# Cones and Foci for Protocol Verification Revisited ⋆

Wan Fokkink[1,2] and Jun Pang[1]

[1] CWI, Department of Software Engineering, PO Box 94079, 1090 GB Amsterdam, The Netherlands, {wan,pangjun}@cwi.nl
[2] Vrije Universiteit Amsterdam, Department of Theoretical Computer Science, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, wanf@cs.vu.nl

**Abstract.** We define a cones and foci proof method, which rephrases the question whether two system specifications are branching bisimilar in terms of proof obligations on relations between data objects. Compared to the original cones and foci method from Groote and Springintveld [22], our method is more generally applicable, and does not require a preprocessing step to eliminate $\tau$-loops. We prove soundness of our approach and give an application.

## 1 Introduction

In order to make data a first class citizen in the study of processes, the language $\mu$CRL [21] combines the process algebra ACP [3] with equational abstract data types [27]. Processes are intertwined with data: Actions and recursion variables are parametrized by data types; an if-then-else construct allows data objects to influence the course of a process; and alternative quantification sums over possibly infinite data domains. Internal activity of a process can be hidden by a hiding operator $\tau_I$, which renames all internal actions (i.e., the actions in the set $I$) into the hidden action $\tau$ [5].

A labeled transition system is associated to each $\mu$CRL specification. Two $\mu$CRL specifications are considered equivalent if the initial states of their labeled transition systems are branching bisimilar [16]. Verification of system correctness boils down to checking whether the implementation of a system (with all internal activity hidden) is branching bisimilar to the specification of the desired external behavior of the system. Checking whether two states are branching bisimilar can be performed efficiently [23]. The $\mu$CRL toolset [7] supports the generation of labeled transition systems, together with reduction modulo branching bisimulation equivalence, and allows model checking of temporal logic formulas [10] via a back-end to the CADP toolset [12].

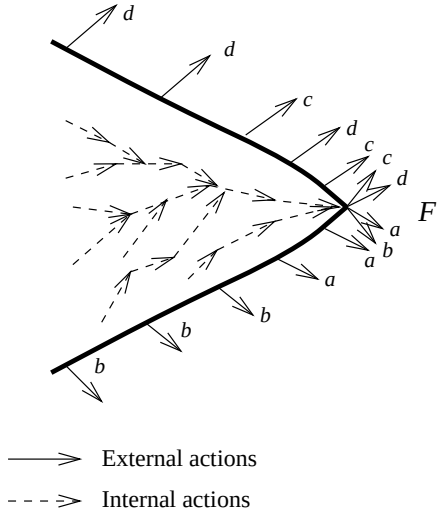
This approach to verify system correctness has three important drawbacks. First, the labeled transition systems of the $\mu$CRL specifications involved must be

⋆ This research is supported by the Dutch Technology Foundation STW under the project CES5008: Improving the quality of embedded systems using formal design and systematic testing.

generated; often the labeled transition system of the implementation of a system cannot be generated, as it is too large, or even infinite. Second, this generation usually requires a specific choice for one network or data domain; in other words, only the correctness of an instantiation of the system is proved. Third, support from and rigorous formalization by theorem provers and proof checkers is not readily available.

Groote and Springintveld [22] introduced the *cones and foci* method, which rephrases the question whether two $\mu$CRL specifications are branching bisimilar in terms of proof obligations on relations between data objects. These proof obligations can be derived by means of algebraic calculations, in general with the help of invariants (i.e., properties of the reachable states) that are proved separately. This method was used in the verification of a considerable number of real-life protocols (e.g., [15, 20, 34]), often with the support of a theorem prover or proof checker.

The main idea of this method is that quite often in the implementation of a system, internal actions progress inertly towards a state in which no internal actions can be executed; such a state is declared to be a *focus point*. The *cone* of a focus point consists of the states that can reach this focus point by a string of internal actions. In the absence of infinite sequences of internal actions, each state belongs to a cone. This core idea is depicted below. Note that the external actions at the edge of the depicted cone can also be executed in the ultimate focus point $F$; this is essential for soundness of the cones and foci method, as otherwise internal actions in the cone would not be inert.




*Linear process equations* [6] constitute a restricted class of $\mu$CRL specifications in some kind of linear format. Algorithms have been developed to transform $\mu$CRL specifications into this linear format [19, 24, 35]. In a linear process equation, the states of the associated labeled transition system are data objects.

Assume that the implementation of a system and its desired external behavior are both given in the form of a linear process equation. In the cones and

foci method, a *state mapping* $\phi$ relates each state of the implementation to a state of the desired external behavior. Groote and Springintveld [22] formulated *matching criteria*, consisting of relations between data objects, which ensure that states $s$ and $\phi(s)$ are branching bisimilar.

If an implementation, with all internal activity hidden, includes infinite sequences of $\tau$-actions, then Groote and Springintveld [22] distinguish between *progressing* and *non-progressing* $\tau$-actions. Their requirements are that (1) there is no infinite sequence of progressing $\tau$-actions, (2) non-progressing $\tau$-actions are only executed at a focus point, and (3) a focus point cannot perform progressing $\tau$-actions. A *pre-abstraction function* divides occurrences of $\tau$-actions in the implementation into progressing and non-progressing ones, and only progressing $\tau$'s are abstracted away; in many cases it is far from trivial to define the proper pre-abstraction. Finally, a special *fair abstraction rule* [2] can be used to try and eliminate the remaining (non-progressing) $\tau$'s.

In this paper, we propose an adaptation of the cones and foci method, in which the cumbersome treatment of infinite sequences of $\tau$-actions is no longer necessary. This improvement of the cones and foci method was conceived during the verification of a sliding window protocol [13], where the adaptation simplified matters considerably. As before, the method deals with linear process equations, requires the definition of a state mapping, and generates the same matching criteria. However, we allow the user to freely assign which states are focus points (instead of prescribing that they are the states in which no progressing $\tau$-actions can be performed), as long as each state is in the cone of a focus point. We do allow infinite sequences of internal actions. Since the meaning of recursive specifications that include infinite sequences of $\tau$-actions is ambiguous, we leave the hiding operator $\tau_I$ around the $\mu$CRL specification of the implementation in place. No distinction between progressing and non-progressing internal actions is needed, and loops of internal actions are eliminated without having to resort to a fair abstraction rule.

We prove that our method is sound modulo branching bisimulation equivalence. Furthermore, we apply our method to the Concurrent Alternating Bit Protocol [26], which served as the main example in [22]. While the old cones and foci method required a typical cumbersome treatment of $\tau$-loops, here we can take these $\tau$-loops in our stride.

*Related Work* In compiler correctness, advances have been made to validate programs at a symbolic level with respect to an underlying simulation notion (e.g., [9, 17, 30]). The methodology surrounding cones and foci incorporates well-known and useful concepts such as the precondition/effect notation [25, 28], invariants and simulations. Linear process equations resemble the UNITY format [8] and recursive applicative program schemes [11]; state mappings are comparable to refinement mappings [29, 32] and simulation [14]. Van der Zwaag [36] gave an adaptation of the cones and foci method from [22] to a timed setting, modulo timed branching bisimulation equivalence. We leave it as an open question whether our innovations for the cones and foci method can also be introduced in this timed setting.

## 2 Preliminaries

### 2.1 $\mu$CRL

$\mu$CRL [21] is a language for specifying distributed systems and protocols in an algebraic style. It is based on process algebra extended with equational abstract data types. In a $\mu$CRL specification, one part specifies the data types, while a second part specifies the process behavior. We do not describe the treatment of data types in $\mu$CRL in detail. For our purpose it is sufficient that processes can be parametrized with data. We assume the data sort of booleans *Bool* with constant $\mathsf{T}$ and $\mathsf{F}$, and the usual connectives $\wedge$, $\vee$, $\neg$ and $\Rightarrow$. For a boolean $b$, we abbreviate $b = \mathsf{T}$ to $b$ and $b = \mathsf{F}$ to $\neg b$.

The specification of a process is constructed from action names, recursion variables and process algebraic operators. Actions and recursion variables carry zero or more data parameters. There are two predefined actions in $\mu$CRL: $\delta$ represents deadlock, and $\tau$ a hidden action. These two actions never carry data parameters.

Processes are represented by process terms, which describe the order in which the actions from a set *Act* may happen. A process term consists of action names and recursion variables combined by process algebraic operators. $p{\cdot}q$ denotes sequential composition and $p+q$ non-deterministic choice, summation $\sum_{d:D} p(d)$ provides the possibly infinite choice over a data type $D$, and the conditional construct $p \lhd b \rhd q$ with $b$ a data term of sort *Bool* behaves as $p$ if $b$ and as $q$ if $\neg b$. Parallel composition $p \parallel q$ interleaves the actions of $p$ and $q$; moreover, actions from $p$ and $q$ may also synchronize to a communication action, when this is explicitly allowed by a predefined communication function. Two actions can only synchronize if their data parameters are semantically the same, which means that communication can be used to represent data transfer from one system component to another. Encapsulation $\partial_H(p)$, which renames all occurrences in $p$ of actions from the set $H$ into $\delta$, can be used to force actions into communication. Finally, hiding $\tau_I(p)$ renames all occurrences in $p$ of actions from the set $I$ into $\tau$. The syntax and semantics of $\mu$CRL are given in [21].

### 2.2 Labeled transition systems

Labeled transition systems (LTSs) capture the operational behavior of concurrent systems. An LTS consists of transitions $s \xrightarrow{a} s'$, denoting that the state $s$ can evolve into the state $s'$ by the execution of action $a$. To each $\mu$CRL specification belongs an LTS, defined by the structural operational semantics for $\mu$CRL in [21].

**Definition 1 (Labeled transition system).** *A* labeled transition system *is a tuple $(S, Lab, \rightarrow, s_0)$, where $S$ is a set of states, $Lab$ a set of transition labels, $\rightarrow \subseteq S \times Lab \times S$ a transition relation, and $s_0$ the initial state. A transition $(s, \ell, s')$ is denoted by $s \xrightarrow{\ell} s'$.*

Here, $S$ consists of $\mu$CRL specifications, and $Lab$ consists of actions from $Act \cup \{\tau\}$ parametrized by data. We define *branching bisimilarity* [16] between states in LTSs. Branching bisimulation is an equivalence relation [4].

**Definition 2 (Branching bisimulation).** *Assume an LTS. A symmetric binary relation $B$ on states is a* branching bisimulation *if $sBt$ and $s \xrightarrow{\ell} s'$ implies:*

- *either $\ell = \tau$ and $s' \mathcal{B}\, t$;*
- *or there is a sequence of (zero or more) $\tau$-transitions $t \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_0$ such that $sBt_0$ and $t_0 \xrightarrow{\ell} t'$ with $s'Bt'$.*

*Two states $s$ and $t$ are* branching bisimilar*, denoted by $s \leftrightarrows_b t$, if there is a branching bisimulation relation $\mathcal{B}$ such that $s \mathcal{B}\, t$.*

The $\mu$CRL toolset [7] supports the generation of labeled transition systems of $\mu$CRL specifications, together with reduction modulo branching bisimulation equivalence and model checking of temporal logic formulas. This approach has been used to analyze a wide range of protocols and distributed systems (e.g., [1, 18, 31, 33]).

In this paper we focus on analyzing protocols and distributed systems on the level of their symbolic specifications.

### 2.3 Linear process equations

A *linear process equation* (LPE) is a one-line $\mu$CRL specification consisting of actions, summations, sequential compositions and conditional constructs. In particular, an LPE does not contain any parallel operators, encapsulations or hidings. In essence an LPE is a vector of data parameters together with a list of condition, action and effect triples, describing when an action may happen and what is its effect on the vector of data parameters. Each $\mu$CRL specification that does not include successful termination can be transformed into an LPE [35].[1]

**Definition 3 (Linear process equation).** *A* linear process equation *is a $\mu$CRL specification of the form*

$$X(d{:}D) = \sum_{a \in Act \cup \{\tau\}} \sum_{e:E} a(f_a(d,e)) \cdot X(g_a(d,e)) \lhd h_a(d,e) \rhd \delta$$

*where $f_a : D \times E \to D$, $g_a : D \times E \to D$ and $h_a : D \times E \to Bool$ for each $a \in Act \cup \{\tau\}$.*

[1] To cover $\mu$CRL specifications with successful termination, LPEs should also include a summand $\sum_{a \in Act \cup \{\tau\}} \sum_{e:E} a(f_a(d,e)) \lhd h_a(d,e) \rhd \delta$. The cones and foci method extends to this setting without any complication. However, this extension would complicate the matching criteria in Definition 7. For the sake of presentation, successful termination is not taken into account here.

The LPE in Definition 3 has exactly one LTS as its solution.[2] In this LTS, the states are data elements $d{:}D$ (where $D$ may be a Cartesian product of $n$ data types, meaning that $d$ is a tuple $(d_1, ..., d_n)$) and the transition labels are actions parametrized with data. The LPE expresses that state $d$ can perform $a(f_a(d, e))$ to end up in state $g_a(d, e)$, under the condition that $h_a(d, e)$ is true. The data type $E$ gives LPEs a more general form, as not only the data parameter $d{:}D$ but also the data parameter $e{:}E$ can influence the parameter of action $a$, the condition $h_a$ and the resulting state $g_a$.

**Definition 4 (Invariant).** *A mapping $\mathcal{I} : D \to Bool$ is an invariant for an LPE, written as in Definition 3, if for all $a \in Act \cup \{\tau\}$, $d{:}D$ and $e{:}E$,*

$$\mathcal{I}(d) \wedge h_a(d, e) \;\Rightarrow\; \mathcal{I}(g_a(d, e)).$$

Intuitively, an invariant characterizes the set of reachable states of an LPE. That is, if $\mathcal{I}(d)$, and if one can involve from state $d$ to state $d'$ in zero or more transitions, then $\mathcal{I}(d')$. Namely, if $\mathcal{I}$ holds in state $d$ and it is possible to execute $a(f_a(d, e))$ in this state (meaning that $h_a(d, e)$), then it is ensured that $\mathcal{I}$ holds in the resulting state $g_a(d, e)$. Invariants tend to play a crucial role in algebraic verifications of system correctness that involve data.

## 3 Cones and foci

In this section, we present our version of the cones and foci method [22]. Suppose that we have an LPE $X(d{:}D)$ (including internal actions from a set $I$, which will be hidden) specifying the implementation of a system, and an LPE $Y(d'{:}D')$ (without internal actions) specifying the desired input/output behavior of this system. Furthermore, assume an invariant $\mathcal{I} : D \to Bool$ characterizing the reachable states of $X$. We want to prove that the implementation exhibits the desired input/output behavior.

We assume the presence of an invariant $\mathcal{I} : D \to Bool$ for $X$. In the cones and foci method, a *state mapping* $\phi : D \to D'$ relates each state of the implementation $X$ to a state of the desired external behavior $Y$. Furthermore, some states in $D$ are designated to be *focus points*. In contrast with the approach of [22], we allow to freely assign focus points, as long as each state $d{:}D$ of $X$ with $\mathcal{I}(d)$ can reach a focus point by a sequence of internal transitions. If a number of *matching criteria* for $d{:}D$ are fulfilled, consisting of relations between data objects, and if $\mathcal{I}(d)$, then the states $d$ and $\phi(d)$ are guaranteed to be branching bisimilar. These matching criteria require that (A) after hiding, all internal transitions of $d$ become invisible, (B) each external transition of $d$ can be mimicked by $\phi(d)$, and (C) if $d$ is a focus point, then vice versa each transition of $\phi(d)$ can be mimicked by $d$.

We start with defining the predicate $FC$, designating the focus points of $X$ in $D$. Next we define the state mapping together with its matching criteria.

[2] LPEs exclude "unguarded" recursive specifications such as $X = X$, which have multiple solutions.

**Definition 5 (Focus point).** *A* focus condition *is a mapping* $FC : D \to Bool$. *If* $FC(d)$, *then* $d$ *is called a* focus point.

**Definition 6 (State mapping).** *A* state mapping *is of the form* $\phi : D \to D'$.

**Definition 7 (Matching criteria).** *Let the LPE* $X$ *be of the form*

$$X(d{:}D) = \sum_{a \in Act} \sum_{e:E} a(f_a(d,e)) \cdot X(g_a(d,e)) \lhd h_a(d,e) \rhd \delta.$$

*Furthermore, let the LPE* $Y$ *be of the form*

$$Y(d'{:}D') = \sum_{a \in Act \setminus I} \sum_{e:E} a(f'_a(d',e)) \cdot Y(g'_a(d',e)) \lhd h'_a(d',e) \rhd \delta.$$

*A state mapping* $\phi : D \to D'$ *satisfies the* matching criteria *for* $d{:}D$ *if for all* $a \in Act \setminus I$ *and* $c \in I$*:*

$$\begin{array}{rl} I & \forall e{:}E\,(h_c(d,e) \Rightarrow \phi(d) = \phi(g_c(d,e)));\\ II & \forall e{:}E\,(h_a(d,e) \Rightarrow h'_a(\phi(d),e));\\ III & FC(d) \Rightarrow \forall e{:}E\,(h'_a(\phi(d),e) \Rightarrow h_a(d,e));\\ IV & \forall e{:}E\,(h_a(d,e) \Rightarrow f_a(d,e) = f'_a(\phi(d),e));\\ V & \forall e{:}E\,(h_a(d,e) \Rightarrow \phi(g_a(d,e)) = g'_a(\phi(d),e)). \end{array}$$

Matching criterion I requires that after hiding, all internal $c$-transitions from $d$ are invisible, meaning that $d$ and $g_c(d,e)$ are branching bisimilar. Criteria II, IV and V express that each external transition of $d$ can be simulated by $\phi(d)$. Finally, criterion III expresses that if $d$ is a focus point, then each external transition of $\phi(d)$ can be simulated by $d$.

**Theorem 1.** *Assume LPEs* $X(d{:}D)$ *and* $Y(d'{:}D')$ *written as in Definition 7. Let* $I \subseteq Act$, *and let* $\mathcal{I} : D \to Bool$ *be an invariant for* $X$. *Suppose that for all* $d{:}D$ *with* $\mathcal{I}(d)$,

1. $\phi : D \to D'$ *satisfies the matching criteria for* $d$, *and*
2. *there is a* $\hat{d}{:}D$ *such that* $FC(\hat{d})$ *and* $d \xrightarrow{c_1} \cdots \xrightarrow{c_n} \hat{d}$ *with* $c_1, \ldots, c_n \in I$ *in the LTS for* $X$.

*Then for all* $d{:}D$ *with* $\mathcal{I}(d)$,

$$\tau_I(X(d)) \leftrightarrow_b Y(\phi(d)).$$

*Proof.* We assume without loss of generality that $D$ and $D'$ are disjoint. Define $B \subseteq D \cup D' \times D \cup D'$ as the smallest relation such that whenever $\mathcal{I}(d)$ for a $d{:}D$ then $dB\phi(d)$ and $\phi(d)Bd$. Clearly, $B$ is symmetric. We show that $B$ is a branching bisimulation relation.

Let $sBt$ and $s \xrightarrow{\ell} s'$. First consider that case where $\phi(s) = t$. By definition of $B$ we have $\mathcal{I}(s)$.

1. If $\ell = \tau$, then $h_c(s,e)$ and $s' = g_c(s,e)$ for some $c \in I$ and $e{:}E$. By matching criterion I, $\phi(g_c(s,e)) = t$. Moreover, $\mathcal{I}(s)$ and $h_c(s,e)$ together imply $\mathcal{I}(g_c(s,e))$. Hence, $g_c(s,e)Bt$.
2. If $\ell \neq \tau$, then $h_a(s,e)$, $s' = g_a(s,e)$ and $\ell = a(f_a(s,e))$ for some $a \in Act\backslash I$ and $e{:}E$. By matching criteria II and IV, $h'_a(t,e)$ and $f_a(s,e) = f'_a(t,e)$. Hence, $t \stackrel{a(f_a(s,e))}{\rightarrow} g'_a(t,e)$. Moreover, $\mathcal{I}(s)$ and $h_a(s,e)$ together imply $\mathcal{I}(g_a(s,e))$, and matching criterion V yields $\phi(g_a(s,e)) = g'_a(t,e)$, so $g_a(s,e)Bg'_a(t,e)$.

Next consider the case where $s = \phi(t)$. Since $s \stackrel{\ell}{\rightarrow} s'$, for some $a \in Act\backslash I$ and $e{:}E$, $h'_a(s,e)$, $s' = g'_a(s,e)$ and $\ell = a(f'_a(s,e))$. By definition of $B$ we have $\mathcal{I}(t)$.

1. If $FC(t)$, then by matching criterion III, $h_a(t,e)$. So by matching criterion IV, $f_a(t,e) = f'_a(s,e)$. Hence, $t \stackrel{a(f'_a(s,e))}{\rightarrow} g_a(t,e)$. Moreover, $\mathcal{I}(t)$ and $h_a(t,e)$ together imply $\mathcal{I}(g_a(t,e))$, and matching criterion V yields $\phi(g_a(t,e)) = g'_a(s,e)$, so $g'_a(s,e)Bg_a(t,e)$.
2. If $\neg FC(t)$, then there is a $\hat{t}{:}D$ with $FC(\hat{t})$ such that $t \stackrel{c_1}{\rightarrow} ... \stackrel{c_n}{\rightarrow} \hat{t}$ with $c_1, \ldots, c_n \in I$ in the LTS for $X$. This implies that $t \stackrel{\tau}{\rightarrow} ... \stackrel{\tau}{\rightarrow} \hat{t}$ in the LTS for $\tau_I(X)$. Invariant $\mathcal{I}$, so also the matching criteria, hold for all states on this $\tau$-path. Repeatedly applying matching criterion I we get $\phi(\hat{t}) = \phi(t) = s$. So matching criterion III together with $h'_a(s,e)$ yields $h_a(\hat{t},e)$. Then by matching criterion IV, $f_a(\hat{t},e) = f'_a(s,e)$, so $t \stackrel{\tau}{\rightarrow} ... \stackrel{\tau}{\rightarrow} \hat{t} \stackrel{a(f'_a(s,e))}{\rightarrow} g_a(\hat{t},e)$. Moreover, $\mathcal{I}(\hat{t})$ and $h_a(\hat{t},e)$ together imply $\mathcal{I}(g_a(\hat{t},e))$, and matching criterion V yields $\phi(g_a(\hat{t},e)) = g'_a(s,e)$, so $sB\hat{t}$ and $g'_a(s,e)Bg_a(\hat{t},e)$.

Concluding, $B$ is a branching bisimulation.

We note that Groote and Springintveld [22] proved for their notion of their cones and foci method that it can be derived from the axioms of $\mu$CRL, which implies that their method is sound modulo branching bisimulation equivalence. We leave it as future work to try and derive our cones and foci method from the axioms of $\mu$CRL.

## 4 Application to the CABP

Groote and Springintveld [22] proved correctness of the Concurrent Alternating Bit Protocol (CABP) [26] as an application of their cones and foci method. Here we redo their correctness proof using our version of the cones and foci method, where in contrast to [22] we can take loops of internal activity in our stride.

In the CABP, data elements $d_1, d_2, \ldots$ are communicated from a data transmitter S to a data receiver R via a lossy channel, so that a message can be corrupted or lost. Therefore, acknowledgments are sent from R to S again via a lossy channel. In the CABP, sending and receiving of acknowledgments is decoupled from R and S, in the form of separate components AS and AR, respectively,

where AS autonomously sends acknowledgments to AR. This ensures a better use of available bandwidth.

S attaches a bit 0 to data elements $d_{2k-1}$ and a bit 1 to data elements $d_{2k}$, and AS sends back the attached bit to acknowledge reception. S keeps on sending a pair $(d_i, b)$ until AR receives the bit $b$ and sends the message $ac$ to S; then S starts sending the next pair $(d_{i+1}, 1-b)$. Alternation of the attached bit enables R to determine whether a received datum is really new, and alternation of the acknowledging bit enables AR to determine which datum is being acknowledged.

The CABP contains unbounded internal behavior, which occurs when a channel eternally corrupts or loses the same datum or acknowledgment. The fair abstraction paradigm [2], which underlies branching bisimulation, says that such infinite sequences of faulty behavior do not exist in reality, because the chance of a channel failing infinitely often is zero. Groote and Springintveld [22] defined a pre-abstraction function to hide all internal actions except those that are executed in focus points, and used Koomen's fair abstraction rule [2] to eliminate the remaining loops of internal actions. In our adaptation of the cones and foci method, focus points can perform internal actions, so neither pre-abstraction nor Koomen's fair abstraction rule are needed here.

The structure of the CABP is shown in Figure 1. The CABP system is built from six components.

- S is a *data transmitter*, which reads data from port 1 and transmits such a datum repeatedly via channel K, until an acknowledgment $ac$ regarding this datum is received from AR.
- K is a lossy *data transmission channel*, which transfers data from S to R. Either it delivers the datum correctly, or it can make two sorts of mistakes: lose the datum or change it into the checksum error $ce$. The non-deterministic choice between the three options is modeled by the action $j$.
- R is a *data receiver*, which receives data from K, sends freshly received data into port 2, and sends an acknowledgment to AS via port 5.
- AS is an *acknowledgment transmitter*, which receives an acknowledgment from R and repeatedly transmits it via L to AR.
- L is a lossy *acknowledgment transmission channel*, which transfers acknowledgments from AS to AR. Either it delivers the acknowledgment correctly, or it can make two sorts of mistakes: lose the acknowledgment or change it into $ae$.
- AR is an *acknowledgment receiver*, which receives acknowledgments from L and passes them on to S.

The components can perform read $r_n(...)$ and send $s_n(...)$ actions to transport data through port $n$. A read and a send action over the same port $n$ can synchronize into a communication action $c_n(...)$. For a detailed description of the data types and each component's specification in $\mu$CRL the reader is referred to [22]. The $\mu$CRL specification of the CABP is obtained by putting the six components in parallel, encapsulating the internal send and read actions at ports $\{3, 4, 5, 6, 7, 8\}$, and hiding the internal communication actions at these ports together with the action $j$.

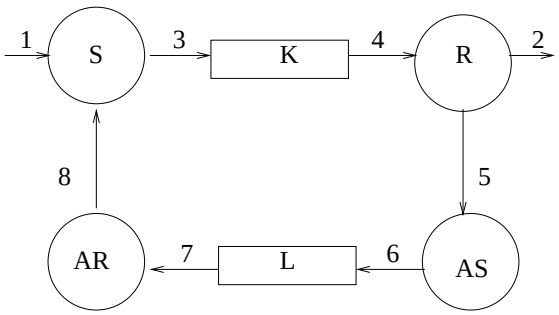



**Fig. 1.** The structure of the CABP

### 4.1 Implementation and external behavior

As a starting point we take the LPE *Sys* that is obtained from the implementation of the CABP; see [22]. It includes the sort *Bit* with elements $b_0$ and $b_1$ and with an inversion function $inv : Bit \to Bit$, and the sort *Nat* of natural numbers. The sort $D$ contains the data elements to be transferred by the protocol. $eq : S \times S \to Bool$ coincides with the equality relation between elements of the sort $S$.

**Definition 8.** *In each summand of the LPE for Sys below, we only present the parameters whose values are changed.*

$$
\begin{aligned}
&Sys(d_s{:}D, b_s{:}Bit, i_s{:}Nat, i'_s{:}Nat, d_r{:}D, b_r{:}Bit,\\
&\quad i_r{:}Nat, d_k{:}D, b_k{:}Bit, i_k{:}Nat, b_l{:}Bit, i_l{:}Nat)\\
&= \textstyle\sum_{d:D} r_1(d)\cdot Sys(d/d_s, 2/i_s) \lhd eq(i_s, 1) \rhd \delta && (1)\\
&+ c_3(\langle d_s, b_s\rangle)\cdot Sys(d_s/d_k, b_s/b_k, 2/i_k) \lhd eq(i_s, 2) \wedge eq(i_k, 1) \rhd \delta && (2)\\
&+ (j\cdot Sys(1/i_k) + j\cdot Sys(3/i_k) + j\cdot Sys(4/i_k)) \lhd eq(i_k, 2) \rhd \delta && (3)\\
&+ c_4(\langle d_k, b_r\rangle)\cdot Sys(d_k/d_r, 2/i_r, 1/i_k) \lhd eq(i_r, 1) \wedge eq(b_r, b_k) \wedge eq(i_k, 3) \rhd \delta && (4)\\
&+ c_4(\langle d_k, b_r\rangle)\cdot Sys(1/i_k) \lhd eq(i_r, 1) \wedge eq(b_r, inv(b_k)) \wedge eq(i_k, 3) \rhd \delta && (5)\\
&+ c_4(ce)\cdot Sys(1/i_k) \lhd eq(i_r, 1) \wedge eq(i_k, 4) \rhd \delta && (6)\\
&+ s_2(d_r)\cdot Sys(3/i_r) \lhd eq(i_r, 2) \rhd \delta && (7)\\
&+ c_5(ac)\cdot Sys(inv(b_r)/b_r, 1/i_r) \lhd eq(i_r, 3) \rhd \delta && (8)\\
&+ c_6(inv(b_r))\cdot Sys(inv(b_r)/b_l, 2/i_l) \lhd eq(i_l, 1) \rhd \delta && (9)\\
&+ (j\cdot Sys(1/i_l) + j\cdot Sys(3/i_l) + j\cdot Sys(4/i_l)) \lhd eq(i_l, 2) \rhd \delta && (10)\\
&+ c_7(b_l)\cdot Sys(1/i_l, 2/i'_s) \lhd eq(i'_s, 1) \wedge eq(b_l, b_s) \wedge eq(i_l, 3) \rhd \delta && (11)\\
&+ c_7(b_l)\cdot Sys(1/i_l) \lhd eq(i'_s, 1) \wedge eq(b_l, inv(b_s)) \wedge eq(i_l, 3) \rhd \delta && (12)\\
&+ c_7(ae)\cdot Sys(1/i_l) \lhd eq(i'_s, 1) \wedge eq(i_l, 4) \rhd \delta && (13)\\
&+ c_8(ac)\cdot Sys(inv(b_s)/b_s, 1/i_s, 1/i'_s) \lhd eq(i_s, 2) \wedge eq(i'_s, 2) \rhd \delta && (14)
\end{aligned}
$$

In the LPE *Sys*, $d_s$, $d_r$ and $d_k$ represent a datum of sort $D$ which S, R and K read at ports 1, 4 and 3, respectively; $b_s$, $b_r$, $b_k$ and $b_l$ are the attached alternating bit for R, S, K and L, respectively. $i_s$, $i'_s$, $i_r$, $i_k$ and $i_l$ are auxiliary parameters that are introduced by the linearization algorithm which transforms the concurrent

specification of the CADP into the LPE *Sys*; these parameters model different states of S, AR, R, K and L, respectively.

The specification of the external behavior of the CABP is a one-datum buffer, which reads a datum at port 1, and sends out this same datum at port 2.

**Definition 9.** *The LPE of the external behavior of the CABP is*

$$B(d{:}D, b{:}Bool) = \textstyle\sum_{d':D} r_1(d')\cdot B(d', \mathsf{F}) \lhd b \rhd \delta + s_2(d)\cdot B(d, \mathsf{T}) \lhd \neg b \rhd \delta.$$

### 4.2 Verification

Let $\Xi$ abbreviate $D \times Bit \times Nat \times Nat \times D \times Bit \times Nat \times D \times Bit \times Nat \times Bit \times Nat$. Furthermore, let $\xi{:}\Xi$ denote $(d_s, b_s, i_s, i'_s, d_r, b_r, i_r, d_k, b_k, i_k, b_l, i_l)$.

Invariants $\mathcal{I}_1$-$\mathcal{I}_5$, $\mathcal{I}_7$ below were taken from [22]. $\mathcal{I}_1$-$\mathcal{I}_5$ describe the range of the data parameters $i_s$, $i'_s$, $i_k$, $i_r$, and $i_l$, respectively. $\mathcal{I}_6$ says that $b_s$ and $b_k$ remain equal until S gets an acknowledgment from AR. $\mathcal{I}_7$ expresses that each component in Figure 1 either has received information about the datum being transmitted which it must forward, or did not yet receive this information.

**Definition 10.**

$$\begin{aligned}
&\mathcal{I}_1(\xi) \equiv eq(i_s, 1) \vee eq(i_s, 2)\\
&\mathcal{I}_2(\xi) \equiv eq(i'_s, 1) \vee eq(i'_s, 2)\\
&\mathcal{I}_3(\xi) \equiv eq(i_k, 1) \vee eq(i_k, 2) \vee eq(i_k, 3) \vee eq(i_k, 4)\\
&\mathcal{I}_4(\xi) \equiv eq(i_r, 1) \vee eq(i_r, 2) \vee eq(i_r, 3)\\
&\mathcal{I}_5(\xi) \equiv eq(i_l, 1) \vee eq(i_l, 2) \vee eq(i_l, 3) \vee eq(i_l, 4)\\
&\mathcal{I}_6(\xi) \equiv \neg eq(i_k, 1) \wedge eq(i_s, 2) \Rightarrow eq(b_s, b_k)\\
&\mathcal{I}_7(\xi) \equiv (eq(i_s, 1) \Rightarrow eq(b_s, inv(b_k)) \wedge eq(b_s, b_r) \wedge eq(d_s, d_k)\\
&\qquad\qquad\qquad\qquad \wedge eq(d_s, d_r) \wedge eq(i'_s, 1) \wedge eq(i_r, 1) \wedge eq(i_k, 1))\\
&\qquad \wedge (eq(b_s, b_k) \Rightarrow eq(d_s, d_k))\\
&\qquad \wedge (eq(i_r, 2) \vee eq(i_r, 3) \Rightarrow eq(d_s, d_r) \wedge eq(b_s, b_r) \wedge eq(b_s, b_k))\\
&\qquad \wedge (eq(b_s, inv(b_r)) \Rightarrow eq(d_s, d_r) \wedge eq(b_s, b_k))\\
&\qquad \wedge (eq(b_s, b_l) \Rightarrow eq(b_s, inv(b_r)))\\
&\qquad \wedge (eq(i'_s, 2) \Rightarrow eq(b_s, b_l)).
\end{aligned}$$

**Lemma 1.** *$\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{I}_3$, $\mathcal{I}_4$, $\mathcal{I}_5$, $\mathcal{I}_6$ and $\mathcal{I}_4 \wedge \mathcal{I}_7$ are invariants of Sys.*

The focus condition for *Sys* is obtained by taking the disjunction of the summands in the LPE in Definition 8 that deal with an external action; these summands are (1) and (7).

**Definition 11.** *The focus condition for Sys is*

$$FC(\xi) = eq(i_s, 1) \vee eq(i_r, 2).$$

We proceed to prove that each state satisfying the invariants above can reach a focus point by a sequence of internal transitions, carrying labels from $I = \{c_3, c_4, c_5, c_6, c_7, c_8, j\}$.

**Lemma 2.** *For each $\xi{:}\Xi$ with $\mathcal{I}_n(\xi)$ for $n = 1$-6, there is a $\hat{\xi}{:}\Xi$ such that FC($\hat{\xi}$) and $\xi \xrightarrow{c_1} \cdots \xrightarrow{c_n} \hat{\xi}$ with $c_1, \ldots, c_n \in I$ in Sys.*

*Proof.* Let $\neg FC(\xi)$; in view of $\mathcal{I}_1$ and $\mathcal{I}_4$ this implies $eq(i_s, 2) \wedge (eq(i_r, 1) \vee eq(i_r, 3))$. In case $eq(i_r, 3)$, we can perform $c_5(ac)$ at summand (8) to arrive a state with $eq(i_s, 2) \wedge eq(i_r, 1)$. By $\mathcal{I}_3$ and summands (2), (3) and (6), we can perform internal actions to reach a state where $eq(i_s, 2) \wedge eq(i_r, 1) \wedge eq(i_k, 3)$. We distinguish two cases.

1. $eq(b_r, b_k)$.
   We can perform $c_4(\langle d_k, b_r \rangle)$ at summand (4) to reach a focus point.
2. $eq(b_r, inv(b_k))$.
   If $i'_s = 2$, then we can perform $c(ac)$ at summand (14) to reach a focus point, so by $\mathcal{I}_2$ we can assume that $i'_s = 1$. If $eq(i_l, 3) \wedge eq(b_l, b_s)$, then by performing $c_7(b_l)$ at summand (11) followed by $c_8(ac)$ at summand (14) we can reach a focus point. Otherwise, by $\mathcal{I}_5$ and summands (10), (12) and (13) we can reach a state where $eq(i_s, 2) \wedge eq(i'_s, 1) \wedge eq(i_r, 1) \wedge eq(i_k, 3) \wedge eq(i_l, 1)$. We can perform $c_6(inv(b_r))$ at summand (9) followed by $j$ at summand (10) to reach a state where $eq(i_s, 2) \wedge eq(i'_s, 1) \wedge eq(i_r, 1) \wedge eq(i_k, 3) \wedge eq(i_l, 3) \wedge eq(b_l, inv(b_r))$. By $eq(b_r, inv(b_k))$, $\mathcal{I}_6$ and $eq(b_l, inv(b_r))$ we have $eq(b_l, b_s)$. Hence, we can perform $c_7(b_l)$ at summand (11) followed by $c_8(ac)$ at summand (14) to reach a focus point.

The state mapping $\phi : \Xi \to D \times Bool$ is defined by

$$\phi(\xi) = \langle d_s, eq(i_s, 1) \vee eq(i_r, 3) \vee \neg eq(b_s, b_r) \rangle.$$

Note that $\phi$ is independent of $i'_s, d_r, d_k, b_k, i_k, b_l, i_l$; we write $\phi(d_s, b_s, i_s, b_r, i_r)$.

**Theorem 2.** *For all $d{:}D$ and $b_0, b_1{:}Bit$,*

$$\tau_I(Sys(d, b_0, 1, 1, d, b_0, 1, d, b_1, 1, b_1, 1)) \leftrightarrows_b B(d, \mathsf{T}).$$

*Proof.* It is easy to check that $\wedge_{n=1}^{7} \mathcal{I}_n(d, b_0, 1, 1, d, b_0, 1, d, b_1, 1, b_1, 1)$.

We obtain the following matching criteria. For class I, we only need to check the summands (4), (8) and (14), as the other nine summands that involve an initial action leave the values of the parameters in $\phi(d_s, b_s, i_s, b_r, i_r)$ unchanged.

1. $eq(i_r, 1) \wedge eq(b_r, b_k) \wedge eq(i_k, 3) \;\Rightarrow\; \phi(d_s, b_s, i_s, b_r, i_r) = \phi(d_s, b_s, i_s, b_r, 2/i_r)$
2. $eq(i_r, 3) \;\Rightarrow\; \phi(d_s, b_s, i_s, b_r, i_r) = \phi(d_s, b_s, i_s, inv(b_r)/b_r, 1/i_r)$
3. $eq(i_s, 2) \wedge eq(i'_s, 2) \;\Rightarrow\; \phi(d_s, b_s, i_s, b_r, i_r) = \phi(d_s, inv(b_s)/b_s, 1/i_s, b_r, i_r)$

The matching criteria for the other four classes are produced by summands (1) and (7). For class II we get:

1. $eq(i_s, 1) \;\Rightarrow\; eq(i_s, 1) \vee eq(i_r, 3) \vee \neg eq(b_s, b_r)$
2. $eq(i_r, 2) \;\Rightarrow\; \neg(eq(i_s, 1) \vee eq(i_r, 3) \vee \neg eq(b_s, b_r))$

For class III we get:

1. $(eq(i_s,1) \vee eq(i_r,2)) \wedge (eq(i_s,1) \vee eq(i_r,3) \vee \neg eq(b_s,b_r)) \;\Rightarrow\; eq(i_s,1)$
2. $(eq(i_s,1) \vee eq(i_r,2)) \wedge \neg(eq(i_s,1) \vee eq(i_r,3) \vee \neg eq(b_s,b_r)) \;\Rightarrow\; eq(i_r,2)$

For class IV we get:

1. $\forall d{:}D\,(eq(i_s,1) \;\Rightarrow\; d = d)$
2. $eq(i_r,2) \;\Rightarrow\; d_r = d_s$

Finally, for class V we get:

1. $\forall d{:}D\,(eq(i_s,1) \;\Rightarrow\; \phi(d/d_s, b_s, 2/i_s, b_r, i_r) = \langle d, \mathsf{F}\rangle)$
2. $eq(i_r,2) \;\Rightarrow\; \phi(d_s,b_s,i_s,b_r,3/i_r) = \langle d_s, \mathsf{T}\rangle$

We proceed to prove the matching criteria.

I.1 Let $eq(i_r,1)$. Then

$$\begin{aligned}\phi(d_s,b_s,i_s,b_r,i_r) &= \langle d_s, eq(i_s,1) \vee eq(1,3) \vee \neg eq(b_s,b_r)\rangle \\ &= \langle d_s, eq(i_s,1) \vee eq(2,3) \vee \neg eq(b_s,b_r)\rangle \\ &= \phi(d_s,b_s,i_s,b_r,2/i_r).\end{aligned}$$

I.2 Let $eq(i_r,3)$. Then by $\mathcal{I}_7$, $eq(b_s,b_r)$. Hence,

$$\begin{aligned}\phi(d_s,b_s,i_s,b_r,i_r) &= \langle d_s, eq(i_s,1) \vee eq(3,3) \vee \neg eq(b_s,b_r)\rangle \\ &= \langle d_s, \mathsf{T}\rangle \\ &= \langle d_s, eq(i_s,1) \vee eq(i_r,3) \vee \neg eq(b_s, inv(b_r))\rangle \\ &= \phi(d_s,b_s,i_s,inv(b_r)/b_r,1/i_r).\end{aligned}$$

I.3 Let $eq(i'_s,2)$. By $\mathcal{I}_7$, $eq(b_s,b_l)$, which together with $\mathcal{I}_7$ yields $eq(b_s,inv(b_r))$. Hence,

$$\begin{aligned}\phi(d_s,b_s,i_s,b_r,i_r) &= \langle d_s, eq(i_s,1) \vee eq(i_r,3) \vee \neg eq(b_s,b_r)\rangle \\ &= \langle d_s, \mathsf{T}\rangle \\ &= \langle d_s, eq(1,1) \vee eq(i_r,3) \vee \neg eq(inv(b_s),b_r)\rangle \\ &= \phi(d_s,inv(b_s)/b_s,1/i_s,b_r,i_r).\end{aligned}$$

II.1 Trivial.

II.2 Let $eq(i_r,2)$. Then clearly $\neg eq(i_r,3)$, and by $\mathcal{I}_7$, $eq(b_s,b_r)$. Furthermore, according to $\mathcal{I}_7$, $eq(i_s,1) \Rightarrow eq(i_r,1)$, so $eq(i_r,2)$ also implies $\neg eq(i_s,1)$.

III.1 If $\neg eq(i_r,2)$, then $eq(i_s,1) \vee eq(i_r,2)$ implies $eq(i_s,1)$. If $eq(i_r,2)$, then by $\mathcal{I}_7$, $eq(b_s,b_r)$, so that $eq(i_s,1) \vee eq(i_r,3) \vee \neg eq(b_s,b_r)$ implies $eq(i_s,1)$.

III.2 If $\neg eq(i_s,1)$, then $eq(i_s,1) \vee eq(i_r,2)$ implies $eq(i_r,2)$. If $eq(i_s,1)$, then the formula $\neg(eq(i_s,1) \vee eq(i_r,3) \vee \neg eq(b_s,b_r))$ is false, so that it implies $eq(i_r,2)$.

IV.1 Trivial.

IV.2 Let $eq(i_r,2)$. Then by $\mathcal{I}_7$, $eq(d_r,d_s)$.

V.1 Let $eq(i_s,1)$. Then by $\mathcal{I}_7$, $eq(i_r,1)$ and $eq(b_s,b_r)$. So for any $d{:}D$,

$$\begin{aligned}\phi(d/d_s,b_s,2/i_s,b_r,i_r) &= \langle d, eq(2,1) \vee eq(1,3) \vee \neg eq(b_s,b_r)\rangle \\ &= \langle d, \mathsf{F}\rangle.\end{aligned}$$

V.2

$$\begin{aligned}\phi(d_s,b_s,i_s,b_r,3/i_r) &= \langle d_s, eq(i_s,1) \vee eq(3,3) \vee \neg eq(b_s,b_r)\rangle \\ &= \langle d_s, \mathsf{T}\rangle.\end{aligned}$$

Note that $\phi(d,b_0,1,b_0,1) = \langle d, \mathsf{T}\rangle$. So by Theorem 1 and Lemma 2,

$$\tau_I(Sys(d,b_0,1,1,d,b_0,1,d,b_1,1,b_1,1)) \underline{\leftrightarrow}_b B(d,\mathsf{T}).$$

**Acknowledgments** Jan Friso Groote is thanked for valuable discussions.

## References

1. T. Arts and I.A. van Langevelde. Correct performance of transaction capabilities. In *Proc. 2nd Conference on Application of Concurrency to System Design*, pp. 35–42. IEEE Computer Society, June 2001.
2. J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51:129–176, 1987.
3. J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.
4. T. Basten. Branching bisimilarity is an equivalence indeed! *Information Processing Letters*, 58:141–147, 1996.
5. J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.
6. M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In *Proc. 5th Conference on Concurrency Theory*, LNCS 836, pp. 401–416. Springer, 1994.
7. S.C.C. Blom, W.J. Fokkink, J.F. Groote, I.A. van Langevelde, B. Lisser, and J.C. van de Pol. $\mu$CRL: A toolset for analysing algebraic specifications. In *Proc. 13th Conference on Computer Aided Verification*, LNCS 2102, pp. 250–254. Springer, 2001.
8. K.M. Chandy and J. Misra. *Parallel Program Design. A Foundation*. Addison Wesley, 1988.
9. A. Cimatti, F. Giunchiglia, P. Pecchiari, B. Pietra, J. Profeta, D. Romano, P. Traverso, and B. Yu. A provably correct embedded verifier for the certification of safety critical software. In *Proc. 9th Conference on Computer Aided Verification*, LNCS 1254, pp. 202–213. Springer, 1997.
10. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2000.
11. B. Courcelle. Recursive applicative program schemes. In *Handbook of Theoretical Computer Science, Volume B, Formal Methods and Semantics*, pp. 459–492. Elsevier, 1990.
12. J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP – a protocol validation and verification toolbox. In *Proc. 8th Conference on Computer-Aided Verification*, LNCS 1102, pp. 437–440. Springer, 1997.
13. W.J. Fokkink, J.F. Groote, and J. Pang. Verification of a sliding window protocol in $\mu$CRL. In preparation.
14. W.J. Fokkink and J.C. van de Pol. Simulation as a correct transformation of rewrite systems. In *Proceedings of 22nd Symposium on Mathematical Foundations of Computer Science*, LNCS 1295, pp. 249–258. Springer, 1997.
15. L.-Å. Fredlund, J.F. Groote, and H.P. Korver. Formal verification of a leader election protocol in process algebra. *Theoretical Computer Science*, 177:459–486, 1997.
16. R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43:555–600, 1996.
17. W. Goerigk and F. Simon. Towards rigorous compiler implementation verification. In *Collaboration between Human and Artificial Societies, Coordination and Agent-Based Distributed Computing*, LNCS 1624, pp. 62–73. Springer, 1999.
18. J. F. Groote, J. Pang, and A.G. Wouters. Analysis of a distributed system for lifting trucks. *Journal of Logic and Algebraic Programming*, 2003. To appear.

19. J. F. Groote, A. Ponse, and Y.S. Usenko. Linearization in parallel pCRL. *Journal of Logic and Algebraic Programming*, 48:39–72, 2001.
20. J.F. Groote, F. Monin, and J.C. van de Pol. Checking verifications of protocols and distributed systems by computer. In *Proc. 9th Conference on Concurrency Theory*, LNCS 1466, pp. 629–655. Springer, 1998.
21. J.F. Groote and A. Ponse. The syntax and semantics of $\mu$CRL. In *Proc. 1st Workshop on the Algebra of Communicating Processes*, Workshops in Computing Series, pp. 26–62. Springer, 1995.
22. J.F. Groote and J. Springintveld. Focus points and convergent process operators. A proof strategy for protocol verification. *Journal of Logic and Algebraic Programming*, 49:31–60, 2001.
23. J.F. Groote and F.W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In *Proc. 17th Colloquium on Automata, Languages and Programming*, LNCS 443, pp. 626–638. Springer, 1990.
24. J.F. Groote and J.J. van Wamel. The parallel composition of uniform processes with data. *Theoretical Computer Science*, 266:631–652, 2001.
25. B. Jonsson. *Compositional Verification of Distributed Systems.* PhD thesis, Department of Computer Science, Uppsala University, 1987.
26. C.P.J. Koymans and J.C. Mulder. A modular approach to protocol verification using process algebra. In *Applications of Process Algebra*, Cambridge Tracts in Theoretical Computer Science 17, pp. 261–306. Cambridge University Press, 1990.
27. J. Loeckx, H.-D. Ehrich, and M. Wolf. *Specification of Abstract Data Types.* Wiley/Teubner, 1996.
28. N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symposium on Principles of Distributed Computing*, pp. 137–151. ACM, 1987.
29. N.A. Lynch and F.W. Vaandrager. Forward and backward simulations. Part I: Untimed systems. *Information and Computation*, 121:214–233, 1995.
30. G. Necula. Translation validation for an optimizing compiler. In *Proc. 2000 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. SIGPLAN Notices 35:83–94. ACM, 2000.
31. J. Pang. Analysis of a security protocol in $\mu$CRL. In *Proc. 4th International Conference on Formal Engineering Methods*, LNCS 2495, pp. 396–400. Springer, 2002.
32. A. Pnueli, M. Siegel, and E. Singerman. Translation validation. In *Proc. 4th Conference on Tools and Algorithms for Construction and Analysis of Systems*, LNCS 1384, pp. 151–166. Springer, 1998.
33. J.C. van de Pol and M. Valero Espada. Formal specification of Javaspaces$^{TM}$ architecture using $\mu$CRL. In *Proc. 5th Conference on Coordination Models and Languages*, LNCS 2315, pp. 274–290. Springer, 2002.
34. C. Shankland and M.B. van der Zwaag. The tree identify protocol of IEEE 1394 in $\mu$CRL. *Formal Aspects of Computing*, 10:509–531, 1998.
35. Y.S. Usenko. Linearization of $\mu$CRL specifications (extended abstract). In *Proc. 3rd International Workshop on Verification and Computational Logic (VCL2002)*, Technical Report DSSE-TR-2002-5. Department of Electronics and Computer Science, University of Southampton, 2002.
36. M.B. van der Zwaag. The cones and foci proof technique for timed transition systems. *Information Processing Letters*, 80(1):33–40, 2001.