Yet Yet on the Bounded Retransmission Protocol

Thérèse Hardin and Brahim Mammass

Laboratoire d'Informatique de Paris 6
Tour 45-55, 4 Place Jussieu 75252 Paris Cedex 05, France
e-mail: Brahim.Mammass@lip6.fr, Therese.Hardin@lip6.fr

Abstract. The aim of this paper is twofold. We first present a correctness proof of the Bounded Retransmission Protocol (BRP), which is easily done by bisimulation in the π-calculus. Then, we compare several works on this protocol, focusing on how the used formalism influences implementation choices and proof techniques.

1 Introduction

The development of communication networks needs more and more sophisticated communication protocols which must be reliable. Traditional verification methods use model checking techniques, but they cannot deal with infinite state systems and more generally with mobility.

Our aim is to elaborate some methodologic guides for designing and proving communication protocols using theorem provers. We choose the BRP as our case study because it is simple but, as it is parameterized, model checking cannot be directly applied. The BRP is a communication protocol, developed at Philips Research Laboratory, that transfers messages from a producer to a consumer over an unreliable physical medium that can lose messages.

The π-calculus [23, 24] is an extension of the process algebra CCS [21] with mobility keeping its algebraic properties. It is more expressive than CCS because it provides possibilities for coding data types, λ-calculus and higher order processes. Moreover, it offers the possibility of introducing new channels and passing channel names between processes. So, on one hand, we present a proof of the BRP using π-calculus bisimulations.

On the other hand, we study some related works in order to compare different approaches. They are the following. Helmink, Sellink and Vaandrager [14] analyze the BRP in the setting of I/O automata [17]. Havelund and Shankar [13] combine model checking and theorem proving techniques to prove the correctness of the BRP. They use PVS [26] as a theorem prover, and SMV [19], Murϕ [20], and an extension of PVS with the modal μ-calculus [15] as model checkers. Groote and Van De Pol [9] use as a formal support μCRL [10], a combination of process algebra and abstract data types, to prove the correctness of the BRP. Abrial [1] designs the BRP by successive refinements in the proof-assistant B [2]. Finally, D'Argenio, Katoen, Ruys and Tretmans [8] analyze the BRP in the setting of timed automata [3]. The correctness of the protocol is checked using UPPAAL [5].

The comparison between these approaches tackles the following questions. Do these works start from the same description ? How are the protocol entities modeled ? What is exactly proved ? What are the difficulties encountered in doing the proofs ? Are they due to the used formalism or to implementation choices ? Is the π-calculus a well-suited framework ?

The paper is structured as follows: section 2 presents the initial informal description of the BRP. In section 3, we complete this description by expliciting some points and we give the point of view of the mentioned papers. The section 4 gives an abstract view of the BRP in the π-calculus and compares it to the others. In section 5, we implement the protocol in the π-calculus. The section 6 presents our correctness proof which proceeds by bisimulation. Finally, in section 7, we present the other implementations and proofs of the BRP and we compare them to ours.

2 The Bounded Retransmission Protocol

Apart from some little and irrelevant variations, all the papers start from the same description which is the following. The BRP communicates messages from a producer to a consumer over an unreliable physical medium that can lose messages. It is a nontrivial extension of the alternating bit protocol [6] that uses timeouts and aborts transmission following a bounded number of transmission attempts. The environment of the protocol consists of the producer and the consumer. The black box view of the system is that it accepts requests $Req(f)$ from the producer to transmit the file f. When transmission of f has been either completed or aborted, the producer receives a confirmation $Conf(c)$, where c is either OK, NOTOK, or DTKW (don't know), respectively indicating that the file was successfully transmitted, aborted, or that the last message in the file was not acknowledged but might have been received by the consumer. The consumer either receives an *Inderr* signal indicating that the file transmission was aborted, or an $Ind(m, i)$ signal where m is the message and i is either FIRST, LAST or INC (incomplete) corresponding to the first, last, or an intermediate message in the file.

Opening the black box, the protocol (figure 1) consists of a sender program at the producer side, a receiver program at the consumer side, and two channels (one-place buffers): a message channel K and an acknowledgment channel L. Both channels are unreliable in that they can lose messages or acknowledgments; but, messages are neither garbled, nor received out of order. Two timers are used. A timer has a fixed period T of time associated. When it is set, a timeout occurs T time units later.

The sender sends each message over the channel K, sets $timer_1$, and then waits for an acknowledgment over the channel L. The $timer_1$ is used to detect the loss of a message or an acknowledgment. The time associated with this timer exceeds the time required to send a message over K and to get the acknowledgment over L. If an acknowledgment comes back within this time, the timer is cleared, and the next message is sent. If the transmission has been completed, the sender confirms *Conf*(OK) to signal a successful transmission to the producer. If there is no acknowledgment, a timeout occurs whereupon the message is retransmitted, and the timer set again. There is a fixed upper bound on the number of such retransmissions (MAX). When this retransmission bound has been reached, the sender aborts transmission and confirms that the transmission failed. Either it confirms *Conf*(NOTOK) or it confirms *Conf*(DTKW).

The receiver waits for messages over the channel K. If the alternating bit of the received message is equal to that of the previous one, the receiver only retransmits an acknowledgment over the channel L. Otherwise, it delivers the message to the consumer, stores its alternating bit, sets $timer_2$, and sends an acknowledgment over the channel L. In both cases, it waits for the subsequent message over the channel K. The time associated with $timer_2$ must exceed the required time to transmit MAX times a message (i.e, $timer_2 \succ \text{MAX} * timer_1$). If $timer_2$ expires, i.e., no new message is received, the receiver sends an *Inderr* signal to the consumer to indicate the transmission abort.

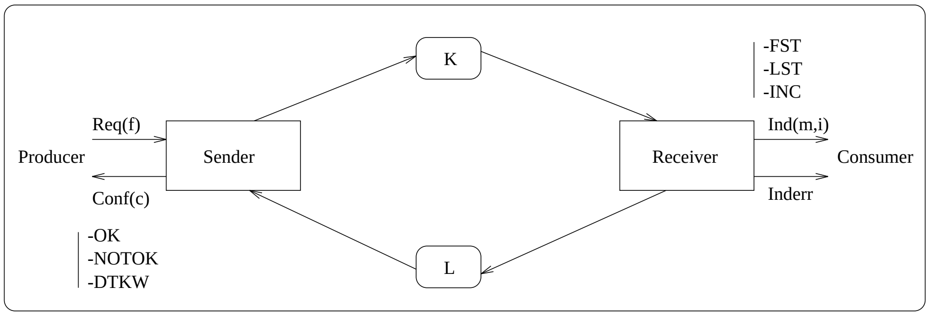

Fig. 1. The BRP protocol

We note that in [8, 9, 14], only one channel is used for indication signals from the receiver to the producer. So, if i equals NOTOK, the notification $Ind(m, i)$ indicates that an abort occurred and m is dummy. This difference seems irrelevant but perhaps may induce some typing difficulties if the protocol is extended by other treatments of m.

3 The complete informal specification

In this section, we give our interpretation of the BRP description, then we compare it to the ones adopted in the other papers.

3.1 Our interpretation

The previous section describes the protocol as a parallel composition of two entities: the sender and the receiver, that run sequentially. Before formalizing the protocol, we must complete the BRP description by expliciting some points. What can be the content of the file ? How can the synchronization between the sender and the receiver be realized ? How does the alternating bit work ? What happens if the first message never arrives at the receiver side ? In which cases must the receiver signal the abort to the consumer ?

We assume that the file may contain zero, one or more messages. If the file is empty, the sender must confirm immediately the transfer with a confirmation OK to the producer. If the file contains one message, this message must be considered as the last message in the file. Moreover, if the transfer is aborted during the transmission of this message, a confirmation DTKW must be sent to the producer.

The synchronization between the sender and the receiver after a transmission abort is achieved by means of $timer_2$. The description gives no indication about its activation. So, we assume that this timer is not enabled when the system starts. Otherwise, $timer_2$ could expire and should be restarted infinitely often leading to an active waiting of the receiver. After getting the first message, the receiver enables $timer_2$.

As $timer_2 \succ \text{MAX} * timer_1$, the sender is the first to detect a transmission abort. The sender may then receive a request to transfer a new file and send its first message while the receiver does not yet detect the abort (i.e., $timer_2$ has not expired). The receiver may consider this message as the next message (or as a duplication of the current message) of the previous file, that is wrong. So, the sender must wait until the receiver detects the abort. Moreover, the two must reinitialize their alternating bit before the beginning of the next transfer. For uniformity, we decide that they reinitialize their alternating bit also when the transfer has been completed.

After the correct termination of a file transmission, two situations are possible. Either the sender receives no new request before expiration of $timer_2$, so the receiver may send a misleading *Inderr* signal to the consumer. Or it receives a new request and sends the first message. If $timer_2$ has not yet expired, the receiver may consider this first message as a duplication of the last message of the previous file because it cannot know the new alternating bit value. To solve these synchronization problems, the sender must signal the end of the current transfer to the receiver before it begins the next one. The receiver can then anticipate the expiration of $timer_2$, then the two reinitialize their alternating bit.

If the first message in the file never arrives at the receiver side, as $timer_2$ is not yet set, the receiver cannot be informed about this abort. But, as decided above, the sender will wait the expiration of $timer_2$. This will lead to a deadlock situation. In a distributed setting, there is no solution to this problem because the abort information has to be transported on an unreliable channel. So, we assume that the first message arrives at least one time at the receiver side.

When an abort occurs during the transmission of an intermediate message, the sender must send a confirmation NOTOK to the producer and the receiver must send an *Inderr* signal to the consumer. If the abort occurs during the transmission of the last message, a confirmation DTKW is sent by the sender. But, at the receiver side, either this message was received, so no *Inderr* signal is sent. Or the message was always lost, an *Inderr* signal must be sent to the consumer. This point is not fully stated in the BRP description.

3.2 Other interpretations

Now, we only point out the differences between the other interpretations and ours. The consequences of their different choices will be discussed in section 7.

In [8, 9], the authors decide that when an abort occurs before the delivery of the first message, the consumer does not need an indication error. Moreover, if the transmission of the next file starts before $timer_2$ expires, the alternating bit scheme is simply continued. This scheme is only interrupted after a failure.

In [14], the acknowledgment consists of three control bits, but is considered as a simple signal in the other presentations.

In [1], the complete informal specification is written in a pseudo-code style. The author decide that $timer_2$ is only started when the received message is not the last one in the file.

4 The abstract view of the BRP

In this section, we consider the system as a black box. Its abstract view is the observable behavior on the external channels *Req*, *Ind*, *Inderr*, and *Conf*, abstracting the communications over the internal channels K and L. We first introduce the polyadic π-calculus [22] which is our formal framework. Then, we give our formal description and compare all the proposed abstract views.

4.1 Syntax and informal semantics of the polyadic π-calculus

Let $x, y, z, u, v, \ldots$ range over $\mathcal{N}$, a set of channel names. Let $A, B, \ldots$ range over a set of agent identifiers; each identifier has a nonnegative arity. We note by $\tilde{x}$ the tuple $x_1, x_2, \ldots, x_n$. Let $P, Q, \ldots$ range over agents (i.e. processes) which are defined as follows:

- 0, an agent which can do nothing.
- $\overline{y}\widetilde{x}.P$, an agent which outputs the tuple $\widetilde{x}$ on channel y; thereafter it behaves as P. In this action, y is the *subject*, $\widetilde{x}$ is the *object*, and both $\widetilde{x}$ and y are *free*.
- $y(\widetilde{x}).P$, an agent which receives a tuple on channel y; thereafter it behaves as P but with the newly received names in place of x_i. In this action, y is the *subject*, $\widetilde{x}$ is *bound*, and y is free.
- $\tau.P$, an agent which performs the silent action τ; thereafter it behaves as P.
- $P + Q$, an agent which behaves like either P or Q.

- $P \mid Q$, an agent representing the parallel composition of P and Q. This agent can do anything that P or Q can do, and moreover if $P = \overline{y}\widetilde{u}.P'$ and $Q = y(\widetilde{x}).Q'$, then $P \mid Q \xrightarrow{\tau} .(P' \mid Q'\{\widetilde{u}/\widetilde{x}\})$ where $Q'\{\widetilde{u}/\widetilde{x}\}$ is the substitution of each occurrence of x_i by u_i in Q'.
- $(\nu\ x)P$, an agent which behaves like P where the name x is local but P can export x.
- $[x = y]P$, an agent which behaves like P if x and y are the same name; otherwise it does nothing.
- $A(y_1, \ldots, y_n)$ is an agent if A is an identifier of arity n; for any such identifier there is a defining equation written $A(x_1, \ldots, x_n) \stackrel{def}{=} P$, where the names $x_1, \ldots, x_n$ are distinct and are the only names which may occur free in P. The agent $A(y_1, \ldots, y_n)$ behaves like P where y_i is substituted for x_i for all $i = 1, \ldots, n$. Agent identifiers provide recursion since the defining equation of A may contain A itself.

The formal operational semantics of agents is defined and explained in [24]. In the sequel, we note $(\nu\ x_1 \ldots x_n)P$ instead of $(\nu\ x_1) \ldots (\nu\ x_n)P$.

4.2 The abstract view of the BRP in the π-calculus

The abstract view is pictured in figure 2 and is expressed by three recursive equations. The file is modeled by a list of messages and we use the usual functions *cons*, *hd* and *tl* on lists.

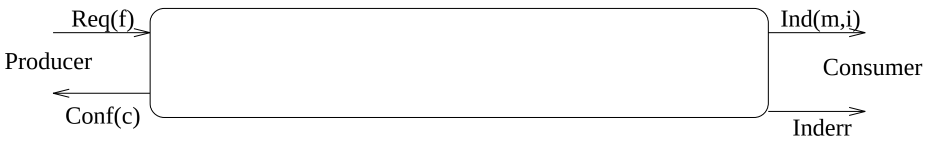

Fig. 2. The abstract view of the BRP

In the initial state S_0, the system waits for a file f on the channel *Req*. If f is empty it returns to S_0, else it processes the first message in the state S_1.

$$S_0 \stackrel{def}{=} \mathtt{Req}(f).([f = Nil].\overline{\mathtt{Conf}}\ \ \mathtt{OK}.S_0\ +\ [f = cons(h,t)].S_1(f)$$

In S_1 and S_2, the τ actions indicate that the choice between the delivery or loss of a message or an acknowledgment is decided by the internal actions.

$$\begin{aligned} S_1(f) \stackrel{def}{=}\ & \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{LAST}.\overline{\mathtt{Conf}}\ \ \mathtt{OK}.S_0 \\ +\ & \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{FIRST}.S_2(tl(f)) \\ +\ & \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{LAST}.\overline{\mathtt{Conf}}\ \mathtt{DTKW}.S_0 \\ +\ & \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{FIRST}.\overline{\mathtt{Conf}}\ \mathtt{NOTOK}.\overline{\mathtt{Inderr}}.S_0 \end{aligned}$$

In the state S_2, the system treats the remaining messages of the list.

$$
\begin{aligned}
S_2(f) \stackrel{def}{=} & \ \tau.\overline{\mathtt{Conf}}\ \mathtt{DTKW}.\overline{\mathtt{Inderr}}.S_0 \\
& + \tau.\overline{\mathtt{Conf}}\ \mathtt{NOTOK}.\overline{\mathtt{Inderr}}.S_0 \\
& + \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{LAST}.\overline{\mathtt{Conf}}\ \mathtt{DTKW}.S_0 \\
& + \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{INC}.\overline{\mathtt{Conf}}\ \mathtt{NOTOK}.\overline{\mathtt{Inderr}}.S_0 \\
& + \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{LAST}.\overline{\mathtt{Conf}}\ \ \mathtt{OK}.S_0 \\
& + \tau.\overline{\mathtt{Ind}}\ hd(f)\ \mathtt{INC}.S_2(tl(f))
\end{aligned}
$$

The specification above does not explicit the loss of messages or acknowledgments but supposes that they may occur. To make these losses explicit, the τ actions must be made observable in the protocol implementation.

4.3 The other abstract views

The abstract view in [9] is defined by four recursive equations written in μCRL. However, only one equation processes the first and the remaining messages of the list. This requires a tag which distinguishes the two cases. This tag is not needed in our case since the first element is treated separately in S_1.

In [14], the authors specify the abstract view by an I/O automaton which has the same input and output actions as the protocol but no internal actions. As the channels are modeled by shared variables, their access managing is part of the abstract view and is described by means of preconditions.

The abstract view in [1] states, in the B language, that the consumer receives a prefix of the file transmitted by the producer. There is no notion of time, even implicitly in message processing. The file is supposed to be transmitted instantaneously.

In [8], the abstract view is provided as a file transfer service described by logical relations between inputs and outputs. This approach leads to some difficulties which we discuss in section 7.

The approach adopted in [13] is different and will be discussed in section 7.

5 The formal implementation of the BRP

We start from the complete informal specification of section 3. To encode the protocol in the π-calculus, we need the types integer, boolean and list, and some functions on these types. They are encoded in the π-calculus [23].

We model the external channels *Req*, *Conf*, *Ind*, and *Inderr* as constant names because they are never bound during the execution of the protocol.

We model $timer_1$ by the agent $T1$ which repeatedly waits for a signal over the channel $time1$, then sends a signal over the channel $timeout1$. To set $timer_1$, the sender must send a signal over $time1$. To reset $timer_1$, the sender must make a rendez-vous over $timeout1$. The $timer_2$ is modeled in the same way.

$T1 \stackrel{def}{=} time1.\overline{timeout1}.T1$
$T2 \stackrel{def}{=} time2.\overline{timeout2}.T2$

The sender S uses locally four variables: $first$, $last$, tag and rn. If $first$ (resp. $last$) equals $True$, then the current message is the first (resp. last) one. The variable tag contains the alternating bit, and rn contains the number of retransmissions. Every message transmitted by the sender contains the informations $first$, $last$, tag and the message data. In the initial state, the sender initializes its variables ($[\![True]\!]tag$ puts True in tag) and waits for a request on the channel Req. When a request is received, the sender starts the transfer of the file.

$$S(K, L, abort, restart) \stackrel{def}{=} [\![True]\!]first.[\![True]\!]tag.[\![0]\!]rn.[\![False]\!]last.$$
$$Wait\_req(K, L, abort, restart, first, last, tag, rn)$$

$$Wait\_req(K, L, abort, restart, first, last, tag, rn) \stackrel{def}{=}$$
$$\texttt{Req}(f).Transfer(K, L, abort, restart, f, first, last, tag, rn)$$

If the file is empty, the sender confirms OK, makes a rendez-vous over the channel $restart$ with the receiver and returns to its initial state. Otherwise, the sender transmits the first message, sets $timer_1$, increments the number of retransmissions rn, and waits for an acknowledgment over the channel L. The function $one(l)$ tests if l is a list of one element.

$$Transfer(K, L, abort, restart, f, first, last, tag, rn) \stackrel{def}{=}$$
$$[f = Nil]\ \overline{\texttt{Conf}}\ \ \texttt{OK}.\overline{restart}.S(K, L, abort, restart)$$
$$+\ [f = cons(head, tail)]\ [\![one(f)]\!]last.\overline{K}\ \ first\ \ last\ \ tag\ \ head.\overline{time1}.$$
$$[\![rn + 1]\!]rn.Wait\_ack(K, L, abort, restart, head, tail, first, last, tag, rn)$$

If an acknowledgment is received, the sender resets $timer_1$, reinitializes rn, complements tag, and transmits the next message in the file. If no acknowledgment is received, $timer_1$ expires and the sender retransmits the message.

$$Wait\_ack(K, L, abort, restart, head, tail, first, last, tag, rn) \stackrel{def}{=}$$
$$L.timeout1.[\![0]\!]rn.Not(tag).[\![False]\!]first.$$
$$Transfer(K, L, abort, restart, tail, first, last, tag, rn)$$
$$+\ timeout1.Retrans(K, L, abort, restart, cons(head, tail), first, last, tag, rn)$$

If the bound of retransmissions is not exceeded, the message is retransmitted. If not, the transfer is aborted. The sender sends a confirmation DTKW (for the last message) or NOTOK (otherwise) to the producer. Then, it makes a rendez-vous with the receiver over the channel $abort$ before it begins a new transfer.

$$Retrans(K, L, abort, restart, f, first, last, tag, rn) \stackrel{def}{=}$$
$$If\ \ equal(rn, \texttt{MAX})\ \ then$$
$$([last = True]\ \overline{\texttt{Conf}}\ \texttt{DTKW}.\overline{abort}.S(K, L, abort, restart)$$
$$+[last = False]\ \overline{\texttt{Conf}}\ \texttt{NOTOK}.\overline{abort}.S(K, L, abort, restart))$$
$$else\ \ Transfer(K, L, abort, restart, f, first, last, tag, rn)$$

The receiver R is described in the same way as the sender in appendix A. The sender and the receiver are linked by the channels K, L, $abort$, and $restart$. These channels are private to the protocol. The channel $abort$ (resp. $restart$) is used to solve the synchronization problems between the sender and the receiver after a transmission abort (resp. after a correct transfer). Introducing these two channels allows us to separate cleanly the two situations.

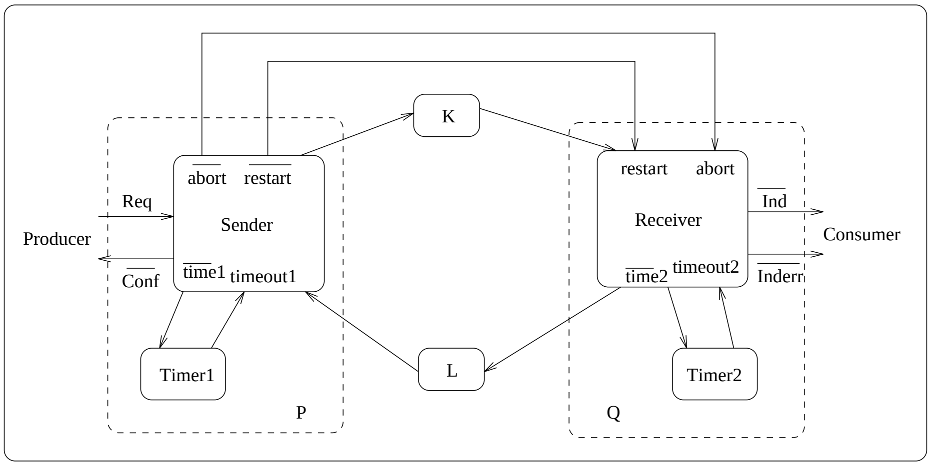

Fig. 3. The implementation of the BRP in the π-calculus

The sender and its timer constitute the component P of the system. They communicate via their private channels $time1$ and $timeout1$. The receiver and its timer constitute the component Q of the system. They communicate via their private channels $time2$ and $timeout2$. These two components communicate via the sender and the receiver over the private channels K, L, $abort$, and $restart$. This is pictured in figure 3.

$$P(K, L, abort, restart) \stackrel{def}{=} (\nu \ time1 \ \ timeout1) \ \ (S(K, L, abort, restart) \mid T1)$$

$$Q(K, L, abort, restart) \stackrel{def}{=} (\nu \ time2 \ \ timeout2) \ \ (R(K, L, abort, restart) \mid T2)$$

The external event corresponding to the loss of a message (resp. loss of an acknowledgment) is modeled by the agent $loss\_msg$ (resp. $loss\_ack$) which can intercept the message (resp. the acknowledgment) and return to its initial state. These two events can happen at any moment.

$loss\_msg \stackrel{def}{=} K(first \ last \ tag \ m).loss\_msg$

$loss\_ack \stackrel{def}{=} L(\ ).loss\_ack$

Hence, the system is completely described by the parallel composition of the components P and Q, and the external events $loss\_msg$ and $loss\_ack$.

$$System \stackrel{def}{=} (\nu\ K\ L\ abort\ restart)$$
$$(P(K, L, abort, restart) \mid Q(K, L, abort, restart) \mid loss\_msg \mid loss\_ack)$$

Note that the configuration of the system does not change during the execution of the protocol: the links are static. However, the mobility would be easily expressed in the π-calculus.

6 The correctness proof of the BRP

The purpose of this section is to prove formally that the implementation of the BRP (*System*) and its abstract view (S_0) have equivalent behaviors so they have the same observational properties, for example *deadlock-freeness*. In the π-calculus, the notion of behavioral equivalence is made mathematically precise by using *bisimulations* [23]. In our proof, we use some algebraic properties of these bisimulations and we recall them in appendix B. Our method is inspired by Orava's and Parrow's method [25]. The proof follows these steps:

1. Analyze the system by applying, repeatedly, the expansion law (E) in order to determine its intermediate states by using strong ground equivalence $\dot{\sim}$. For example, the system $(\overline{x}y.P \mid x(u).Q)$ is expanded to $(\overline{x}y.P + x(u).Q + \tau.(P \mid Q\{x/u\}))$, then we iterate the expansion on the new states $\overline{x}y.P$, $x(u).Q$ and $\tau.(P \mid Q\{x/u\})$. This step leads to a set RE_0 of mutual recursive equations between the obtained states.
2. Build the fix-point of RE_0. This results in a new set RE_1 of mutual recursive equations.
3. Simplify RE_1 by using the τ-laws, by identifying and substituting in the equations equivalent expressions up to weak bisimulation $\simeq$, and by eliminating τ-loops from equations using the law (L) (the law (U1) cannot be applied if the equations are not guarded). This step leads to a new reduced set RE_2 of mutual recursive equations.
4. Build the fix-point ABS of the equations defining the abstract view.
5. Finally, prove that RE_2 is a solution of ABS. Then, by applying the law (U1), conclude that RE_2 and ABS are equivalent.

Starting from the implementation of the BRP (*System*), the step 1 is first applied separately to the components P and Q, then it is applied to the parallel composition of their expansion with the external events *loss_msg* and *loss_ack*. This technique has a great advantage: it is *modular* in that we never have to analyze and compute on the whole system description at once.

Because of lack of space, we cannot give the complete proof. The step 1 results in twenty four equations parameterized by the file to be transferred. The step 3 leads to a system of three equations which is proved equivalent to the abstract view. The complete proof is described in an internal report [18]. It is done manually, requires about three man-month and its writing is about 50 pages.

The method we use is really interesting. It provides a clear distinction between the implementation and the abstraction of the system, proving the equivalence of the two views. Someone who wants to use the protocol as a component of a more complex system has just to use its abstract view which is simple and provides exactly its observable behavior.

7 Other implementations and proofs of the BRP

Now, we give a sketchy comparison between the implementations and proofs of the BRP presented in the studied papers, focusing on the following aspects: the modeling of time, the synchronization between the sender and the receiver, the implementation difficulties and the proof approach.

7.1 The modeling of time

Like us, in [1, 9, 13, 14], the formalism does not provide explicit time. The modeling of time in [9] is close to ours. The timers just have to expire, and the authors only care about scheduling of actions. In [1, 13, 14], the timers are represented by timer events. For example, the timeout event corresponding to $timer_1$ is defined to occur when a message or an acknowledgment is lost. In [8], the authors used timed automata. A timed automaton is a classical finite state automaton equipped with clock variables and state invariants which constrain the amount of time the system may idle in a state. So, the protocol verification allows them to obtain tight constraints on the amount of the timers.

7.2 The synchronization between the sender and the receiver

In [9], the synchronization between the sender and the receiver, done via the channels *abort* and *restart* in our case, is enforced by two extra signals *lost* and *ready*. To avoid that a message arrives after $timer_1$ expires, the channels K and L send a signal *lost* to $timer_1$ indicating that a timeout may occur. When an abort occurs, the sender sends a signal *ready* to the receiver asking it to stop $timer_2$. Then, the receiver returns a signal *ready* to the sender allowing it to transfer a new list. Since there is a strong connection between the sender, the receiver, $timer_1$ and $timer_2$, the resulting implementation is not modular.

In [13, 14], the loss of a message or an acknowledgment causes a timeout action of $timer_1$. After an abort, the sender starts a new timer called $timer_3$. When $timer_2$ expires, the receiver generates a timeout action for $timer_3$ so that the sender can proceed and handle the next request. This solution requires that $timer_3 \succ timer_2$ and can be hardly reused if time constraints have to be changed.

In [1], a loss of the last acknowledgment causes a misleading abort of the sender. The receiver considers that the transfer is already completed, so any retransmission done by the sender will not be acknowledged. Furthermore, the variable representing the number of retransmissions is still shared by the sender and the receiver in the last refinement.

In [8], urgent transitions, performed without any delay, are used to forbid the sender to stay arbitrarily long in a state and to avoid that the receiver times out without abortion of the sender. After a failure, an additional delay of $SYNC$ units is set to the sender to ensure that it does not start transmitting a new file before the receiver has properly reacted to the failure.

7.3 The implementation and the proof

In [9], the BRP is defined in μCRL as the parallel composition of its components: the sender, the receiver, $timer_1$ and $timer_2$, as we have done. The authors use the *branching bisimulation*, a strong variant of weak bisimulation (no difference observable) which is a model of μCRL theory [11]. They prove manually the equivalence between the protocol and its abstract view by applying the Recursive Specification Principle (equivalent to our law (U1)). But in their case, the protocol can start transmission of a list in two distinct modes: either the receiver knows the next alternating bit (after a successful transmission), either no (after a transmission abort). For this reason, the intermediate system is defined by eight equations considering the two modes. In our case, this system is simply the abstract view and is defined by three equations, so the proof is facilitated. Their proof is mechanized in the proof-assistant Coq [7]. The authors encode the syntax, axioms and rules of μCRL in Coq. However, they do not use the Recursive Specification Principle, but instead encode the system of recursive equations by a unique equation in Coq. So, their BRP implementation in μCRL is compact and formal, but the proof in Coq required a detailed encoding so that the resulting Coq specification is fairly large.

In [14], the authors specify each component of the protocol by an I/O automaton (the sender, the receiver, and channels K and L). Then, they define the full protocol as the parallel composition of these I/O automata. However, the model forces them to specify, for all possible states, what happens if an input action occurs. This leads to the explosion of the I/O automata. The correctness criteria of the protocol is a refinement argument showing that the BRP I/O automata implements the abstract view I/O automata. The authors prove that the BRP is deadlock-free. Moreover, a number of protocol invariants is presented. The proofs of these invariants lead to the following conclusions. The protocol may use a single bidirectional medium to implement both channels K and L. At each reachable state of the protocol, at most one of the four components enables a locally controlled action; this means that the protocol operates in a fully sequential way. The only information conveyed by an acknowledgment is the fact of its arrival itself, the rest is redundant. Finally, the field $first$ of the messages conveys no information and is redundant; it can be determined by the state of the receiver and the other fields of the transmitted message. However, the most difficulties with I/O automata verifications is finding the appropriate automata, the refinement relation and the invariants. The safety part of the proofs is mechanically checked using Coq. The notions from I/O automata theory are encoded directly for the BRP. So, it is difficult to reuse this encoding for other applications. The authors have not checked the liveness property because this would have required a considerable effort.

In [1], the author constructs formally the protocol by successive refinements. The implicit time in the abstract view is extended gradually to obtain the implementation. Each refinement step is proved to satisfy the properties expressed in the preceding one. This construction approach required seven refinements which deal with gradual distribution of various aspects of the protocol that are global in the abstract view. The first and the second refinement introduce in the sender and in the receiver variables which express the termination of the protocol. The third, fourth and fifth refinements are concerned with the distribution of the data transmission. The remaining refinements are concerned with the localization of the control in the protocol. The deadlock-freeness property is proved provided the protocol is performed in a fully sequential way. Moreover, the termination of the protocol is proved by determining a sequence of natural number expressions that decrease lexicographically after each protocol action. But, the most difficulty of this work is to find the appropriate refinements; this task is nontrivial.

In [8], the BRP is modeled by a network of timed automata. Channels K and L are modeled as queues of unbounded capacity, and the data is removed from the transmitted message. The authors verify the protocol in UPPAAL which reduces the verification problem to solving a set of constraints on clock variables. So, they obtain precise amounts of the timers. However, data in UPPAAL is restricted to clocks and integers and value passing at synchronization is not supported. For these reasons, the data was removed from the transmitted message. If data is included in the model, this would lead to an explosion of the amount of states and transitions in UPPAAL. Value passing is modeled by shared variables assignments, this requires to split some transitions. Channels K and L were reduced from unbounded queues to one-place buffers. Conditions like $a \neq b$ are not handled by UPPAAL. So, some transitions are splitted in locations that must be performed atomically. To this end, committed locations that forbid interference with the actions of other timed automata are introduced. The properties of the FTS specification are not invariant and can hardly be expressed using the property language of UPPAAL. Moreover, since the data is removed from their specification, properties of the FTS concerning the transmitted data are not checked. So, the protocol is only checked for small values of the file length and the number of retransmissions. The correctness of the protocol when omitting the timing aspects is also checked using SPIN [12]. The FTS description in PROMELA, the modeling language provided by SPIN, is obtained by a straightforward translation of the abstract view of [9]. Moreover, the BRP implementation is close to that of [9]. So, the remarks on [9] can be made here.

7.4 The abstraction approach

Now, we discuss the opposed approach taken in [13] which starts from an implementation to deduce an abstract view. The authors first analyze a scaled-down version (i.e. finite state system) of the BRP using Murϕ, a state exploration tool, as a debugging aid. Then, they translate the Murϕ description into PVS and modify manually a few of the PVS declarations to obtain the infinite state implementation. This yields two PVS theories. The first one contains the protocol itself. It is modeled by a predicate that holds for a sequence of reachable states. Their modeling of the synchronization between the sender and the receiver is the same as [14]. So, the remarks on [14] apply here. Moreover, the BRP implementation in PVS is too detailed and not so formal. The second theory contains the correctness criteria which is defined by an invariant. This invariant needs to be greatly strengthened in order to be provable, and this invariant strengthening is the real challenge of the proof. Finally, from the complete implementation in PVS, they deduce a finite state abstraction which bound the resources of unboundedness in the state space that are the message data, the number of retransmissions and the file length. They show that the mapping between the implementation and the abstract view preserves the initialization predicate, the next-state relation and the properties. They used the model checkers SMV, Murϕ and an extension of PVS with the modal μ-calculus for the final model checking. However, the most difficulty of Havelund's and Shanker's approach is to find the protocol abstraction: no technique is provided to mechanize the abstraction research. For example, to find the abstraction of the sliding window protocol is a real challenge.

8 Conclusions

We first present our correctness proof of the BRP using the π-calculus. The major advantages of our approach are the following. The description of the protocol is compact and entirely formal. Moreover, the exhaustive analysis of all possible cases gives a good understanding of the protocol; it allows us to detect several implementation errors. In fact, the BRP implementation was modified three times. The approach is modular since we never have to handle the whole protocol description at once. So, the implementation can be reused easily if specification changes occur. Furthermore, our correctness criteria is highly informative because the protocol is proved equivalent to the specification representing its external behavior. Finally, the π-calculus laws are simple and the proof by bisimulation is purely procedural. So, large parts of the proof can be mechanized.

Without the help of a prover, the exhaustive analysis of all possible cases is tedious and is hard to control and to maintain after implementation changes due to errors detection. Our aim is to elaborate a general methodology for the design of communication protocols which allows to reduce the effort to prove their correctness by bisimulation in the π-calculus. Since our BRP implementation is modular, actually, we are investigating a compositional proof of the BRP by using the relativized bisimulation [16]. Next, we want to extend the methodology in order to prove mobile protocols and liveness properties. The proofs are done by hand for the moment, another objective is to mechanize at least parts of the proofs.

Having compared with other works, the π-calculus appears as a really convenient framework for encoding and analyzing communication protocols. Some other languages like μ-CRL also offer some possibilities of modularity and coding of data but the π-calculus has a functional treatment of names, which allows extensions of protocols by mobility features.

References

1. Abrial, J-R.: Specification and Design of a Transmission Protocol by Successive Refinements using B, 1997.
2. Abrial, J-R.: The B-Book. Cambridge University Press, 1996.
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science, **126** (1994) p183–235.
4. Alur, R., Henzinger, T., Sontag, E.D.: Hybrid Systems III. LNCS 1066, Springer-Verlag, 1996.
5. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: UPPAAL – A tool suite for the automatic verification of real-time systems. In [4], p232-243.
6. Bartlett, K.A., Scantlebury, R.A., Wilkinson, P.T.: A note on reliable full-duplex transmission over half-duplex links. Communications of the ACM, **12(5)** (1969) p260–261.
7. Cornes, C., Courant, J., Filliatre, J.C., Huet, G., Manoury, P., Paulin-Mohring, C., Munoz, C., Murthy, C., Parent, C., Saibi, A., Werner, B.: The Coq Proof Assistant Reference Manual version 5.10. Technical Report, INRIA Rocquencourt, France, February 1995.
8. D'Argenio, P.R., Katoen, J.P., Ruys, T.C., Tretmans, J.: The Bounded Retransmission Protocol must be on Time !. TACAS'97.
9. Groote, J.F., Van de Pol, J.: A Bounded Retransmission Protocol for Large Data Packets. CAV'96, LNCS 1101, 1996.
10. Groote, J.F., Ponse, A.: The syntax and semantics of μCRL. Technical report CS-R9076, CWI, Amsterdam, December 1990.
11. Groote, J.F., Ponse, A.: Proof theory for μCRL: a language for processes with data. In Andrews, D.J., Groote, J.F., and Middelburg, C.A., editors, Proc. of the Int. Workshop on Semantics of Specification Languages, p232–251. Workshops in Computing, Springer Verlag, 1994.
12. Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice-Hall, 1991.
13. Havelund, K., Shankar, N.: Experiments in Theorem Proving and Model Checking for Protocol Verification. In Proceeding of FME, March 1996, Oxford.
14. Helmink, L., Sellink, M.P.A., Vaandrager, F.W.: Proof checking a data link protocol. In Barandregt, H., and Nipkow, T., editors, Types for proofs and programs, LNCS 806, p127–165, Springer-Verlag, 1994.
15. Janssen, G.: ROBDD Software. Department of Electrical Engineering, Eindhoven University of Technology, October 1993.
16. Larsen, K., Milner, R.: A Complete Protocol Verification Using Relativized Bisimulation. In Proceeding 14th Colloquium on Automata, Languages and Programming, LNCS 267, Springer-Verlag, 1987.
17. Lynch, N.A., Tuttle, M.R.: Hierarchical Correctness Proofs for Distributed Algorithms. In Proceeding of the 6th Annual Symposium on Principles of Distributed Computing, New York, p137–151, ACM Press, 1987.

18. Mammass, B.: A proof of the Bounded Retransmission Protocol in the π-calculus. Technical report, LIP6, 1997.
19. McMillan, K.L.: Symbolic Model ChecKing. Kluwer Academic Publishers, Boston, 1993.
20. Melton, R., Dill, D.L., Norris Ip., C.: Murphi Annotated Reference Manual, version 2.6. Technical Report, Stanford University, Palo Alto, California, USA, November 1993.
21. Milner, R.: Communication and Concurrency. Prentice-Hall, 1989.
22. Milner, R.: The polyadic π-calculus: a tutorial. LFCS, technical report ECS-LFCS-91-180, October 1991.
23. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, Part 1. LFCS, technical report ECS-LFCS-89-85, June 1989.
24. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, Part 2. LFCS, technical report ECS-LFCS-89-86, June 1989.
25. Orava, F., Parrow, J.: An Algebraic Verification of a Mobile Network. Formal Aspects of Computing, 4(6), p497–543, 1992.
26. Owre, S., Rushby, J., Shankar, N., Henke, F.von.: Formal Verification For Fault-tolerant Architectures: Prolegomena to the Design of PVS. IEEE Transactions on Software Engineering, 21(2), p107–125, February 1995.

A The implementation of the receiver in the π-calculus

$$\begin{array}{l}
R(K,L,abort,restart) \stackrel{def}{=} [\![False]\!]rtag.[\![False]\!]end.[\![False]\!]t2on. \\
\qquad\qquad Wait_msg(K,L,abort,restart,rtag,end,t2on)
\end{array}$$

$$\begin{array}{l}
Wait_msg(K,L,abort,restart,rtag,end,t2on) \stackrel{def}{=} \\
\quad K(first\ \ last\ \ tag\ \ m).Treat(K,L,abort,restart,first,last,tag,m,rtag,end,t2on) \\
\quad + \ If\ \ equal_bool(t2on,True)\ \ then \\
\qquad\quad timeout2.abort.Abort(K,L,abort,restart,end) \\
\quad + \ restart.If\ \ equal_bool(t2on,True)\ \ then\ \ timeout2.R(K,L,abort,restart) \\
\qquad\qquad else\ \ R(K,L,abort,restart)
\end{array}$$

$$\begin{array}{l}
Treat(K,L,abort,restart,first,last,tag,m,rtag,end,t2on) \stackrel{def}{=} \\
\quad If\ \ equal_bool(tag,rtag)\ \ then \\
\qquad \overline{L}.Wait_msg(K,L,abort,restart,rtag,end,t2on) \\
\quad else \\
\qquad If\ \ equal_bool(first,True)\ \ then \\
\qquad\quad [\![tag]\!]rtag.Indicate(K,L,abort,restart,first,last,m,rtag,end,t2on) \\
\qquad else\ \ [\![tag]\!]rtag.timeout2. \\
\qquad\qquad Indicate(K,L,abort,restart,first,last,m,rtag,end,t2on)
\end{array}$$

$$\begin{array}{l}
Indicate(K,L,abort,restart,first,last,m,rtag,end,t2on) \stackrel{def}{=} \\
\quad If\ \ equal_bool(last,True)\ \ then \\
\qquad [\![True]\!]end.\overline{\mathtt{Ind}}\ m\ \textsc{last}.\overline{L}.\overline{time2}.[\![True]\!]t2on. \\
\qquad\qquad Wait_msg(K,L,abort,restart,rtag,end,t2on) \\
\quad else\ \ If\ \ equal_bool(first,True)\ \ then
\end{array}$$

$$
\begin{array}{ll}
& \overline{\texttt{Ind}}\ m\ \textsc{first}.\overline{L}.\overline{time2}.[\![True]\!]t2on. \\
& \quad Wait\_msg(K, L, abort, restart, rtag, end, t2on) \\
else & \overline{\texttt{Ind}}\ m\ \textsc{inc}.\overline{L}.\overline{time2}.[\![True]\!]t2on. \\
& \quad Wait\_msg(K, L, abort, restart, rtag, end, t2on)
\end{array}
$$

$$
\begin{array}{lll}
Abort(K,L, abort, restart, end) & \stackrel{def}{=} & If\ \ equal\_bool(end, True)\ \ then\ \ R(K, L, abort, restart) \\
& & else\ \ \overline{\texttt{Inderr}}.R(K, L, abort, restart)
\end{array}
$$

B The π-calculus algebraic theory

The *strong ground equivalence* $\dot{\sim}$ corresponds to behavioral a equivalence where the precise amount of internal actions τ is significant. For example, we distinguish the agent $\tau.\tau.0$ from the agent $\tau.0$. In contrast, the *weak ground equivalence* $\dot{\simeq}$ identifies this two agents; the internal actions τ are significant only insofar as they preempt other actions.

The algebraic laws for strong ground equivalence $\dot{\sim}$, as stated in [24], are described below. To state them in a compact way, we define the derived prefix $\overline{x}(\widetilde{y}).P$ to mean $(\nu\ \widetilde{y})\overline{x}\widetilde{y}.P$ when $x \neq y$, and let α, β range over ordinary and derived prefixes. Let $fn(P)$ (resp. $bn(P)$) be the set of free (resp. bound) names in P. Hereafter, $=$ is used instead of $\dot{\sim}$ to allow different interpretations of the laws.

- (A) $P \equiv Q \vdash P = Q$ (α-conversion)
- (C0) $P = Q \vdash \tau.P = \tau.Q$, $P + R = Q + R$, $(\nu\ x)P = (\nu\ x)Q$
  $\overline{x}y.P = \overline{x}y.Q$, $P \mid R = Q \mid R$, $[x = y]P = [x = y]Q$
- (C1) $x(y).P = x(y).Q$ iff $P\{z/y\} = Q\{z/y\}$, $\forall\ z$
- 0 is a zero for $+$, and $+$ is idempotent, commutative and associative.
- (R0) $(\nu\ x)P = P$ (if $x \notin$ fn(P))
  (R1) $(\nu\ x)(\nu\ y)P = (\nu\ y)(\nu\ x)P$
  (R2) $(\nu\ x)(P + Q) = (\nu\ x)P + (\nu\ x)Q$
  (R3) $(\nu\ x)\alpha.P = \alpha.(\nu\ x)P$ (if x is not in α)
  (R4) $(\nu\ x)\alpha.P = 0$ (if x is the subject of α)
- (M0) $[x = y]P = 0$ if $x \neq y$, (M1) $[x = x]P = P$
- (I) $A(\widetilde{y}) = P\{\widetilde{y}/\widetilde{x}\}$ if $A(\widetilde{x}) \stackrel{def}{=} P$
- 0 is a zero for $\mid$, and $\mid$ is commutative and associative.
  (P3) $(\nu\ x)(P \mid Q) = P \mid (\nu\ x)Q$ (if $x \notin fn(P)$)
- (E) Let $P = \sum_i \alpha_i.P_i$, $Q = \sum_j \beta_j.Q_j$ where $bn(\alpha_i) \cap fn(Q) = \emptyset\ \forall\ i$ and $bn(\beta_j) \cap fn(P) = \emptyset\ \forall\ j$. Then
  $P \mid Q = \sum_i \alpha_i.(P_i \mid Q) + \sum_j \beta_j.(P \mid Q_j) + \sum_{\alpha_i comp \beta_j} \tau.R_{ij}$
  where the relation $\alpha_i\ comp\ \beta_j$ (α_i complements β_j) holds in the following four cases, which also define R_{ij}:
  1. α_i is $\overline{x}u$ and β_j is $x(v)$; then R_{ij} is $P_i \mid Q_j\{u/v\}$
  2. α_i is $\overline{x}(u)$ and β_j is $x(v)$; then R_{ij} is $(\nu\ w)(P_i\{w/u\} \mid Q_j\{w/v\})$ (where w is not free in $(\nu\ u)P_i$ or in $(\nu\ v)Q_j$)
  3. the two others are the converse.

The weak ground equivalence $\dot{\simeq}$ is strictly weaker that strong ground equivalence $\dot{\sim}$ and also satisfies the laws described above. In addition, it satisfies the well known τ-laws [21], these are:

- (T0) $\alpha.\tau.P \dot{\simeq} \alpha.P$
- (T1) $P + \tau.P \dot{\simeq} \tau.P$
- (T2) $\alpha.(P + \tau.Q) + \alpha.Q \dot{\simeq} \alpha.(P + \tau.Q)$.

In order to eliminate τ-loops from recursively defined agents (see [21]):

- (L) If $A = P + \tau.A$ and $B = \tau.P$ then $A \dot{\simeq} B$

We define *strong (non-ground) equivalence* $\sim$ as strong ground equivalence under all substitutions σ of non-constant names, i.e., $P \sim Q$ iff $P\sigma \dot{\sim} Q\sigma$, for all substitutions σ from non-constant names to names. We define *weak (non-ground) equivalence* $\simeq$ in a similar way.

The main use of the non-ground equivalences is in the laws for recursively defined agents which we adopt from [24]. To formulate them, we need some additional notations. Let E, F, ... represent agent expressions; these are like agents with "holes" where agents or agent identifiers can be inserted. Let $E(P_1, \ldots, P_n)$ be the agent which is the result of inserting $P_1, \ldots, P_n$ into E. Two agent expressions E and F are (strongly/weakly) equivalent if $E(\widetilde{P})$ is (strongly/weakly) equivalent to $F(\widetilde{P})$ for all $P_1, \ldots, P_n$.

The first law for recursion (U0) means that if the right hand sides of definitions are transformed, respecting equivalence, then the agent defined is the same up to equivalence. This law holds for strong and weak non-ground equivalence (but fails for the ground equivalences).

(U0) Suppose that $E_1, \ldots, E_n$ and $F_1, \ldots, F_n$ are expressions and $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ identifiers such that for all i: $E_i = F_i$ and $A_i(\widetilde{x_i}) = E_i(A_1, \ldots, A_n)$ and $B_i(\widetilde{x_i}) = F_i(B_1, \ldots, B_n)$ Then $A_i(\widetilde{x_i}) = B_i(\widetilde{x_i})$ for all i.

The second law (U1) means that if two agents satisfy the same set of recursive equations, then the agents are equivalent. This law holds for strong non-ground equivalence provided $E_1, \ldots, E_n$ are *weakly guarded* (i.e., all occurrences of P_j in $E_i(P_1, \ldots, P_n)$ are within a prefix operator). Furthermore, it holds for weak non-ground equivalence provided $E_1, \ldots, E_n$ are *guarded* (i.e., all occurrences of P_j in $E_i(P_1, \ldots, P_n)$ are within an output or input prefix operator), and *sequential* (i.e., no E_i contains a parallel composition).

(U1) Suppose that $E_1, \ldots, E_n$ are expressions and $P_1, \ldots, P_n$ and $Q_1, \ldots, Q_n$ are agents such that for all i: $P_i = E_i(P_1, \ldots, P_n)$ and $Q_i = E_i(Q_1, \ldots, Q_n)$ Then $P_i = Q_i$ for all i.

This article was processed using the LaTeX macro package with LLNCS style