A Correctness Proof of the Bakery Protocol in μCRL

Jan Friso Groote

Department of Philosophy, Utrecht University

Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

`jfg@phil.ruu.nl`

Henri Korver

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

`henri@cwi.nl`

Abstract

A specification of a 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 data types and elementary rules and axioms from process algebra.

AMS Subject Classification (1991): 68Q60, 68T15.
CR Subject Classification (1991): F.3.1.
Keywords & Phrases: Abstract data types, Bakery Protocol, process algebra, proof checking, protocol verification.
Note: A slightly different version of this paper appeared as [GK95].

1 Introduction

The main purpose of this paper is to show that μCRL, or in more general terms process algebra with abstract data types, offers a framework for reasoning about distributed systems. This is done by the verification of a bakery protocol, which is a non trivial protocol with unbounded state space. Neither process algebra nor data type theory seems to form a suitable vehicle for the verification of this protocol on their own, showing that the verification capacities of μCRL go beyond those found in both its constituents. Actually, this observation has been confirmed by the verification of a number of other 'difficult' protocols (see e.g. [BG93a, GP93, KS94]). 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. All these are essential in the verification given in this paper.

Our work structurally differs from the more conventional 'assertional' verification techniques (see [Apt81, Apt84, CM88]). 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 3 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 [Sel93, BG93b, GP93, KS94]. 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 calculations that we may expect that also the proof in this paper is not completely flawless. The only way to systematically increase the reliability of proofs is by having

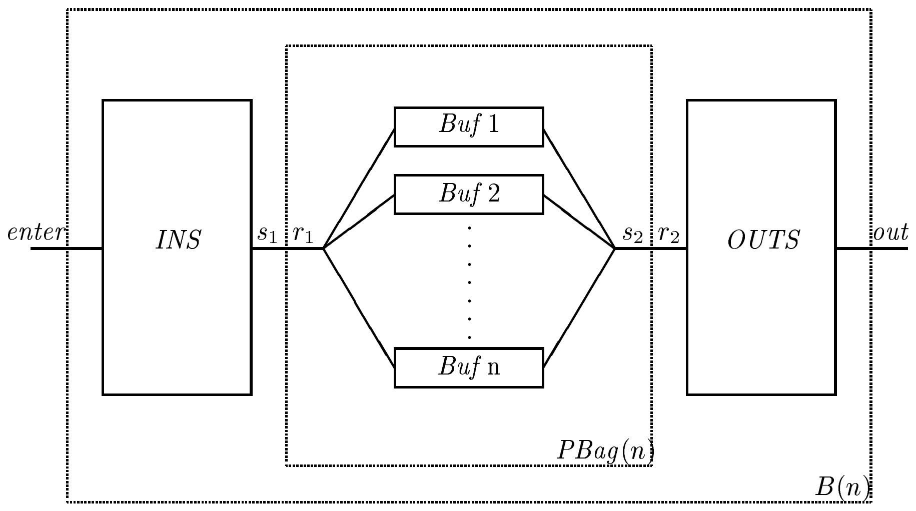

Figure 1: The Bakery Protocol.

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 initially the proof was not easy to find due to the large number of possible proof strategies, the resulting proof follows a reasonable and straightforward line of thought. This is promising, because we think that if we get more skill and experience in doing calculations such as given in this 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 a well-known protocol that has been used over the centuries, especially in bakery shops, and prove its correctness in the proof system for μCRL [GP94b, GP94a]. We assume that the reader is familiar with μCRL which is a straightforward combination of process algebra [BW90] and abstract data types [EM85]. But see Appendix B for the axiom system of μCRL.

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 data types, which are in this case booleans and natural numbers, for granted. These data types are specified in appendix A. 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 A.

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 have a sort $queue_d$ that consists of queues of customers (see appendix A for its specification). In order to attach a number to a customer the data type *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.

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

We also need queues which can contain pairs. Therefore, the data type $queue_p$ is introduced in appendix A.

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

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

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'.

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

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

The process $\mathit{INS}(n, i)$ 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 $\mathit{OUTS}(n, i)$ 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*.

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

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

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

where $I = \{c_1, c_2\}$ and $H = \{r_1, s_1, r_2, s_2\}$.

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 $\delta \parallel \mathit{Buf} = \mathit{Buf}$ 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:

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

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 Protocol. 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 data terms (possibly containing variables). 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 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 process terms.

The following lemma expresses that μCRL supports the Excluded Middle Principle.

Lemma 4.1 (*Excluded Middle Principle*). *Let ϕ and ψ be two arbitrary property-formulas over a given μCRL specification.*

1. *If $\phi \rightarrow \psi$ and $\neg\phi \rightarrow \psi$ then ψ.*
2. *If $b = \mathsf{t} \rightarrow \psi$ and $b = \mathsf{f} \rightarrow \psi$ then ψ.*

Proof.

1. See [GP94a].
2. Immediate by 4.1.1 and BOOL2.

$\square$

Lemma 4.1.2 is an instance of the Excluded Middle Principle which is often applied in μCRL proofs, for instance to prove the conditional identities in the next lemma.

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. $(x \lhd b \rhd \delta) \mathbin{⌊\!⌊} y = (x \mathbin{⌊\!⌊} y) \lhd b \rhd \delta$,

6. $x \mid (y \lhd b \rhd \delta) = (x \mid y) \lhd b \rhd \delta$,

7. $p[e/d] = p \lhd eq(d,e) \rhd \delta + p[e/d]$, *provided that* $eq(d,e) \to d = e$,

8. $a(d) \cdot x \mid b(e) \cdot y = (a(d) \cdot x \mid b(e) \cdot y) \lhd eq(d,e) \rhd \delta$, *provided that* $d = e \to eq(d,e)$.

These identities are frequently used in the verification of the Bakery Protocol and μCRL verifications in general.

Proof.

1–7. Easy with axioms COND1, COND2 and Lemma 4.1.2.

8. By COND1, COND2, CF and Lemma 4.1.2.

□

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. We first introduce an auxiliary proposition, which enables us to identify processes via summand inclusion.

Proposition 4.3. *Let $p \subseteq q$ be a shorthand for $q = q + p$ expressing that p is a summand of q. Then, we have:*

$$p \subseteq q \land q \subseteq p \to p = q.$$

Proof. $q = q + p \overset{\text{A1}}{=} p + q = p$. □

Sometimes it is more convenient to reason with summands instead of equations.

Lemma 4.4 (*Sum Elimination*). *Assume that there is an equality function eq such that $eq(d,e) = \mathsf{t} \leftrightarrow d = e$. Then*

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

Proof. By showing summand inclusion in both directions (Proposition 4.3).

⊇: Consider the following obvious identity

$$(1) \qquad \sum_{d:D} (p \lhd eq(d,e) \rhd \delta) \overset{\text{SUM3}}{=} \sum_{d:D} (p \lhd eq(d,e) \rhd \delta) + p \lhd eq(d,e) \rhd \delta$$

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

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

Note that the substitution does not affect both sum constructs above because in μCRL substitutions do not change bound variables (see [GP94a]). By applying axiom COND1 to the second summand on the right hand side of equation (2), we get

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

$\subseteq$: By the calculation:

$$\begin{array}{lcl} p[e/d] & \overset{\text{SUM1}}{=} & \sum_{d:D}(p[e/d]) \\ & \overset{4.2.7}{=} & \sum_{d:D}(p \triangleleft eq(d,e) \triangleright \delta + p[e/d]) \\ & \overset{\text{SUM4}}{=} & \sum_{d:D}(p \triangleleft eq(d,e) \triangleright \delta) + \sum_{d:D}(p[e/d]) \\ & \overset{\text{SUM1}}{=} & \sum_{d:D}(p \triangleleft eq(d,e) \triangleright \delta) + p[e/d]. \end{array}$$

$\square$

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

Lemma 4.5 (*Left Merge with SUM and COND*)**.** *We assume that the variable d does not occur free in term q.*

1. $\sum_{d:D}(a(d) \cdot p) \mathbin{\lfloor\!\lfloor} q = \sum_{d:D}(a(d) \cdot (p \parallel q))$
2. $\sum_{d:D}(a(d) \cdot p \triangleleft c \triangleright \delta) \mathbin{\lfloor\!\lfloor} q = \sum_{d:D}(a(d) \cdot (p \parallel q) \triangleleft c \triangleright \delta)$

Proof.

1. By the following calculation

$$\begin{array}{ll} \multicolumn{2}{l}{\sum_{d:D}(a(d) \cdot p) \mathbin{\lfloor\!\lfloor} q} \\ \overset{\text{SUM6}}{=} & \sum_{d:D}(a(d) \cdot p \mathbin{\lfloor\!\lfloor} q) \\ \overset{\text{CM3}}{=} & \sum_{d:D}(a(d) \cdot (p \parallel q)) \end{array}$$

2. By the following calculation

$$\begin{array}{ll} \multicolumn{2}{l}{\sum_{d:D}(a(d) \cdot p \triangleleft c \triangleright \delta) \mathbin{\lfloor\!\lfloor} q} \\ \overset{\text{SUM6}}{=} & \sum_{d:D}(a(d) \cdot p \triangleleft c \triangleright \delta \mathbin{\lfloor\!\lfloor} q) \\ \overset{4.2.5}{=} & \sum_{d:D}(a(d) \cdot p \mathbin{\lfloor\!\lfloor} q \triangleleft c \triangleright \delta) \\ \overset{\text{CM3}}{=} & \sum_{d:D}(a(d) \cdot (p \parallel q) \triangleleft c \triangleright \delta) \end{array}$$

$\square$

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 4.5.2, we often use a boolean term c instead of a boolean variable b to be as general as possible. Otherwise b can never be substituted by a term containing a variable d, as substitution may not create bound variables.

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

Lemma 4.6 (*Communication with SUM and COND*)**.** *Let $d, e : D$ and $d' : D'$ be variables, and let $t : D$ a term. Assume there is an equality function eq such that $eq(d,e) = \mathsf{t} \leftrightarrow d = e$, and variable d does not occur free in terms q and c_2. Variable d' does not occur free in p, c and c_1.*

1. $\sum_{d:D}(a(d) \cdot p) \mid b(e) \cdot q = (a(e) \mid b(e)) \cdot (p[e/d] \parallel q)$
2. $\sum_{d:D}(a(d) \cdot p \triangleleft c \triangleright \delta) \mid b(e) \cdot q = (a(e) \mid b(e)) \cdot (p[e/d] \parallel q) \triangleleft c[e/d] \triangleright \delta$
3. $\sum_{d:D}(a(d) \cdot p \triangleleft c \triangleright \delta) \mid \sum_{d':D'}(b(t) \cdot q) = \sum_{d':D'}((a(t) \mid b(t)) \cdot (p[t/d] \parallel q) \triangleleft c[t/d] \triangleright \delta)$

4. $\sum_{d:D}(a(d)\cdot p \lhd c_1 \rhd \delta) \mid \sum_{d':D'}(b(t)\cdot q \lhd c_2 \rhd \delta) =$
$\sum_{d':D'}((a(t)\mid b(t))\cdot(p[t/d] \parallel q) \lhd c_1[t/d] \text{ and } c_2 \rhd \delta)$

Proof.

1. By the following calculation

$$\begin{aligned}
&\textstyle\sum_{d:D}(a(d)\cdot p) \mid b(e)\cdot q\\
\overset{\text{SUM7}}{=}\quad &\textstyle\sum_{d:D}(a(d)\cdot p \mid b(e)\cdot q)\\
\overset{4.2.8}{=}\quad &\textstyle\sum_{d:D}((a(d)\cdot p \mid b(e)\cdot q) \lhd eq(d,e) \rhd \delta)\\
\overset{\text{SumEl.}}{=}\quad & a(e)\cdot p[e/d] \mid b(e)\cdot q\\
\overset{\text{CM7}}{=}\quad & (a(e)\mid b(e))\cdot(p[e/d] \parallel q)
\end{aligned}$$

Note that the assumption $eq(d,e) = \mathsf{t} \leftrightarrow d = e$ is needed for application of lemmas 4.2.8 and 4.4 (Sum Elimination) in the calculation above.

2. By the following calculation

$$\begin{aligned}
&\textstyle\sum_{d:D}(a(d)\cdot p \lhd c \rhd \delta) \mid b(e)\cdot q\\
\overset{\text{SUM7}}{=}\quad &\textstyle\sum_{d:D}((a(d)\cdot p \lhd c \rhd \delta) \mid b(e)\cdot q)\\
\overset{4.2.6}{=}\quad &\textstyle\sum_{d:D}((a(d)\cdot p \mid b(e)\cdot q) \lhd c \rhd \delta)\\
\overset{4.2.8}{=}\quad &\textstyle\sum_{d:D}(((a(d)\cdot p \mid b(e)\cdot q) \lhd eq(d,e) \rhd \delta) \lhd c \rhd \delta)\\
\overset{\text{SumEl.}}{=}\quad & (a(e)\cdot p[e/d] \mid b(e)\cdot q) \lhd c \rhd \delta\\
\overset{\text{CM7}}{=}\quad & (a(e)\mid b(e))\cdot(p[e/d] \parallel q) \lhd c \rhd \delta
\end{aligned}$$

Then by the substitution rule SUB (see [GP94a]) we have

$$\sum_{d:D}(a(d)\cdot p \lhd c \rhd \delta) \mid b(e)\cdot q = (a(e)\mid b(e))\cdot(p[e/d] \parallel q) \lhd c[e/d] \rhd \delta$$

as the substitution $[e/d]$ does not effect the right-hand side of this equation.

3. By the following calculation

$$\begin{aligned}
&\textstyle\sum_{d:D}(a(d)\cdot p \lhd c \rhd \delta) \mid \sum_{d':D'}(b(t)\cdot q)\\
\overset{\text{SUM7}}{=}\quad &\textstyle\sum_{d':D'}(\sum_{d:D}(a(d)\cdot p \lhd c \rhd \delta) \mid (b(t)\cdot q))\\
\overset{4.6.2}{=}\quad &\textstyle\sum_{d':D'}((a(t)\mid b(t))\cdot(p[t/d] \parallel q) \lhd c[t/d] \rhd \delta)
\end{aligned}$$

4. In a similar way as the proof of 4.6.3.

□

In the proofs given above we often needed an equality function eq for comparing elements of data type D. In order to have the rather desirable property that $eq(d,e) \leftrightarrow d = e$, one can extend the specification of D with a selector function *if* and four axioms. This is formulated in Lemma 4.7. The axioms are due to Jan Bergstra.

$$
\begin{array}{ll}
\textbf{sort} & D \\
\textbf{func} & \ldots \\
 & eq : D \times D \to \textbf{Bool} \\
 & if : \textbf{Bool} \times D \times D \to D \\
\textbf{var} & d, e : D \\
\textbf{rew} & \ldots \\
 & if(\mathsf{t}, d, e) = d \\
 & if(\mathsf{f}, d, e) = e \\
 & eq(d, d) = \mathsf{t} \\
 & if(eq(d, e), d, e) = e
\end{array}
$$

Lemma 4.7. *Let d, e be variables of sort D, and let eq be the equality function specified above. Then, $eq(d, e) \leftrightarrow d = e$.*

5 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 use the fact that $PBag(n)$ behaves as a 'standard' bounded bag which is usually described by a data type bag. This fact has already be proven in μCRL (see [Kor94]). Then we show that this bounded bag combined with the in-sequencer *INS* and the out-sequencer *OUTS* is equal to the bounded queue described above.

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

The standard behaviour of a bounded bag is specified as follows:

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

The D in *DBag* refers to 'Data'. Although the data type $queue_p$ is actually a queue, it has been extended with functions *test* and *rem* (*rem*ove) which makes it possible to use it as a bag.

The next theorem taken from [Kor94] says that $PBag(n)$ behaves like a bounded bag of size n.

Theorem 5.1. *$PBag(n) = DBag(n, \varnothing_p)$.*

Proof. By induction on n and RSP (see [Kor94]). □

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 situations above we 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 matter. 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.

```
proc $CQ_0(n : nat, i : nat, b : queue_d) =$
            $\tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}($
                $INS(n, i +_n size(b)) \parallel$
                $DBag(n, number(i, n, b)) \parallel$
                $OUTS(n, i)))$
            $\lhd size(b) \leq n \text{ and } i < n \rhd \delta$

     $CQ_1(n : nat, i : nat, b : queue_d) =$
            $\tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}($
                $INS(n, i +_n size(b)) \parallel$
                $DBag(n, number(i +_n 1, n, untoe(b))) \parallel$
                $out(toe(b))\ OUTS(n, i +_n 1)))$
            $\lhd 0 < size(b) \leq n+1 \text{ and } i < n \rhd \delta$

     $CQ_2(n : nat, i : nat, b : queue_d) =$
            $\tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}($
                $s_1(\langle hd(b), i +_n size(tl(b))\rangle)\ INS(n, i +_n size(b)) \parallel$
                $DBag(n, number(i, n, tl(b))) \parallel$
                $OUTS(n, i)))$
            $\lhd 0 < size(b) \leq n+1 \text{ and } i < n \rhd \delta$

     $CQ_3(n : nat, i : nat, b : queue_d) =$
            $\tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}($
                $s_1(\langle hd(b), i +_n size(tl(b))\rangle)\ INS(n, i +_n size(b)) \parallel$
                $DBag(n, number(i +_n 1, n, tl(untoe(b)))) \parallel$
                $out(toe(b))\ OUTS(n, i +_n 1)))$
            $\lhd 1 < size(b) \leq n+2 \text{ and } i < n \rhd \delta$
```

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

Now let us consider say $CQ_0(n, i, b)$ and 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 5.2.3.

$$\begin{aligned}
&\textbf{proc } G_0(n:nat, i:nat, b:queue_d) = \\
&\quad \textstyle\sum_{d:D}(enter(d)\, G_2(n,i,in(d,b))) \lhd size(b) \leq n \text{ and } i < n \rhd \delta + \\
&\quad \tau\, G_1(n,i,b) \lhd 0 < size(b) \leq n \text{ and } i < n \rhd \delta
\end{aligned}$$

$$\begin{aligned}
&G_1(n:nat, i:nat, b:queue_d) = \\
&\quad \textstyle\sum_{d:D}(enter(d)\, G_3(n,i,in(d,b))) \\
&\qquad\qquad \lhd 0 < size(b) \leq n+1 \text{ and } i < n \rhd \delta + \\
&\quad out(toe(b))\, G_0(n, i +_n 1, untoe(b)) \\
&\qquad\qquad \lhd 0 < size(b) \leq n+1 \text{ and } i < n \rhd \delta
\end{aligned}$$

$$\begin{aligned}
&G_2(n:nat, i:nat, b:queue_d) = \\
&\quad \tau\, G_0(n,i,b) \lhd 0 < size(b) \leq n \text{ and } i < n \rhd \delta + \\
&\quad \tau\, G_3(n,i,b) \lhd 1 < size(b) \leq n+1 \text{ and } i < n \rhd \delta
\end{aligned}$$

$$\begin{aligned}
&G_3(n:nat, i:nat, b:queue_d) = \\
&\quad \tau\, G_1(n,i,b) \lhd 1 < size(b) \leq n+1 \text{ and } i < n \rhd \delta + \\
&\quad out(toe(b))\, G_2(n, i +_n 1, untoe(b)) \\
&\qquad\qquad \lhd 1 < size(b) \leq n+2 \text{ and } i < n \rhd \delta
\end{aligned}$$

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.

$$\begin{aligned}
&\textbf{proc } Q_0(n:nat, i:nat, b:queue_d) = \\
&\quad (Q(n+2,b) \lhd empty(b) \rhd \tau\, Q(n+2,b)) \\
&\quad \lhd size(b) \leq n \text{ and } i < n \rhd \delta
\end{aligned}$$

$$\begin{aligned}
&Q_1(n:nat, i:nat, b:queue_d) = \\
&\quad Q(n+2,b) \lhd 0 < size(b) \leq n+1 \text{ and } i < n \rhd \delta
\end{aligned}$$

$$\begin{aligned}
&Q_2(n:nat, i:nat, b:queue_d) = \\
&\quad \tau\, Q(n+2,b) \lhd 0 < size(b) \leq n+1 \text{ and } i < n \rhd \delta
\end{aligned}$$

$$\begin{aligned}
&Q_3(n:nat, i:nat, b:queue_d) = \\
&\quad (Q(n+2,b) \lhd eq(size(b), n+2) \rhd \tau\, Q(n+2,b)) \\
&\qquad\qquad \lhd 1 < size(b) \leq n+2 \text{ and } i < n \rhd \delta
\end{aligned}$$

Theorem 5.2. *Let $i, n : nat$, $b : queue_d$.*

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

Proof.

1. $$\begin{aligned}
B(n) &= \tau_I \partial_H(INS(n,0) \parallel PBag(n) \parallel OUTS(n,0)) \\
&\overset{5.1}{=} \tau_I \partial_H(INS(n,0) \parallel DBag(n,\varnothing_p) \parallel OUTS(n,0))
\end{aligned}$$

$$= \quad CQ_0(n, 0, \varnothing_d)$$

2. Immediate using the definitions of $Q(n{:}nat)$ and $Q_0(n, i{:}nat, b{:}queue_d)$.

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 (see appendix B) 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 4.5 and 4.6 and the applications of some lemmas about data given in appendix A.

$$
\begin{array}{ll}
\multicolumn{2}{l}{CQ_0(n,i,b)} \\
= & \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,b)) \parallel OUTS(n,i))) \\
& \lhd size(b) \le n \text{ and } i < n \rhd \delta \\
\\
\overset{4.5}{=} & \sum_{d:D}(enter(d) \\
& \quad \tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}(\\
& \qquad s_1(\langle d, i +_n size(b)\rangle)\, INS(n, i +_n size(b) +_n 1) \parallel \\
& \qquad DBag(n, number(i,n,b)) \parallel OUTS(n,i)))) \\
& \lhd size(b) \le n \text{ and } i < n \rhd \delta \\
+ & \tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}(\\
& \quad (\sum_{p:Pair}(s_2(p)\, DBag(n, rem(p, number(i,n,b)))) \\
& \qquad\qquad \lhd test(p, number(i,n,b)) \rhd \delta) \parallel \\
& \quad \sum_{d:D}(r_2(\langle d,i\rangle)\, out(d))\, OUTS(n, i +_n 1)) \,\lfloor\!\lfloor \\
& \quad INS(n, i +_n size(b)))) \\
& \lhd size(b) \le n \text{ and } i < n \rhd \delta \\
\\
\overset{4.6.3}{=} & \sum_{d:D}(enter(d)\, CQ_2(n,i,in(d,b))) \lhd size(b) \le n \text{ and } i < n \rhd \delta \\
+ & \tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}(\\
& \quad \sum_{d:D}(c_1(\langle d,i\rangle) \\
& \qquad (DBag(n, rem(\langle d,i\rangle, number(i,n,b))) \parallel \\
& \qquad out(d)\; OUTS(n, i +_n 1)) \\
& \qquad \lhd test(\langle d,i\rangle, number(i,n,b)) \rhd \delta)) \,\lfloor\!\lfloor \\
& \quad INS(n, i +_n size(b)))) \\
& \lhd size(b) \le n \text{ and } i < n \rhd \delta \\
\\
\overset{\text{A.6.1, Sum El.}}{=} & \\
& \sum_{d:D}(enter(d)\, CQ_2(n,i,in(d,b))) \lhd size(b) \le n \text{ and } i < n \rhd \delta \\
+ & \tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}(\\
& \quad (c_1(\langle toe(b), i\rangle) \\
& \qquad (DBag(n, rem(\langle toe(b), i\rangle, number(i,n,b)))) \parallel \\
& \qquad out(toe(b))\; OUTS(n, i +_n 1)) \,\lfloor\!\lfloor \\
& \quad INS(n, i +_n size(b)))) \\
& \lhd 0 < size(b) \le n \text{ and } i < n \rhd \delta \\
\\
\overset{\text{A.6.3,TI2}}{=} & \sum_{d:D}(enter(d)\, CQ_2(n,i,in(d,b))) \lhd size(b) \le n \text{ and } i < n \rhd \delta \\
+ & \tau \cdot \tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}(\\
& \qquad INS(n, i +_n size(b)) \parallel
\end{array}
$$

$$\begin{array}{ll}
& \quad DBag(n, number(i +_n 1, n, untoe(b))) \parallel \\
& \quad out(toe(b))\ OUTS(n, i +_n 1))) \\
& \lhd 0 < size(b) \leq n \text{ and } i < n \rhd \delta \\
\\
= & \sum_{d:D}(enter(d)\ CQ_2(n, i, in(d, b))) \lhd size(b) \leq n \text{ and } i < n \rhd \delta \\
+ & \tau\ CQ_1(n, i, b) \lhd 0 < size(b) \leq n \text{ and } i < n \rhd \delta
\end{array}$$

The calculations showing that $CQ_j(n, i, b)$ satisfy the equation defining G_1 are analogous to the ones given above. They are omitted here but can be found in the full version [GK92].

We continue by showing that the processes $CQ_j(n, i, b)$ also satisfy the equation for G_2.

$$\begin{array}{ll}
CQ_2(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, tl(b))) \parallel OUTS(n, i))) \\
& \lhd 0 < size(b) \leq n + 1 \text{ and } i < n \rhd \delta \\
\\
\overset{4.6.1,\ 4.6.3}{=} & \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) \\
& \quad\quad (INS(n, i +_n size(b)) \parallel \\
& \quad\quad\ DBag(n, in(\langle hd(b), i +_n size(tl(b))\rangle, \\
& \quad\quad\quad\quad number(i, n, tl(b))))) \\
& \quad \lhd size(b) \leq n \rhd \delta) \mathbin{⌊\!⌊} OUTS(n, i))) \\
& \lhd 0 < size(b) \leq n + 1 \text{ and } i < n \rhd \delta \\
+ & \tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}(\\
& \quad \sum_{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 \lhd test(\langle d, i\rangle, number(i, n, tl(b))) \rhd \delta) \mathbin{⌊\!⌊} \\
& \quad s_1(\langle hd(b), i +_n size(tl(b))\rangle)\ INS(n, i +_n size(b)))) \\
& \lhd 0 < size(b) \leq n + 1 \text{ and } i < n \rhd \delta \\
\\
\overset{TI2,\ A.6.7}{=} & \tau\ CQ_0(n, i, b) \lhd 0 < size(b) \leq n \text{ and } i < n \rhd \delta \\
+ & \tau_{\{c_1,c_2\}}(\partial_{\{r_1,s_1,r_2,s_2\}}(\\
& \quad \sum_{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 \lhd test(\langle d, i\rangle, number(i, n, tl(b))) \rhd \delta) \mathbin{⌊\!⌊} \\
& \quad s_1(\langle hd(b), i +_n size(tl(b))\rangle)\ INS(n, i +_n size(b)))) \\
& \lhd 0 < size(b) \leq n + 1 \text{ and } i < n \rhd \delta \\
\\
\overset{A.6.2,\ Sum\ El.,\ TI2}{=} & \\
& \tau\ CQ_0(n, i, b) \lhd 0 < size(b) \leq n \text{ and } i < n \rhd \delta \\
+ & \tau \cdot \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, rem(\langle toe(b), i\rangle, number(i, n, tl(b)))) \parallel \\
& \quad\quad out(toe(b))\ OUTS(n, i +_n 1))) \\
& \lhd 1 < size(b) \leq n + 1 \text{ and } i < n \rhd \delta \\
\\
\overset{A.6.8}{=} & \tau\ CQ_0(n, i, b) \lhd 0 < size(b) \leq n \text{ and } i < n \rhd \delta
\end{array}$$

$$+ \quad \tau\, CQ_3(n,i,b) \triangleleft 1 < size(b) \leq n+1 \text{ and } i < n \triangleright \delta$$

That $CQ_j(n,i,b)$ even satisfy the equation for G_3 can be shown in a similar way as above. These calculations are omitted here but can be found in [GK92].

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.

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

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

$$\begin{array}{ll}
\multicolumn{2}{l}{Q_1(n,i,b)} \\
= & Q(n+2,b) \triangleleft 0 < size(b) \leq n+1 \text{ and } i < n \triangleright \delta \\
= & \sum_{d:D}(enter(d)\, Q(n+2,in(d,b))) \\
 & \qquad\qquad\qquad \triangleleft 0 < size(b) \leq n+1 \text{ and } i < n \triangleright \delta \\
+ & out(toe(b))\, Q(n+2,untoe(b)) \triangleleft 0 < size(b) \leq n \text{ and } i < n \triangleright \delta \\
\overset{\text{T1}}{=} & \sum_{d:D}(enter(d)\, Q_3(n,i,in(d,b))) \\
 & \qquad\qquad\qquad \triangleleft 0 < size(b) \leq n+1 \text{ and } i < n \triangleright \delta \\
+ & out(toe(b))\, Q_0(n, i +_n 1, untoe(b)) \\
 & \qquad\qquad\qquad \triangleleft 0 < size(b) \leq n+1 \text{ and } i < n \triangleright \delta
\end{array}$$

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

$$\begin{array}{ll}
\multicolumn{2}{l}{Q_2(n,i,b)} \\
= & \tau\, Q(n+2,b) \triangleleft 0 < size(b) \leq n+1 \text{ and } i < n \triangleright \delta \\
= & \tau\, Q(n+2,b) \triangleleft 0 < size(b) \leq n \text{ and } i < n \triangleright \delta \\
+ & \tau\, Q(n+2,b) \triangleleft 1 < size(b) \leq n+1 \text{ and } i < n \triangleright \delta \\
\overset{\text{T1}}{=} & \tau\, Q_0(n,i,b) \triangleleft 0 < size(b) \leq n \text{ and } i < n \triangleright \delta \\
+ & \tau\, Q_3(n,i,b) \triangleleft 1 < size(b) \leq n+1 \text{ and } i < n \triangleright \delta
\end{array}$$

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

$$\begin{array}{ll}
\multicolumn{2}{l}{Q_3(n,i,b)} \\
= & (Q(n+2,b) \triangleleft eq(size(b),n+2) \triangleright \tau\, Q(n+2,b)) \\
 & \triangleleft 1 < size(b) \leq n+2 \text{ and } i < n \triangleright \delta \\
\overset{\text{T2}}{=} & (out(toe(b))\, Q(n+2,untoe(b)) \triangleleft eq(size(b),n+2) \triangleright \\
 & \qquad (\tau\, Q(n+2,b) + out(toe(b))\, Q(n+2,untoe(b)))) \\
 & \triangleleft 1 < size(b) \leq n+2 \text{ and } i < n \triangleright \delta
\end{array}$$

$$
\begin{array}{ll}
\overset{4.2.4,\,\mathrm{A}.3.5,\,\mathrm{A}.1.1}{=} & \\
 & \tau\, Q(n+2,b) \lhd 1 < size(b) < n+2 \text{ and } i < n \rhd \delta \\
+ & out(toe(b))\, Q(n+2, untoe(b)) \\
 & \qquad\qquad \lhd 1 < size(b) \leq n+2 \text{ and } i < n \rhd \delta \\
\overset{\mathrm{T1}}{=} & \tau\, Q_1(n,i,b) \lhd 1 < size(b) < n+2 \text{ and } i < n \rhd \delta \\
+ & out(toe(b))\, Q_2(n, i +_n 1, untoe(b)) \\
 & \qquad\qquad \lhd 1 < size(b) \leq n+2 \text{ and } i < n \rhd \delta
\end{array}
$$

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

$$Q(n+2) \overset{5.2.2}{=} Q_0(n,0,\varnothing_d) \overset{5.2.3}{=} CQ_0(n,0,\varnothing_d) \overset{5.2.1}{=} B(n).$$

□

Acknowledgements

We would like to thank Jos Baeten, Jan Bergstra, Wan Fokkink, Gert-Jan Kamsteeg, Alban Ponse and Frits Vaandrager for their comments on earlier drafts of this paper. In particular, we are very grateful to Jos van Wamel for spotting dozens of bugs in, and helpful comments on our verification of the Bakery Protocol.

A Elementary data types

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

A.1 About booleans

The predefined booleans extended with their well-known connectives *not, and, or*[2] are often used in the Bakery proof. The (rewrite) rules for the added connectives are consistent with the predefined rules BOOL1 and BOOL2.

$$
\begin{array}{ll}
\textbf{sort} & \textbf{Bool} \\
\textbf{func} & \mathsf{t}, \mathsf{f} : \rightarrow \textbf{Bool} \\
 & \mathrm{not} : \textbf{Bool} \rightarrow \textbf{Bool} \\
 & \mathrm{and} : \textbf{Bool} \times \textbf{Bool} \rightarrow \textbf{Bool} \\
 & \mathrm{or} : \textbf{Bool} \times \textbf{Bool} \rightarrow \textbf{Bool} \\
\textbf{var} & b, b_1, b_2, b_3 : \textbf{Bool} \\
\textbf{rew} & \mathrm{not}(\mathsf{t}) = \mathsf{f} \\
 & \mathrm{not}(\mathsf{f}) = \mathsf{t} \\
 & \mathsf{t} \text{ and } b = b \\
 & \mathsf{f} \text{ and } b = \mathsf{f} \\
 & \mathsf{t} \text{ or } b = \mathsf{t} \\
 & \mathsf{f} \text{ or } b = b
\end{array}
$$

Lemma A.1.

1. $p \lhd b \rhd q = q \lhd \mathrm{not}(b) \rhd p,$

[2]The symbols $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ are reserved in the proof system of μCRL as operators for connecting properties.

2. $p \lhd b_1$ or $b_2 \rhd \delta = p \lhd b_1 \rhd \delta + p \lhd b_2 \rhd \delta$,

3. b_1 and $b_1 = b_1$,

4. b_1 or $b_1 = b_1$,

5. b_1 and $b_2 = b_2$ and b_1.

Proof. By COND1, COND2 and Lemma 4.1. □

A.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), $\geq, \leq, <, >, \mathit{if}, \mathit{eq}$ used in this paper are specified. (We will use infix notation wherever we find it convenient to do so.)

sort nat
func $0 :\rightarrow \mathit{nat}$
$S, P : \mathit{nat} \rightarrow \mathit{nat}$
$+, -, : \mathit{nat} \times \mathit{nat} \rightarrow \mathit{nat}$
$\mathit{eq}, \geq, \leq, <, >: \mathit{nat} \times \mathit{nat} \rightarrow \mathbf{Bool}$
$\mathit{if} : \mathbf{Bool} \times \mathit{nat} \times \mathit{nat} \rightarrow \mathit{nat}$
var $n, m, z : \mathit{nat}$
rew $P(0) = 0$
$P(S(n)) = n$
$n + 0 = n$
$n + S(m) = S(n + m)$
$n - 0 = n$
$n - S(m) = P(n - m)$
$\mathit{eq}(n, n) = \mathsf{t}$
$\mathit{if}(\mathit{eq}(n, m), n, m) = m$
$n \geq 0 = \mathsf{t}$
$0 \geq S(n) = \mathsf{f}$
$S(n) \geq S(m) = n \geq m$
$n \leq m = m \geq n$
$n > m = n \geq S(m)$
$n < m = S(n) \leq m$
$\mathit{if}(\mathsf{t}, n, m) = n$
$\mathit{if}(\mathsf{f}, n, m) = m$

The *if* function given above will be used for specifying modulo arithmetic in section A.3.

Notation A.2. We write $n \leq m$ for $n \leq m = \mathsf{t}$. Idem for $\geq$,$>$ and $<$. We write $\mathit{eq}(n, m)$ for $\mathit{eq}(n, m) = \mathsf{t}$. We write 1 for $S(0)$ and 2 for $S(S(0))$.

Lemma A.3.

1. $\mathit{eq}(n, m) = \mathsf{t} \leftrightarrow n = m$,

2. $n = m \vee \neg(n = m)$,

3. $n + (m + z) = (n + m) + z$,

4. $n + m = m + n$,

5. $n \geq m = \mathit{not}(n < m)$,

6. *etc.*

Proof. By (nested) induction on *nat* (see [TD88]). □

A.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.

$$
\begin{array}{ll}
\textbf{func} & \text{mod} : nat \times nat \to nat \\
 & + : nat \times nat \times nat \to nat \\
\textbf{var} & n, n', m : nat \\
\textbf{rew} & n \text{ mod } 0 = n \\
 & n \text{ mod } m = \mathit{if}(n \geq m, (n-m) \text{ mod } m, n) \\
 & n +_m n' = (n + n') \text{ mod } m
\end{array}
$$

Lemma A.4.

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. $n_2 > 0 \to (n_1 \text{ mod } n_2) < n_2$,
4. $(n_1 \text{ mod } n_2) \text{ mod } n_2 = n_1 \text{ mod } n_2$.

A.4 About data queues

In this section, we specify the data type $queue_d$ 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 $eq : D \times D \to \textbf{Bool}$ that has the desired property $eq(d, e) = \mathsf{t} \leftrightarrow d = e$.

$$
\begin{array}{ll}
\textbf{sort} & D \\
\textbf{func} & d_\perp : D \\
 & d_1, \ldots, d_n : D \\
 & eq : D \times D \to \textbf{Bool}
\end{array}
$$

The specification of $queue_d$ is given below.

$$
\begin{array}{ll}
\textbf{sort} & queue_d \\
\textbf{func} & \varnothing_d : \to queue_d \\
 & in, rem : D \times queue_d \to queue_d \\
 & test : D \times queue_d \to \textbf{Bool} \\
 & size : queue_d \to nat \\
 & hd, toe : queue_d \to D \\
 & tl, untoe : queue_d \to queue_d \\
 & \mathit{if} : \textbf{Bool} \times queue_d \times queue_d \to queue_d \\
 & empty : queue_d \to \textbf{Bool} \\
\textbf{var} & d, e : D \\
 & b, c : queue_d \\
\textbf{rew} & test(d, \varnothing_d) = \mathsf{f} \\
 & test(d, in(e, b)) = \mathit{if}(eq(d, e), \mathsf{t}, test(d, b)) \\
 & rem(d, \varnothing_d) = \varnothing_d \\
 & rem(d, in(e, b)) = \mathit{if}(eq(d, e), b, in(e, rem(d, b))) \\
 & size(\varnothing_d) = 0 \\
 & size(in(d, b)) = S(size(b)) \\
 & hd(\varnothing_d) = d_\perp \\
 & hd(in(d, b)) = d
\end{array}
$$

$$\begin{aligned}
&toe(\varnothing_d) = d_\perp\\
&toe(in(d,\varnothing_d)) = d\\
&toe(in(d,in(e,b))) = toe(in(e,b))\\
&tl(\varnothing_d) = \varnothing_d\\
&tl(in(d,b)) = b\\
&untoe(\varnothing_d) = \varnothing_d\\
&untoe(in(d,\varnothing_d)) = \varnothing_d\\
&untoe(in(d,in(e,b))) = in(d,untoe(in(e,b)))\\
&empty(b) = eq(size(b),0)
\end{aligned}$$

Lemma A.5.

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

The function *number* is specified as follows.

$$\begin{array}{ll}
\textbf{sort} & queue_p\\
\textbf{func} & \varnothing_p :\to queue_p\\
 & number : nat \times nat \times queue_d \to queue_p\\
\textbf{rew} & number(i,n,\varnothing_d) = \varnothing_p\\
 & number(i,n,in(d,b)) = in(\langle d, i +_n size(b)\rangle, number(i,n,b))
\end{array}$$

Here $queue_p$ is exactly the same data type as $queue_d$ except that it contains elements of sort $Pair$ instead of sort D.

Lemma A.6.

1. $size(b) \leq n \to$
   $test(\langle d,i\rangle, number(i,n,b)) = (eq(d,toe(b))$ and $size(b) > 0)$,
2. $size(b) \leq n \to$
   $test(\langle d,i\rangle, number(i,n,b)) = (size(b) \geq 1$ and $eq(d,toe(b)))$,
3. $size(b) \leq n \to$
   $rem(\langle toe(b),i\rangle, number(i,n,b)) = number(i +_n 1, n, untoe(b))$,
4. $size(b) > 0 \to size(tl(b)) < n = size(b) \leq n$,
5. $size(b) > 0 \to$
   $size(number(i +_n 1, n, tl(untoe(b)))) < n = size(b) < n + 2$,
6. $size(number(i,n,b)) = size(b)$,
7. $size(b) \leq n \to$
   $in(\langle hd(b), i +_n size(tl(b))\rangle, number(i,n,tl(b))) = number(i,n,b)$,
8. $1 < size(b) \leq n \to$
   $rm(\langle toe(b),i\rangle, number(i,n,tl(b))) = number(i +_n 1, n, tl(untoe(b)))$.

A1	$x + y = y + x$	CF	$n(\overline{t}) \mid m(\overline{t}) = \begin{cases} \gamma(n,m)(\overline{t}) & \text{if } \gamma(n,m)\downarrow \\ \delta & \text{otherwise} \end{cases}$
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	$x \parallel y = x \mathbin{⌊\!⌊} y + y \mathbin{⌊\!⌊} x + x \mid y$	CT1	$\tau \mid x = \delta$
CM2	$a \mathbin{⌊\!⌊} x = a \cdot x$	CT2	$x \mid \tau = \delta$
CM3	$a \cdot x \mathbin{⌊\!⌊} y = a \cdot (x \parallel y)$		
CM4	$(x + y) \mathbin{⌊\!⌊} z = x \mathbin{⌊\!⌊} z + y \mathbin{⌊\!⌊} z$	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	$\partial_H(n(\overline{t})) = n(\overline{t}) \quad \text{if } n \notin H$
CM7	$a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$	D2	$\partial_H(n(\overline{t})) = \delta \quad \text{if } n \in H$
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 1: The axioms of ACP in μCRL.

B An overview of the proof theory for μCRL

B.1 The proof system

In [GP94a] a proof system has been given which allows to prove identities about processes with data. Table 1 lists the axioms of ACP in μCRL, followed by the axioms of Standard Concurrency in Table 2 and the axioms for hiding in Table 3. For an explanation of these axioms we refer to [BW90], except for the following points. In the tables x, y are process variables and p, q are process terms in which the variable d may occur. The letters $t_1, \ldots, t_n$ 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 $\tilde{\gamma}$. We write $\gamma(n, m) \downarrow$ if γ is defined on n and m.

Tables 4, 5 and 6 lists the typical μCRL axioms and rules for interaction between data and processes. The axioms for summation are denoted by SUM, the axioms for the conditional by COND and the rules for the booleans by BOOL.

Beside the axioms and rules mentioned above, μCRL incorporates two other important proof principles. First, it supports an principle for induction not only on data but also on data in processes. The second principle is RSP (Recursive Specification Principle) taken from [BW90] extended to processes with data. Informally, it says that each guarded recursive specification has at most one solution.

$(x \mathbin{⌊\!⌊} y) \mathbin{⌊\!⌊} z = x \mathbin{⌊\!⌊} (y \parallel z)$	$(x \mid y) \mid z = x \mid (y \mid z)$
$x \parallel \delta = x\delta$	$x \mid (y \mathbin{⌊\!⌊} z) = (x \mid y) \mathbin{⌊\!⌊} z$
$x \mid y = y \mid x$	$x \mid (y \mid z) = \delta$ Handshaking

Table 2: Axioms of Standard Concurrency (SC).

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 3: Axioms for abstraction.

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

Table 4: Axioms for summation.

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

Table 5: Axioms for the conditional construct.

BOOL1	$\neg(\mathsf{t} = \mathsf{f})$
BOOL2	$\neg(b = \mathsf{t}) \rightarrow b = \mathsf{f}$

Table 6: Axioms for **Bool**.

B.2 Adding τ-laws to the proof system

The proof system as presented above 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. One can add the τ-laws of Table 7 taken from MILNER [Mil89] to the proof system. These axioms correspond to the well-known *observation equivalence.* These axioms can be

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

Table 7: τ-laws for observation equivalence.

added to the proof system under the restriction that the a and b in the ACP axioms of μCRL do not range over τ. Otherwise, we are able to derive inconsistent identities (see [BW90], page 165). The axioms in Table 8 model the interaction between τ and the other operators (see [BW90]).

TM1	$\tau \mathbin{⌊\!⌊} x$	$=$	τx
TM2	$\tau x \mathbin{⌊\!⌊} y$	$=$	$\tau(x \parallel y)$
TC3	$\tau x \mid y$	$=$	$x \mid y$
TC4	$x \mid \tau y$	$=$	$x \mid y$

Table 8: Completing τ-laws for observation equivalence.

Another point is that the axiom $(x \mid y) \mathbin{⌊\!⌊} z = x \mid (y \mathbin{⌊\!⌊} z)$ of standard concurrency is not consistent in the context of the τ-laws given in Table 7. It must be replaced by the weaker axiom $(x \mid ay) \mathbin{⌊\!⌊} z = x \mid (ay \mathbin{⌊\!⌊} z)$.

The RSP rule mentioned above is restricted to guarded systems of process-equations. A definition for guardedness that works in a setting of Milner's observation equivalence can be given as follows:

Definition B.1 (*Guardedness of* G)**.** A term p is a *guard* iff:

- $p \equiv \delta$,
- $p \equiv n(t_1, \ldots, t_n)$ or $p \equiv n$ and n is an action label,
- $p \equiv q_1 \circ q_2$ with $\circ \in \{+, \lhd t \rhd\}$ and q_1 and q_2 are guards,
- $p \equiv q_1 \circ q_2$ with $\circ \in \{\cdot, \parallel, \mathbin{⌊\!⌊}\}$ and q_1 or q_2 are guards,
- $p \equiv q_1 \mid q_2$,
- $p \equiv \sum_{x:S}(q_1)$ and q_1 is a guard,
- $p \equiv \partial_{nl}(q_1)$ with 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 $\circ \in \{+, \parallel, \mid, \lhd t \rhd\}$, and N is guarded in q_1 and q_2,
- $r \equiv q_1 \circ q_2$ with $\circ \in \{\cdot, \mathbin{⌊\!⌊}\}$ and N is guarded in q_1, and q_1 is a *guard* or N is guarded in q_2,
- $r \equiv \sum_{x:S}(q_1)$ and N is guarded in q_1,

- $r \equiv \partial_{nl}(q_1)$ with nl being a list of names, 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 left-hand side of the equations of G is a node,
- if N is a node of $IDG(G)$ and $N = r \in G$, then there is an edge $N \rightarrow N'$ for any node N' that appears unguarded in r.

We call G *guarded* iff

- $IDG(G)$ is well founded, i.e. does not contain an infinite path, and
- none of the right-hand sides of G has a subterm of the form $\tau_{nl}(q)$, where nl is a list of names.

□

References

[SoSL94] 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, Springer Verlag, 1994.

[Apt81] 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.

[Apt84] K.R. Apt. Ten years of Hoare's logic, a survey, part II: Nondeterminism. *Theoretical Computer Science*, 28:83–109, 1984.

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

[BG93a] M.A. Bezem and J.F. Groote. A correctness proof of a one bit sliding window protocol in μCRL. Technical Report 99, Logic Group Preprint Series, Utrecht University, 1993. To appear in the Computer Journal, Volume 37(4), 1994.

[BG93b] M.A. Bezem and J.F. Groote. A formal verification of the alternating bit protocol in the calculus of constructions. Technical Report 88, Logic Group Preprint Series, Utrecht University, March 1993.

[CM88] K.M. Chandy and J. Misra. *Parallel Program Design. A Foundation.* Addison-Wesley, 1988.

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

[GK92] J.F. Groote and H. Korver. A correctness proof of the bakery protocol in μCRL. Logic Group Preprint Series 80, Dept. of Philosophy, Utrecht University, October 1992.

[GK95] J.F. Groote and H.P. Korver. A correctness proof of the bakery protocol in μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes,* Utrecht, 1994, Workshops in Computing, pages 63–86. Springer-Verlag, 1995.

[GP93] J.F. Groote and J.C. van de Pol. A bounded retransmission protocol for large data packets. A case study in computer checked verification. Technical Report 100, Logic Group Preprint Series, Utrecht University, 1993.

[GP94a] J.F. Groote and A. Ponse. Proof theory for μCRL: a language for processes with data. In D.J. Andrews, e.a. [SoSL94], pages 232–251. Full version is available as CWI Report CS-R9138, Amsterdam, The Netherlands, August 1991.

[GP94b] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In this volume.

[GW89] 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. Full version available as Report CS-R9120, CWI, Amsterdam, 1991.

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

[KS94] H. Korver and J. Springintveld. A computer-checked verification of Milner's scheduler. In M. Hagiya and J.C. Mitchel, editors. *Proceedings of the 2^{nd} International Symposium on Theoretical Aspects of Computer Software, TACS '94,* Sendai, Japan, pages 161–178, volume 789 of *Lecture Notes in Computer Science.* Springer-Verlag, 1994.

[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 Andrews et al. [SoSL94], pages 315–339.

[TD88] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics, An Introduction (vol I).* North-Holland, 1988.