A Correctness Proof of a One-bit Sliding Window Protocol in μCRL

M.A. Bezem and J.F. Groote,

Utrecht University, Department of Philosophy, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

We model a one-bit sliding window protocol and prove that its external behaviour is a bi-directional buffer of capacity 2. The proof is given in μCRL, which is a process algebra extended with data. Due to the abundant parallelism in this protocol, the behaviour is quite complicated. The complexity has been mastered by explicitly identifying invariants and foci of cones in the protocol. Both concepts seem promising as tools for the verification of larger and more complex protocols.

1. INTRODUCTION

Sliding window protocols are widely used in data communication networks. These protocols make efficient use of data communication media with long transfer delays. Many data packets may be sent from the sender to the receiver before a data packet is acknowledged. As a consequence there may be many data packets and acknowledgements simultaneously floating around in the protocol. Sliding window protocols have a high degree of parallelism.

It is not an easy task to develop correct parallel systems. This gets even harder when the degree of parallelism increases, because then the number of states of the system increases more than proportionally. Mathematical and logical techniques are required to cope with the sheer complexity of such systems; too many — even small systems — turn out to be erroneous. A striking example is the third sliding window protocol in [13]. This protocol deadlocks (see [5] and [3] for an explanation). Several techniques have been developed to prove that parallel systems operate correctly, among which there are I/O-automata [9], Hoare Logics [11] and process algebras [1, 8, 10].

The technique adopted in this paper is process algebra in the ACP style [1]. More specifically, we are using an extension of process algebra with data which is called μCRL [6]. The reason for taking process algebra is that it is designed to deal with parallellism and, in particular, abstraction. It allows for algebraic, semantical and logical approaches. This contrasts with the other techniques. I/O-automata mainly employ the semantical approach and assertional methods are based on Hoare Logic. We think that process algebra has the capability to incorporate the effective aspects of existing formalisms, and to add its own. The current verification provides positive evidence for this standpoint.

In this paper we provide a correctness proof for a bi-directional sliding window protocol that has buffer size 1. Since only one auxiliary bit is required to ensure reliable transmission of data, it is called the *One-bit Sliding Window Protocol*. Although several verifications of one-bit sliding window protocols exist [4, 12, 14, 15] this particular verification is interesting for at least three reasons.

First, the protocol has been formulated such that the external behaviour of the protocol exactly matches that of a bi-directional buffer with capacity 2. We believe that this should be one of the major concerns of a developer of a protocol. Only when a protocol has a 'pleasant' external behaviour, it can easily be used within larger systems without introducing unexpected malbehaviour. The internal behaviour of the protocol can of course be rather complicated in order to achieve the external behaviour with optimal use of resources. The external behaviours of the sliding window protocols in [4, 14, 15] are more complicated than the one presented here.

Second, the protocol has a high degree of internal parallelism. Most protocols that have been studied in the setting of process algebra have an inherent sequential structure. This includes previous process algebraic verifications of the sliding window protocols. As a consequence, the state spaces of these protocols are relatively small. Typically, alternating bit protocols have about 20 states (depending on how states are counted) and this allows for an exhaustive algebraic characterization of the state space. The one-bit sliding window protocol as described in this paper has an estimated number of states of about 10000. In order to cope with this, we use two techniques that we believe to be important for the practice of process algebraic verification. The first is the use invariants in process algebra with data [2]. The second technique is the identification of structures in the state space of the protocol, called *cones* and *foci*.

The third reason why we find the current verification interesting is that it embodies a structure of proof that seems more general, and different from existing verifications. The description of the protocol is first expanded

to a linear equation, using some *ad hoc* optimizations wherever possible. In a straightforward way the state space induced by the linear equation is restricted using an invariant. The invariant allows to make convenient substitutions, during the further course of calculations. Then we prove that a slightly adapted version of the bi-directional buffer of size 2 is a solution of the obtained linear equation. This reduces to straightforward calculations if a case distinction is made between the states that are focus points of cones, and the states that are not.

From the current experiment we have drawn the conclusion that verification of more complicated systems in process algebra is possible, as the current paper provides techniques that enable to characterize properties of state spaces, without the need to give an explicit enumeration of it. Moreover, it hints towards a more modularized approach to protocol verification: find invariants, identify the focus points and use these to prove the correctness at last.

The current verification has been spelled out in some detail in this document, with almost all detailed calculation steps provided. As a consequence the verification is rather lengthy. On the one hand, this is worrysome. As the calculations are long it is hard to obtain the core of the proof of correctness out of the actual proof. On the other hand, there is no real problem in the length of the proof. The calculations are straightforward after the outline of the proof has been provided. Most calculations could have been omitted as they are trivial. We have chosen not to do so, as the kind of calculations is fairly new.

2. THE PROTOCOL AND ITS EXTERNAL BEHAVIOUR

We describe the one-bit sliding window protocol (OSWP) in μCRL. The language μCRL is defined in [6]. It consists of the main process algebra primitives in the style of ACP [1] together with a straightforward extension with data. For an explanation of μCRL see for instance [7]. The proof of the correctness of the OSWP is given in the style of [7], but we have additionally listed the axioms that have been used in Appendix A. The data types of a general kind that have been used and that are of a general character have been listed in Appendix B. The overall structure of the OSWP is depicted in Figure 1. There are two unreliable channels that transfer data frames and two S/R components at both sides that send and receive data via the channels. The protocol simultaneously transfers data from left to right and from right to left.

We have a standard sort **Bool** with two elements t and f and the usual boolean operations. The data elements to be transferred are provided by some given sort D. In order to transmit data we require a data type bit which contains two elements e_0 and e_1 and an invertor inv. Furthermore, we use a data type *frame*

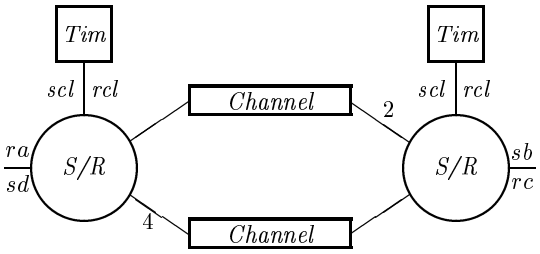

FIGURE 1. The structure of the one-bit sliding window protocol

which contains the elements to be transferred by the data channel. Each frame contains a data element d from D and two bits. The first bit is the alternating bit which is attached to successive d's, and the second bit is an 'piggy backed' acknowledgement. For each data type E there are operations $eq : E \times E \to \mathbf{Bool}$ and $if : \mathbf{Bool} \times E \times E \to E$ representing, respectively, equality on the data type and if-then-else.

sort $D, bit, frame$
func $e_0, e_1 : bit$
$inv : bit \to bit$
$\langle \cdot, \cdot, \cdot \rangle : D \times bit \times bit \to frame$
$dat : frame \to D$
$bit_1, bit_2 : frame \to bit$
$f_{dum} : frame$
var $d : D, b_1, b_2 : bit$
rew $inv(e_0) = e_1$
$inv(e_1) = e_0$
$dat(\langle d, b_1, b_2 \rangle) = d$
$bit_1(\langle d, b_1, b_2 \rangle) = b_1$
$bit_2(\langle d, b_1, b_2 \rangle) = b_2$

We first provide an intuitive description of an S/R component. It reads data via gate ra and sends frames via gate s. It receives acknowledgement frames via r_4 and it delivers data elements via sb. It continuously receives time-outs from a timer Tim via rcl. The parameters of an S/R component have the following intended meaning. Variable p contains the bit of the last frame sent. Variable q contains the bit of the last frame received. The boolean *ready* is true iff a datum can be read from an outside user, i.e. if its internal buffer is free. The boolean *rec* is true iff the receiver does not have a value to be transferred to a receiving instance. The third boolean *sts* is true in case the protocol should not send a frame into the channel; false in case there is a reason to send a frame (after a timeout or after reading a datum to be transmitted). The variable d represents the datum to be transmitted and e the datum which is received.

proc $S(ready, rec, sts{:}\mathbf{Bool}, d, e{:}D, p, q{:}bit) =$
$\sum_{d:D} ra(d)\, S(\mathsf{f}, rec, \mathsf{f}, d, e, inv(p), q) \lhd ready \rhd \delta +$

$\delta \lhd rec \rhd sb(e)\, S(ready, \mathsf{t}, sts, d, e, p, q)+$
$rcl\, S(ready, rec, \mathsf{f}, d, e, p, q)+$
$\sum_{f:D,b_1,b_2:bit} r_4(\langle f, b_1, b_2\rangle)$
$(S(eq(b_2,p), \mathsf{f}, sts, d, f, p, inv(q))$
$\lhd rec \wedge eq(b_1, inv(q)) \rhd$
$S(eq(b_2,p), rec, sts, d, e, p, q))+$
$\delta \lhd sts \rhd s(\langle d, p, q\rangle)\, S(ready, rec, \mathsf{t}, d, e, p, q)$

The following equation defines a simple timer. It just generates time-outs all the time. In fact time-outs are not needed in every state of the protocol, but they do never harm. For reasons of simplicity we let the timer generate time-outs independently of the state of the protocol, although this makes the protocol less efficient.

proc $Tim = scl\, Tim$

The channels are defined as follows. We have chosen for a description where the state of the channel is explicitly recorded using status flags of the type *status*, as this is convenient in the expansion of the protocol. If the status is *read*, then the channel can only read a new frame. If the status is *del*, then the channel must deliver a frame. If the status is *choice*, then the channel has two options: either to go to the status *del*, after which the frame in the channel will be delivered, or to go to the status *read*, which means that the current frame in the channel will be lost.

sort $status$
func $read, choice, del : status$
$eq_{read}, eq_{choice}, eq_{del} : status \to \mathbf{Bool}$
rew $eq_{read}(read) = \mathsf{t}$
$eq_{read}(choice) = \mathsf{f}$
$eq_{read}(del) = \mathsf{f}$
$eq_{choice}(read) = \mathsf{f}$
$eq_{choice}(choice) = \mathsf{t}$
$eq_{choice}(del) = \mathsf{f}$
$eq_{del}(read) = \mathsf{f}$
$eq_{del}(choice) = \mathsf{f}$
$eq_{del}(del) = \mathsf{t}$

proc $C(f{:}frame, st{:}status) =$
$\sum_{f:F} r(f)\, C(f, choice) \lhd eq_{read}(st) \rhd \delta +$
$(i\, C(f, del) + i\, C(f, read)) \lhd eq_{choice}(st) \rhd \delta +$
$s_2(f)\, C(f, read) \lhd eq_{del}(st) \rhd \delta$

Combining these elements yields a bi-directional OSWP-protocol.

act rcl, scl, i, r
$s, c, r_2, s_2, c_2, r_4, s_4, c_4 : frame$
$ra, rc, sb, sd : D$
comm $rcl|scl = i, r|s = c, r_2|s_2 = c_2, r_4|s_4 = c_4$
proc $OSWP(d_1, e_1, d_2, e_2{:}D, p_1, p_2{:}Bit) =$
$\tau_I \partial_H(\rho_{R_1}(\partial_{H'}(S(\mathsf{t},\mathsf{t},\mathsf{t}, d_1, e_1, p_1, p_2) \parallel$
$Tim \parallel C(f_{dum}, read))) \parallel$
$\rho_{R_2}(\partial_{H'}(S(\mathsf{t},\mathsf{t},\mathsf{t}, d_2, e_2, p_2, p_1) \parallel$
$Tim \parallel C(f_{dum}, read))))$

where $I = \{c, c_2, c_4, i\}$, $H = \{r_2, s_2, r_4, s_4\}$, $H' = \{r, s, rcl, scl\}$, $R_1 = \{sb \to sd\}$ and $R_2 = \{ra \to rc, s_2 \to s_4, r_4 \to r_2\}$.

The external behaviour of the protocol should be that of a bidirectional queue DQ of size 2. The data type $Queue$ is specified in Appendix B.

proc $Q(b{:}Queue, n{:}nat) =$
$\sum_{d:D}(ra(d)\, Q(in(d,b), n) \lhd size(b) < n \rhd \delta) +$
$sd(toe(b))\, Q(untoe(b), n) \lhd size(b) > 0 \rhd \delta$

$DQ(b_1, b_2{:}Queue, n{:}nat) =$
$\rho_{\{sd \to sb\}}(Q(b_1, n)) \parallel \rho_{\{ra \to rc\}}(Q(b_2, n))$

The following theorem states the correctness of the one-bit sliding window protocol.

THEOREM 2.1 *For all* d_1, e_1, d_2, $e_2{:}D$, p_1, $p_2{:}Bit$

$$OSWP(d_1, e_1, d_2, e_2, p_1, p_2) = DQ(\emptyset, \emptyset, 2)$$

This theorem is repeated and proved as Theorem 8.2.

3. LINEARIZATION

The remainder of this article is devoted to proving Theorem 2.1. In the current section *OSWP* is related to a linear equation. In the next section we prove some invariant properties. In Section 6. we introduce the notions *cone* and *focus* in relation to the OSWP. In Section 7. we show that a bi-directional buffer with an idle loop satisfies the linear equation provided in the current section. This is the longest part of the proof. In the final Section 8. it is then straightforward to show that the bi-directional buffer behaves the same as a bi-directional buffer with an idle loop. With this result the proof of Theorem 2.1 follows directly.

We first expand the OSWP protocol. In order to do so, we define the auxiliary processes D and E. For processes with long parameter lists (typically E below), we use the following notational convention: in a recursive call, we only make explicit those parameters that change with respect to the original call. For example, d/d_1 means that d is the new value for parameter d_1, $inv(p_1)/p_1$ means that parameter p_1 is inverted. Parameters which are not explicitly shown do not change.

proc $D(ready, rec, sts{:}\mathbf{Bool}, d, e{:}D, p, q{:}bit,$
$f{:}frame, st{:}status) =$
$\partial_{H'}(S(ready, rec, sts, d, e, p, q) \parallel Tim \parallel C(f, st))$

$E(rdy_1, rec_1, sts_1{:}\mathbf{Bool}, d_1, e_1{:}D, p_1, q_1{:}bit,$
$f_1{:}frame, st_1{:}status, rdy_2, rec_2, sts_2{:}\mathbf{Bool},$
$d_2, e_2{:}D, p_2, q_2{:}bit, f_2{:}frame, st_2{:}status) =$
$\partial_H(\rho_{R_1}(D(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1))$
$\parallel \rho_{R_2}(D(rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2)))$

LEMMA 3.1 *For all* rdy, rdy_1, rdy_2, rec, rec_1, rec_2, sts, sts_1, $sts_2{:}\mathbf{Bool}$, d, d_1, d_2, e, e_1, $e_2{:}D$, p, p_1, p_2,

q, q_1, q_2:bit, f, f', f_1, f_2:frame, st, st_1, st_2:status the equations in Table 1 hold.

Proof Straightforward. □

4. INVARIANT

We define the following predicate in order to establish some invariant properties between the variables in E. In Section 7. this invariant will be used. The operator $\neg$ binds stronger than $\wedge$, which binds stronger than $\rightarrow$.

$$
\begin{aligned}
&I(rdy, d, e, p, q, f, f') = \\
&(rdy \rightarrow eq(bit_1(f), p) \wedge \\
&\qquad eq(p, q) \wedge eq(e, d) \wedge eq(bit_2(f'), p)) \wedge \\
&(\neg rdy \wedge eq(bit_1(f), p) \rightarrow eq(d, dat(f))) \wedge \\
&(\neg rdy \wedge eq(p, q) \rightarrow eq(bit_1(f), p) \wedge eq(e, d)) \wedge \\
&(\neg rdy \wedge eq(bit_2(f'), p) \rightarrow eq(p, q))
\end{aligned}
$$

To support the intuition that

$$I(rdy_1, d_1, e_2, p_1, q_2, f_1, f_2)$$

is an invariant of E, consider the following trace of E. Initially, rdy_1 is true and p_1 equals q_2. After an action $ra(d)$, rdy_1 becomes false, d_1 becomes d, and p_1 is inverted. Via $c(\langle d, p_1, q_1\rangle)$ the frame containing d is communicated to the channel. This is marked by $eq(bit_1(f_1), p_1) \wedge eq(d, dat(f_1))$ becoming true. Via $c_2(f_1)$ the frame is delivered, at which bit q_2 is inverted, thus matching p_1 again, and e_2 is made equal to d_1. Finally, the receipt of the frame is acknowledged by communicating a frame f_2 back, whose acknowledging bit $bit_2(f_2)$ matches p_1. The invariant describes the increase of the match between corresponding bits and corresponding data during the part of the execution in which rdy_1 is false, culminating in a perfect match $eq(bit_1(f_1), p_1) \wedge eq(p_1, q_2) \wedge eq(e_2, d_1)) \wedge eq(bit_2(f_2), p_1)$ marking the completion of one cycle in the execution. Thereafter rdy_1 can safely be made true, expressing readyness to enter a new cycle while maintaining the invariant. Of course the above description concerns only one direction; the other direction is completely symmetric.

LEMMA 4.1 *Both $I(rdy_1, d_1, e_2, p_1, q_2, f_1, f_2)$ and $I(rdy_2, d_2, e_1, p_2, q_1, f_2, f_1)$ are invariants of E.*

Proof We follow the definition of an invariant in [2, Corollary 3.9]. As both invariants are symmetric, we only prove $I(rdy_1, d_1, e_2, p_1, q_2, f_1, f_2)$ an invariant. So, assume $I(rdy_1, d_1, e_2, p_1, q_2, f_1, f_2)$. We must verify that the invariant is maintained during the course of the process. The following five cases are non trivial. The numbers refer to the number of the summand in the definition of E. Most of the summands are conditionals and the condition may be used in the proof of the invariant. However, we only mention conditions that are actually used in the proof below.

(1) $rdy_1 \rightarrow I(\mathrm{f}, d, e_2, inv(p_1), q_2, f_1, f_2)$. As rdy_1, it follows from the invariant that $eq(p_1, q_2)$, $eq(bit_1(f_1), p_1)$ and $eq(bit_2(f_2), p_1)$. Hence, $\neg eq(bit_1(f_1), inv(p_1))$ and $\neg eq(bit_2(f_2), inv(p_1))$. The invariant $I(\mathrm{f}, d, e_2, inv(p_1), q_2, f_1, f_2)$ is a direct consequence of these three observations.

(3) $I(eq(bit_2(f_2), p_1), d_1, e_2, p_1, q_2, f_1, f_2)$. This invariant of course holds if $rdy_1 \leftrightarrow eq(bit_2(f_2), p_1)$. So, assume $rdy_1 \leftrightarrow \neg eq(bit_2(f_2), p_1)$. Then, as $rdy_1 \rightarrow eq(bit_2(f_2), p_1)$ by the invariant, we have $\neg rdy_1$. From the invariant it follows that $eq(p_1, q_2)$ and thus, $eq(bit_1(f_1), p_1)$ and $eq(e_2, d_1)$. Hence $I(eq(bit_2(f_2), p_1), d_1, e_2, p_1, q_2, f_1, f_2)$ holds.

(6) $I(rdy_1, d_1, e_2, p_1, q_2, \langle d_1, p_1, q_1\rangle, f_2)$ follows trivially after substitution.

(9) We must show that

$$
\begin{aligned}
&eq(bit_1(f_1), inv(q_2)) \rightarrow \\
&I(rdy_1, d_1, dat(f_1), p_1, inv(q_2), f_1, f_2).
\end{aligned}
$$

As $eq(bit_1(f_1), inv(q_2))$, it follows that $\neg eq(bit_1(f_1), q_2)$. So, it follows, using the invariant, that $\neg rdy_1$, $\neg eq(p_1, q_2)$ and $\neg eq(bit_2(f_2), p_1)$. Hence, $eq(p_1, inv(q_2))$. So, it follows that $eq(bit_1(f_1), p_1)$ and hence, again using the invariant, $eq(d_1, dat(f_1))$. It follows that $I(rdy_1, dat(f_1), p_1, inv(q_2), f_1, f_2)$.

(12) $I(rdy_1, d_1, e_2, p_1, q_2, f_1, \langle d_2, p_2, q_2\rangle)$. This case follows trivially, after substitution.

□

5. A BI-DIRECTIONAL BUFFER WITH AN IDLE LOOP

In this short section we define a bi-directional two-bit buffer B with an idle loop. Using KFAR to eliminate the idle loop, B can easily be seen to be equal to the bi-directional two-bit buffer $DQ(\emptyset, \emptyset, 2)$ specifying the external behaviour of the OSWP. On the other hand, B can be proved equal to the OSWP, where the idle loop is used to subsume some internal loops of the OSWP. Thus B is an important auxiliary process that serves as an intermediate between the OSWP and $DQ(\emptyset, \emptyset, 2)$.

In the definition of B, buffer positions are represented using explicit variables: d_1, e_2 represent the buffer content from left to right (d_2, e_1 the buffer from right to left). If a buffer contains only one element, this element is present in both parameters d_1, e_2 (d_2, e_1).

$$
\begin{aligned}
\textbf{proc}\ \ & B(d_1, e_2, size_1, d_2, e_1, size_2) = \\
& \textstyle\sum_{d:D} ra(d)\, B(d, d, 1, d_2, e_1, size_2) \\
& \qquad \triangleleft eq(size_1, 0) \triangleright \delta + \\
& \textstyle\sum_{d:D} ra(d)\, B(d, e_2, 2, d_2, e_1, size_2) \\
& \qquad \triangleleft eq(size_1, 1) \triangleright \delta + \\
& sb(e_2)\, B(d_1, d_1, size_1 - 1, d_2, e_1, size_2) \\
& \qquad \triangleleft size_1 > 0 \triangleright \delta + \\
& \textstyle\sum_{d:D} rc(d)\, B(d_1, e_2, size_1, d, d, 1) \\
& \qquad \triangleleft eq(size_2, 0) \triangleright \delta + \\
& \textstyle\sum_{d:D} rc(d)\, B(d_1, e_2, size_1, d, e_1, 2) \\
& \qquad \triangleleft eq(size_2, 1) \triangleright \delta + \\
& sd(e_1)\, B(d_1, e_2, size_1, d_2, d_2, size_2 - 1)
\end{aligned}
$$

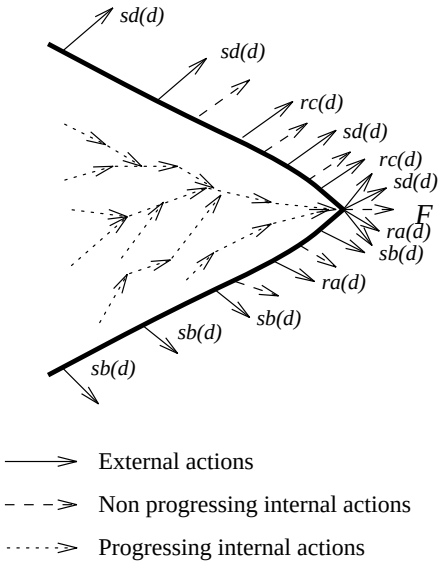

FIGURE 2. A cone and focus point for the OSWP

$$\triangleleft size_2 > 0 \triangleright \delta + i\,B(d_1, e_2, size_1, d_2, e_1, size_2)$$

6. CONES AND FOCI

At this point we use an explicit distinction between so-called focus and non focus points of cones. As far as we know these heuristic notions have not been identified anywhere else in the literature. Therefore we explain these notions first. We should stress that the notion of cones and foci are comparable to notions as invariants and induction hypotheses. They are in general not uniquely determined, and there are often alternatives that also work well.

Focus points of a distributed system are specific points in the state space of such a system, where all external behaviour can immediately, i.e. without first doing internal steps be performed. For the OSWP the focus points are those points where as much as possible of the data has been transferred from the senders to the corresponding receivers, and all received data have been acknowledged. Given the focus points of the protocol, we can divide transitions corresponding to internal behaviour into two classes: those that progress towards a focus point, and those that don't. Generally, finding focus points and a natural distinction between progressing and non progressing internal actions is done simultaneously. A typical example of a progressing internal step in the OSWP is a time-out which causes a lost frame to be retransmitted. A typical example of non progressing internal action is a time-out which causes a frame to be retransmitted that arrived already. A characteristic of a focus point is that in such a state all internal behaviour is non progressing. The *cone* of a focus point F is the set of states which can be reached in a finite number of progressing internal steps and that, the closer to F, the more external behaviour is possible. Note that cycles of progressing internal actions should be excluded as otherwise not every progressing internal step would bring one closer to the focus point. Note furthermore that all points in the cone of F must be weakly bisimulation equivalent.

In Figure 2 we have depicted a focus point F of the OSWP and its corresponding cone. Note the increase in possible external behaviour while progressing towards focus point F in Figure 2: first received data can be delivered $(sb(d), sd(d))$, and in a later stage also new data can be read $(ra(d), rc(d))$.

For the OSWP we characterize the focus points using the following focus condition *FC*. We provide some intuition first, restricting attention to one direction. The protocol is in a focus point in one of the following three situations: no datum neither left nor right, no datum left but a datum right, a datum left as well as right. Note that a datum left and no datum right is excluded in a focus point. In that case the protocol must first send the datum from the sender to the receiver. In all three situations we must exclude any progressing internal steps, for which we will use sts_1 and st_1.

Ignoring sts_1 and st_1 first, we get

$$rdy_1 \wedge eq(p_1, q_2) \wedge rec_2$$
$$rdy_1 \wedge eq(p_1, q_2) \wedge \neg rec_2$$
$$\neg rdy_1 \wedge \neg eq(p_1, q_2) \wedge \neg rec_2$$

By the invariant $rdy_1 \rightarrow eq(p_1, q_2)$, so the disjunction of the three conjunctions above simplifies to:

$$rdy_1 \vee (\neg eq(p_1, q_2) \wedge \neg rec_2)$$

The values of the variables sts_1, sts_2, st_1 and st_2 in focus points are not very essential. If $sts_1 = \mathsf{f}$, this means that a timer signalled that a retransmission should take place. Dispatching the retransmission is considered as 'progress'. So, handing over a frame to, the channel and delivering the frame at the receivers are considered progressing internal actions. Consequently, the states where $sts_1 = \mathsf{f}$ or $st_1 \neq read$ cannot be focus points. Only if $st_1 = read$ and $sts_1 = \mathsf{t}$ no remaining message need to be transferred at the receiver and so we additionally require that in a focus point $st_1 = read$ and $sts_1 = \mathsf{t}$. Doing the same for st_2 and sts_2, the focus points are those points satisfying the following *focus condition FC*:

$$\begin{aligned} FC = &(\neg rec_2 \vee rdy_1) \wedge (\neg eq(p_1, q_2) \vee rdy_1) \wedge \\ &(\neg rec_1 \vee rdy_2) \wedge (\neg eq(p_2, q_1) \vee rdy_2) \wedge \\ &eq_{read}(st_1) \wedge eq_{read}(st_2) \wedge sts_1 \wedge sts_2. \end{aligned}$$

Now that we have established the focus condition, we make the distinction between progressing and non progressing transitions explicit. In the case of the OSWP, all transitions labelled by actions c, c_2, c_4 are progressing transitions. Transitions labelled by i (time-outs) can be both progressing and non progressing. Action i is non progressing if it represents the loss of a datum

in a channel, or a time-out when there is nothing to retransmit. In all other cases i is progressing. Assume a condition C distinguishes between states in which a transition i is progressing and states in which not. The idea is then to write $i\ldots = i\ldots \lhd C \rhd \delta + i\ldots \lhd \neg C \rhd \delta$. Now the first i on the right hand side labels progressing transitions and the the second i labels non progressing transitions. The first i will below be renamed to τ_i to stress that this i will be hidden in the sequel. The other i's will be hidden in the last part of the calculation. Actually, we use $i\ldots = i\ldots + i\ldots \lhd \neg C \rhd \delta$ for technical convenience.

Given the distinction between progressing and non progressing transitions, we will define in the next section a partial abstraction by which progressing transitions are replaced by τ's. It is important to note that in every state one can do only finitely many progressing steps before arriving at a focus point. We prepare the partial abstraction by defining a process F (see Table 2) which results from E by replacing actions c, c_2, c_4 and i labelling *progressing* transitions by $\tau_c, \tau_{c_2}, \tau_{c_4}$ and τ_i, respectively. The partial abstraction $\tau_{\{\tau_c,\tau_{c_2},\tau_{c_4},\tau_i\}}F$ will be used in the next section. Note that the τ_i's etc. are no silent steps but visible actions. Define $R = \{\tau_c \to c, \tau_{c_2} \to c_2, \tau_{c_4} \to c_4, \tau_i \to i\}$. The following lemma simply states that undoing the replacements in F yields E and that c, c_2, c_4 do not occur any longer in F.

LEMMA 6.1 *Let* rdy_1, rdy_2, rec_1, rec_2, sts_1, sts_2:**Bool**, d_1, d_2, e_1, e_2:D, p_1, p_2, q_1, q_2:*bit*, f_1, f_2:*frame*, st_1, st_2:*status*. *We have*

- $E(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2) = \rho_R(F(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2))$,
- $\tau_{\{c,c_2,c_4\}}(F(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1 st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2)) = F(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2)$.

Proof As mentioned above the τ_i's etc. are no silent steps but visible actions. Hence the recursion equations defining E and F are trivially guarded. Now the first part of the lemma follows immediately using RSP. The second part is obvious since c, c_2, c_4 do not occur in F □

7. MAIN LEMMA USING A PARTIAL ABSTRACTION

The most substantial step in the correctness proof of the one-bit sliding window protocol is provided by the following lemma. It states, roughly, that the partial abstraction $\tau_{\{\tau_c,\tau_{c_2},\tau_{c_4},\tau_i\}}F$ equals the buffer B from Section 5., provided that the invariant and the focus condition are taken into account. This lemma is a direct application of the Concrete Invariant Corollary which has been defined in [2]. The proof of the lemma is lengthy. This has two reasons. The first one is that all calculations are spelled out in detail. Actually, the proof of this lemma becomes an easy exercise, once the recursion equation for X is provided and is shown guarded, provided the reader has some skill in process algebraic calculations. As we think that most readers are not skilled in this respect, we provide the full proof. Another reason for the length of the proof is a more serious one, and seems to be inherent to protocols. The description of the expanded protocol requires 14 lines. The recursion equation for X necessarily resembles this equation. When showing that certain terms are a solution for X, terms get a size which is proportional to the defining equation for X. Every calculation step requires repeating these large terms. Therefore, the proof becomes lengthy.

Before providing the lemma, we stress once more the fact that we use both the focus condition and the invariant in this lemma.

LEMMA 7.1 *For all* rdy_1, rdy_2, rec_1, rec_2, sts_1, sts_2:**Bool**, d_1, d_2, e_1, e_2:D, p_1, p_2, q_1, q_2:*bit*, f_1, f_2:*frame*, st_1, st_2:*status it holds that*

$$I(rdy_1, rec_1, d_1, e_2, p_1, q_2, f_1, f_2, st_1) \wedge I(rdy_2, rec_2, d_2, e_1, p_2, q_1, f_2, f_1, st_2)$$

implies

$$B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \lhd FC \rhd \tau B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1))$$

is equivalent to

$$\tau_{\{\tau_c,\tau_{c_2},\tau_{c_4},\tau_i\}}(F(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2))).$$

where

$$\begin{array}{ll} \textbf{func} & det : \textbf{Bool} \times \textbf{Bool} \to Nat \\ \textbf{rew} & det(\mathsf{t},\mathsf{t}) = 0 \\ & det(\mathsf{t},\mathsf{f}) = 1 \\ & det(\mathsf{f},\mathsf{t}) = 1 \\ & det(\mathsf{f},\mathsf{f}) = 2 \end{array}$$

Proof This fact is proven using the Invariant Corollary from [2, Corollary 3.9]. Let Φ_X be the process operator corresponding to recursion equation in Table 3. The only difference with the defining equations of E and F is that the τ_i's etc. have become silent steps τ. As a consequence, X has the same invariants as E, in particular

$$I(rdy_1, rec_1, d_1, e_2, p_1, q_2, f_1, f_2, st_1) \wedge I(rdy_2, rec_2, d_2, e_1, p_2, q_1, f_2, f_1, st_2)$$

is an invariant of X according to Lemma 4.1.

- $C(f, read) = C(f', read)$.
- $D(rdy, rec, sts, d, e, p, q, f, st) =$
  $\sum_{d:D} ra(d)\, D(\mathsf{f}, rec, \mathsf{f}, d, e, inv(p), q, f, st) \lhd rdy \rhd \delta +$
  $\delta \lhd rec \rhd sb(e)\, D(rdy, \mathsf{t}, sts, d, e, p, q, f, st) +$
  $\sum_{f:D,b_1,b_2:bit} r_4(\langle f, b_1, b_2\rangle)\, (D(eq(b_2, p), \mathsf{f}, sts, d, f, p, inv(q), f, st)$
  $\lhd rec \wedge eq(b_1, inv(q)) \rhd D(eq(b_2, p), rec, sts, d, e, p, q, f, st)) +$
  $i\, D(rdy, rec, \mathsf{f}, d, e, p, q, f, st) +$
  $(i\, D(rdy, rec, sts, d, e, p, q, f, del) + i\, D(rdy, rec, sts, d, e, p, q, f, read)) \lhd eq_{choice}(st) \rhd \delta +$
  $c(\langle d, p, q\rangle)\, D(rdy, rec, \mathsf{t}, d, e, p, q, \langle d, p, q\rangle, choice) \lhd eq_{read}(st) \wedge \neg sts \rhd \delta +$
  $s_2(f)\, D(rdy, rec, sts, d, e, p, q, f, read) \lhd eq_{del}(st) \rhd \delta$
- $E(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2) =$
  $\sum_{d:D} ra(d)\, E(\mathsf{f}/rdy_1, \mathsf{f}/sts_1, d/d_1, inv(p_1)/p_1) \lhd rdy_1 \rhd \delta +$
  $\delta \lhd rec_1 \rhd sd(e_1)\, E(\mathsf{t}/rec_1) +$
  $c_4(f_2)(E(eq(bit_2(f_2), p_1)/rdy_1, \mathsf{f}/rec_1, dat(f_2)/e_1, inv(q_1)/q_1, read/st_2)$
  $\lhd rec_1 \wedge eq(bit_1(f_2), inv(q_1)) \rhd E(eq(bit_2(f_2), p_1)/rdy_1, read/st_2)) \lhd eq_{del}(st_2) \rhd \delta +$
  $i\, E(\mathsf{f}/sts_1) +$
  $(i\, E(del/st_1) + i\, E(read/st_1)) \lhd eq_{choice}(st_1) \rhd \delta +$
  $c(\langle d_1, p_1, q_1\rangle)\, E(\mathsf{t}/sts_1, \langle d_1, p_1, q_1\rangle/f_1, choice/st_1) \lhd eq_{read}(st_1) \wedge \neg sts_1 \rhd \delta +$
  $\sum_{d:D} rc(d)\, E(\mathsf{f}/rdy_2, \mathsf{f}/sts_2, d/d_2, inv(p_2)/p_2) \lhd rdy_2 \rhd \delta +$
  $\delta \lhd rec_2 \rhd sb(e_2)\, E(\mathsf{t}/rec_2) +$
  $c_2(f_1)\, (E(read/st_1, eq(bit_2(f_1), p_2)/rdy_2, \mathsf{f}/rec_2, dat(f_1)/e_2, inv(q_2)/q_2)$
  $\lhd rec_2 \wedge eq(bit_1(f_1), inv(q_2)) \rhd E(read/st_1, eq(bit_2(f_1), p_2)/rdy_2)) \lhd eq_{del}(st_1) \rhd \delta +$
  $i\, E(\mathsf{f}/sts_2) +$
  $(i\, E(del/st_2) + i\, E(read/st_2)) \lhd eq_{choice}(st_2) \rhd \delta +$
  $c(\langle d_2, p_2, q_2\rangle)\, E(\mathsf{t}/sts_2, \langle d_2, p_2, q_2\rangle/f_2, choice/st_2) \lhd eq_{read}(st_2) \wedge \neg sts_2 \rhd \delta$

TABLE 1. Properties of C, D and E

proc $F(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2) =$
$\sum_{d:D} ra(d)\, F(\mathsf{f}/rdy_1, \mathsf{f}/sts_1, d/d_1, inv(p_1)/p_1) \lhd rdy_1 \rhd \delta +$
$\delta \lhd rec_1 \rhd sd(e_1)\, F(\mathsf{t}/rec_1) +$
$\tau_{c_4}\, (F(eq(bit_2(f_2), p_1)/rdy_1, \mathsf{f}/rec_1, dat(f_2)/e_1, inv(q_1)/q_1, read/st_2)$
$\lhd rec_1 \wedge eq(bit_1(f_2), inv(q_1)) \rhd F(eq(bit_2(f_2), p_1)/rdy_1, read/st_2)) \lhd eq_{del}(st_2) \rhd \delta +$
$i\, F(\mathsf{f}/sts_1) +$
$\tau_i\, F(\mathsf{f}/sts_1) \lhd ((\neg eq(p_1, q_2) \wedge rec_2) \vee (eq(p_2, q_1) \wedge \neg rdy_2)) \wedge eq_{read}(st_1) \wedge sts_1 \rhd \delta +$
$(\tau_i\, F(del/st_1) + i\, F(read/st_1)) \lhd eq_{choice}(st_1) \rhd \delta +$
$\tau_c\, F(\mathsf{t}/sts_1, \langle d_1, p_1, q_1\rangle/f_1, choice/st_1) \lhd eq_{read}(st_1) \wedge \neg sts_1 \rhd \delta +$
$\sum_{d:D} rc(d)\, F(\mathsf{f}/rdy_2, \mathsf{f}/sts_2, d/d_2, inv(p_2)/p_2) \lhd rdy_2 \rhd \delta +$
$\delta \lhd rec_2 \rhd sb(e_2)\, F(\mathsf{t}/rec_2) +$
$\tau_{c_2}\, (F(read/st_1, eq(bit_2(f_1), p_2)/rdy_2, \mathsf{f}/rec_2, dat(f_1)/e_2, inv(q_2)/q_2)$
$\lhd rec_2 \wedge eq(bit_1(f_1), inv(q_2)) \rhd F(read/st_1, eq(bit_2(f_1), p_2)/rdy_2)) \lhd eq_{del}(st_1) \rhd \delta +$
$i\, F(\mathsf{f}/sts_2) +$
$\tau_i\, F(\mathsf{f}/sts_2) \lhd ((\neg eq(p_2, q_1) \wedge rec_1) \vee (eq(p_1, q_2) \wedge \neg rdy_1)) \wedge eq_{read}(st_2) \wedge sts_2 \rhd \delta +$
$(\tau_i\, F(del/st_2) + i\, F(read/st_2)) \lhd eq_{choice}(st_2) \rhd \delta +$
$\tau_c\, F(\mathsf{t}/sts_2, \langle d_2, p_2, q_2\rangle/f_2, choice/st_2) \lhd eq_{read}(st_2) \wedge \neg sts_2 \rhd \delta$

TABLE 2. The definition of process F

We have the following three proof obligations:

First we must show that Φ_X is convergent. Note that, due to the replacement of τ_i's by τ's, this fact does not follow from the guardedness of the recursion equations defining E and F.

Second, we must show that the term

$$\lambda rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2.\ \tau_{\{\tau_c,\tau_{c_2},\tau_{c_4},\tau_i\}}(F(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2))$$

satisfies the recursion equation of X, for all values of the parameters satisfying the invariant.

Third, we must show that the parameterised process

$$B_{FC} = \lambda rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2.\ B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \lhd FC \rhd \tau B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1))$$

satisfies the recursion equation of X, for all values of the parameters satisfying the invariant.

The second point follows straightforwardly by applying the hiding axioms, without using the invariant. The first and third point will be dealt with in the Appendices C and D.

8. FINAL CALCULATIONS

In this section we first show how B and the bi-directional queue DQ are related. We use an auxiliary process R for this purpose. Then we collect all obtained results into the proof of Theorem 2.1 (=Theorem 8.2). Define the following process

$$\textbf{proc}\ \ R(d, e, n) = Q(\emptyset, 2) \lhd eq(n, 0) \rhd \delta + Q(in(e, \emptyset), 2) \lhd eq(n, 1) \rhd \delta + Q(in(d, in(e, \emptyset)), 2) \lhd eq(n, 2) \rhd \delta.$$

LEMMA 8.1 *For all d_1, d_2, e_1, d_2:D and $size_1$, $size_2$:Nat we have*

$$\rho_{\{sd\to sb\}}(R(d_1, e_2, size_1)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2)) = \tau_{\{i\}}(B(d_1, e_2, size_1, d_2, e_1, size_2)).$$

Proof We show that both sides of the equation satisfy the following guarded recursive specification.

$$\begin{aligned} Y(d_1, e_2, size_1, d_2, e_1, size_2) = {} & \textstyle\sum_{d:D} ra(d)\, Y(d, d, 1, d_2, e_1, size_2) \lhd eq(size_1, 0) \rhd \delta + \\ & \textstyle\sum_{d:D} ra(d)\, Y(d, e_2, 2, d_2, e_1, size_2) \lhd eq(size_1, 1) \rhd \delta + \\ & sb(e_2)\, Y(d_1, d_1, size_1 - 1, d_2, e_1, size_2) \lhd size_1 > 0 \rhd \delta + \\ & \textstyle\sum_{d:D} rc(d)\, Y(d_1, e_2, size_1, d, d, 1) \lhd eq(size_2, 0) \rhd \delta + \\ & \textstyle\sum_{d:D} rc(d)\, Y(d_1, e_2, size_1, d, e_1, 2) \lhd eq(size_2, 1) \rhd \delta + \\ & sd(e_1)\, Y(d_1, e_2, size_1, d_2, d_2, size_2 - 1) \lhd size_2 > 0 \rhd \delta. \end{aligned}$$

First we show that

$$\lambda d_1, e_2, size_1, d_2, e_2, size_2.\ \rho_{\{sd\to sb\}}(R(d_1, e_2, size_1)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))$$

satisfies the above specification of Y. The proof is in Table 4. It follows immediately with KFAR that

$$\lambda d_1, e_2, size_1, d_2, e_1, size_2.\ \tau_{\{i\}}(B(d_1, e_2, size_1, d_2, e_1, size_2))$$

also satisfies the specification of Y. □

THEOREM 8.2 *For all d_1, d_2, e_1, e_2:D and p_1, p_2:bit it follows that*

$$OSWP(d_1, e_1, d_2, e_2, p_1, p_2) = DQ(\emptyset, \emptyset, 2).$$

Proof We collect all results that we have obtained until now (see Table 5). □

Acknowledgements. The authors are partly supported by the Netherlands Computer Science Research Foundation (SION) with financial support of the Netherlands Organization for Scientific Research (NWO).

REFERENCES

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

[2] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. Technical Report 98, Logic Group Preprint Series, Utrecht University, 1993.

[3] J.J. Brunekreef. A formal specification of three sliding window protocols. Technical Report P9102, Programming Research Group, University of Amsterdam, 1991.

[4] R.A. Groenveld. Verification of a sliding window protocol by means of process algebra. Report P8701, Programming Research Group, University of Amsterdam, 1987.

[5] J.F. Groote. *Process Algebra and Structured Operational Semantics.* PhD thesis, University of Amsterdam, November 1991.

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

[7] J.F. Groote and A. Ponse. μCRL: A base for analysing processes with data. In E. Best and G. Rozenberg, editors, *Proceedings 3rd Workshop on Concurrency and Compositionality, Goslar, GMD-Studien Nr. 191*, pages 125–130. Universität Hildesheim, May 1991.

$$
\begin{aligned}
&X(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2) = \\
&\quad \textstyle\sum_{d:D} ra(d)\, X(\mathsf{f}/rdy_1, \mathsf{f}/sts_1, d/d_1, inv(p_1)/p_1) \lhd rdy_1 \rhd \delta + \\
&\quad \delta \lhd rec_1 \rhd sd(e_1)\, X(\mathsf{t}/rec_1) + \\
&\quad \tau\,(X(eq(bit_2(f_2), p_1)/rdy_1, \mathsf{f}/rec_1, dat(f_2)/e_1, inv(q_1)/q_1, read/st_2) \\
&\qquad \lhd rec_1 \wedge eq(bit_1(f_2), inv(q_1)) \rhd X(eq(bit_2(f_2), p_1)/rdy_1, read/st_2)) \lhd eq_{del}(st_2) \rhd \delta + \\
&\quad i\, X(\mathsf{f}/sts_1) + \\
&\quad \tau\, X(\mathsf{f}/sts_1) \lhd (\neg eq(p_1, q_2) \wedge rec_2) \vee (eq(p_2, q_1) \wedge \neg rdy_2) \wedge eq_{read}(st_1) \wedge sts_1 \rhd \delta + \\
&\quad (\tau\, X(del/st_1) + i\, X(read/st_1)) \lhd eq_{choice}(st_1) \rhd \delta + \\
&\quad \tau\, X(\mathsf{t}/sts_1, \langle d_1, p_1, q_1\rangle/f_1, choice/st_1) \lhd eq_{read}(st_1) \wedge \neg sts_1 \rhd \delta + \\
&\quad \textstyle\sum_{d:D} rc(d)\, X(\mathsf{f}/rdy_2, \mathsf{f}/sts_2, d/d_2, inv(p_2)/p_2) \lhd rdy_2 \rhd \delta + \\
&\quad \delta \lhd rec_2 \rhd sb(e_2)\, X(\mathsf{t}/rec_2) + \\
&\quad \tau\,(X(read/st_1, eq(bit_2(f_1), p_2)/rdy_2, \mathsf{f}/rec_2, dat(f_1)/e_2, inv(q_2)/q_2) \\
&\qquad \lhd rec_2 \wedge eq(bit_1(f_1), inv(q_2)) \rhd X(read/st_1, eq(bit_2(f_1), p_2)/rdy_2)) \lhd eq_{del}(st_1) \rhd \delta + \\
&\quad i\, X(\mathsf{f}/sts_2) + \\
&\quad \tau\, X(\mathsf{f}/sts_2) \lhd (\neg eq(p_2, q_1) \wedge rec_1) \vee (eq(p_1, q_2) \wedge \neg rdy_1) \wedge eq_{read}(st_2) \wedge sts_2 \rhd \delta + \\
&\quad (\tau\, X(del/st_2) + i\, X(read/st_2)) \lhd eq_{choice}(st_2) \rhd \delta + \\
&\quad \tau\, X(\mathsf{t}/sts_2, \langle d_2, p_2, q_2\rangle/f_2, choice/st_2) \lhd eq_{read}(st_2) \wedge \neg sts_2 \rhd \delta
\end{aligned}
$$

TABLE 3. The defining equation for the process X

$$
\begin{aligned}
&\rho_{\{sd\to sb\}}(R(d_1, e_2, size_1)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2)) = \\
&\textstyle\sum_{d:D} ra(d)\,(\rho_{\{sd\to sb\}} Q(in(d, \emptyset), 2) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))) \lhd eq(size_1, 0) \rhd \delta + \\
&\textstyle\sum_{d:D} ra(d)(\rho_{\{sd\to sb\}} Q(in(d, in(e_2, \emptyset)), 2) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2)) \lhd eq(size_1, 1) \rhd \delta + \\
&sd(e_2)\,(\rho_{\{sd\to sb\}} Q(\emptyset, 2)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))) \lhd eq(size_1, 1) \rhd \delta + \\
&sd(e_2)\,(\rho_{\{sd\to sb\}} Q(in(d_1, \emptyset), 2)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))) \lhd eq(size_1, 2) \rhd \delta + \\
&\textstyle\sum_{d:D} ra(d)\,(\rho_{\{sd\to sb\}} Q(in(d, \emptyset), 2) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))) \lhd eq(size_1, 0) \rhd \delta + \\
&\textstyle\sum_{d:D} ra(d)(\rho_{\{sd\to sb\}} Q(in(d, in(e_2, \emptyset)), 2) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2)) \lhd eq(size_1, 1) \rhd \delta + \\
&sd(e_2)\,(\rho_{\{sd\to sb\}} Q(\emptyset, 2)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))) \lhd eq(size_1, 1) \rhd \delta + \\
&sd(e_2)\,(\rho_{\{sd\to sb\}} Q(in(d_1, \emptyset), 2)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))) \lhd eq(size_1, 2) \rhd \delta = \\
&\textstyle\sum_{d:D} ra(d)\,(\rho_{\{sd\to sb\}}(R(d, d, 1)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))) \lhd eq(size_1, 0) \rhd \delta + \\
&\textstyle\sum_{d:D} ra(d)(\rho_{\{sd\to sb\}} R(d, e_2, 2)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2)) \lhd eq(size_1, 1) \rhd \delta + \\
&sd(e_2)\,(\rho_{\{sd\to sb\}} R(d_1, d_1, size_1 - 1)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))) \lhd size_1 > 0) \rhd \delta + \\
&\textstyle\sum_{d:D} ra(d)\,(\rho_{\{sd\to sb\}}(R(d, d, 1)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))) \lhd eq(size_1, 0) \rhd \delta + \\
&\textstyle\sum_{d:D} ra(d)(\rho_{\{sd\to sb\}} R(d, e_2, 2)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2)) \lhd eq(size_1, 1) \rhd \delta + \\
&sd(e_2)\,(\rho_{\{sd\to sb\}} R(d_1, d_1, size_1 - 1)) \parallel \rho_{\{ra\to rc\}}(R(d_2, e_1, size_2))) \lhd size_1 > 0) \rhd \delta
\end{aligned}
$$

TABLE 4. Part of the proof of Lemma 8.1

$$
\begin{aligned}
&DQ(\emptyset,\emptyset,2) = \\
&\rho_{\{sd\to sb\}}(Q(\emptyset,2)) \parallel \rho_{\{ra\to rc\}}(Q(\emptyset,2)) = \\
&\rho_{\{sd\to sb\}}(R(d_1,d_1,0)) \parallel \rho_{\{ra\to rc\}}(R(d_2,d_2,0)) \overset{\text{Lemma 8.1}}{=} \\
&\tau_{\{i\}}(B(d_1,d_1,0,d_2,d_2,0)) \overset{\text{Lemma 7.1}}{=} \\
&\text{(since both } FC \text{ and the invariant hold for the parameters of } F) \\
&\tau_{\{i\}}(\tau_{\{\tau_c,\tau_{c_2},\tau_{c_4},\tau_i\}}(F(\mathsf{t},\mathsf{t},\mathsf{t},d_1,e_1,p_1,p_2,\langle d_1,p_1,p_2\rangle,read, \\
&\qquad\qquad \mathsf{t},\mathsf{t},\mathsf{t},d_2,e_2,p_2,p_1,\langle d_2,p_2,p_1\rangle,read))) = \\
&\tau_{\{i,\tau_c,\tau_{c_2},\tau_{c_4},\tau_i,c_2,c_4,c\}}(F(\mathsf{t},\mathsf{t},\mathsf{t},d_1,e_1,p_1,p_2,\langle d_1,p_1,p_2\rangle,read, \\
&\qquad\qquad \mathsf{t},\mathsf{t},\mathsf{t},d_2,e_2,p_2,p_1,\langle d_2,p_2,p_1\rangle,read)) = \\
&\tau_I(\rho_R(F(\mathsf{t},\mathsf{t},\mathsf{t},d_1,e_1,p_1,p_2,\langle d_1,p_1,p_2\rangle,read,\mathsf{t},\mathsf{t},\mathsf{t},d_2,e_2,p_2,p_1,\langle d_2,p_2,p_1\rangle,read)) = \\
&\tau_I(E(\mathsf{t},\mathsf{t},\mathsf{t},d_1,e_1,p_1,p_2,\langle d_1,p_1,p_2\rangle,read,\mathsf{t},\mathsf{t},\mathsf{t},d_2,e_2,p_2,p_1,\langle d_2,p_2,p_1\rangle,read)) = \\
&\tau_I(\partial_H(\rho_{R_1}(\partial_{H'}(S(\mathsf{t},\mathsf{t},\mathsf{t},d_1,e_1,p_1,p_2) \parallel Tim \parallel C(\langle d_1,p_1,p_2\rangle,read))) \parallel \\
&\qquad\qquad \rho_{R_2}(\partial_{H'}(S(\mathsf{t},\mathsf{t},\mathsf{t},d_1,e_1,p_2,p_1) \parallel Tim \parallel C(\langle d_2,p_2,p_1\rangle,read))))) \overset{\text{Lemma 3.1}}{=} \\
&\tau_I(\partial_H(\rho_{R_1}(\partial_{H'}(S(\mathsf{t},\mathsf{t},\mathsf{t},d_1,e_1,p_1,p_2) \parallel Tim \parallel C(f_{dum},read))) \parallel \\
&\qquad\qquad \rho_{R_2}(\partial_{H'}(S(\mathsf{t},\mathsf{t},\mathsf{t},d_2,e_2,p_2,p_1) \parallel Tim \parallel C(f_{dum},read))))) = \\
&OSWP(d_1,e_1,d_2,e_2,p_1,p_2).
\end{aligned}
$$

TABLE 5. Proof of Theorem 8.2

[8] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.

[9] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.

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

[11] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6(4):319–340, 1976.

[12] A.A. Schoone. *Assertional Verification in Distributed Computing.* PhD thesis, Utrecht University, 1991.

[13] A.S. Tanenbaum. *Computer networks.* Prentice-Hall International, Englewood Cliffs, 1981.

[14] F.W. Vaandrager. Verification of two communication protocols by means of process algebra. Report CS-R8608, CWI, Amsterdam, 1986.

[15] J.J. van Wamel. A study of a one bit sliding window protocol in ACP. In *Proceedings of the Second Workshop on Protocol Verification,* Mierlo, The Netherlands. Technische Universiteit Eindhoven, 1992.

APPENDIX A

We list the axioms that we have used in the Tables 6 to 9. These are the axioms of μCRL^1-algebras [2], together with some laws for true concurrency, the principle KFAR, Milner's first and second τ-laws and some laws to handle data. Furthermore we have assumed the Recursive Definition Principle (RDP, every recursion equation has at least one solution) and the Recursive Specification Principle (RSP, every guarded recursion equation has at most one solution). The latter principle is also used in the form of CL-RSP from [2]. In the tables, x, y, z range over processes, a, b are actions c, d represent either τ, δ or an action $a(d)$, p, q range over functions from data to processes, D is an arbitrary data type and d represents an element of D.

APPENDIX B

Below we list the axioms defining the standard data types that we use. We start with the booleans, then define the standard functions *eq* and *if* and end with the natural numbers and queues.

8.1. Booleans

sort	**Bool**
func	$\mathsf{t},\mathsf{f} :\to$ **Bool**
	$\neg$: **Bool** $\to$ **Bool**
	$\wedge$: **Bool** $\times$ **Bool** $\to$ **Bool**
	$\vee$: **Bool** $\times$ **Bool** $\to$ **Bool**
var	b : **Bool**
rew	$\neg\mathsf{t} = \mathsf{f}$
	$\neg\mathsf{f} = \mathsf{t}$
	$\mathsf{t}\wedge b = b$
	$\mathsf{f}\wedge b = \mathsf{f}$
	$\mathsf{t}\vee b = \mathsf{t}$
	$\mathsf{f}\vee b = b$

A1	$x+y=y+x$	SUM1	$\Sigma_{d:D} x = x$
A2	$x+(y+z)=(x+y)+z$	SUM3	$\Sigma p = \Sigma p + p(d)$
A3	$x+x=x$	SUM4	$\Sigma_{d:D}(p(d)+q(d)) = \Sigma p + \Sigma q$
A4	$(x+y)\cdot z = x\cdot z + y\cdot z$	SUM5	$\Sigma_{d:D}(p(d)\cdot z) = (\Sigma p)\cdot z$
A5	$(x\cdot y)\cdot z = x\cdot(y\cdot z)$	SUM11	$(\forall d \in D\ p(d)=q(d)) \rightarrow \Sigma p = \Sigma q$
A6	$x+\delta = x$		
A7	$\delta\cdot x = \delta$	B1	$\neg(\mathsf{t}=\mathsf{f})$
		B2	$\neg(b=\mathsf{t}) \rightarrow b=\mathsf{f}$
T1	$x\cdot\tau = x$	C1	$z \lhd \mathsf{t} \rhd z' = z$
T2	$\tau\cdot x + x = \tau\cdot x$	C2	$z \lhd \mathsf{f} \rhd z' = z'$

TABLE 6. Axioms for pCRL$_\tau$ (pico CRL)

SUM6	$\Sigma_{d:D}(p(d) \between z) = (\Sigma p) \between z$	CF	$a(d)\|b(e) = \begin{cases} \gamma(a,b)(d) & \text{if } d=e \text{ and} \\ & \gamma(a,b) \text{ defined} \\ \delta & \text{otherwise} \end{cases}$
SUM7	$\Sigma_{d:D}(p(d)\|z) = (\Sigma p)\|z$		
SUM8	$\Sigma_{d:D}(\partial_H(p(d))) = \partial_H(\Sigma p)$		
SUM9	$\Sigma_{d:D}(\tau_I(p(d))) = \tau_I(\Sigma p)$		
SUM10	$\Sigma_{d:D}(\rho_R(p(d))) = \rho_R(\Sigma p)$	CD1	$\delta\|x = \delta$
		CD2	$x\|\delta = \delta$
CM1	$x \parallel y = x \between y + y \between x + x\|y$	CT1	$\tau\|x = \delta$
CM2	$c \between x = c\cdot x$	CT2	$x\|\tau = \delta$
CM3	$c\cdot x \between y = c\cdot(x \parallel y)$		
CM4	$(x+y) \between z = x \between z + y \between z$	DD	$\partial_H(\delta) = \delta$
CM5	$c\cdot x\|d = (c\|d)\cdot x$	DT	$\partial_H(\tau) = \tau$
CM6	$c\|d\cdot x = (c\|d)\cdot x$	D1	$\partial_H(a(d)) = a$ if $a \notin H$
CM7	$c\cdot x\|d\cdot y = (c\|d)\cdot(x \parallel y)$	D2	$\partial_H(a(d)) = \delta$ if $a \in H$
CM8	$(x+y)\|z = x\|z + y\|z$	D3	$\partial_H(x+y) = \partial_H(x) + \partial_H(y)$
CM9	$x\|(y+z) = x\|y + x\|z$	D4	$\partial_H(x\cdot y) = \partial_H(x)\cdot\partial_H(y)$

TABLE 7. μCRL axioms for the encapsulation operator and the merges

TID	$\tau_I(\delta) = \delta$		RD	$\rho_R(\delta) = \delta$	
TIT	$\tau_I(\tau) = \tau$		RT	$\rho_R(\tau) = \tau$	
TI1	$\tau_I(a(d)) = a(d)$	if $a \notin I$	R1	$\rho_R(a(d)) = R(a)(d)$	
TI2	$\tau_I(a(d)) = \tau$	if $a \in I$			
TI3	$\tau_I(x+y) = \tau_I(x) + \tau_I(y)$		R3	$\rho_R(x+y) = \rho_R(x) + \rho_R(y)$	
TI4	$\tau_I(x\cdot y) = \tau_I(x)\cdot\tau_I(y)$		R4	$\rho_R(x\cdot y) = \rho_R(x)\cdot\rho_R(y)$	

TABLE 8. μCRL axioms for the hiding and renaming operators

8.2. The functions *eq* and *if*

For every data type we assume the functions *eq* and *if* available. The axioms that are listed here are essentially used in the expansions. Assume some arbitrary data type E. We have

func $eq : E \times E \to \mathbf{Bool}$
$if : \mathbf{Bool} \times E \times E \to E$
var $d_1, d_2 : E$
rew $if(\mathsf{t}, d_1, d_2) = d_1$
$if(\mathsf{f}, d_1, d_2) = d_2$
$eq(d_1, d_1) = \mathsf{t}$
$if(eq(d_1, d_2), d_1, d_2) = d_2$

The following lemma is crucial in the verification

LEMMA 8.3 *For all* d_1, d_2:D *we have that* $eq(d_1, d_2) = \mathsf{t} \leftrightarrow d_1 = d_2$.

8.3. Natural numbers

First we specify some elementary sorts and functions. We use infix notation wherever we find it convenient to do so. Moreover, we write $n \leq m$ for $(n \leq m) = \mathsf{t}$. Idem for $\geq$, $>$ and $<$.

sort *nat*
func $0 :\to nat$
$S, P : nat \to nat$
$+, - : nat \times nat \to nat$
$\geq, \leq, <, >: nat \times nat \to \mathbf{Bool}$
$if : \mathbf{Bool} \times nat \times nat \to nat$
$1, 2 :\to nat$
var $n, m : nat$
rew $P(0) = 0$
$P(S(n)) = n$
$n + 0 = n$
$n + S(m) = S(n + m)$
$n - 0 = n$
$n - S(m) = P(n - m)$
$eq(0, 0) = \mathsf{t}$
$eq(0, S(n)) = \mathsf{f}$
$eq(S(n), 0) = \mathsf{f}$
$eq(S(n), S(m)) = eq(n, m)$
$n \geq 0 = \mathsf{t}$
$0 \geq S(n) = \mathsf{f}$
$S(n) \geq S(m) = n \geq m$
$n \leq m = m \geq n$
$n > m = n \geq S(m)$
$n < m = S(n) \leq m$
$1 = S(0)$
$2 = S(1)$

8.4. The data type Queue

sort *queue*
func $\emptyset :\to queue$
$in : D \times queue \to queue$
$size : queue \to nat$
$toe : queue \to D$
$untoe : queue \to queue$
var $d, e : D$
$b, c : queue$
rew $size(\emptyset) = 0$
$size(in(d, b)) = S(size(b))$
$toe(in(d, \emptyset) = d$
$toe(in(d, in(e, b))) = toe(in(e, b))$
$untoe(\emptyset) = \emptyset$
$untoe(in(d, \emptyset)) = \emptyset$
$untoe(in(d, in(e, b))) = in(d, untoe(in(e, b)))$

APPENDIX C

We start with an exhaustive list, in order of appearance, of the unguarded recursive calls of process

$$X(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2),$$

followed by the condition under which the call may take place.

- RC1
  $X(eq(bit_2(f_2), p_1)/rdy_1, \mathsf{f}/rec_1, dat(f_2)/e_1, inv(q_1)/q_1, read/st_2)$
  $rec_1 \wedge eq(bit_1(f_2), inv(q_1)) \wedge eq_{del}(st_2)$

- RC2
  $X(eq(bit_2(f_2), p_1)/rdy_1, read/st_2)$
  $\neg(rec_1 \wedge eq(bit_1(f_2), inv(q_1))) \wedge eq_{del}(st_2)$

- RC3
  $X(\mathsf{f}/sts_1)$
  $(\neg eq(p_1, q_2) \wedge rec_2) \vee (eq(p_2, q_1) \wedge \neg rdy_2) \wedge eq_{read}(st_1) \wedge sts_1$

- RC4
  $X(del/st_1)$
  $eq_{choice}(st_1)$

- RC5
  $X(\mathsf{t}/sts_1, \langle d_1, p_1, q_1 \rangle / f_1, choice/st_1)$
  $eq_{read}(st_1) \wedge \neg sts_1$

- RC6
  $X(read/st_1, eq(bit_2(f_1), p_2)/rdy_2, \mathsf{f}/rec_2, dat(f_1)/e_2, inv(q_2)/q_2)$
  $rec_2 \wedge eq(bit_1(f_1), inv(q_2)) \wedge eq_{del}(st_1)$

- RC7
  $X(read/st_1, eq(bit_2(f_1), p_2)/rdy_2))$
  $\neg(rec_2 \wedge eq(bit_1(f_1), inv(q_2))) \wedge eq_{del}(st_1)$

- RC8
  $X(\mathsf{f}/sts_2)$
  $(\neg eq(p_2, q_1) \wedge rec_1) \vee (eq(p_1, q_2) \wedge \neg rdy_1) \wedge eq_{read}(st_2) \wedge sts_2$

- RC9
  $X(del/st_2)$
  $eq_{choice}(st_2)$

- RC10
  $X(\mathrm{t}/sts_2, \langle d_2, p_2, q_2\rangle/f_2, choice/st_2)$
  $eq_{read}(st_2) \wedge \neg sts_2$

The conditional recursive calls RC1–10 define a directed graph on the state space of X. If this graph does not contain an infinite path, then a guarded recursive call must occur after a finite number of unguarded ones. Hence the convergence of Φ_X in the sense of [2] is equivalent to the well-foundedness of the graph.

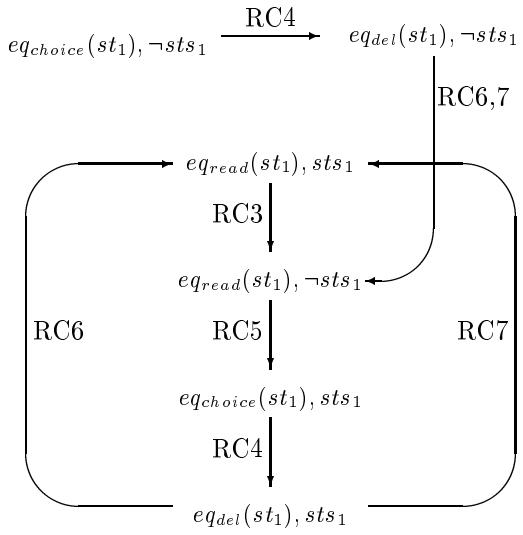

FIGURE 3. An abstraction of the graph defined by RC3–7

The proof of the well-foundedness of the graph is greatly simplified by the following decomposition. Observe that the parameters rdy_1, rec_1, q_1, st_2, sts_2, f_2 are affected only by the recursive calls RC1,2,8–10. Symmetrically, parameters rdy_2, rec_2, q_2, st_1, sts_1, f_1 are affected only by RC3–7. Observe furthermore that all other parameters either are not affected by any recursive call (the p_i's), or do not occur in any condition (the d_i's, e_i's and $dat(f_i)$'s). Unfortunately, these two sets of recursive calls are not completely independent of each other: q_2 occurs in conditions of RC1,2,8–10, and, symmetrically, q_1 in conditions of RC3–7. There is no other overlap.

For a moment we restrict ourselves to the acyclicity of one component, or rather subgraph, namely the one defined by RC3–7. It is helpful to partition the state space in six classes determined by the pair (st_1, sts_1). In all classes but two (those with $eq_{del}(st_1)$) there is at most one recursive call possible. If $eq_{read}(st_1) \wedge sts_1$ then at most RC3 is possible, bringing us in the class $eq_{read}(st_1) \wedge \neg sts_1$. Now only RC5 is possible, bringing us in the class $eq_{choice}(st_1) \wedge sts_1$. If $eq_{choice}(st_1)$, then RC4 brings us to $eq_{del}(st_1)$, in which either RC6 or RC7 is possible, depending on the boolean $rec_2 \wedge eq(bit_1(f_1), inv(q_2))$. Here we have the possibility of a loop, since as well RC6 as RC7 brings us back to $eq_{read}(st_1)$. The situation is depicted in the Figure 3. It should be remarked that this figure represents an abstraction of the graph on the state space. Transitions in the state space correspond by projection with transitions in the figure, but the cycles in the figure do not correspond to cycles or infinite paths in the original graph. On the contrary: by taking the conditions into account the cycles in the figure are used to exclude infinite paths in the original graph.

Inspection of RC6 shows that the left loop in Figure 3 can occur only once: only in case rec_2, and after RC6, by f/rec_2, we have $\neg rec_2$ for once and for all (no other recursive call affects rec_2). By symmetry RC1 can also occur only once. Furthermore, RC5 is on both the left and the right loop in Figure 3.

The right loop via RC7 is a bit more complicated. We first make the following observation. After recursive call RC5, by $\langle d_1, p_1, q_1\rangle/f_1$, we have $eq(bit_1(f_1), p_1)$ for once and for all. In this situation, the condition of RC7 implies $\neg(rec_2 \wedge eq(p_1, inv(q_2)))$. This means that in the condition of RC3 the disjunct $(eq(p_2, q_1) \wedge \neg rdy_2)$ must be true to enable this recursive call. However, by $eq(bit_2(f_1), p_2)/rdy_2$ in RC7, looping would be prevented if $eq(bit_2(f_1), q_1)$.

We are now in a position to complete our argument. Assume the original graph contains an infinite path, say P. Then either infinitely many recursive calls RC3–7, or infinitely many recursive calls RC1,2,8–10 are executed along P. Let us assume infinitely many calls RC3–7, as the other case is fully symmetric. Recall that RC1 is executed at most once. As RC5 has infinitely many occurrences along P, we may choose an occurrence of RC5 after which no RC1 occurs in P. Since RC1 is the only call which affects q_1, q_1 does not change after the abovementioned occurrence of RC5, and we have both $eq(bit_1(f_1), p_1)$ and $eq(bit_2(f_1), q_1)$ for once and for all. As also RC6 is executed at most once along P, we must encounter an occurrence of RC7 after the abovementioned occurrence of RC5. Now we arrive at a contradiction using the argument from the previous paragraph. It follows that the original graph cannot contain an infinite path.

There exists a function from the state space to natural numbers which decreases along the paths. This follows from the results above. Conversely, given such a function, it follows that the graph does not contain an infinite path. We give such a function below. The reader can easily verify that it has the desired property. To find such a function is not so easy. The function below is bounded by 28, which means that the graph does not contain any path longer than that.

Let $(x_1, x_2, x_3, x_3', x_4, x_4')^i$ abbreviate

$$if(eq_{read}(st_i), if(sts_i, x_1, x_2), if(eq_{choice}(st_i), if(sts_i, x_3, x_3'), if(sts_i, x_4, x_4')))$$

Define

$$g(rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2)$$

by

$$\begin{array}{l} if(rec_1, 8, 0) + \\ if(rec_2, 8, 0) + \\ if(rec_2 \wedge \neg eq(p_1, q_2), \\ \quad if(eq(bit_1(f_1), p_1), (4,3,2,5,1,4)^1, (4,3,6,5,5,4)^1), \\ \quad if(eq(bit_2(f_1), q_1), (if(\neg rdy_2 \wedge eq(p_2, q_1), 4, 0), \\ \qquad 3,2,5,1,4)^1, (4,3,6,5,5,4)^1)) + \\ if(rec_1 \wedge \neg eq(p_2, q_1), \\ \quad if(eq(bit_1(f_2), p_2), (4,3,2,5,1,4)^2, (4,3,6,5,5,4)^2), \\ \quad if(eq(bit_2(f_2), q_2), (if(\neg rdy_1 \wedge eq(p_1, q_2), 4, 0), \\ \qquad 3,2,5,1,4)^2, (4,3,6,5,5,4)^2)) \end{array}$$

APPENDIX D

We must show that the term

$$\begin{array}{l} B = \lambda rdy_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, \\ \qquad rdy_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2. \\ B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2), \\ \quad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \\ \triangleleft FC \triangleright \\ \tau B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2), \\ \quad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \end{array}$$

satisfies the recursion equation of X, for all values of the parameters satisfying the invariant. This fact is proven by distinguishing between the cases FC and $\neg FC$. Since $\tau\,\tau = \tau$ and $a\,\tau = a$ for any action a, the case distinction between FC and $\neg FC$ is irrelevant for calculating the right hand side of the equation, and shall therefore be ignored.

First assume $I(rdy_1, d_1, e_2, p_1, q_2, f_1, f_2)$ and $I(rdy_2, d_2, e_1, p_2, q_1, f_2, f_1)$ and FC. The calculation is exhibited in Table 10 Inspection of the resulting term in Table 10 indeed shows that B matches the right hand side of the defining equation of X provided FC.

Note that at $(*)$ in Table 10 we use that FC and the invariant imply:

- $rdy_1 = eq(p_1, q_2)$,
- $rdy_2 = eq(p_2, q_1)$,
- $eq(det(rdy_1, rec_2), 1) = rdy_1 \wedge \neg rec_2$,
- $eq(det(rdy_2, rec_1), 1) = rdy_2 \wedge \neg rec_1$.

Furthermore we use at $(*)$ and below the following trivial consequences of the definition of det:

- $det(rdy_1, rec_2) > 0 = \neg rec_2$,
- $det(rdy_2, rec_1) > 0 = \neg rec_1$,
- $det(rdy_2, \mathsf{f}) - 1 = det(rdy_2, \mathsf{t})$,
- $det(rdy_1, \mathsf{f}) - 1 = det(rdy_1, \mathsf{t})$,
- $eq(det(rdy_1, rec_2), 0) = rdy_1 \wedge rec_2$,
- $eq(det(rdy_2, rec_1), 0) = rdy_2 \wedge rec_1$.

Now assume $I(rdy_1, d_1, e_2, p_1, q_2, f_1, f_2)$ and $I(rdy_2, d_2, e_1, p_2, q_1, f_2, f_1)$ and $\neg FC$. The expansion from Table 11 uses the auxiliary calculations from Tables 14,15 and 16. Recall that $x \sqsupseteq y$ if and only if $x = y + z$ for some z. The first summand of the right hand side of the equation in Table 11 can now be justified using the axiom $\tau\,x = \tau\,x + x$, which implies $\tau\,x = \tau\,x + y$ for any $x \sqsupseteq y$. Note that we used $rdy_1 \leftrightarrow ((rdy_1 \wedge rec_2) \vee (rdy_1 \wedge (\neg rec_2)))$. The second summand is justified using the same technique and the calculation below; the other summands are either trivial or follow by symmetry. The calculation proceeds by proving the right hand side of the equation in Table 11 equal to the term in Table 12, for which we use that $\neg FC$ is equivalent to

$$\begin{array}{l} eq_{del}(st_1) \vee eq_{del}(st_2) \vee \\ eq_{choice}(st_1) \vee eq_{choice}(st_2) \vee \\ (eq_{read}(st_1) \wedge \neg sts_1) \vee (eq_{read}(st_2) \wedge \neg sts_2) \vee \\ (((\neg eq(p_1, q_2) \wedge rec_2) \vee (eq(p_2, q_1) \wedge \neg rdy_2)) \wedge \\ eq_{read}(st_1) \wedge sts_1) \vee \\ (((\neg eq(p_2, q_1) \wedge rec_1) \vee \\ (eq(p_1, q_2) \wedge \neg rdy_1)) \wedge eq_{read}(st_2) \wedge sts_2) \end{array}$$

Finally we prove the terms from Tables 12 and 13 are equal, using that the invariant implies:

- $eq(bit_2(f_2), p_1) \vee eq(p_1, q_2) = rdy_1 \vee eq(p_1, q_2)$.
- $eq(bit_1(f_2), inv(q_1)) \rightarrow \neg rdy_2 \wedge \neg eq(p_2, q_1)$.
- $eq(bit_1(f_2), inv(q_1)) \rightarrow if(\mathsf{f}, d_2, dat(f_2)) = if(\mathsf{t}, d_2, e_1)$.
- $eq(bit_1(f_2), inv(q_1)) \rightarrow det(rdy_2 \vee eq(p_2, inv(q_1)), \mathsf{f}) = det(rdy_2 \vee eq(p_2, q_1), \mathsf{t})$.

□

KFAR	$p(d) = i\,p(d) + y \rightarrow \tau\,\tau_{\{i\}}(p(d)) = \tau\,\tau_{\{i\}}(y)$
T+	$\tau_I(\tau_{I'}(x)) = \tau_{I \cup I'}(x)$
R+	$\tau_I(\rho_R(x)) = \tau_{I'}(x)$ if $R(I') \subseteq I$
SC1	$(x \between y) \between z = x \between (y \between z)$
SC3	$x \mid y = y \mid x$
SC4	$(x \mid y) \mid z = x \mid (y \mid z)$
SC5	$x \mid (y \between z) = (x \mid y) \between z$

TABLE 9. Some extra axioms needed in the verification

$$
\begin{aligned}
&B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) = \\
&\\
&\textstyle\sum_{d:D} ra(d)\, B(d, d, 1, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \lhd eq(\mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), 0) \rhd \delta + \\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d_1, e_2), 2, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \lhd eq(\mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), 1) \rhd \delta + \\
&sb(e_2)\, B(d_1, d_1, \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2) - 1, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \lhd \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2) > 0 \rhd \delta + \\
&\textstyle\sum_{d:D} rc(d)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d, d, 1) \\
&\qquad \lhd eq(\mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1), 0) \rhd \delta + \\
&\textstyle\sum_{d:D} rc(d)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d, \mathit{if}(rec_1, d_2, e_1), 2) \\
&\qquad \lhd eq(\mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1), 1) \rhd \delta + \\
&sd(e_1)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, d_2, \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1) - 1) \\
&\qquad \lhd \mathit{det}(rdy_2, rec_1) > 0 \rhd \delta + \\
&i\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, \\
&\qquad \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) =^{(*)} \\
&\\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), \mathit{det}(\mathsf{f}, rec_2), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \lhd rdy_1 \wedge rec_2 \rhd \delta + \\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), \mathit{det}(\mathsf{f}, rec_2), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \lhd rdy_1 \wedge \neg rec_2 \rhd \delta + \\
&sb(e_2)\, B(d_1, \mathit{if}(\mathsf{t}, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), \mathsf{t}), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \lhd \neg rec_2 \rhd \delta + \\
&\textstyle\sum_{d:D} rc(d)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, p_2), rec_2), d, \mathit{if}(rec_1, d, e_1), \mathit{det}(\mathsf{f}, rec_1)) \\
&\qquad \lhd rdy_2 \wedge rec_1 \rhd \delta + \\
&\textstyle\sum_{d:D} rc(d)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d, \mathit{if}(rec_1, d, e_1), \mathit{det}(\mathsf{f}, rec_1)) \\
&\qquad \lhd rdy_2 \wedge \neg rec_1 \rhd \delta + \\
&sd(e_1)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, \mathit{if}(\mathsf{t}, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), \mathsf{t})) \\
&\qquad \lhd \neg rec_1 \rhd \delta + \\
&i\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) = \\
&\\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), \mathit{det}(\mathsf{f} \vee eq(inv(p_1), q_2), rec_2), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \lhd rdy_1 \rhd \delta + \\
&\delta \lhd rec_2 \rhd sb(e_2)\, B(d_1, \mathit{if}(\mathsf{t}, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), \mathsf{t}), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) + \\
&\textstyle\sum_{d:D} rc(d)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d, \mathit{if}(rec_1, d, e_1), \mathit{det}(\mathsf{f} \vee eq(inv(p_2), q_1), rec_1)) \\
&\qquad \lhd rdy_2 \rhd \delta + \\
&\delta \lhd rec_1 \rhd sd(e_1)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, \mathit{if}(\mathsf{t}, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), \mathsf{t})) + \\
&i\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)).
\end{aligned}
$$

TABLE 10. Calculation in the case FC

$$
\begin{aligned}
&\tau\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) = \\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), \mathit{det}(\mathsf{f} \vee eq(inv(p_1), q_2), rec_2), \\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \lhd rdy_1 \rhd \delta + \\
&\delta \lhd rec_1 \rhd sd(e_1)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), \\
&\qquad d_2, d_2, \mathit{det}(rdy_2 \vee eq(p_2, q_1), \mathsf{t})) + \\
&\tau\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), \\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) + \\
&\textstyle\sum_{d:D} rc(d)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), \\
&\qquad d, \mathit{if}(rec_1, d, e_1), \mathit{det}(\mathsf{f} \vee eq(inv(p_2), q_1), rec_1)) \lhd rdy_2 \rhd \delta + \\
&\delta \lhd rec_2 \rhd sb(e_2)\, B(d_1, d_1, \mathit{det}(rdy_1 \vee eq(p_1, q_2), \mathsf{t}), \\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) + \\
&\tau\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), \\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) + \\
&i\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), \\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) =^{(**)}
\end{aligned}
$$

TABLE 11. First expansion of B in the case $\neg FC$

$$
\begin{aligned}
&\textstyle\sum_{d:D} ra(d)\, B(d, if(rec_2, d, e_2), det(\mathsf{f} \vee eq(inv(p_1), q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \lhd rdy_1 \rhd \delta +\\
&\delta \lhd rec_1 \rhd sd(e_1)\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d_2, d_2, det(rdy_2 \vee eq(p_2, q_1), \mathsf{t})) +\\
&\tau\,(B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1))\\
&\qquad \lhd rec_1 \wedge eq(bit_1(f_2), inv(q_1)) \rhd\\
&\qquad\quad B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1))) \lhd eq_{del}(st_1) \rhd \delta +\\
&i\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad\quad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) +\\
&\tau\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1))\\
&\qquad \lhd ((\neg eq(p_1, q_2) \wedge rec_2) \vee (eq(p_2, q_1) \wedge \neg rdy_2)) \wedge eq_{read}(st_1) \wedge sts_1 \rhd \delta +\\
&\tau\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \lhd eq_{choice}(st_1) \rhd \delta +\\
&i\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \lhd eq_{choice}(st_1) \rhd \delta +\\
&\tau\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \lhd eq_{read}(st_1) \wedge \neg sts_1 \rhd \delta +\\
&\textstyle\sum_{d:D} rc(d)\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d, if(rec_1, d, e_1), \mathsf{f} \vee eq(inv(p_2), q_1), rec_1)) \lhd rdy_1 \rhd \delta +\\
&\delta \lhd rec_2 \rhd sb(e_2)\, B(d_1, d_1, , det(rdy_1 \vee eq(p_1, q_2), \mathsf{t}),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) +\\
&\tau\,(B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1))\\
&\qquad \lhd rec_2 \wedge eq(bit_1(f_1), inv(q_2)) \rhd\\
&\qquad\quad B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1))) \lhd eq_{del}(st_2) \rhd \delta +\\
&i\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad\quad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) +\\
&\qquad \tau\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 eq(p_1, q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 eq(p_2, q_1), rec_1))\\
&\lhd ((\neg eq(p_2, q_1) \wedge rec_1) \vee (eq(p_1, q_2) \wedge \neg rdy_1)) \wedge eq_{read}(st_2) \wedge sts_2 \rhd \delta +\\
&\tau\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \lhd eq_{choice}(st_2) \rhd \delta +\\
&i\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \lhd eq_{choice}(st_2) \rhd \delta +\\
&\tau\, B(d_1, if(rec_2, d_1, e_2), det(rdy_1 \vee eq(p_1, q_2), rec_2),\\
&\qquad d_2, if(rec_1, d_2, e_1), det(rdy_2 \vee eq(p_2, q_1), rec_1)) \lhd eq_{read}(st_2) \wedge \neg sts_2 \rhd \delta =^{(***)}
\end{aligned}
$$

TABLE 12. Further calculation using $\neg FC$

$$
\begin{aligned}
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), \mathit{det}(\mathsf{f} \vee \mathit{eq}(\mathit{inv}(p_1), q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1) \triangleleft rdy_1 \triangleright \delta +\\
&\delta \triangleleft rec_1 \triangleright sd(e_1)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(\mathsf{t}, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), \mathsf{t})) +\\
&\tau\,(B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(\mathit{eq}(bit_2(f_2), p_1) \vee \mathit{eq}(p_1, q_2)), rec_2),\\
&\qquad d_2, \mathit{if}(\mathsf{f}, d_2, dat(f_2)), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, \mathit{inv}(q_1)), \mathsf{f}))\\
&\qquad \triangleleft rec_1 \wedge \mathit{eq}(bit_1(f_2), \mathit{inv}(q_1)) \triangleright\\
&\qquad\quad B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(\mathit{eq}(bit_2(f_2), p_1) \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad\qquad d_2, \mathit{if}(rec_1, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1))) \triangleleft \mathit{eq}_{del}(st_2) \triangleright \delta +\\
&i\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) +\\
&\tau\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1))\\
&\qquad \triangleleft((\neg \mathit{eq}(p_1, q_2) \wedge rec_2) \vee (\mathit{eq}(p_2, q_1) \wedge \neg rdy_2)) \triangleright \delta +\\
&\tau\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) \triangleleft \mathit{eq}_{choice}(st_1) \triangleright \delta +\\
&i\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) \triangleleft \mathit{eq}_{choice}(st_1) \triangleright \delta +\\
&\tau\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) \triangleleft \mathit{eq}_{read}(st_1) \wedge \neg sts_1 \triangleright \delta +\\
&\textstyle\sum_{d:D} rc(d)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_2 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d, e_1), \mathit{det}(\mathsf{f} \vee \mathit{eq}(\mathit{inv}(p_2), q_1), rec_1)) \triangleleft rdy_2 \triangleright \delta +\\
&\delta \triangleleft rec_2 \triangleright sb(e_2)\, B(d_1, \mathit{if}(\mathsf{t}, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), \mathsf{t}),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) +\\
&\tau\,(B(d_1, \mathit{if}(\mathsf{f}, d_1, dat(f_1)), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, \mathit{inv}(q_2)), \mathsf{f}),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(\mathit{eq}(bit_2(f_1), p_2) \vee \mathit{eq}(p_2, q_1)), rec_1))\\
&\qquad \triangleleft rec_2 \wedge \mathit{eq}(bit_1(f_1), \mathit{inv}(q_2)) \triangleright\\
&\qquad\quad B(d_1, \mathit{if}(rec_2, d_1, \mathit{if}(rec_2, d_1, e_2)), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(\mathit{eq}(bit_2(f_1), p_2) \vee \mathit{eq}(p_2, q_1), rec_1)) \triangleleft \mathit{eq}_{del}(st_1) \triangleright \delta +\\
&i\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) +\\
&\tau\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1))\\
&\qquad \triangleleft((\neg \mathit{eq}(p_2, q_1) \wedge rec_1) \vee (\mathit{eq}(p_1, q_2) \wedge \neg rdy_1)) \triangleright \delta +\\
&\tau\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) \triangleleft \mathit{eq}_{choice}(st_2) \triangleright \delta +\\
&i\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) \triangleleft \mathit{eq}_{choice}(st_2) \triangleright \delta +\\
&\tau\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) \triangleleft \mathit{eq}_{read}(st_2) \wedge \neg sts_2 \triangleright \delta
\end{aligned}
$$

TABLE 13. Further calculations using the invariant

$$
\begin{aligned}
&B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) \supseteq\\
&\textstyle\sum_{d:D} ra(d)\, B(d, d, 1, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1))\\
&\qquad\qquad \triangleleft \mathit{eq}(\mathit{det}(rdy_1 \vee \mathit{eq}(p_1, q_2), rec_2), 0) \triangleright \delta =\\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), 1, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1))\\
&\qquad\qquad \triangleleft (rdy_1 \vee \mathit{eq}(p_1, q_2)) \wedge rec_2) \triangleright \delta \supseteq\\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), 1, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1))\\
&\qquad\qquad \triangleleft rdy_1 \wedge rec_2 \triangleright \delta =\\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), \mathit{det}(\mathsf{f}, \mathsf{t}), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1))\\
&\qquad\qquad \triangleleft rdy_1 \wedge rec_2 \triangleright \delta = (\text{since } rdy_1 \rightarrow \neg \mathit{eq}(\mathit{inv}(p_1), q_2) \text{ by the invariant})\\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), \mathit{det}(\mathsf{f} \vee \mathit{eq}(\mathit{inv}(p_1), q_2)), rec_2),\\
&\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee \mathit{eq}(p_2, q_1), rec_1)) \triangleleft rdy_1 \wedge rec_2 \triangleright \delta.
\end{aligned}
$$

TABLE 14. Auxiliary calculation 1

$$
\begin{aligned}
&B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \supseteq \\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d_1, e_2), 2, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \triangleleft eq(\mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), 1) \triangleright \delta \supseteq \\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d_1, e_2), 2, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \triangleleft eq(\mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), 1) \wedge \neg rec_2 \triangleright \delta = \\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), 2, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \triangleleft (rdy_1 \vee eq(p_1, q_2)) \wedge \neg rec_2 \triangleright \delta \supseteq \\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), 2, d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \triangleleft rdy_1 \wedge \neg rec_2 \triangleright \delta =^{\dagger} \\
&\textstyle\sum_{d:D} ra(d)\, B(d, \mathit{if}(rec_2, d, e_2), \mathit{det}(\mathsf{f} \vee eq(inv(p_1), q_2), rec_2), \\
&\qquad\qquad d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \\
&\qquad \triangleleft rdy_1 \wedge \neg rec_2 \triangleright \delta.
\end{aligned}
$$

TABLE 15. Auxiliary calculation 2

$$
\begin{aligned}
&B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, \mathit{if}(rec_1, d_2, e_1), \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) \supseteq \\
&sd(e_1)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, d_2, \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) - 1) \\
&\qquad \triangleleft \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) > 0 \triangleright \delta = \\
&sd(e_1)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, d_2, \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) - 1) \\
&\qquad \triangleleft \neg(rdy_2 \vee eq(p_2, q_1)) \vee \neg rec_1 \triangleright \delta \supseteq \\
&sd(e_1)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, d_2, \mathit{det}(rdy_2 \vee eq(p_2, q_1), rec_1)) - 1) \\
&\qquad \triangleleft \neg rec_1 \triangleright \delta = \\
&\delta \triangleleft rec_1 \triangleright sd(e_1)\, B(d_1, \mathit{if}(rec_2, d_1, e_2), \mathit{det}(rdy_1 \vee eq(p_1, q_2), rec_2), d_2, d_2, \mathit{det}(rdy_2 \vee eq(p_2, q_1), \mathsf{t})).
\end{aligned}
$$

TABLE 16. Auxiliary calculation 3