are assigned consecutive numbers which is modelled by the function $number(i, n, b)$. It makes a queue of pairs ($queue_p$) out of a queue of customers $b : queue_d$ by taking the elements in b and number it from the end to the start of b, starting with i, modulo n. Below the four situations are described as processes $CQ_0, \ldots, CQ_3$. The number n indicates the size of the queue (the actual size of the queue is $n + 2$), b is the queue of customers and i is the number on the ticket of the first customer. Note that the behaviour of CQ_j is chosen to be δ if $i \geq n$ or the size of the queue b is too small or too large. In these cases the values of i, n and b do not make sense. For instance in case of CQ_3 if $size(b) < 2$, then there cannot be customers in both INS and $OUTS$, which does not conform to the intention of CQ_3.

$$
\begin{aligned}
&\textbf{proc } CQ_0(n : nat, i : nat, b : queue_d) = \\
&\qquad \tau(\{c_1, c_2\}, \partial(\{r_1, s_1, r_2, s_2\}, \\
&\qquad\quad INS(n, i +_n size(b)) \parallel \\
&\qquad\quad DBag(n, number(i, n, b)) \parallel \\
&\qquad\quad OUTS(n, i))) \\
&\qquad \triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta \\
\\
&CQ_1(n : nat, i : nat, b : queue_d) = \\
&\qquad \tau(\{c_1, c_2\}, \partial(\{r_1, s_1, r_2, s_2\}, \\
&\qquad\quad INS(n, i +_n size(b)) \parallel \\
&\qquad\quad DBag(n, number(i +_n 1, n, untoe(b))) \parallel \\
&\qquad\quad out(toe(b))\, OUTS(n, i +_n 1))) \\
&\qquad \triangleleft size(b) > 0 \text{ and } size(b) \leq n + 1 \text{ and } i < n \triangleright \delta \\
\\
&CQ_2(n : nat, i : nat, b : queue_d) = \\
&\qquad \tau(\{c_1, c_2\}, \partial(\{r_1, s_1, r_2, s_2\}, \\
&\qquad\quad s_1(\langle hd(b), i +_n size(tl(b))\rangle)\, INS(n, i +_n size(b)) \parallel \\
&\qquad\quad DBag(n, number(i, n, tl(b))) \parallel \\
&\qquad\quad OUTS(n, i))) \\
&\qquad \triangleleft size(b) > 0 \text{ and } size(b) \leq n + 1 \text{ and } i < n \triangleright \delta \\
\\
&CQ_3(n : nat, i : nat, b : queue_d) = \\
&\qquad \tau(\{c_1, c_2\}, \partial(\{r_1, s_1, r_2, s_2\}, \\
&\qquad\quad s_1(\langle hd(b), i +_n size(tl(b))\rangle)\, INS(n, i +_n size(b)) \parallel \\
&\qquad\quad DBag(n, number(i +_n 1, n, tl(untoe(b)))) \parallel \\
&\qquad\quad out(toe(b))\, OUTS(n, i +_n 1))) \\
&\qquad \triangleleft size(b) > 1 \text{ and } size(b) \leq n + 2 \text{ and } i < n \triangleright \delta
\end{aligned}
$$

Note that obviously $CQ_0(n, 0, \varnothing_d) = B(n)$ if $n > 0$.

Now lets consider say $CQ_0(n, i, b)$ and lets pose the question what the behaviour of CQ_0 would be. The process $CQ_0(n, i, b)$ can perform an action $enter(d)$ and arrive in the situation $CQ_2(n, i, in(d, b))$, namely the situation where customer d is busy picking a ticket. If $size(b) > 0$, there is a customer in $CQ_0(n, i, b)$ that can become served by the baker. So, via an internal step $CQ_0(n, i, b)$ becomes $CQ_1(n, i, b)$. This analysis can be made in all four cases. In other words, CQ_j substituted for G_j should satisfy the equations below. Indeed this is confirmed by theorem 6.2.3.

$$
\textbf{proc } G_0(n : nat, i : nat, b : queue_d) =
$$

$$\Sigma(d:D, enter(d)\, G_2(n,i,in(d,b))) \triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta +$$
$$\tau\, G_1(n,i,b) \triangleleft size(b) > 0 \text{ and } size(b) \leq n \text{ and } i < n \triangleright \delta$$

$$G_1(n:nat, i:nat, b:queue_d) =$$
$$\Sigma(d:D, enter(d)\, G_3(n,i,in(d,b))) \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta +$$
$$out(toe(b))\, G_0(n, i +_n 1, untoe(b)) \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta$$

$$G_2(n:nat, i:nat, b:queue_d) =$$
$$\tau\, G_0(n,i,b) \triangleleft size(b) > 0 \text{ and } size(b) \leq n \text{ and } i < n \triangleright \delta +$$
$$\tau\, G_3(n,i,b) \triangleleft size(b) > 1 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta$$

$$G_3(n:nat, i:nat, b:queue_d) =$$
$$\tau\, G_1(n,i,b) \triangleleft size(b) > 1 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta +$$
$$out(toe(b))\, G_2(n, i +_n 1, untoe(b)) \triangleleft size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \triangleright \delta$$

Now it is tempting to state that the queue $Q(n,b)$ is a solution of $G_0(n,i,b)$. But this can not easily be shown. The most important reason is that Q must perform τ-steps in a rather irregular way in order to be a solution. We model this by defining the following four processes Q_j $(j = 0,1,2,3)$. Obviously, $Q_0(n,0,\varnothing_d)$ is equal to $Q(n+2)$. $Q_j(n,i,b)$ is also a solution for $G_j(n,i,b)$ from which it follows that $Q_j(n,i,b) = CQ(n,i,b)$. Combination of these facts leads to the correctness of the protocol.

proc $Q_0(n:nat, i:nat, b:queue_d) =$
$$(Q(n+2,b) \triangleleft empty(b) \triangleright \tau\, Q(n+2,b))$$
$$\triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta$$

$$Q_1(n:nat, i:nat, b:queue_d) =$$
$$Q(n+2,b) \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta$$

$$Q_2(n:nat, i:nat, b:queue_d) =$$
$$\tau\, Q(n+2,b) \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta$$

$$Q_3(n:nat, i:nat, b:queue_d) =$$
$$(Q(n+2,b) \triangleleft eq(size(b), n+2) \triangleright \tau\, Q(n+2,b))$$
$$\triangleleft size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \triangleright \delta$$

Theorem 6.2. *Let $i, n : nat$, $b : queue_d$.*

1. $n > 0 \rightarrow B(n) = CQ_0(n, 0, \varnothing_d)$,
2. $n > 0 \rightarrow Q(n+2) = Q_0(n, 0, \varnothing_d)$,
3. $CQ_j(n,i,b) = Q_j(n,i,b)$ *for* $j = 0,1,2,3$,
4. $n > 0 \rightarrow Q(n+2) = B(n)$.

Proof.

1. Direct via the definition.
2. Also immediate using the definition.
3. We show that both $CQ_j(n,i,b)$ and $Q_j(n,i,b)$ are solutions for the equations defining $G_j(n,i,b)$. As $G_j(n,i,b)$ is guarded this immediately implies that $CQ_j(n,i,b) = Q_j(n,i,b)$ $(j = 0,1,2,3)$.

First we show that the processes $CQ_j(n,i,b)$ satisfy the equations for G_j and then we do the same for $Q_j(n,i,b)$. In each case the proof consists of a straightforward expansion using theorem 4.4 and 4.5 and the applications of some lemmas about data given in appendix B.

$$
\begin{aligned}
& CQ_0(n,i,b) \\
&= \quad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\}, \\
&\qquad INS(n, i +_n size(b)) \parallel \\
&\qquad DBag(n, number(i,n,b)) \parallel \\
&\qquad OUTS(n,i))) \\
&\qquad \triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta
\end{aligned}
$$

$$
\begin{aligned}
&\overset{4.4,\,D1,\,TI1,\,SUM8,\,SUM9}{=} \\
&\qquad \Sigma(d:D, enter(d) \\
&\qquad\quad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\}, \\
&\qquad\qquad s_1(\langle d, i +_n size(b)\rangle)\, INS(n, i +_n size(b) +_n 1) \parallel \\
&\qquad\qquad DBag(n, number(i,n,b)) \parallel \\
&\qquad\qquad OUTS(n,i)))) \\
&\qquad\quad \triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta \\
&\quad + \quad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\}, \\
&\qquad\quad (\Sigma(p:Pair, s_2(p)\, DBag(n, rem(p, number(i,n,b))) \\
&\qquad\qquad\qquad \triangleleft test(p, number(i,n,b)) \triangleright \delta) \mid \\
&\qquad\quad \Sigma(d:D, r_2(\langle d,i\rangle)\, out(d))\, OUTS(n, i +_n 1)) \between \\
&\qquad\quad INS(n, i +_n size(b)))) \\
&\qquad\quad \triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta
\end{aligned}
$$

$$
\begin{aligned}
&\overset{4.5.3}{=} \quad \Sigma(d:D, enter(d)\, CQ_2(n,i,in(d,b))) \triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta \\
&+ \quad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\}, \\
&\qquad \Sigma(d:D, c_1(\langle d,i\rangle) \\
&\qquad\quad (DBag(n, rem(\langle d,i\rangle, number(i,n,b))) \parallel \\
&\qquad\quad out(d)\, OUTS(n, i +_n 1)) \\
&\qquad\quad \triangleleft test(\langle d,i\rangle, number(i,n,b)) \triangleright \delta)) \between \\
&\qquad INS(n, i +_n size(b)))) \\
&\qquad \triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta
\end{aligned}
$$

$$
\begin{aligned}
&\overset{B.10.8,\,Sum\ El.}{=} \quad \Sigma(d:D, enter(d)\, CQ_2(n,i,in(d,b))) \triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta \\
&+ \quad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\}, \\
&\qquad (c_1(\langle toe(b), i\rangle) \\
&\qquad\quad (DBag(n, rem(\langle toe(b), i\rangle, number(i,n,b)))) \parallel \\
&\qquad\quad out(toe(b))\, OUTS(n, i +_n 1)) \between \\
&\qquad INS(n, i +_n size(b)))) \\
&\qquad \triangleleft size(b) \leq n \text{ and } size(b) > 0 \text{ and } i < n \triangleright \delta
\end{aligned}
$$

$$
\begin{aligned}
&\overset{B.10.11,\,D1,\,TI2}{=} \quad \Sigma(d:D, enter(d)\, CQ_2(n,i,in(d,b))) \triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta \\
&+ \quad \tau\, \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\}, \\
&\qquad\quad INS(n, i +_n size(b)) \parallel \\
&\qquad\quad DBag(n, number(i +_n 1, n, untoe(b))) \parallel \\
&\qquad\quad out(toe(b))\, OUTS(n, i +_n 1))) \\
&\qquad \triangleleft size(b) \leq n \text{ and } size(b) > 0 \text{ and } i < n \triangleright \delta \\
&\\
&= \quad \Sigma(d:D, enter(d)\, CQ_2(n,i,in(d,b))) \triangleleft size(b) \leq n \text{ and } i < n \triangleright \delta
\end{aligned}
$$

$$+ \quad \tau\, CQ_1(n,i,b) \triangleleft size(b) > 0 \text{ and } size(b) \leq n \text{ and } i < n \triangleright \delta$$

Now we show that $CQ_j(n,i,b)$ also satisfy the equation defining G_1.

$$\begin{aligned}
&CQ_1(n,i,b)\\
&\quad = \quad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\},\\
&\qquad INS(n, i +_n size(b)) \parallel\\
&\qquad DBag(n, number(i +_n 1, n, untoe(b))) \parallel\\
&\qquad out(toe(b))\, OUTS(n, i +_n 1)))\\
&\qquad \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta\\
&\quad \overset{4.4,\,D1,\,TI1}{=} \quad \Sigma(d:D, enter(d)\\
&\qquad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\},\\
&\qquad\quad s_1(\langle d, i +_n size(b) +_n 1\rangle)\, INS(n, i +_n size(b) +_n 1) \parallel\\
&\qquad\quad DBag(n, number(i +_n 1, n, untoe(b))) \parallel\\
&\qquad\quad out(toe(b))\, OUTS(n, i +_n 1))))\\
&\qquad \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta\\
&\quad + \quad out(toe(b))\\
&\qquad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\},\\
&\qquad\quad INS(n, i +_n size(b)) \parallel\\
&\qquad\quad DBag(n, number(i +_n 1, n, untoe(b))) \parallel\\
&\qquad\quad OUTS(n, i +_n 1)))\\
&\qquad \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta\\
&\quad = \quad \Sigma(d:D, enter(d)\, CQ_3(n,i,in(d,b)))\\
&\qquad\qquad \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta\\
&\quad + \quad out(toe(b))\, CQ_0(n, i +_n 1, untoe(b))\\
&\qquad\qquad \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta
\end{aligned}$$

The processes $CQ_j(n,i,b)$ also satisfy the equation for G_2.

$$\begin{aligned}
&CQ_2(n,i,b)\\
&\quad = \quad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\},\\
&\qquad s_1(\langle hd(b), i +_n size(tl(b))\rangle)\, INS(n, i +_n size(b)) \parallel\\
&\qquad DBag(n, number(i, n, tl(b))) \parallel\\
&\qquad OUTS(n,i)))\\
&\qquad \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta\\
&\quad = \quad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\},\\
&\qquad ((s_1(\langle hd(b), i +_n size(tl(b))\rangle)\, INS(n, i +_n size(b)) \mid\\
&\qquad\quad \Sigma(p:Pair, r_1(p)\, DBag(n, in(p, number(i,n,tl(b))))))\\
&\qquad\quad \triangleleft size(tl(b)) < n \triangleright \delta) \mathbin{\underline{\parallel}}\\
&\qquad OUTS(n,i)))\\
&\qquad \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta\\
&\quad + \quad \tau(\{c_1,c_2\}, \partial(\{r_1,s_1,r_2,s_2\},\\
&\qquad \Sigma(p:Pair, s_2(p)\, DBag(n, rem(p, number(i,n,tl(b)))))\\
&\qquad\qquad \triangleleft test(p, number(i,n,tl(b))) \triangleright \delta \mid\\
&\qquad\quad \Sigma(d:D, r_2(\langle d,i\rangle)\, out(d)\, OUTS(n, i +_n 1))) \mathbin{\underline{\parallel}}\\
&\qquad s_1(\langle hd(b), i +_n size(tl(b))\rangle)\, INS(n, i +_n size(b))))\\
&\qquad \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta
\end{aligned}$$

$$
\begin{array}{lll}
\overset{4.5.1,\ 4.5.3,\ SC3}{=} & \tau(\{c_1,c_2\},\partial(\{r_1,s_1,r_2,s_2\}, & \\
& \quad (c_1(\langle hd(b), i +_n size(tl(b))\rangle) & \\
& \quad\quad (INS(n, i +_n size(b)) \parallel & \\
& \quad\quad\ DBag(n, in(\langle hd(b), i +_n size(tl(b))\rangle, number(i,n,tl(b))))) & \\
& \quad \triangleleft size(b) \leq n \triangleright \delta) \between & \\
& \quad OUTS(n,i))) & \\
& \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta & \\
+ & \tau(\{c_1,c_2\},\partial(\{r_1,s_1,r_2,s_2\}, & \\
& \quad \Sigma(d:D, c_2(\langle d,i\rangle) & \\
& \quad\quad (DBag(n, rem(\langle d,i\rangle, number(i,n,tl(b)))) \parallel & \\
& \quad\quad\ out(d)\ OUTS(n, i +_n 1)) & \\
& \quad \triangleleft test(\langle d,i\rangle, number(i,n,tl(b))) \triangleright \delta) \between & \\
& \quad s_1(\langle hd(b), i +_n size(tl(b))\rangle)\ INS(n, i +_n size(b)))) & \\
& \triangleleft size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta &
\end{array}
$$

$$
\begin{array}{lll}
\overset{B.10.9,\ B.10.10,\ Sum\ El.,D1,TI2}{=} & & \\
& \tau\, CQ_0(n,i,b) \triangleleft size(b) > 0 \text{ and } size(b) \leq n \text{ and } i < n \triangleright \delta & \\
+ & \tau\, \tau(\{c_1,c_2\},\partial(\{r_1,s_1,r_2,s_2\}, & \\
& \quad s_1(\langle hd(b), i +_n size(tl(b))\rangle)\ INS(n, i +_n size(b)) \parallel & \\
& \quad DBag(n, rem(\langle toe(b), i\rangle, number(i,n,tl(b)))) \parallel & \\
& \quad out(toe(b))\ OUTS(n, i +_n 1))) & \\
& \triangleleft size(b) > 1 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta & \\
& & \\
= & \tau\, CQ_0(n,i,b) \triangleleft size(b) > 0 \text{ and } size(b) \leq n \text{ and } i < n \triangleright \delta & \\
+ & \tau\, CQ_3(n,i,b) \triangleleft size(b) > 1 \text{ and } size(b) \leq n+1 \text{ and } i < n \triangleright \delta &
\end{array}
$$

And $CQ_j(n,i,b)$ even satisfy the equation for G_3.

$$
\begin{array}{lll}
CQ_3(n,i,b) & & \\
= & \tau(\{c_1,c_2\},\partial(\{r_1,s_1,r_2,s_2\}, & \\
& \quad s_1(\langle hd(b), i +_n size(tl(b))\rangle)\ INS(n, i +_n size(b)) \parallel & \\
& \quad DBag(n, number(i +_n 1, n, tl(untoe(b)))) \parallel & \\
& \quad out(toe(b))\ OUTS(n, i +_n 1))) & \\
& \triangleleft size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \triangleright \delta & \\
& & \\
= & \tau(\{c_1,c_2\},\partial(\{r_1,s_1,r_2,s_2\}, & \\
& \quad (s_1(\langle hd(b), i +_n size(tl(b))\rangle)\ INS(n, i +_n size(b)) \mid & \\
& \quad\ \Sigma(p:Pair, r_1(p)\, DBag(n, in(p, number(i +_n 1, n, tl(untoe(b))))))) & \\
& \quad \triangleleft size(number(i +_n 1, n, tl(untoe(b)))) < n \triangleright \delta \between & \\
& \quad out(toe(b))\ OUTS(n, i +_n 1))) & \\
& \triangleleft size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \triangleright \delta & \\
+ & out(toe(b)) & \\
& \quad \tau(\{c_1,c_2\},\partial(\{r_1,s_1,r_2,s_2\}, & \\
& \quad\quad s_1(\langle hd(b), i +_n size(tl(b))\rangle)\ INS(n, i +_n size(b)) \parallel & \\
& \quad\quad DBag(n, number(i +_n 1, n, tl(untoe(b)))) \parallel & \\
& \quad\quad OUTS(n, i +_n 1))) & \\
& \triangleleft size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \triangleright \delta & \\
& & \\
\overset{4.5.3}{=} & \tau(\{c_1,c_2\},\partial(\{r_1,s_1,r_2,s_2\}, & \\
& \quad c_1(\langle hd(b), i +_n size(tl(b))\rangle) &
\end{array}
$$

$$
\begin{array}{lll}
 & & (INS(n, i +_n size(b)) \parallel \\
 & & \quad DBag(n, in(\langle hd(b), i +_n size(tl(b))\rangle, number(i +_n 1, n, tl(untoe(b))))) \between \\
 & & \quad out(toe(b))\, OUTS(n, i +_n 1))))) \\
 & & \lhd size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \rhd \delta \\
 & & \quad size(number(i +_n 1, n, tl(untoe(b)))) < n \rhd \delta \\
 & + & out(toe(b))\, CQ_2(n, i +_n 1, untoe(b)) \\
 & & \quad \lhd size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \rhd \delta
\end{array}
$$

$$
\begin{array}{lll}
 & \overset{\text{B.10.13, B.10.15,D1,TI2}}{=} & \tau\,\tau(\{c_1, c_2\}, \partial(\{r_1, s_1, r_2, s_2\}, \\
 & & \quad INS(n, i +_n size(b)) \parallel \\
 & & \quad DBag(n, number(i +_n 1, n, untoe(b))) \parallel \\
 & & \quad out(toe(b))\, OUTS(n, i +_n 1))) \\
 & & \lhd size(b) > 1 \text{ and } size(b) < n+2 \text{ and } i < n \rhd \delta \\
 & + & out(toe(b))\, CQ_2(n, i +_n 1, untoe(b)) \\
 & & \quad \lhd size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \rhd \delta \\
 & & \\
 & = & \tau\, CQ_1(n, i, b) \lhd size(b) > 1 \text{ and } size(b) < n+2 \text{ and } i < n \rhd \delta \\
 & + & out(toe(b))\, CQ_2(n, i +_n 1, untoe(b)) \\
 & & \quad \lhd size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \rhd \delta
\end{array}
$$

Now we show that the processes $Q_j(n, i, b)$ are solutions for the equations for $G_0, \ldots, G_3$. It is worth noting that the only place where the τ-laws are used is below.

$$
\begin{array}{lll}
Q_0(n, i, b) & & \\
 & = & (Q(n+2, b) \lhd empty(b) \rhd \tau\, Q(n+2, b)) \\
 & & \lhd size(b) \leq n \text{ and } i < n \rhd \delta \\
 & \overset{\text{T2}}{=} & (\Sigma(d : D, enter(d)\, Q(n+2, in(d, b))) \\
 & & \quad \lhd empty(b) \rhd \\
 & & \quad \Sigma(d : D, enter(d)\, Q(n+2, in(d, b))) + \tau\, Q(n+2, b))) \\
 & & \lhd size(b) \leq n \text{ and } i < n \rhd \delta \\
 & = & (\Sigma(d : D, enter(d)\, Q(n+2, in(d, b))) \lhd size(b) \leq n \rhd \delta) + \tau\, Q_1(n, i, b) \lhd size(b) > 0 \rhd \delta) \\
 & & \quad \lhd empty(b) \rhd \\
 & & \quad (\Sigma(d : D, enter(d)\, Q(n+2, in(d, b))) \lhd size(b) \leq n \rhd \delta) + \tau\, Q_1(n, i, b) \lhd size(b) > 0 \rhd \delta) \\
 & & \lhd size(b) \leq n \text{ and } i < n \rhd \delta \\
 & \overset{\text{4.2.1, T1}}{=} & \Sigma(d : D, enter(d)\, Q_2(n, i, in(d, b))) \lhd size(b) \leq n \text{ and } i < n \rhd \delta \\
 & + & \tau Q_1(n, i, b) \lhd size(b) > 0 \text{ and } size(b) \leq n \text{ and } i < n \rhd \delta
\end{array}
$$

The processes $Q_j(n, i, b)$ are also a solution of the defining equation for G_1.

$$
\begin{array}{lll}
Q_1(n, i, b) & & \\
 & = & Q(n+2, b) \lhd size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \rhd \delta \\
 & = & \Sigma(d : D, enter(d)\, Q(n+2, in(d, b))) \lhd size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \rhd \delta \\
 & + & out(toe(b))\, Q(n+2, untoe(b)) \lhd size(b) > 0 \text{ and } size(b) \leq n \text{ and } i < n \rhd \delta \\
 & \overset{\text{T1}}{=} & \Sigma(d : D, enter(d)\, Q_3(n, i, in(d, b))) \lhd size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \rhd \delta \\
 & + & out(toe(b))\, Q_0(n, i +_n 1, untoe(b)) \lhd size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \rhd \delta
\end{array}
$$

Note that we need that $n > 0$ to show that the processes $Q_j(n, i, b)$ are a solution for the equation for G_2.

$$
Q_2(n, i, b)
$$

$$
\begin{array}{lll}
= & \tau\, Q(n+2,b) \lhd size(b) > 0 \text{ and } size(b) \leq n+1 \text{ and } i < n \rhd \delta \\
= & \tau\, Q(n+2,b) \lhd size(b) > 0 \text{ and } size(b) \leq n \text{ and } i < n \rhd \delta \\
+ & \tau\, Q(n+2,b) \lhd size(b) > 1 \text{ and } size(b) \leq n+1 \text{ and } i < n \rhd \delta \\
\overset{\text{T1}}{=} & \tau\, Q_0(n,i,b) \lhd size(b) > 0 \text{ and } size(b) \leq n \text{ and } i < n \rhd \delta \\
+ & \tau\, Q_3(n,i,b) \lhd size(b) > 1 \text{ and } size(b) \leq n+1 \text{ and } i < n \rhd \delta
\end{array}
$$

And last, we show that $CQ_j(n,i,b)$ also satisfies the equation for Q_3.

$$
\begin{array}{ll}
Q_3(n,i,b) & \\
= & (Q(n+2,b) \lhd eq(size(b), n+2) \rhd \tau\, Q(n+2,b)) \\
& \lhd size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \rhd \delta \\
\overset{\text{T2}}{=} & (out(toe(b))\, Q(n+2, untoe(b)) \lhd eq(size(b), n+2) \rhd \\
& \qquad (\tau\, Q(n+2,b) + out(toe(b))\, Q(n+2, untoe(b)))) \\
& \lhd size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \rhd \delta \\
\overset{\text{4.2.4, B.5.17, B.2.1}}{=} & \\
& \tau\, Q(n+2,b) \lhd size(b) > 1 \text{ and } size(b) < n+2 \text{ and and } i < n \rhd \delta \\
+ & out(toe(b))\, Q(n+2, untoe(b)) \lhd size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \rhd \delta \\
\overset{\text{T1}}{=} & \tau\, Q_1(n,i,b) \lhd size(b) > 1 \text{ and } size(b) < n+2 \text{ and and } i < n \rhd \delta \\
+ & out(toe(b))\, Q_2(n, i +_n 1, untoe(b)) \lhd size(b) > 1 \text{ and } size(b) \leq n+2 \text{ and } i < n \rhd \delta
\end{array}
$$

4. Now, given the calculations above, this proof is straightforward:

$$Q(n+2) \overset{6.2.2}{=} Q_0(n,0,\varnothing_d) \overset{6.2.3}{=} CQ_0(n,0,\varnothing_d) \overset{6.2.1}{=} B(n).$$

□

A An overview of the proof theory for μCRL

We assume that the reader is familiar with the basic concepts of ACP and its data extension μCRL. Below, we summarise the most relevant definitions of the proof system for μCRL as given in [9] and formulate some revisions for guaranteeing compatibility with additional τ-laws.

A.1 Formulating properties for a μCRL specification

In the proof theory, the properties of a μCRL specification are stated by *property-formulas.* A property-formula is either an identity between terms, or an application of the operators $\mathcal{F}$ ('Falsum'), $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ known from propositional logic (see e.g. [5]) between such identities.

Definition A.1. *(Property-formulas).* A *property-formula* is defined inductively in the following way:

- $\mathcal{F}$ is a *property-formula,*
- $p_1 = p_2$ is a *property-formula* where p_1 and p_2 are open *process* terms,
- $t_1 = t_2$ is a *property-formula* where t_1 and t_2 are open *data* terms of the same sort,
- $\neg(\phi)$ is a *property-formula* iff ϕ is a *property-formula,*
- $(\phi \circ \psi)$ with $\circ \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ is a *property-formula* iff both ϕ and ψ are *property-formulas.*

End Definition A.1.

For deriving the fundamental properties induced by $=$ and the propositional connectives ($\neg, \vee, \wedge, \rightarrow, \leftrightarrow$), the proof system contains the so called 'logical' axioms and rules for 'natural deduction' which are close to intuitive reasoning (see e.g. [5]). We do not mention these logical rules here and refer to [9] because 'natural deduction' hardly plays a role in the proofs given in this paper.

A.2 Proof rules for data equivalence

A.2.1 Axioms for the predefined booleans

μCRL has one predefined sort called **Bool** for supporting the conditional construct $(p_1 \lhd c \rhd p_2)$. In order to guarantee a proper axiomatisation of the conditional construct, the proof system includes two predefined axioms for the booleans presented in table 4.

B1	$\neg(T = F)$
B2	$\neg(b = T) \rightarrow b = F$

Table 4: Axioms for **Bool**.

Axiom B1 states that the booleans T and F are different and axiom B2 expresses that there are at most two boolean values, represented by T and F.

A.2.2 The induction rule

Most data properties of a μCRL specification are proven by induction on the structure of data terms. The following definition tells us on which data terms the induction takes place.

Definition A.2. *(Constructors).* Let C be a subset of the function declarations occurring in a μCRL specification. We say that C is a *constructor set of the sort* S iff all functions in C have target sort S, and any closed data term of sort S can be proved equal to a data term in which all occurring functions with target sort S are contained in C.

End Definition A.2.

The induction on a data variable x_i of sort S_i follows the construction of terms of sort S_i using the constructors in the constructor set:

$$C_i \stackrel{\text{def}}{=} \{f_{ij} : S_1^{ij} \times \cdots \times S_{l_{ij}}^{ij} \rightarrow S_i \mid 1 \leq j \leq k_i\ ,\ k_i > 0\ ,\ l_{ij} \geq 0\}.$$

Here k_i is called the cardinality of sort S_i, i.e. the number of functions needed to generate the term set of sort S_i.

For applying induction on variables $x_1 : S_1, \ldots, x_m : S_m$, we have an induction rule IND($\overline{C}$) as given in table 5 parameterised by the constructor sets $\overline{C} \equiv C_1, \ldots, C_m$, specifying the construction of terms on which the induction has to be be followed.

A.3 Proof rules for process equivalence

A.3.1 Axioms of ACP

Table 6 shows the ACP axioms [3] for a μCRL specification E. In this table, a and b range over δ and the actions occurring in specification E the n_i range over a set of gate names $\mathcal{N}$. The set $Comm(E)$ is the commutative and associative closure of all communications declared in E. In CF2, D1 and D2 a function *label*() is used that extracts the gate name of an atomic action, and is the identity for δ and τ. In D1-4, $H \subseteq \mathcal{N}$ denotes the set of gate names to be encapsulated.

$$\mathrm{IND}(\overline{C}) \qquad \Big(\bigwedge_{i=1}^{m}\Big(\bigwedge_{j=1}^{k_i}\Big(\bigwedge_{\sigma\in I_{ij}}\sigma(\phi)\Big)\rightarrow\phi[f_{ij}(z_1^{ij},\ldots,z_{l_{ij}}^{ij})/x_i]\Big)\Big)\rightarrow\phi$$

where

- $(f_{ij}:S_1^{ij}\times\ldots\times S_{l_{ij}}^{ij}\rightarrow S_i)\in C_i$ with $l_{ij}\geq 0$,
- $x_i:S_i$, $z_n^{ij}:S_n^{ij}$ with $1\leq n\leq l_{ij}$ are data variables,
- $\sigma\in I_{ij} \iff \sigma$ is the identity, except that it maps x_k to some y_k, where
  - $1\leq k\leq m$,
  - $y_k\in\{x_1,\ldots,x_m\}\cup\{z_n^{ij}\mid 1\leq n\leq l_{ij}\}$,
  - $y_i\not\equiv x_i$,
  - if $1\leq k<k'\leq m$, then $y_k\not\equiv y_{k'}$.

Table 5: The induction rule $\mathrm{IND}(C_1,\ldots,C_m)$.

A1	$x+y=y+x$	CF1	$n_1\mid n_2=n_3$	if $n_1\mid n_2=n_3\in Comm(E)$
A2	$x+(y+z)=(x+y)+z$	CF1′	$n_1(t_1,\ldots,t_m)\mid n_2(t_1,\ldots,t_m)=n_3(t_1,\ldots,t_m)$	if $n_1\mid n_2=n_3\in Comm(E)$
A3	$x+x=x$	CF2	$a\mid b=\delta$	if $\forall n\in\mathcal{N}\,.\,label(a)\mid label(b)=n\notin Comm(E)$
A4	$(x+y)\cdot z=x\cdot z+y\cdot z$	CF2′	$\neg(t_i=t_i')\rightarrow n_1(t_1,\ldots,t_m)\mid n_2(t_1',\ldots,t_m')=\delta$	for some $1\leq i\leq m$
A5	$(x\cdot y)\cdot z=x\cdot(y\cdot z)$	CF2″	$n_1(t_1,\ldots,t_m)\mid n_2(t_1',\ldots,t_{m'}')=\delta$	if $m\neq m'$
A6	$x+\delta=x$	D1	$\partial(H,a)=a$	if $label(a)\notin H$
A7	$\delta\cdot x=\delta$	D2	$\partial(H,a)=\delta$	if $label(a)\in H$
CM1	$x\parallel y=x\between y+y\between x+x\mid y$	D3	$\partial(H,x+y)=\partial(H,x)+\partial(H,y)$	
CM2	$a\between x=a\cdot x$	D4	$\partial(H,x\cdot y)=\partial(H,x)\cdot\partial(H,y)$	
CM3	$a\cdot x\between y=a\cdot(x\parallel y)$			
CM4	$(x+y)\between z=x\between z+y\between z$			
CM5	$a\cdot x\mid b=(a\mid b)\cdot x$			
CM6	$a\mid b\cdot x=(a\mid b)\cdot x$			
CM7	$a\cdot x\mid b\cdot y=(a\mid b)\cdot(x\parallel y)$			
CM8	$(x+y)\mid z=x\mid z+y\mid z$			
CM9	$x\mid(y+z)=x\mid y+x\mid z$			

Table 6: The ACP axioms for a μCRL specification E.

A.3.2 Axioms of Standard Concurrency

In table 7 some axioms for the merge operators, known as the Standard Concurrency laws (see [3]) are presented. These axioms are derivable for closed process terms without recursion.

SC1 $(x \between y) \between z = x \between (y \parallel z)$	SC4 $(x \mid y) \mid z = x \mid (y \mid z)$
SC2 $x \between \delta = x\delta$	SC5 $x \mid (ay \between z) = (x \mid ay) \between z$
SC3 $x \mid y = y \mid x$	

Table 7: Axioms of Standard Concurrency (SC).

Using the axioms of standard concurrency we can derive the following identities.

Theorem A.3.

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

A.3.3 Axioms for the abstraction operator

For abstraction in μCRL we present the axioms in table 8. Here $I \subseteq \mathcal{N}$ is the set of gate names to be abstracted from.

TI1	$\tau(I, a) = a$	if $label(a) \notin I$
TI2	$\tau(I, a) = \tau$	if $label(a) \in I$
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 8: Axioms for abstraction.

A.3.4 Axioms for the sum operator

In ACP the summation over a certain data domain D is treated on meta-level by using abbreviations like $\sum_{d \in \{d_1, \ldots, d_n\}} a(d) \equiv a(d_1) + \ldots + a(d_n)$. In contrast with ACP, μCRL has official syntax 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 9) [3] making it possible to reason with sum expressions in a formal way.

The axioms SUM hold for all substitutions with the convention that *substitutions do not change the binding in a sum construct.* For example, $\Sigma(d : D, a(d))[e/d] = \Sigma(d : D, a(d))$ is a true identity and $\Sigma(d : D, a(d))[e/d] = \Sigma(d : D, a(e))$ is a false identity. To compensate the restricted way of substitution, the SUM axioms are formulated with open process terms p, p_1, p_2, which are more general than ordinary process variables like x and y. [4]

Another consequence of the particular substitution mechanism, is that the congruence rule for equational derivability does not longer hold in places where bound variables occur and that we therefore

[3] Axiom SUM10 can be skipped because we do not consider renaming (ρ) in this paper.

[4] In the original paper [9] the sum axioms are formulated with ordinary variables (x, y) and a substitution operator (σ). We do the formulation with open process terms (p, p_1, p_2) to avoid a formal introduction of a substitution operator.

need a rule like SUM11 for axiomatising the 'congruence property' of the sum operator separately. For instance, a semantically obvious identity as $\Sigma(d:D, \delta \cdot a(d)) = \Sigma(d:D,\delta)$ is not derivable without the application of SUM11.

As a final remark, we note that SUM11 may only be applied when data variable d does not occur free in the assumptions of derivation $\mathcal{D}$. Otherwise one can derive false property-formulas such as $d = 1 \rightarrow \Sigma(d:D, a(d)) = \Sigma(d:D, a(1))$.

SUM1	$\Sigma(d:D,p) = p$	if d not free in p
SUM2	$\Sigma(d:D,p) = \Sigma(e:D, p[e/d])$	if e not free in p
SUM3	$\Sigma(d:D,p) = \Sigma(d:D,p) + p$	
SUM4	$\Sigma(d:D,p_1+p_2) = \Sigma(d:D,p_1) + \Sigma(d:D,p_2)$	
SUM5	$\Sigma(d:D,p_1 \cdot p_2) = \Sigma(d:D,p_1) \cdot p_2$	if d not free in p
SUM6	$\Sigma(d:D,p_1 \between p_2) = \Sigma(d:D,p_1) \between p_2$	if d not free in p
SUM7	$\Sigma(d:D,p_1 \mid p_2) = \Sigma(d:D,p_1) \mid p_2$	if d not free in p
SUM8	$\Sigma(d:D,\partial(H,p)) = \partial(H,\Sigma(d:D,p))$	
SUM9	$\Sigma(d:D,\tau(I,p)) = \tau(I,\Sigma(d:D,p))$	
SUM10	$\Sigma(d:D,\rho(R,p)) = \rho(R,\Sigma(d:D,p))$	
SUM11	$\dfrac{\begin{array}{c}\mathcal{D}\\ p_1 = p_2\end{array}}{\Sigma(d:D,p_1) = \Sigma(d:D,p_2)}$	provided d not free in the assumptions of $\mathcal{D}$

Table 9: Axioms for the sum operator.

A.3.5 Axioms for the conditional construct

The axioms for the *conditional* construct $p \lhd c \rhd q$, are given in table 10. In the original paper [9], these axioms are denoted by COND1 and COND2, but here we use the more compact C1 and C2. Axiom C1 expresses that process $p \lhd t \rhd q$ behaves as p if the data term t evaluates to true (T) and C2 expresses that it behaves as q if t evaluates to false (F).

C1	$x \lhd T \rhd y$	$=$	x
C2	$x \lhd F \rhd y$	$=$	y

Table 10: Axioms for the conditional construct.

A.3.6 The rule RSP

For deriving identities between infinite processes, the proof theory for μCRL contains (an extended version of) the Recursive Specification Principle (RSP, see e.g. [3]). The idea of RSP is that if two series of process terms $p_1(\bar{x}_1), \ldots, p_m(\bar{x}_m)$ and $q_1(\bar{x}_1), \ldots, q_m(\bar{x}_m)$ both form a solution for a guarded system of process equations $G = G_1, \ldots, G_m$ than for every $i \in \{1, \ldots, m\}$ the process terms $p_i(\bar{x}_i)$ and $q_i(\bar{x}_i)$ are considered equal. [5] Each equation G_i in system G has at its left-hand side a fresh

[5] $p(\bar{x})$ denotes a process term p that is possibly parametrised by the variables $\bar{x} = x_1, \ldots, x_n$. In case $n = 0$ then $\bar{x} = \langle\rangle$ denoting the empty sequence of variables meaning that process term p is not parametrised.

(possibly parameterised) identifier $n_i(\bar{x}_i)$ and at its right-hand side a 'process term' that may contain (parametrised) identifiers that appear freshly at the left-hand side of another equation G_j $(j \neq i)$ in G.

A sequence of process terms $p_1(\bar{x}_1), \ldots, p_m(\bar{x}_m)$ is a solution of an equational system $G = G_1, \ldots, G_m$ if for every $i \in \{1, \ldots, m\}$ the equation

$$G_i[\lambda \bar{x}_j \,.\, p_j(\bar{x}_j)/n_j]_{j=i}^{m} \tag{4}$$

is derivable in the proof system. Equation 4 is defined as the equation obtained by substituting $\lambda \bar{x}_j \,.\, p(\bar{x}_j)$ for the n_j-occurrences in G_i, and then repeatedly performing β-conversion on the respective arguments of the identifier n_i. For any identifier without arguments only the substitution of p is performed.

The rule RSP displayed in table 11 is restricted to guarded systems of process-equations. A definition for guardedness that works in a setting of observational equivalence can be given as follows:

Definition A.4. *(Guardedness of G).* A term p is a *guard* iff:

- $p \equiv \delta$, or
- $p \equiv n(t_1, \ldots, t_n)$ or $p \equiv n$ and n is an action label, or
- $p \equiv q_1 \circ q_2$ with $\circ \in \{+, \lhd t \rhd\}$ and q_1 and q_2 are guards, or
- $p \equiv q_1 \circ q_2$ with $\circ \in \{\cdot, \|, \lfloor\!\lfloor \}$ and q_1 or q_2 are guards, or
- $p \equiv q_1 \,|\, q_2$, or
- $p \equiv \Sigma(x : S, q_1)$ and q_1 is a guard, or
- $p \equiv C(nl, q_1)$ with $C \in \{\partial, \rho\}$ and nl being a list of names, and q_1 is a guard.

Let G be a system of process-equations and let N be the left-hand side of one of the equations of G. We say that N is *guarded* in r, where r is a subterm of one of the right-hand sides of G, iff

- $r \equiv q_1 \circ q_2$ with $\circ \in \{+, \|, \lfloor\!\lfloor, |, \lhd t \rhd\}$, and N is guarded in q_1 and q_2,
- $r \equiv q_1 \cdot q_2$ with N is guarded in q_1, and q_1 is a guard or N is guarded in q_2,
- $r \equiv \Sigma(x : S, q_1)$ and N is guarded in q_1,
- $r \equiv C(nl, q_1)$ with $C \in \{\partial, \rho\}$ and nl being a list of names (or in the case of ρ a renaming scheme), and N is guarded in q_1,
- $r \equiv \delta$ or $r \equiv \tau$,
- $r \equiv n'$ for a *name* n' and $N \not\equiv n'$,
- $r \equiv n'(u_1, \ldots, u_{m'})$ and $N \not\equiv n'(x_{i1}, \ldots, x_{im_i})$.

If N is not guarded in r we say that N appears *unguarded* in r.

The *Identifier Dependency Graph* of G, notation $IDG(G)$, is constructed as follows:

- each fully instantiated left-hand side of the equations of G is a node,
- if $N = r \in G$, M appears unguarded in r, N' is obtained by instantiating N via a closed substitution σ, and M' is obtained by instantiating M via σ (where variables in M bound in r are supposed to be different from those in N), then there is an edge $N' \to M'$.

We call G *guarded* iff $IDG(G)$ is well founded, i.e. does not contain an infinite path.

End Definition A.4.

RSP $\quad (\bigwedge_{i=1}^{m}(G_i[\lambda \bar{x}_j \,.\, p_j(\bar{x}_j)/n_j]_{j=1}^{m} \wedge G_i[\lambda \bar{x}_j \,.\, q_j(\bar{x}_j)/n_j]_{j=1}^{m})) \;\rightarrow p_k(\bar{x}_k) = q_k(\bar{x}_k)$

where

- $G_1, \ldots, G_m$ is a *guarded* system of process equations,
- for $1 \leq i \leq m$ the $p_i(\bar{x}_i)$ and $q_i(\bar{x}_i)$ are process terms,
- the notation $[\ldots]_{j=1}^{m}$ abbreviates the m given, consecutive substitutions.

Table 11: The rule RSP.

B Elementary datatypes

In this appendix, we specify the datatypes that are used in the specification and in proof of the bakery protocol. Furthermore, the properties of the datatypes needed for the verification of the bakery are presented as data laws together with their proofs.

B.1 About booleans

The predefined booleans extended with their well-known connectives *not, and, or* [6] are often used in the bakery proof. Note that the (rewrite) rules for the added connectives are consistent with the predefined rules B1 and B2.

sort **Bool**
func $T, F :\rightarrow$ **Bool**

func not : **Bool** $\rightarrow$ **Bool**
and : **Bool** $\times$ **Bool** $\rightarrow$ **Bool**
or : **Bool** $\times$ **Bool** $\rightarrow$ **Bool**
var b, b_1, b_2, b_3 : **Bool**
rew $\mathrm{not}(T) = F$
$\mathrm{not}(F) = T$
T and $b = b$
F and $b = F$
T or $b = T$
F or $b = b$

Lemma B.1. *$\{T, F\}$ is a constructor set of sort* **Bool**.

Proof. By B2 every term t : **Bool** is either equal to T or to F. □

Lemma B.2.

1. $p \lhd b \rhd q = q \lhd \mathrm{not}(b) \rhd p$,
2. $p \lhd b_1$ or $b_2 \rhd \delta = p \lhd b_1 \rhd \delta + p \lhd b_2 \rhd \delta$,
3. $x \lhd b \rhd y = (x + z \lhd \mathrm{not}(b) \rhd \delta) \lhd b \rhd y$,
4. $\neg(p \lhd b_1$ or $b_2 \rhd q = p \lhd b_1 \rhd q + p \lhd b_2 \rhd q)$,

[6] The symbols $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ are reserved in the proof system as operators for connecting properties (see definition A.1).

5. $p \lhd b_1$ and $b_2 \rhd q = (p \lhd b_1 \rhd q) \lhd b_2 \rhd q$,
6. $p \lhd b_1$ or $b_2 \rhd q = p \lhd b_1 \rhd (p \lhd b_2 \rhd q)$,
7. b_1 and $b_1 = b_1$,
8. b_1 or $b_1 = b_1$,
9. b_1 and $b_2 = b_2$ and b_1,
10. b_1 or $b_2 = b_2$ or b_1,
11. $(b_1$ and $b_2)$ and $b_3 = b_1$ and $(b_2$ and $b_3)$,
12. $(b_1$ or $b_2)$ or $b_3 = b_1$ or $(b_2$ or $b_3)$,
13. $(b_1$ and $b_2)$ or $b_3 = (b_1$ or $b_3)$ and $(b_2$ or $b_3)$,
14. $(b_1$ or $b_2)$ and $b_3 = (b_1$ and $b_3)$ or $(b_2$ and $b_3)$,
15. b_1 and $(b_1$ or $b_2) = b_1$,
16. b_1 or $(b_1$ and $b_2) = b_1$.

Proof. By C1, C2 and lemma 4.1. □

B.2 About natural numbers

The natural numbers represented by sort *nat* play an important role in both the specification and the verification of the bakery protocol. Below the operators on natural numbers $0, P$ (Predecessor), $S, +, -$ (monus), $\geq, \leq, <, >, if, eq$ used in this paper are specified. (We will use infix notation wherever we find it convenient to do so.)

sort nat

func $0 :\rightarrow nat$
$S, P : nat \rightarrow nat$
$+, -, : nat \times nat \rightarrow nat$
$eq, \geq, \leq, <, >: nat \times nat \rightarrow \mathbf{Bool}$
$if : \mathbf{Bool} \times nat \times nat \rightarrow nat$

var $n, m, z : nat$

rew $P(0) = 0$
$P(S(n)) = n$
$n + 0 = n$
$n + S(m) = S(n + m)$
$n - 0 = n$
$n - S(m) = P(n - m)$
$eq(0, 0) = T$
$eq(0, S(n)) = F$
$eq(S(n), 0) = F$
$eq(S(n), S(m)) = eq(n, m)$
$n \geq 0 = T$
$0 \geq S(n) = F$
$S(n) \geq S(m) = n \geq m$
$n \leq m = m \geq n$
$n > m = n \geq S(m)$
$n < m = S(n) \leq m$
$if(T, n, m) = n$
$if(F, n, m) = m$

The *if* operator does not occur in the verification and specification but is used for specifying modulo arithmetic in section B.3.

Notation B.3. We write $n \leq m$ for $n \leq m = T$. Idem for $\geq$,$>$ and $<$. We write $eq(n, m)$ for $eq(n, m) = T$. We write 1 for $S(0)$ and 2 for $S(S(0))$.

End Notation B.3.

Lemma B.4. *$\{0, S\}$ is a constructor set for sort nat.*

Lemma B.5.

1. $eq(n, m) = T \leftrightarrow n = m$,
2. $n \leq m = eq(n - m, 0)$,
3. $n < m = eq(S(n) - m, 0)$,
4. $n = m \vee \neg(n = m)$,
5. $n \leq m \wedge m \leq n \rightarrow n = m$,
6. $n \leq m \vee m \leq n \rightarrow n = m$,
7. $n \leq m \leftrightarrow n = m \vee n < m$,
8. $n = m \vee n < m \vee m < n$,
9. $(\neg n \leq m) \leftrightarrow m < n$,
10. $n + S(0) = S(n)$,
11. $S(0) + n = S(n)$,
12. $n + (m + z) = (n + m) + z$,
13. $n + m = m + n$,
14. $n - m = S(n) \leftrightarrow n = m + S(z)$,
15. $n < m \wedge m < z \rightarrow n < z$,
16. $n \leq m \wedge m \leq z \rightarrow n \leq z$,
17. $n \geq m = not(n < m)$,
18. $P(n) = n - S(0)$,
19. *etc.*

Proof. By (nested) induction on *nat* (see [12]). □

B.3 About modulo arithmetic

On top of the natural numbers *nat* we specify the *mod* operator and the $+_m$ operator (addition modulo m) as follows.

func	$\text{mod} : nat \times nat \rightarrow nat$
	$+ : nat \times nat \times nat \rightarrow nat$
var	$n, n', m : nat$
rew	$n \text{ mod } 0 = 0$
	$n \text{ mod } m = if(n \geq m, n \text{ mod } (n - m), n)$
	$n +_m n' = (n + n') \text{ mod } m$

Lemma B.6.

1. $n_1 +_m n_2 = n_2 +_m n_1$,
2. $(n_1 +_m n_2) +_m n_3 = n_1 +_m (n_2 +_m n_3)$,
3. $n_2 > 0 \rightarrow (n_1 \bmod n_2) < n_2$,
4. $(n_1 \bmod n_2) \bmod n_2 = n_1 \bmod n_2$.

Proof. Straightforward with induction. □

B.4 About data queues

B.4.1 Sort $queue_d$

In this section, we specify the datatype $queue_d$ which can contain elements of a set D of customers. D may be finite or infinite, but it should at least contain a bottom element $d_\perp$ for denoting an undefined data element. We assume that D is equipped with an equality function $eq : D \times D \rightarrow \mathbf{Bool}$ that has the obvious property $eq(d, e) = T \leftrightarrow d = e$.

sort	D
func	$d_\perp : D$
	$d_1, \ldots, d_n : D$
	$eq : D \times D \rightarrow \mathbf{Bool}$

The specification of $queue_d$ is given below.

sort	$queue_d$
func	$\varnothing_d :\rightarrow queue_d$
	$in, rem : D \times queue_d \rightarrow queue_d$
	$test : D \times queue_d \rightarrow \mathbf{Bool}$
	$size : queue_d \rightarrow nat$
	$hd, toe : queue_d \rightarrow D$
	$tl, untoe : queue_d \rightarrow queue_d$
	$if : \mathbf{Bool} \times queue_d \times queue_d \rightarrow queue_d$
	$empty : queue_d \rightarrow \mathbf{Bool}$
var	$d, e : D$
	$b, c : queue_d$
rew	$test(d, \varnothing_d) = F$
	$test(d, in(e, b)) = if(eq(d, e), T, test(d, b))$
	$rem(d, \varnothing_d) = \varnothing_d$
	$rem(d, in(e, b)) = if(eq(d, e), b, in(e, rem(d, b)))$
	$size(\varnothing_d) = 0$
	$size(in(d, b)) = S(size(b))$
	$hd(\varnothing_d) = d_\perp$
	$hd(in(d, b)) = d$
	$toe(\varnothing_d) = d_\perp$
	$toe(in(d, \varnothing_d)) = d$
	$toe(in(d, in(e, b))) = toe(in(e, b))$
	$tl(\varnothing_d) = \varnothing_d$
	$tl(in(d, b)) = b$
	$untoe(\varnothing_d) = \varnothing_d$
	$untoe(in(d, \varnothing_d)) = \varnothing_d$
	$untoe(in(d, in(e, b))) = in(d, untoe(in(e, b)))$
	$empty(b) = eq(size(b), 0)$

Lemma B.7. $\{\varnothing_d, in\}$ *is a constructor set for the datasort* $queue_d$.

Lemma B.8.

1. $size(untoe(b)) = size(b) - 1$,
2. $size(tl(b)) = size(b) - 1$,
3. $\neg b = \varnothing_d \rightarrow in(hd(b), tl(b)) = b$,
4. $test(p, tl(b)) \rightarrow in(hd(b), rem(d, tl(b))) = rem(d, b)$,
5. $size(b) \leq 0 \leftrightarrow b = \varnothing_d$,
6. $\neg b = \varnothing_d \rightarrow test(d, tl(b))$ or $eq(d, hd(d)) = test(d, b)$.

Proof. Straightforward with induction on $b : queue_d$. □

B.4.2 Sort $queue_p$

In this section, we specify the datatype $queue_p$ which can contain elements of sort $Pair$ as specified in section 2. The μCRL specification of $queue_p$ is given below.

sort $queue_p$

func
$\varnothing_p :\rightarrow queue_p$
$in, rem : Pair \times queue_p \rightarrow queue_p$
$test : Pair \times queue_p \rightarrow \mathbf{Bool}$
$size : queue_p \rightarrow nat$
$hd, toe : queue_p \rightarrow Pair$
$tl, untoe : queue_p \rightarrow queue_p$
$if : \mathbf{Bool} \times queue_p \times queue_p \rightarrow queue_p$
$empty : queue_p \rightarrow \mathbf{Bool}$
$number : nat \times nat \times queue_d \rightarrow queue_p$

var
$d : D$
$p, p' : Pair$
$b, c : queue_p$

rew
$test(p, \varnothing_p) = F$
$test(p, in(p', b)) = if(eq(p, p'), T, test(p, b))$
$rem(p, \varnothing_p) = \varnothing_p$
$rem(p, in(p', b)) = if(eq(p, p'), b, in(p', rem(p, b)))$
$size(\varnothing_p) = 0$
$size(in(p, b)) = S(size(b))$
$hd(\varnothing_p) = \langle d_\perp, 0\rangle$
$hd(in(p, b)) = p$
$toe(\varnothing_p) = \langle d_\perp, 0\rangle$
$toe(in(p, \varnothing_p) = p$
$toe(in(p, in(p', b))) = toe(in(p', b))$
$tl(\varnothing_p) = \varnothing_p$
$tl(in(p, b)) = b$
$untoe(\varnothing_p) = \varnothing_p$
$untoe(in(p, \varnothing_p)) = \varnothing_p$
$untoe(in(p, in(p', b))) = in(p, untoe(in(p', b)))$
$empty(b) = eq(size(b), 0)$
$number(i, n, \varnothing_d) = \varnothing_p$
$number(i, n, in(d, b)) = in(\langle d, i +_n size(tl(b))\rangle, number(i, n, b))$

Lemma B.9. $\{\varnothing_p, in\}$ *is a constructor set for the datasort* $queue_p$.

Lemma B.10.

1. $size(untoe(b)) = size(b) - 1$,
2. $size(tl(b)) = size(b) - 1$,
3. $\neg b = \varnothing_p \rightarrow in(hd(b), tl(b)) = b$,
4. $\mathrm{test}(p, tl(b)) = T \rightarrow in(hd(b), rem(p, tl(b))) = rem(p, b)$,
5. $size(b) \leq 0 \leftrightarrow b = \varnothing_p$,
6. $b = \varnothing_p \rightarrow size(b) \leq n$,
7. $\neg b = \varnothing_p \rightarrow test(p, tl(b))$ or $eq(p, hd(p)) = test(p, b)$,
8. $size(b) \leq n \rightarrow test(\langle d, i\rangle, number(i, n, b)) = eq(p, toe(b))$ and $size(b) > 0$
9. $size(b) \leq n \rightarrow test(\langle d, i\rangle, number(i, n, b)) = size(b) \geq 1$ and $eq(d, toe(b))$,
10. $size(b) > 0 \rightarrow in(\langle hd(b), i +_n size(tl(b))\rangle, number(i, n, tl(b))) = number(i, n, b)$,
11. $size(b) \leq n \rightarrow rem(\langle toe(b), i\rangle, number(i, n, b)) = number(i +_n 1, n, untoe(b))$,
12. $size(b) > 0 \rightarrow size(tl(b)) < n = size(b) \leq n$,
13. $size(b) \rightarrow size(number(i +_n 1, n, tl(untoe(b)))) < n = size(b) < n + 2$,
14. $size(number(i, n, b)) = size(b)$,
15. $n > 0 \rightarrow$
    $size(b) > 0$ and $size(b) \leq n + 1 =$
    $size(b) > 1$ and $size(b) \leq n + 1$ or $size(b) > 0$ and $size(b) \leq n$.

Proof. Straightforward with induction on $b : queue_p$. □

References

[1] K.R. Apt. Ten years of Hoare's logic, a survey, part I. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, 1981.

[2] K.R. Apt. Ten years of Hoare's logic, a survey, part II: Nondeterminism. *Theoretical Computer Science*, 28:83–109, 1984.

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

[4] K.M. Chandy and J. Misra. *Parallel Program Design. A Foundation.* Addison-Wesley, 1988.

[5] D. van Dalen. *Logic and Structure.* Springer-Verlag, 1983.

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

[7] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In G.X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, 1989.

[8] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. Technical Report CS-R9076, CWI, Amsterdam, 1990.

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

[10] L. Lamport. A new solution to Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455, 1974.

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

[12] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics. An Introduction. Part I and part II*, volume 121 and 123 of *Studies in Logic and the Foundation of Mathematics.* North-Holland, 1988.

[13] M.P.A. Sellink. An implementation of μCRL in COQ. Technical Report (to appear), Utrecht University, 1992.