Verifying a Sliding Window Protocol in μCRL

Wan Fokkink[1,2], Jan Friso Groote[1,3], Jun Pang[1],
Bahareh Badban[1], and Jaco van de Pol[1]

[1] CWI, Embedded Systems Group
[2] Vrije Universiteit Amsterdam, Theoretical Computer Science Group
[3] Eindhoven University of Technology, Systems Engineering Group
{wan,pangjun,badban,vdpol}@cwi.nl, J.F.Groote@tue.nl

Abstract. We prove the correctness of a sliding window protocol with an arbitrary finite window size n and sequence numbers modulo $2n$. We show that the sliding window protocol is branching bisimilar to a queue of capacity $2n$. The proof is given entirely on the basis of an axiomatic theory, and was checked with the help of PVS.

1 Introduction

Sliding window protocols [6] (SWPs) ensure successful transmission of messages from a sender to a receiver through a medium, in which messages may get lost. Their main characteristic is that the sender does not wait for an incoming acknowledgement before sending next messages, for optimal use of bandwidth. This is the reason why many data communication systems include the SWP, in one of its many variations.

In SWPs, both the sender and the receiver maintain a buffer. In practice the buffer at the receiver is often much smaller than at the sender, but here we make the simplifying assumption that both buffers can contain up to n messages. By providing the messages with sequence numbers, reliable in-order delivery without duplications is guaranteed. The sequence numbers can be taken modulo $2n$ (and not less, see [42] for a nice argument). The messages at the sender are numbered from i to $i+n$ (modulo $2n$); this is called a *window*. When an acknowledgement reaches the sender, indicating that k messages have arrived correctly, the window *slides* forward, so that the sending buffer can contain messages with sequence numbers $i+k$ to $i+k+n$ (modulo $2n$). The window of the receiver slides forward when the first element in this window is passed on to the environment.

Within the process algebraic community, SWPs have attracted much attention. We provide a comparison with verifications of SWPs in Section 8, and restrict here to the context in which the current paper was written. After the advent of process algebra in the early 80's of last century, it was observed that simple protocols, such as the alternating bit protocol, could readily be verified. In an attempt to show that more difficult protocols could also be dealt with, SWPs were considered. Middeldorp [31] and Brunekreef [4] gave specifications in ACP [1] and PSF [30], respectively. Vaandrager [43], Groenveld [12], van Wamel [44] and Bezem and Groote [3] manually verified one-bit SWPs, in which the windows have size one. Starting in 1990, we attempted to prove the most complex SWP from [42] (not taking into account additional features such as duplex message passing and piggybacking) correct using the process algebraic language μCRL [16]. This turned out to be unexpectedly hard, which is shown by the 13 year it took to finish the current paper, and led to significant developments in the realm of process algebraic proof techniques for protocol verification. We therefore consider the current paper as a true milestone in process algebraic verification.

Our first observation was that the external behaviour of the protocol, as given in [42], was unclear. We adapted the SWP such that it nicely behaves as a queue of capacity $2n$. The second observation was that the SWP of [42] contained a deadlock [13, Stelling 7], which could only occur after at least n messages were transmitted. This error was communicated to Tanenbaum, and has been repaired in more recent editions of [42]. Another bug in the μCRL specification of the SWP was detected by means of a model checking analysis. A first attempt to prove the resulting SWP correct led to the verification of a bakery protocol [14], and to the development of the *cones and foci* proof method [19, 9]. This method rephrases the question whether two system specifications are branching bisimilar in terms of proof obligations on relations between data objects. It plays an essential role in the proof in the current paper, and has been used to prove many other protocols and distributed algorithms correct. But the correctness proof required an additional idea, already put forward by Schoone [37], to first perform the proof with unbounded sequence numbers, and to separately eliminate modulo arithmetic.

We present a specification in μCRL of a SWP with buffer size $2n$ and window size n, for arbitrary n. The medium between the sender and the receiver is modelled as a lossy queue of capacity one. We manually prove that the external behaviour of this protocol is branching bisimilar [10] to a FIFO queue of capacity $2n$. This proof is entirely based on the axiomatic theory underlying μCRL and the axioms characterising the data types. It implies both safety and liveness of the protocol (the latter under the assumption of fairness). First, we linearise the specification, meaning that we get rid of parallel operators. Moreover, communication actions are stripped from their data parameters. Then we eliminate modulo arithmetic, using the proof principle CL-RSP [2], which states that each linear specification has a unique solution (modulo branching bisimulation). Finally, we apply the cones and foci technique, to prove that the linear specification without modulo arithmetic is branching bisimilar to a FIFO queue of capacity $2n$. All lemmas for the data types, all invariants and all correctness proofs have been checked using PVS. The PVS files are available via `http://www.cwi.nl/~badban/swp.html`. Ongoing research is to extend the current verification to a setting where the medium is modelled as a lossy queue of unbounded capacity, and to include duplex message passing and piggybacking.

In this extended abstract we omitted most equational definitions of the data types, most lemmas regarding these data types, part of the invariants and part of the correctness proofs. The reader is referred to the full version of the paper [8], for these definitions and proofs.

2 μCRL

μCRL [16] (see also [18]) is a language for specifying distributed systems and protocols in an algebraic style. It is based on the process algebra ACP [1] extended with equational abstract data types [28]. In a μCRL specification, one part specifies the data types, while a second part specifies the process behaviour.

The data types needed for our μCRL specification of a SWP are presented in Section 3. In this section we focus on the process part of μCRL. Processes are represented by process terms, which describe the order in which the actions from a set $\mathcal{A}$ may happen. A process term consists of action names and recursion variables combined by process algebraic operators. Actions and recursion variables may carry data parameters. There are two predefined actions outside $\mathcal{A}$: δ represents deadlock, and τ a hidden action. These two actions never carry data parameters. $p{\cdot}q$ denotes sequential composition and $p+q$ non-deterministic choice. Summation $\sum_{d:D} p(d)$ provides the possibly infinite choice over a data type D, and the conditional construct $p \lhd b \rhd q$ with b a data term of sort *Bool* behaves as p if b and as q if $\neg b$. Parallel composition $p \parallel q$ interleaves the actions of p and q; moreover, actions from p and q may also synchronise to a communication action, when this is explicitly allowed by a predefined communication function. Two actions can only synchronise if their data parameters are equal. Encapsulation $\partial_{\mathcal{H}}(p)$, which renames all occurrences in p of actions from the set $\mathcal{H}$ into δ, can be used to force actions into communication. Hiding $\tau_{\mathcal{I}}(p)$ renames all occurrences in p of actions from the set $\mathcal{I}$ into τ. Finally, processes can be specified by means of recursive equations $X(d_1{:}D_1, \ldots, d_n{:}D_n) \approx p$, where X is a recursion variable, d_i a data parameter of type D_i for $i = 1, \ldots, n$, and p a process term (possibly containing recursion variables and the parameters d_i). A recursive specification is linear if it is of the form

$$X(d_1{:}D_1, \ldots, d_n{:}D_n) \approx \sum_{i=1}^{\ell} \sum_{z_i:Z_i} a_i(e_1^i, \ldots, e_{m_i}^i){\cdot}X(d_1^i, \ldots, d_n^i) \lhd b_i \rhd \delta.$$

To each μCRL specification belongs a directed graph, called a labelled transition system, which is defined by the structural operational semantics of μCRL (see [16]). In this labelled transition system, the states are process terms, and the edges are labelled with parameterised actions. Branching bisimulation $\underline{\leftrightarrow}_b$ [10] and strong bisimulation $\underline{\leftrightarrow}$ [33] are two well-established equivalence relations on the states in labelled transition systems. Conveniently, strong bisimulation equivalence implies branching bisimulation equivalence. The proof theory of μCRL from [15] is sound modulo branching bisimulation equivalence, meaning that if $p \approx q$ can be derived from it then $p \underline{\leftrightarrow}_b q$.

The goal of this paper is to prove that the initial state of the forthcoming μCRL specification of a SWP is branching bisimilar to a FIFO queue. We use three proof principles from the literature:

- *Sum elimination* [14] states that a summation over a data type from which only one element can be selected can be removed.
- *CL-RSP* [2] states that the solutions of a linear μCRL specification that does not contain any infinite τ sequence are all strongly bisimilar.

- The *cones and foci* method from [9, 19] rephrases the question whether two linear μCRL specifications $\tau_{\mathcal{I}}(S_1)$ and S_2 are branching bisimilar, where S_2 does not contain actions from some set $\mathcal{I}$ of internal actions, in terms of data equalities. A *state mapping* ϕ relates each state in S_1 to a state in S_2. Furthermore, some states in S_1 are declared to be *focus points*, by means of a predicate *FC*. The *cone* of a focus point consists of the states in S_1 that can reach this focus point by a string of actions from $\mathcal{I}$. It is required that each reachable state in S_1 is in the cone of a focus point. If a number of *matching criteria* are satisfied, then $\tau_{\mathcal{I}}(S_1)$ and S_2 are branching bisimilar.

3 Data Types

In this section, the data types used in the μCRL specification of the SWP are presented: booleans, natural numbers supplied with modulo arithmetic, and buffers.

Booleans and Natural Numbers. Bool is the data type of booleans. t and f denote true and false, $\wedge$ and $\vee$ conjunction and disjunction, $\rightarrow$ and $\leftrightarrow$ implication and logic equivalence, and $\neg$ negation. For a boolean b, we abbreviate $b = \mathsf{t}$ to b and $b = \mathsf{f}$ to $\neg b$. Unless otherwise stated, data parameters in boolean formulas are universally quantified.

For each data type D in this paper there is an operation $\mathit{if} : \mathit{Bool} \times D \times D \rightarrow D$ with as defining equations $\mathit{if}(\mathsf{t}, d, e) = d$ and $\mathit{if}(\mathsf{f}, d, e) = e$. Furthermore, for each data type D in this paper one can easily define a mapping $\mathit{eq} : D \times D \rightarrow \mathit{Bool}$ such that $\mathit{eq}(d, e)$ holds if and only if $d = e$ can be derived. For notational convenience we take the liberty to write $d = e$ instead of $\mathit{eq}(d, e)$.

Nat is the data type of natural numbers. 0 denotes zero, $S(n)$ the successor of n, $+$, $\dot{-}$ and $\cdot$ addition, monus (also called proper subtraction) and multiplication, and $\leq$, $<$, $\geq$ and $>$ less-than(-or-equal) and greater-than(-or-equal). Usually, the sign for multiplication is omitted, and $\neg(i = j)$ is abbreviated to $i \neq j$. As binding convention, $\{=, \neq\} > \{\cdot\} > \{+, \dot{-}\} > \{\leq, <, \geq, >\} > \{\neg\} > \{\wedge, \vee\} > \{\rightarrow, \leftrightarrow\}$.

Since the buffers at the sender and the receiver in the sliding window are of size $2n$, calculations modulo $2n$ play an important role. $i|_n$ denotes i modulo n, while $i \;\mathit{div}\; n$ denotes i integer divided by n.

Buffers. The sender and the receiver in the SWP both maintain a buffer containing the sending and the receiving window, respectively (outside these windows both buffers are empty). Let Δ be the set of data elements that can be communicated between sender and receiver. The buffers are modelled as a list of pairs (d, i) with $d \in \Delta$ and $i \in \mathit{Nat}$, representing that position (or sequence number) i of the buffer is occupied by datum d. The data type *Buf* is specified as follows, where $[]$ denotes the empty buffer: $[] :\rightarrow \mathit{Buf}$ and $\mathit{in} : \Delta \times \mathit{Nat} \times \mathit{Buf} \rightarrow \mathit{Buf}$. $q|_n$ denotes buffer q with all sequence numbers taken modulo n. $[]|_n = []$ and $\mathit{in}(d, i, q)|_n = \mathit{in}(d, i|_n, q|_n)$. $\mathit{test}(i, q)$ produces t if and only if position i in q is occupied, $\mathit{retrieve}(i, q)$ produces the datum that resides at position i in buffer q (if this position is occupied), and $\mathit{remove}(i, q)$ is obtained by emptying position

i in buffer q. $\mathit{release}(i,j,q)$ is obtained by emptying positions i up to j excluded in q. $\mathit{release}|_n(i,j,q)$ does the same modulo n:

$$\begin{aligned}\mathit{release}(i,j,q) &= \mathit{if}\,(i \geq j, q, \mathit{release}(S(i), j, \mathit{remove}(i,q)))\\ \mathit{release}|_n(i,j,q) &= \mathit{if}\,(i|_n{=}j|_n, q, \mathit{release}|_n(S(i), j, \mathit{remove}(i,q)))\end{aligned}$$

$\mathit{next\text{-}empty}(i,q)$ produces the first empty position in q, counting upwards from sequence number i onward. $\mathit{next\text{-}empty}|_n(i,q)$ does the same modulo n.

$$\begin{aligned}\mathit{next\text{-}empty}(i,q) &= \mathit{if}\,(\mathit{test}(i,q), \mathit{next\text{-}empty}(S(i),q), i)\\ \mathit{next\text{-}empty}|_n(i,q) &= \begin{cases}\mathit{next\text{-}empty}(i|_n, q|_n) & \text{if } \mathit{next\text{-}empty}(i|_n, q|_n) < n\\ \mathit{next\text{-}empty}(0, q|_n) & \text{otherwise}\end{cases}\end{aligned}$$

Intuitively, $\mathit{in\text{-}window}(i,j,k)$ produces $\mathtt{t}$ if and only if j lies in the range from i to $k \dot{-} 1$, modulo n, where n is greater than i, j and k.

$$\mathit{in\text{-}window}(i,j,k) = i \leq j < k \vee k < i \leq j \vee j < k < i$$

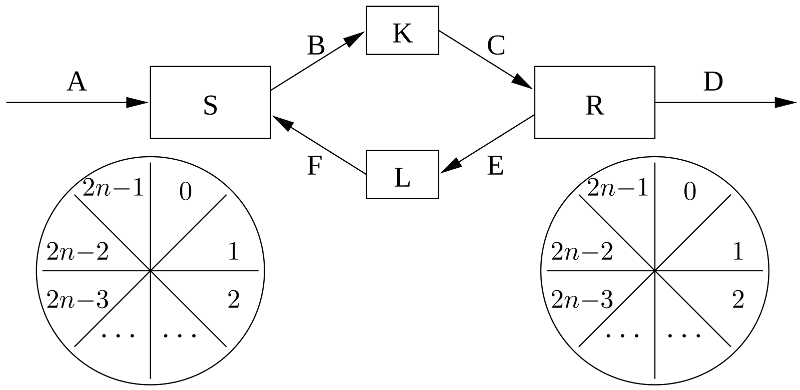
Lists. The data type *List* of lists is used in the specification of the desired external behaviour of the SWP: a FIFO queue of capacity $2n$. It is specified by the empty list $\langle\rangle :\to \mathit{List}$ and $\mathit{in} : \mathit{\Delta} \times \mathit{List} \to \mathit{List}$. $\mathit{length}(\lambda)$ denotes the length of λ, $\mathit{top}(\lambda)$ produces the datum at the top of λ, $\mathit{tail}(\lambda)$ is obtained by removing the top position in λ, $\mathit{append}(d,\lambda)$ adds datum d at the end of λ, and $\lambda {+\!\!+} \lambda'$ represents list concatenation. Furthermore, $q[i..j\rangle$ is the list containing the elements in buffer q at positions i up to but not including j. An empty position in q, in between i and j, gives rise to an occurrence of the default datum d_0 in $q[i..j\rangle$.

$$q[i..j\rangle = \begin{cases}\langle\rangle & \text{if } i \geq j\\ \mathit{in}(\mathit{retrieve}(i,q), q[S(i)..j\rangle) & \text{if } i < j \wedge \mathit{test}(i,q)\\ \mathit{in}(d_0, q[S(i)..j\rangle) & \text{if } i < j \wedge \neg\mathit{test}(i,q)\end{cases}$$

4 Sliding Window Protocol

In this section, a μCRL specification of a SWP is presented, together with its desired external behaviour.

Specification of a Sliding Window Protocol. A sender **S** stores data elements that it receives via channel A in a buffer of size $2n$, in the order in which they are received. **S** can send a datum, together with its sequence number in the buffer, to a receiver **R** via a medium that behaves as lossy queue of capacity one, represented by the medium **K** and the channels B and C. Upon reception, **R** may store the datum in its buffer, where its position in the buffer is dictated by the attached sequence number. In order to avoid a possible overlap between the sequence numbers of different data elements in the buffers of **S** and **R**, no more than one half of the buffers of **S** and **R** may be occupied at any time; these halves are called the sending and the receiving window, respectively. **R** can pass on a datum that resides at the first position in its window via channel D; in that case the receiving window slides forward by one position. Furthermore, **R** can send the sequence number of the first empty position in (or just outside) its window as an acknowledgement to **S** via a medium that behaves as lossy queue of capacity one, represented by the medium **L** and the channels E and F. If **S** receives this acknowledgement, its window slides accordingly.



The sender **S** is modelled by the process $\mathbf{S}(\ell, m, q)$, where q is a buffer of size $2n$, ℓ the first position in the sending window, and m the first empty position in (or just outside) the sending window. Data elements can be selected at random for transmission from (the filled part of) the sending window.

$$\begin{aligned}
&\mathbf{S}(\ell{:}Nat, m{:}Nat, q{:}Buf)\\
&\approx \textstyle\sum_{d:\Delta} r_{\mathrm{A}}(d)\cdot\mathbf{S}(\ell, S(m)|_{2n}, in(d,m,q)) \lhd\ \textit{in-window}(\ell, m, (\ell+n)|_{2n}) \rhd \delta\\
&+ \textstyle\sum_{k:Nat} s_{\mathrm{B}}(retrieve(k,q),k)\cdot\mathbf{S}(\ell,m,q) \lhd\ test(k,q) \rhd \delta\\
&+ \textstyle\sum_{k:Nat} r_{\mathrm{F}}(k)\cdot\mathbf{S}(k, m, release|_{2n}(\ell,k,q))
\end{aligned}$$

The receiver **R** is modelled by the process $\mathbf{R}(\ell', q')$, where q' is a buffer of size $2n$ and ℓ' the first position in the receiving window.

$$\begin{aligned}
&\mathbf{R}(\ell'{:}Nat, q'{:}Buf)\\
&\approx \textstyle\sum_{d:\Delta}\sum_{k:Nat} r_{\mathrm{C}}(d,k)\cdot(\mathbf{R}(\ell', in(d,k,q')) \lhd\ \textit{in-window}(\ell', k, (\ell'+n)|_{2n}) \rhd \mathbf{R}(\ell',q'))\\
&+ s_{\mathrm{D}}(retrieve(\ell',q'))\cdot\mathbf{R}(S(\ell')|_{2n}, remove(\ell',q')) \lhd\ test(\ell',q') \rhd \delta\\
&+ s_{\mathrm{E}}(\textit{next-empty}|_{2n}(\ell',q'))\cdot\mathbf{R}(\ell',q')
\end{aligned}$$

For $i \in \{\mathrm{B}, \mathrm{C}, \mathrm{E}, \mathrm{F}\}$, s_i and r_i can communicate, resulting in c_i.

Finally, the mediums **K** and **L**, which have capacity one, may lose frames between **S** and **R**. The action j indicates an internal choice.

$$\begin{aligned}
\mathbf{K} &\approx \textstyle\sum_{d:\Delta}\sum_{k:Nat} r_{\mathrm{B}}(d,k)\cdot(j\cdot s_{\mathrm{C}}(d,k)+j)\cdot\mathbf{K}\\
\mathbf{L} &\approx \textstyle\sum_{k:Nat} r_{\mathrm{E}}(k)\cdot(j\cdot s_{\mathrm{F}}(k)+j)\cdot\mathbf{L}
\end{aligned}$$

The initial state of the SWP is expressed by $\tau_{\mathcal{I}}(\partial_{\mathcal{H}}(\mathbf{S}(0,0,[]) \parallel \mathbf{R}(0,[]) \parallel \mathbf{K} \parallel \mathbf{L}))$, where the set $\mathcal{H}$ consists of the read and send actions over the internal channels B, C, E, and F, while the set $\mathcal{I}$ consists of the communication actions over these internal channels together with j.

External Behaviour. Data elements that are read from channel A by $\mathbf{S}$ should be sent into channel D by $\mathbf{R}$ in the same order, and no data elements should be lost. In other words, the SWP is intended to be a solution for the linear specification

$$\begin{aligned}\mathbf{Z}(\lambda{:}List) \approx\ & \textstyle\sum_{d:\Delta} r_{\mathrm{A}}(d){\cdot}\mathbf{Z}(append(d,\lambda)) \lhd length(\lambda) < 2n \rhd \delta \\ & + s_{\mathrm{D}}(top(\lambda)){\cdot}\mathbf{Z}(tail(\lambda)) \lhd length(\lambda) > 0 \rhd \delta\end{aligned}$$

Note that $r_{\mathrm{A}}(d)$ can be performed until the list λ contains $2n$ elements, because in that situation the sending and receiving windows will be filled. Furthermore, $s_{\mathrm{D}}(top(\lambda))$ can only be performed if λ is not empty.

The remainder of this paper is devoted to proving the following theorem.

Theorem 1. $\tau_{\mathcal{I}}(\partial_{\mathcal{H}}(\mathbf{S}(0,0,[]) \parallel \mathbf{R}(0,[]) \parallel \mathbf{K} \parallel \mathbf{L})) \mathrel{\underline{\leftrightarrow}_b} \mathbf{Z}(\langle\rangle)$.

5 Transformations of the Specification

The starting point of our correctness proof is a linear specification $\mathbf{N}_{mod}$, in which no parallel operators occur. $\mathbf{N}_{mod}$ can be obtained from the μCRL specification of the SWP without the hiding operator, by means of a linearisation algorithm presented in [17]. $\mathbf{N}_{mod}$ contains five extra parameters: $e{:}D$ and $g, g', h, h'{:}Nat$. Intuitively, g (resp. g') equals zero when medium $\mathbf{K}$ (resp. $\mathbf{L}$) is inactive, equals one when $\mathbf{K}$ (resp. $\mathbf{L}$) just received a datum, and equals two if $\mathbf{K}$ (resp. $\mathbf{L}$) decides to pass on this datum. Furthermore, e (resp. h) equals the datum that is being sent from $\mathbf{S}$ to $\mathbf{R}$ (resp. the position of this datum in the sending window) while $g \neq 0$, and equals the dummy value d_0 (resp. 0) while $g = 0$. Finally h' equals the first empty position in the receiving window while $g' \neq 0$ and equals 0 while $g' = 0$. Furthermore, data arguments are stripped from communication actions, and these actions are renamed to a fresh action c. For the sake of presentation, we only present parameters whose values are changed.

$$\begin{aligned}
&\mathbf{N}_{mod}(\ell{:}Nat, m{:}Nat, q{:}Buf, \ell'{:}Nat, q'{:}Buf, g{:}Nat, e{:}D, h{:}Nat, g'{:}Nat, h'{:}Nat)\\
\approx\ & \textstyle\sum_{d:\Delta} r_{\mathrm{A}}(d){\cdot}\mathbf{N}_{mod}(m{:=}S(m)|_{2n}, q{:=}in(d,m,q)) \lhd in\text{-}window(\ell, m, (\ell+n)|_{2n}) \rhd \delta\\
+\ & \textstyle\sum_{k:Nat} c{\cdot}\mathbf{N}_{mod}(g{:=}1, e{:=}retrieve(k,q), h{:=}k) \lhd test(k,q) \wedge g = 0 \rhd \delta\\
+\ & j{\cdot}\mathbf{N}_{mod}(g{:=}0, e{:=}d_0, h{:=}0) \lhd g = 1 \rhd \delta\\
+\ & j{\cdot}\mathbf{N}_{mod}(g{:=}2) \lhd g = 1 \rhd \delta\\
+\ & c{\cdot}\mathbf{N}_{mod}(q'{:=}in(e,h,q'), g{:=}0, e{:=}d_0, h{:=}0) \lhd in\text{-}window(\ell', h, (\ell'+n)|_{2n}) \wedge g = 2 \rhd \delta\\
+\ & c{\cdot}\mathbf{N}_{mod}(g{:=}0, e{:=}d_0, h{:=}0) \lhd \neg in\text{-}window(\ell', h, (\ell'+n)|_{2n}) \wedge g = 2 \rhd \delta\\
+\ & s_{\mathrm{D}}(retrieve(\ell', q')){\cdot}\mathbf{N}_{mod}(\ell'{:=}S(\ell')|_{2n}, q'{:=}remove(\ell', q')) \lhd test(\ell', q') \rhd \delta\\
+\ & c{\cdot}\mathbf{N}_{mod}(g'{:=}1, h'{:=}next\text{-}empty|_{2n}(\ell', q')) \lhd g' = 0 \rhd \delta\\
+\ & j{\cdot}\mathbf{N}_{mod}(g'{:=}0, h'{:=}0) \lhd g' = 1 \rhd \delta\\
+\ & j{\cdot}\mathbf{N}_{mod}(g'{:=}2) \lhd g' = 1 \rhd \delta\\
+\ & c{\cdot}\mathbf{N}_{mod}(\ell{:=}h', q{:=}release|_{2n}(\ell, h', q), g'{:=}0, h'{:=}0) \lhd g' = 2 \rhd \delta
\end{aligned}$$

Theorem 2.

$$\tau_{\mathcal{I}}(\partial_{\mathcal{H}}(\mathbf{S}(0,0,[]) \parallel \mathbf{R}(0,[]) \parallel \mathbf{K} \parallel \mathbf{L})) \leftrightarrow \tau_{\{c,j\}}(\mathbf{N}_{mod}(0,0,[],0,[],0,d_0,0,0,0)).$$

The specification of $\mathbf{N}_{nonmod}$ is obtained by eliminating all occurrences of $|_{2n}$ from $\mathbf{N}_{mod}$, and replacing $\mathit{in\text{-}window}(\ell, m, (\ell+n)|_{2n}$ by $m < \ell + n$ and $\mathit{in\text{-}window}(\ell', h, (\ell'+n)|_{2n}$ by $\ell' \leq h < \ell' + n$.

Theorem 3. $\mathbf{N}_{mod}(0,0,[],0,[],0,d_0,0,0,0) \leftrightarrow \mathbf{N}_{nonmod}(0,0,[],0,[],0,d_0,0,0,0)$.

The proof of Theorem 2, using a linearisation algorithm [17] and a simple renaming, is omitted. The proof of Theorem 3 is shown in Section 7.1.

6 Properties of Data and Invariants of $\mathbf{N}_{nonmod}$

Lemma 1 collects results on modulo arithmetic related to buffers. Simpler lemmas on modulo arithmetic, buffers, the *next-empty* operation, and lists can be found in the full version of this paper [8]. We use those lemmas without mention.

Lemma 1. *The lemmas below hold for modulo arithmetic related to buffers.*

1. $\forall j{:}Nat(\mathit{test}(j,q) \rightarrow i \leq j < i+n) \wedge i \leq k \leq i+n \rightarrow \mathit{test}(k,q) = \mathit{test}(k|_{2n}, q|_{2n})$
2. $\forall j{:}Nat(\mathit{test}(j,q) \rightarrow i \leq j < i+n) \wedge \mathit{test}(k,q) \rightarrow \mathit{retrieve}(k,q) = \mathit{retrieve}(k|_{2n}, q|_{2n})$
3. $i \leq k < i+n \rightarrow \mathit{in\text{-}window}(i|_{2n}, k|_{2n}, (i+n)|_{2n})$
4. $\mathit{in\text{-}window}(i|_{2n}, k|_{2n}, (i+n)|_{2n}) \rightarrow k+n < i \vee i \leq k < i+n \vee k \geq i+2n$
5. $\forall j{:}Nat(\mathit{test}(j,q) \rightarrow i \leq j < i+n) \wedge \mathit{test}(k, q|_{2n}) \rightarrow \mathit{in\text{-}window}(i|_{2n}, k, (i+n)|_{2n})$

Invariants of a system are properties of data that are satisfied throughout the reachable state space of the system. Lemma 2 collects 9 invariants of $\mathbf{N}_{nonmod}$ that are needed in the correctness proofs in the current paper.

Lemma 2. *The invariants below hold for* $\mathbf{N}_{nonmod}(\ell, m, q, \ell', q', g, e, h, g', h')$.

1. $\mathit{max}\{h', \ell\} \leq \mathit{next\text{-}empty}(\ell', q')$
2. $g \neq 0 \;\rightarrow\; h < m$
3. $\mathit{next\text{-}empty}(\ell', q') \leq \min\{m, \ell'+n\}$
4. $\mathit{test}(i,q) \;\leftrightarrow\; \ell \leq i < m$
5. $\ell \leq m \leq \ell+n \leq \ell'+2n$
6. $g \neq 0 \;\rightarrow\; \mathit{next\text{-}empty}(\ell', q') \leq h+n$
7. $g \neq 0 \wedge \mathit{test}(h,q) \;\rightarrow\; \mathit{retrieve}(h,q) = e$
8. $g \neq 0 \wedge \mathit{test}(h,q') \;\rightarrow\; \mathit{retrieve}(h,q') = e$
9. $\ell \leq i \wedge j \leq \mathit{next\text{-}empty}(i,q') \;\rightarrow\; q[i..j\rangle = q'[i..j\rangle$

7 Correctness of $\mathbf{N}_{mod}$

7.1 Equality of $\mathbf{N}_{mod}$ and $\mathbf{N}_{nonmod}$

In this section we present a proof of Theorem 3. It suffices to prove that for all $\ell, m, \ell', h, h' : Nat$, $q, q' : Buf$, $e : \Delta$ and $g, g' \leq 2$,

$$\mathbf{N}_{mod}(\ell|_{2n}, m|_{2n}, q|_{2n}, \ell'|_{2n}, q'|_{2n}, g, e, h|_{2n}, g', h'|_{2n})$$
$$\leftrightarrow \mathbf{N}_{nonmod}(\ell|_{2n}, m|_{2n}, q|_{2n}, \ell'|_{2n}, q'|_{2n}, g, e, h|_{2n}, g', h'|_{2n})$$

Proof. We show that $\mathbf{N}_{mod}(\ell|_{2n}, m|_{2n}, q|_{2n}, \ell'|_{2n}, q'|_{2n}, g, e, h|_{2n}, g', h'|_{2n})$ is a solution for the defining equation of $\mathbf{N}_{nonmod}(\ell, m, q, \ell', q', g, e, h, g', h')$. Hence, we must derive the following equation.

$$
\begin{array}{ll}
\quad \mathbf{N}_{mod}(\ell|_{2n}, m|_{2n}, q|_{2n}, \ell'|_{2n}, q'|_{2n}, g, e, h|_{2n}, g', h'|_{2n}) & \\
\approx \sum_{d:\Delta} r_{\mathrm{A}}(d)\cdot\mathbf{N}_{mod}(m{:=}S(m)|_{2n}, q{:=}in(d,m,q)|_{2n}) \lhd m < \ell + n \rhd \delta & (A)\\
+ \sum_{k:Nat} c\cdot\mathbf{N}_{mod}(g{:=}1, e{:=}retrieve(k,q), h{:=}k|_{2n}) \lhd test(k,q) \wedge g = 0 \rhd \delta & (B)\\
+ j\cdot\mathbf{N}_{mod}(g{:=}0, e{:=}d_0, h{:=}0) \lhd g = 1 \rhd \delta & (C)\\
+ j\cdot\mathbf{N}_{mod}(g{:=}2) \lhd g = 1 \rhd \delta & (D)\\
+ c\cdot\mathbf{N}_{mod}(q'{:=}in(e,h,q')|_{2n}, g{:=}0, e{:=}d_0, h{:=}0) \lhd \ell' \leq h < \ell' + n \wedge g = 2 \rhd \delta & (E)\\
+ c\cdot\mathbf{N}_{mod}(g{:=}0, e{:=}d_0, h{:=}0) \lhd \neg(\ell' \leq h < \ell' + n) \wedge g = 2 \rhd \delta & (F)\\
+ s_{\mathrm{D}}(retrieve(\ell', q'))\cdot\mathbf{N}_{mod}(\ell'{:=}S(\ell')|_{2n}, q'{:=}remove(\ell', q')|_{2n}) \lhd test(\ell', q') \rhd \delta & (G)\\
+ c\cdot\mathbf{N}_{mod}(g'{:=}1, h'{:=}next\text{-}empty(\ell', q')|_{2n}) \lhd g' = 0 \rhd \delta & (H)\\
+ j\cdot\mathbf{N}_{mod}(g'{:=}0, h'{:=}0) \lhd g' = 1 \rhd \delta & (I)\\
+ j\cdot\mathbf{N}_{mod}(g'{:=}2) \lhd g' = 1 \rhd \delta & (J)\\
+ c\cdot\mathbf{N}_{mod}(\ell{:=}h'|_{2n}, q{:=}release(\ell, h', q)|_{2n}, g'{:=}0, h'{:=}0) \lhd g' = 2 \rhd \delta & (K)
\end{array}
$$

In order to prove this, we instantiate the parameters in the defining equation of $\mathbf{N}_{mod}$ with $\ell|_{2n}, m|_{2n}, q|_{2n}, \ell'|_{2n}, q'|_{2n}, g, e, h|_{2n}, g', h'|_{2n}$.

$$
\begin{array}{l}
\quad \mathbf{N}_{mod}(\ell|_{2n}, m|_{2n}, q|_{2n}, \ell'|_{2n}, q'|_{2n}, g, e, h|_{2n}, g', h'|_{2n})\\
\approx \sum_{d:\Delta} r_{\mathrm{A}}(d)\cdot\mathbf{N}_{mod}(m{:=}S(m|_{2n})|_{2n}, q{:=}in(d, m|_{2n}, q|_{2n}))\\
\qquad\qquad\qquad\qquad \lhd in\text{-}window(\ell|_{2n}, m|_{2n}, (\ell|_{2n} + n)|_{2n}) \rhd \delta\\
+ \sum_{k:Nat} c\cdot\mathbf{N}_{mod}(g{:=}1, e{:=}retrieve(k, q|_{2n}), h{:=}k) \lhd test(k, q|_{2n}) \wedge g = 0 \rhd \delta\\
+ j\cdot\mathbf{N}_{mod}(g{:=}0, e{:=}d_0, h{:=}0) \lhd g = 1 \rhd \delta\\
+ j\cdot\mathbf{N}_{mod}(g{:=}2) \lhd g = 1 \rhd \delta\\
+ c\cdot\mathbf{N}_{mod}(q'{:=}in(e, h|_{2n}, q'|_{2n}), g{:=}0, e{:=}d_0, h{:=}0)\\
\qquad\qquad\qquad\qquad \lhd in\text{-}window(\ell'|_{2n}, h|_{2n}, (\ell'|_{2n} + n)|_{2n}) \wedge g = 2 \rhd \delta\\
+ c\cdot\mathbf{N}_{mod}(g{:=}0, e{:=}d_0, h{:=}0)\\
\qquad\qquad\qquad\qquad \lhd \neg in\text{-}window(\ell'|_{2n}, h|_{2n}, (\ell'|_{2n} + n)|_{2n}) \wedge g = 2 \rhd \delta\\
+ s_{\mathrm{D}}(retrieve(\ell'|_{2n}, q'|_{2n}))\cdot\mathbf{N}_{mod}(\ell'{:=}S(\ell'|_{2n})|_{2n}, q'{:=}remove(\ell'|_{2n}, q'|_{2n}))\\
\qquad\qquad\qquad\qquad \lhd test(\ell'|_{2n}, q'|_{2n}) \rhd \delta\\
+ c\cdot\mathbf{N}_{mod}(g'{:=}1, h'{:=}next\text{-}empty|_{2n}(\ell'|_{2n}, q'|_{2n})) \lhd g' = 0 \rhd \delta\\
+ j\cdot\mathbf{N}_{mod}(g'{:=}0, h'{:=}0) \lhd g' = 1 \rhd \delta\\
+ j\cdot\mathbf{N}_{mod}(g'{:=}2) \lhd g' = 1 \rhd \delta\\
+ c\cdot\mathbf{N}_{mod}(\ell{:=}h'|_{2n}, q{:=}release|_{2n}(\ell|_{2n}, h'|_{2n}, q|_{2n}), g'{:=}0, h'{:=}0) \lhd g' = 2 \rhd \delta
\end{array}
$$

To equate the eleven summands in both specifications, we obtain a number of proof obligations. Here, we focus on summands A, B, and E.

A $m < \ell + n = in\text{-}window(\ell|_{2n}, m|_{2n}, (\ell|_{2n} + n)|_{2n})$.
$m < \ell + n \leftrightarrow \ell \leq m < \ell + n$ (Inv. 2.5) $\rightarrow in\text{-}window(\ell|_{2n}, m|_{2n}, (\ell + n)|_{2n})$ (Lem. 1.3). Reversely, $in\text{-}window(\ell|_{2n}, m|_{2n}, (\ell + n)|_{2n}) \rightarrow m{+}n < \ell \vee \ell \leq m < \ell + n \vee m \geq \ell{+}2n$ (Lem. 1.4) $\leftrightarrow m < \ell{+}n$ (Inv. 2.5). Since $(\ell + n)|_{2n} = (\ell|_{2n} + n)|_{2n}$, we have $m < \ell + n = in\text{-}window(\ell|_{2n}, m|_{2n}, (\ell|_{2n} + n)|_{2n})$.

B Below we equate the entire summand B of the two specifications. The conjunction $g = 0$ and the argument $g{:=}1$ of summand B are omitted, because they are irrelevant for this derivation.
By Inv. 2.4 and 2.5, $\mathit{test}(j, q) \rightarrow \ell \leq j < \ell + n$. So by Lem. 1.5, $\mathit{test}(k', q|_{2n})$ implies $\mathit{in\text{-}window}(\ell|_{2n}, k', (\ell+n)|_{2n})$. $\mathit{test}(k', q|_{2n})$ implies $k' = k'|_{2n}$, and by Lem. 1.4, $k' + n < \ell|_{2n} \vee \ell|_{2n} \leq k' < \ell|_{2n} + n \vee k' \geq \ell + 2n$. $k' = k'|_{2n} < 2n$ implies $k' + n < \ell|_{2n} \vee \ell|_{2n} \leq k' < \ell|_{2n} + n$.

$$
\begin{array}{lr}
\sum_{k:Nat} c{\cdot}\mathbf{N}_{mod}(e{:=}\mathit{retrieve}(k, q), h{:=}k|_{2n}) & \\
\quad \lhd\ \mathit{test}(k, q)\ \rhd\ \delta & \\
\approx \sum_{k:Nat} c{\cdot}\mathbf{N}_{mod}(e{:=}\mathit{retrieve}(k, q), h{:=}k|_{2n}) & \\
\quad \lhd\ \mathit{test}(k, q) \wedge \ell \leq k < \ell + n\ \rhd\ \delta & \text{(Inv. 2.4, 2.5)} \\
\approx \sum_{k:Nat} c{\cdot}\mathbf{N}_{mod}(e{:=}\mathit{retrieve}(k|_{2n}, q|_{2n}), h{:=}k|_{2n}) & \\
\quad \lhd\ \mathit{test}(k|_{2n}, q|_{2n}) \wedge \ell \leq k < \ell + n\ \rhd\ \delta & \text{(Lem. 1.1, 1.2)} \\
\approx \sum_{k':Nat} \sum_{k:Nat} c{\cdot}\mathbf{N}_{mod}(e{:=}\mathit{retrieve}(k', q|_{2n}), h{:=}k') & \\
\quad \lhd\ \mathit{test}(k', q|_{2n}) \wedge \ell \leq k < \ell + n \wedge k' = k|_{2n}\ \rhd\ \delta & \text{(sum elimination)} \\
\approx \sum_{k':Nat} \sum_{k:Nat} c{\cdot}\mathbf{N}_{mod}(e{:=}\mathit{retrieve}(k', q|_{2n}), h{:=}k') & \\
\quad \lhd\ \mathit{test}(k', q|_{2n}) \wedge k = (\ell\ \mathit{div}\ 2n)2n + k' \wedge \ell|_{2n} \leq k' < \ell|_{2n} + n \wedge k' = k|_{2n}\ \rhd\ \delta & \\
+ \sum_{k':Nat} \sum_{k:Nat} c{\cdot}\mathbf{N}_{mod}(e{:=}\mathit{retrieve}(k', q|_{2n}), h{:=}k') & \\
\quad \lhd\ \mathit{test}(k', q|_{2n}) \wedge k = S(\ell\ \mathit{div}\ 2n)2n + k' \wedge k' + n < \ell|_{2n} \wedge k' = k|_{2n}\ \rhd\ \delta & \\
\approx \sum_{k':Nat} c{\cdot}\mathbf{N}_{mod}(e{:=}\mathit{retrieve}(k', q|_{2n}), h{:=}k') & \\
\quad \lhd\ \mathit{test}(k', q|_{2n}) \wedge \ell|_{2n} \leq k' < \ell|_{2n} + n \wedge k' = k'\ \rhd\ \delta & \\
+ \sum_{k':Nat} c{\cdot}\mathbf{N}_{mod}(e{:=}\mathit{retrieve}(k', q|_{2n}), h{:=}k') & \\
\quad \lhd\ \mathit{test}(k', q|_{2n}) \wedge k' + n < \ell|_{2n} \wedge k' = k'\ \rhd\ \delta & \text{(sum elimination)} \\
\approx \sum_{k':Nat} c{\cdot}\mathbf{N}_{mod}(e{:=}\mathit{retrieve}(k', q|_{2n}), h{:=}k') & \\
\quad \lhd\ \mathit{test}(k', q|_{2n})\ \rhd\ \delta & \text{(see above)}
\end{array}
$$

E $g = 2 \rightarrow \ell' \leq h < \ell' + n = \mathit{in\text{-}window}(\ell'|_{2n}, h|_{2n}, (\ell'+n)|_{2n})$.
Let $g = 2$. We have $\ell' \leq \mathit{next\text{-}empty}(\ell', q')$, and by Inv. 2.6 together with $g = 2$, $\mathit{next\text{-}empty}(\ell', q') \leq h + n$, so $\ell' \leq h + n$. Furthermore, by Inv. 2.2 together with $g = 2$, $h < m$, by Inv. 2.5, $m \leq \ell' + 2n$. Hence, $h < \ell' + 2n$. So using Lem. 1.3 and 1.4, it follows that $\ell' \leq h < \ell' + n = \mathit{in\text{-}window}(\ell'|_{2n}, h|_{2n}, (\ell'+n)|_{2n})$.

Equality of other summands can be derived without much difficulty. Hence, we prove that $\mathbf{N}_{mod}(\ell|_{2n}, m|_{2n}, q|_{2n}, \ell'|_{2n}, q'|_{2n}, g, e, h|_{2n}, g', h'|_{2n})$ is a solution for the specification of $\mathbf{N}_{nonmod}(\ell, m, q, \ell', q', g, e, h, g', h')$. By CL-RSP, they are strongly bisimilar.

7.2 Correctness of $\mathbf{N}_{nonmod}$

We prove that $\mathbf{N}_{nonmod}$ is branching bisimilar to the FIFO queue $\mathbf{Z}$ of capacity $2n$ (see Section 4), using the cones and foci method [9].

Let Ξ abbreviate $Nat \times Nat \times Buf \times Nat \times Buf \times Nat \times \Delta \times Nat \times Nat \times Nat$. Furthermore, let $\xi{:}\Xi$ denote $(\ell, m, q, \ell', q', g, e, h, g', h')$. The state mapping $\phi : \Xi \rightarrow List$, which maps states of $\mathbf{N}_{nonmod}$ to states of $\mathbf{Z}$, is defined by:

$$\phi(\xi) = q'[\ell'..\mathit{next\text{-}empty}(\ell', q')\rangle {+\!\!+} q[\mathit{next\text{-}empty}(\ell', q')..m\rangle$$

Intuitively, ϕ collects the data elements in the sending and receiving windows, starting at the first position of the receiving window (i.e., ℓ') until the first empty position in this window, and then continuing in the sending window until the first empty position in that window (i.e., m). Note that ϕ is independent of e, g, ℓ, h, g', h'; we therefore write $\phi(m, q, \ell', q')$.

The focus points are those states where either the sending window is empty (meaning that $\ell = m$), or the receiving window is full and all data elements in the receiving window have been acknowledged, meaning that $\ell = \ell' + n$. That is, the focus condition for $\mathbf{N}_{nonmod}(\ell, m, q, \ell', q', g, e, h, g', h')$ is

$$FC(\ell, m, q, \ell', q', g, e, h, g', h') \;:=\; \ell = m \vee \ell = \ell' + n$$

Lemma 3. *For each $\xi{:}\Xi$ where the invariants in Lemma 2 hold, there is a $\hat{\xi}{:}\Xi$ with $FC(\hat{\xi})$ such that $\mathbf{N}_{nonmod}(\xi) \xrightarrow{c_1} \cdots \xrightarrow{c_n} \mathbf{N}_{nonmod}(\hat{\xi})$, where $c_1, \ldots, c_n \in \mathcal{I}$.*

Proof. In case $g \neq 0$ in ξ, by summands C, E and F, we can perform one or two communication actions to a state where $g = 0$. By Inv. 2.3, $\textit{next-empty}(\ell', q') \leq \min\{m, \ell'+n\}$. We prove by induction on $\min\{m, \ell'+n\} - \textit{next-empty}(\ell', q')$ that for each state ξ' where $g = 0$ and the invariants in Lemma 2 hold, a focus point can be reached.
Base Case: $\textit{next-empty}(\ell', q') = \min\{m, \ell' + n\}$.
In case $g' \neq 0$ in ξ', by summands I and K, we can perform communication actions to a state where $g' = 0$ and $\textit{next-empty}(\ell', q') = \min\{m, \ell' + n\}$. By summands H, J and K we can perform three communication actions to a state $\hat{\xi}$ where $\ell = h' = \textit{next-empty}(\ell', q') = \min\{m, \ell' + n\}$. Then $\ell = m$ or $\ell = \ell' + n$, so $FC(\hat{\xi})$.
Induction Case: $\textit{next-empty}(\ell', q') < \min\{m, \ell' + n\}$.
By Inv. 2.1, $\ell \leq \textit{next-empty}(\ell', q') < m$. By Inv. 2.4, $\textit{test}(\textit{next-empty}(\ell', q'), q)$. Furthermore, $\ell' \leq \textit{next-empty}(\ell, q') < \ell' + n$. Hence, by summands B, D and E from ξ' we can perform three communication actions to a state ξ''. In ξ'', $g := 0$, and in comparison to ξ', m and ℓ' remain the same, while $q' := \textit{in}(d, \textit{next-empty}(\ell', q'), q')$ where d denotes $\textit{retrieve}(\textit{next-empty}(\ell', q'), q)$. Since $\textit{next-empty}(\ell', \textit{in}(d, \textit{next-empty}(\ell', q'), q'))$ $= \textit{next-empty}(S(\textit{next-empty}(\ell', q')), q') > \textit{next-empty}(\ell', q')$, we can apply the induction hypothesis to conclude that from ξ'' a focus point can be reached.

Theorem 4. *For all $e{:}\Delta$, $\tau_{\{c,j\}}(\mathbf{N}_{nonmod}(0, 0, [], 0, [], 0, e, 0, 0, 0)) \,\underline{\leftrightarrow}_b\, \mathbf{Z}(\langle\rangle)$.*

Proof. By the cones and foci method we obtain the following matching criteria (cf. [9]). Trivial matching criteria are left out.

$$\begin{cases}
\text{I.1}: & \ell' \leq h < \ell' + n \wedge g = 2 \;\rightarrow\; \phi(m, q, \ell', q') = \phi(m, q, \ell', \textit{in}(e, h, q')) \\
\text{I.2}: & g' = 2 \;\rightarrow\; \phi(m, q, \ell', q') = \phi(m, \textit{release}(\ell, h', q), \ell', q') \\
\text{II.1}: & m < \ell + n \;\rightarrow\; \textit{length}(\phi(m, q, \ell', q')) < 2n \\
\text{II.2}: & \textit{test}(\ell', q') \;\rightarrow\; \textit{length}(\phi(m, q, \ell', q')) > 0 \\
\text{III.1}: & (\ell = m \vee \ell = \ell' + n) \wedge \textit{length}(\phi(m, q, \ell', q')) < 2n \;\rightarrow\; m < \ell + n \\
\text{III.2}: & (\ell = m \vee \ell = \ell' + n) \wedge \textit{length}(\phi(m, q, \ell', q')) > 0 \;\rightarrow\; \textit{test}(\ell', q') \\
\text{IV}: & \textit{test}(\ell', q') \;\rightarrow\; \textit{retrieve}(\ell', q') = \textit{top}(\phi(m, q, \ell', q')) \\
\text{V.1}: & m < \ell + n \;\rightarrow\; \phi(S(m), \textit{in}(d, m, q), \ell', q') = \textit{append}(d, \phi(m, q, \ell', q')) \\
\text{V.2}: & \textit{test}(\ell', q') \;\rightarrow\; \phi(m, q, S(\ell'), \textit{remove}(\ell', q')) = \textit{tail}(\phi(m, q, \ell', q'))
\end{cases}$$

I.1 $\ell' \leq h < \ell' + n \wedge g = 2 \rightarrow \phi(m, q, \ell', q') = \phi(m, q, \ell', \textit{in}(e, h, q'))$.

Case 1: $h \neq \textit{next-empty}(\ell', q')$).

Let $g = 2$. Since $\textit{next-empty}(\ell', \textit{in}(e, h, q')) = \textit{next-empty}(\ell', q')$, it follows that $\phi(m, q, \ell', \textit{in}(e, h, q')) = \textit{in}(e, h, q')[\ell'..\textit{next-empty}(\ell', q')\rangle {+\!\!+} q[\textit{next-empty}(\ell', q')..m\rangle$.

CASE 1.1: $\ell' \leq h < \mathit{next\text{-}empty}(\ell', q')$).
$\mathit{test}(h, q')$, so by Inv. 2.8 together with $g = 2$, $\mathit{retrieve}(h, q') = e$. Hence,
$\mathit{in}(e, h, q')[\ell'..\mathit{next\text{-}empty}(\ell', q')\rangle = q'[\ell'..\mathit{next\text{-}empty}(\ell', q')\rangle$.
CASE 1.2: $\neg(\ell' \leq h \leq \mathit{next\text{-}empty}(\ell', q'))$.
$\mathit{in}(e, h, q')[\ell'..\mathit{next\text{-}empty}(\ell', q')\rangle = q'[\ell'..\mathit{next\text{-}empty}(\ell', q')\rangle$.
CASE 2: $h = \mathit{next\text{-}empty}(\ell', q')$.
Let $g = 2$. The derivation splits into two parts.

(1) $\mathit{in}(e, h, q')[\ell'..h\rangle = q'[\ell'..h\rangle$.

(2) By Inv. 2.1, $\ell \leq h$, and by Inv. 2.2 together with $g = 2$, $h < m$. Thus, by Inv. 2.4, $\mathit{test}(h, q)$. So by Inv. 2.7 together with $g = 2$, $\mathit{retrieve}(h, q) = e$. Hence,

$$\begin{aligned}
& \mathit{in}(e, h, q')[h..\mathit{next\text{-}empty}(S(h), q')\rangle \\
= \;& \mathit{in}(e, \mathit{in}(e, h, q')[S(h)..\mathit{next\text{-}empty}(S(h), q')\rangle) \\
= \;& \mathit{in}(e, q'[S(h)..\mathit{next\text{-}empty}(S(h), q')\rangle) \\
= \;& \mathit{in}(e, q[S(h)..\mathit{next\text{-}empty}(S(h), q')\rangle) && \text{(Inv. 2.9)} \\
= \;& q[h..\mathit{next\text{-}empty}(S(h), q')\rangle
\end{aligned}$$

Finally, we combine (1) and (2). We recall that $h = \mathit{next\text{-}empty}(\ell', q')$.

$$\begin{aligned}
& \mathit{in}(e, h, q')[\ell'..\mathit{next\text{-}empty}(\ell', \mathit{in}(e, h, q'))\rangle \\
& {+\!\!+} q[\mathit{next\text{-}empty}(\ell', \mathit{in}(e, h, q'))..m\rangle \\
= \;& \mathit{in}(e, h, q')[\ell'..\mathit{next\text{-}empty}(S(h), q')\rangle \\
& {+\!\!+} q[\mathit{next\text{-}empty}(S(h), q')..m\rangle \\
= \;& (\mathit{in}(e, h, q')[\ell'..h\rangle {+\!\!+} \mathit{in}(e, h, q')[h..\mathit{next\text{-}empty}(S(h), q')\rangle) \\
& {+\!\!+} q[\mathit{next\text{-}empty}(S(h), q')..m\rangle \\
= \;& q'[\ell'..h\rangle {+\!\!+} q[h..\mathit{next\text{-}empty}(S(h), q')\rangle \\
& {+\!\!+} q[\mathit{next\text{-}empty}(S(h), q')..m\rangle && (1), (2) \\
= \;& q'[\ell'..h\rangle {+\!\!+} q[h..m\rangle
\end{aligned}$$

I.2 $g' = 2 \to \phi(m, q, \ell', q') = \phi(m, \mathit{release}(\ell, h', q), \ell', q')$.
By Inv. 2.1, $h' \leq \mathit{next\text{-}empty}(\ell', q')$.
So $\mathit{release}(\ell, h', q)[\mathit{next\text{-}empty}(\ell', q')..m\rangle = q'[\mathit{next\text{-}empty}(\ell', q')..m\rangle$.

II.1 $m < \ell + n \to \mathit{length}(\phi(m, q, \ell', q')) < 2n$.
Let $m < \ell + n$. By Inv. 2.3, $\mathit{next\text{-}empty}(\ell', q') \leq \ell' + n$. Hence,

$$\begin{aligned}
& \mathit{length}(q'[\ell'..\mathit{next\text{-}empty}(\ell', q')\rangle {+\!\!+} q[\mathit{next\text{-}empty}(\ell', q')..m\rangle) \\
= \;& \mathit{length}(q'[\ell'..\mathit{next\text{-}empty}(\ell', q')\rangle) + \mathit{length}(q[\mathit{next\text{-}empty}(\ell', q')..m\rangle)) \\
= \;& (\mathit{next\text{-}empty}(\ell', q') \dot{-} \ell') + (m \dot{-} \mathit{next\text{-}empty}(\ell', q')) \\
\leq \;& n + (m \dot{-} \ell) && \text{(Inv. 2.1)} \\
< \;& 2n
\end{aligned}$$

II.2 $\mathit{test}(\ell', q') \to \mathit{length}(\phi(m, q, \ell', q')) > 0$.
$\mathit{test}(\ell', q')$ yields $\mathit{next\text{-}empty}(\ell', q') = \mathit{next\text{-}empty}(S(\ell'), q') \geq S(\ell')$. Hence,
$\mathit{length}(\phi(m, q, \ell', q')) = (\mathit{next\text{-}empty}(\ell', q') \dot{-} \ell') + (m \dot{-} \mathit{next\text{-}empty}(\ell', q')) > 0$.

III.1 $(\ell = m \lor \ell = \ell' + n) \land \mathit{length}(\phi(m, q, \ell', q')) < 2n \to m < \ell + n$.
CASE 1: $\ell = m$.
Then $m < \ell + n$ holds trivially.
CASE 2: $\ell = \ell' + n$.
By Inv. 2.3, $\mathit{next\text{-}empty}(\ell', q') \leq \ell' + n$. Hence,

$$\begin{aligned}
& \mathit{length}(\phi(m, q, \ell', q')) \\
= \;& (\mathit{next\text{-}empty}(\ell', q') \dot{-} \ell') + (m \dot{-} \mathit{next\text{-}empty}(\ell', q')) \\
\leq \;& ((\ell' + n) \dot{-} \ell') + (m \dot{-} \ell) && \text{(Inv. 2.1)} \\
= \;& n + (m \dot{-} \ell)
\end{aligned}$$

So $\mathit{length}(\phi(m,q,\ell',q')) < 2n$ implies $m < \ell + n$.

III.2 $(\ell = m \vee \ell = \ell' + n) \wedge \mathit{length}(\phi(m,q,\ell',q')) > 0 \to \mathit{test}(\ell',q')$.

CASE 1: $\ell = m$.

Since $m \dot{-} \mathit{next\text{-}empty}(\ell',q') \leq (m \dot{-} \ell)$ (Inv. 2.1) $= 0$, we have $\mathit{length}(\phi(m,q,\ell',q')) = \mathit{next\text{-}empty}(\ell',q') \dot{-} \ell'$.

Hence, $\mathit{length}(\phi(m,q,\ell',q')) > 0$ yields $\mathit{next\text{-}empty}(\ell',q') > \ell'$, which implies $\mathit{test}(\ell',q')$.

CASE 2: $\ell = \ell' + n$.

Then by Inv. 2.1, $\mathit{next\text{-}empty}(\ell',q') \geq \ell' + n$, which implies $\mathit{test}(\ell',q')$.

IV $\mathit{test}(\ell',q') \ \to \ \mathit{retrieve}(\ell',q') = \mathit{top}(\phi(m,q,\ell',q'))$.

$\mathit{test}(\ell',q')$ implies $\mathit{next\text{-}empty}(\ell',q') = \mathit{next\text{-}empty}(S(\ell'),q') \geq S(\ell')$.

So $q'[\ell'..\mathit{next\text{-}empty}(\ell',q')\rangle = \mathit{in}(\mathit{retrieve}(\ell',q'), q'[S(\ell')..\mathit{next\text{-}empty}(\ell',q')\rangle)$.

Hence, $\mathit{top}(\phi(m,q,\ell',q')) = \mathit{retrieve}(\ell',q')$.

V.1 $m < \ell + n \to \phi(S(m), \mathit{in}(d,m,q), \ell', q') = \mathit{append}(d, \phi(m,q,\ell',q'))$.

$$\begin{aligned}
&q'[\ell'..\mathit{next\text{-}empty}(\ell',q')\rangle +\!\!+ \mathit{in}(d,m,q)[\mathit{next\text{-}empty}(\ell',q')..S(m)\rangle \\
&= q'[\ell'..\mathit{next\text{-}empty}(\ell',q')\rangle +\!\!+ \mathit{append}(d, q[\mathit{next\text{-}empty}(\ell',q')..m\rangle) \\
&= \mathit{append}(d, q'[\ell'..\mathit{next\text{-}empty}(\ell',q')\rangle +\!\!+ q[\mathit{next\text{-}empty}(\ell',q')..m\rangle)
\end{aligned}$$

V.2 $\mathit{test}(\ell',q') \to \phi(m,q,S(\ell'),\mathit{remove}(\ell',q')) = \mathit{tail}(\phi(m,q,\ell',q'))$.

$\mathit{test}(\ell',q')$ implies $\mathit{next\text{-}empty}(\ell',q') = \mathit{next\text{-}empty}(S(\ell'),q')$. Hence,

$$\begin{aligned}
&\mathit{remove}(\ell',q')[S(\ell')..\mathit{next\text{-}empty}(S(\ell'),\mathit{remove}(\ell',q'))\rangle \\
&\quad +\!\!+ q[\mathit{next\text{-}empty}(S(\ell'),\mathit{remove}(\ell',q'))..m\rangle \\
&= \mathit{remove}(\ell',q')[S(\ell')..\mathit{next\text{-}empty}(S(\ell'),q')\rangle +\!\!+ q[\mathit{next\text{-}empty}(S(\ell'),q')..m\rangle \\
&= \mathit{remove}(\ell',q')[S(\ell')..\mathit{next\text{-}empty}(\ell',q')\rangle +\!\!+ q[\mathit{next\text{-}empty}(\ell',q')..m\rangle \\
&= q'[S(\ell')..\mathit{next\text{-}empty}(\ell',q')\rangle +\!\!+ q[\mathit{next\text{-}empty}(\ell',q')..m\rangle \\
&= \mathit{tail}(q'[\ell'..\mathit{next\text{-}empty}(\ell',q')\rangle +\!\!+ q[\mathit{next\text{-}empty}(\ell',q')..m\rangle)
\end{aligned}$$

7.3 Correctness of the Sliding Window Protocol

Finally, we can prove Theorem 1.

Proof.

$$\begin{aligned}
&\tau_{\mathcal{I}}(\partial_{\mathcal{H}}(\mathbf{S}(0,0,[]) \parallel \mathbf{R}(0,[]) \parallel \mathbf{K} \parallel \mathbf{L})) && \\
\leftrightarrow\ &\tau_{\{c,j\}}(\mathbf{N}_{mod}(0,0,[],0,[],0,d_0,0,0,0)) && \text{(Thm. 2)} \\
\leftrightarrow\ &\tau_{\{c,j\}}(\mathbf{N}_{nonmod}(0,0,[],0,[],0,d_0,0,0,0)) && \text{(Thm. 3)} \\
\leftrightarrow_b\ &\mathbf{Z}(\langle\rangle) && \text{(Thm. 4)}
\end{aligned}$$

8 Related Work

Sliding window protocols have attracted considerable interest from the formal verification community. In this section we present an overview. Many of these verifications deal with unbounded sequence numbers, in which case modulo arithmetic is avoided, or with a fixed finite window size. The papers that do treat arbitrary finite window sizes mostly restrict to safety properties.

Infinite window size. Stenning [41] studied a SWP with unbounded sequence numbers and an infinite window size, in which messages can be lost, duplicated or reordered. A timeout mechanism is used to trigger retransmission. Stenning gave informal manual proofs of some safety properties. Knuth [26] examined more general principles behind Stenning's protocol, and manually verified some safety properties. Hailpern [20] used temporal logic to formulate safety and liveness properties for Stenning's protocol, and established their validity by informal reasoning. Jonsson [23] also verified both safety and liveness properties of the protocol, using temporal logic and a manual compositional verification technique.

Fixed finite window size. Richier *et al.* [34] specified a SWP in a process algebra based language Estelle/R, and verified safety properties for window size up to eight using the model checker Xesar. Madelaine and Vergamini [29] specified a SWP in Lotos, with the help of the simulation environment Lite, and proved some safety properties for window size six. Holzmann [21, 22] used the Spin model checker to verify both safety and liveness properties of a SWP with sequence numbers up to five. Kaivola [25] verified safety and liveness properties using model checking for a SWP with window size up to seven. Godefroid and Long [11] specified a full duplex SWP in a guarded command language, and verified the protocol for window size two using a model checker based on Queue BDDs. Stahl *et al.* [40] used a combination of abstraction, data independence, compositional reasoning and model checking to verify safety and liveness properties for a SWP with window size up to sixteen. The protocol was specified in Promela, the input language for the Spin model checker. Smith and Klarlund [38] specified a SWP in the high-level language IOA, and used the theorem prover MONA to verify a safety property for unbounded sequence numbers with window size up to 256. Latvala [27] modeled a SWP using Colored Petri nets. A liveness property was model checked with fairness constraints for window size up to eleven.

Arbitrary finite window size. Cardell-Oliver [5] specified a SWP using higher order logic, and manually proved and mechanically checked safety properties using HOL. (Van de Snepscheut [39] noted that what Cardell-Oliver claims to be a liveness property is in fact a safety property.) Schoone [37] manually proved safety properties for several SWPs using assertional verification. Van de Snepscheut [39] gave a correctness proof of a SWP as a sequence of correctness preserving transformations of a sequential program. Paliwoda and Sanders [32] specified a reduced version of what they call a SWP (but which is in fact very similar to the bakery protocol from [14]) in the process algebra CSP, and verified a safety property modulo trace semantics. Röckl and Esparza [35] verified the correctness of this bakery protocol modulo weak bisimulation using Isabelle/HOL, by explicitly checking a bisimulation relation. Jonsson and Nilsson [24] used an automated reachability analysis to verify safety properties for a SWP with arbitrary sending window size and receiving window size one. Rusu [36] used the theorem prover PVS to verify both safety and liveness properties for a SWP with unbounded sequence numbers. Chkliaev *et al.* [7] used a timed state machine in

PVS to specify a SWP in which messages can be lost, duplicated or reordered, and proved some safety properties with the mechanical support of PVS.

References

1. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.
2. M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In *Proc. CONCUR'94*, LNCS 836, pp. 401–416. Springer, 1994.
3. M.A. Bezem and J.F. Groote. A correctness proof of a one bit sliding window protocol in μCRL. *The Computer Journal*, 37(4):289–307, 1994.
4. J.J. Brunekreef. Sliding window protocols. In S. Mauw and G. Veltink, eds, *Algebraic Specification of Protocols.* Cambridge Tracts in Theoretical Computer Science 36, pp. 71–112. Cambridge University Press, 1993.
5. R. Cardell-Oliver. Using higher order logic for modelling real-time protocols. In *Proc. TAPSOFT'91*, LNCS 494, pp. 259–282. Springer, 1991.
6. V.G. Cerf and R.E. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, COM-22:637–648, 1974.
7. D. Chkliaev, J. Hooman, and E. de Vink. Verification and improvement of the sliding window protocol. In *TACAS'03*, LNCS 2619, pp. 113–127. Springer, 2003.
8. W.J. Fokkink, J.F. Groote, J. Pang, B. Badban, and J.C. van de Pol. Verifying a sliding window protocol in μCRL. Technical Report SEN-R0308, CWI, 2003.
9. W.J. Fokkink and J. Pang. Cones and foci for protocol verification revisited. In *Proc. FOSSACS'03*, LNCS 2620, pp. 267–281. Springer, 2003.
10. R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.
11. P. Godefroid and D.E. Long. Symbolic protocol verification with Queue BDDs. *Formal Methods and System Design*, 14(3):257–271, 1999.
12. R.A. Groenveld. Verification of a sliding window protocol by means of process algebra. Report P8701, University of Amsterdam, 1987.
13. J.F. Groote. *Process Algebra and Structured Operational Semantics.* PhD thesis, University of Amsterdam, 1991.
14. J.F. Groote and H.P. Korver. Correctness proof of the bakery protocol in μCRL. In *Proc. ACP'94*, Workshops in Computing, pp. 63–86. Springer, 1995.
15. J.F. Groote and A. Ponse. Proof theory for μCRL: A language for processes with data. In *Proc. SoSL'93*, Workshops in Computing, pp. 232–251. Springer, 1994.
16. J.F. Groote and A. Ponse. Syntax and semantics of μCRL. In *Proc. ACP'94*, Workshops in Computing, pp. 26–62. Springer, 1995.
17. J.F. Groote, A. Ponse, and Y.S. Usenko. Linearization of parallel pCRL. *Journal of Logic and Algebraic Programming*, 48(1/2):39–72, 2001.
18. J.F. Groote and M. Reniers. Algebraic process verification. In *Handbook of Process Algebra*, pp. 1151–1208. Elsevier, 2001.
19. J.F. Groote and J. Springintveld. Focus points and convergent process operators: A proof strategy for protocol verification. *Journal of Logic and Algebraic Programming*, 49(1/2):31–60, 2001.
20. B.T. Hailpern. *Verifying Concurrent Processes Using Temporal Logic.* LNCS 129, Springer, 1982.
21. G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1991.

22. G.J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279-295, 1997.
23. B. Jonsson. *Compositional Verification of Distributed Systems.* PhD thesis, Uppsala University, 1987.
24. B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *TACAS'00*, LNCS 1785, pp. 220–234. Springer, 2000
25. R. Kaivola. Using compositional preorders in the verification of sliding window protocol. In *Proc. CAV'97*, LNCS 1254, pp. 48–59. Springer, 1997.
26. D.E. Knuth. Verification of link-level protocols. *BIT*, 21:21–36, 1981.
27. T. Latvala. Model checking LTL properties of high-level Petri nets with fairness constraints. In *Proc. APN'01*, LNCS 2075, pp. 242–262. Springer, 2001.
28. J. Loeckx, H.-D. Ehrich, and M. Wolf. *Specification of Abstract Data Types.* Wiley/Teubner, 1996.
29. E. Madelaine and D. Vergamini. Specification and verification of a sliding window protocol in Lotos. In *Proc. FORTE'91*, IFIP Transactions, pp. 495-510. North-Holland, 1991.
30. S. Mauw and G.J. Veltink. A process specification formalism. *Fundamenta Informaticae*, 13(2):85–139, 1990.
31. A. Middeldorp. Specification of a sliding window protocol within the framework of process algebra. Report FVI 86-19, University of Amsterdam, 1986.
32. K. Paliwoda and J.W. Sanders. An incremental specification of the sliding-window protocol. *Distributed Computing*, 5:83–94, 1991.
33. D.M.R. Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conference*, LNCS 104, pp. 167–183. Springer, 1981.
34. J.L. Richier, C. Rodriguez, J. Sifakis, and J. Voiron. Verification in Xesar of the sliding window protocol. In *Proc. PSTV'87*, pp. 235–248. North-Holland, 1987.
35. C. Röckl and J. Esparza. Proof-checking protocols using bisimulations. In *Proc. CONCUR'99*, LNCS 1664, pp. 525–540. Springer, 1999.
36. V. Rusu. Verifying a sliding-window protocol using PVS. In *Proc. FORTE'01*, Conference Proceedings 197, pp. 251-268. Kluwer, 2001.
37. A.A. Schoone. *Assertional Verification in Distributed Computing.* PhD thesis, Utrecht University, 1991.
38. M.A. Smith and N. Klarlund. Verification of a sliding window protocol using IOA and MONA. In *Proc. FORTE/PSTV'00*, pp. 19–34. Kluwer, 2000.
39. J.L.A. van de Snepscheut. The sliding window protocol revisited. *Formal Aspects of Computing*, 7(1):3–17, 1995.
40. K. Stahl, K. Baukus, Y. Lakhnech, and M. Steffen. Divide, abstract, and model-check. In *Proc. SPIN'99*, LNCS 1680, pp. 57–76. Springer, 1999.
41. N.V. Stenning. A data transfer protocol. *Computer Networks*, 1(2):99–110, 1976.
42. A.S. Tanenbaum. *Computer Networks.* Prentice Hall, 1981.
43. F.W. Vaandrager. Verification of two communication protocols by means of process algebra. Report CS-R8608, CWI, Amsterdam, 1986.
44. J.J. van Wamel. A study of a one bit sliding window protocol in ACP. Report P9212, University of Amsterdam, 1992.