Cones and Foci: A Mechanical Framework for Protocol Verification

Wan Fokkink[1,2], Jun Pang[1] and Jaco van de Pol[1]
[1]CWI, Software Engineering Department
[2]Vrije Universiteit, Department of Theoretical Computer Science

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, our method is more generally applicable, and does not require a preprocessing step to eliminate τ-loops. We prove soundness of our approach and present a set of rules to prove the reachability of focus points. Our method has been formalized and proved correct using PVS. Thus we have established a framework for mechanical protocol verification. We apply this framework to the Concurrent Alternating Bit Protocol.

Keywords: protocol verification, branching bisimulation, process algebra, PVS

1 Introduction

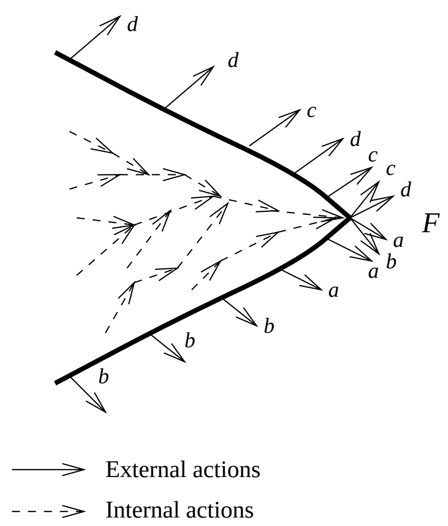
Protocol verification with the help of a theorem prover is often rather ad hoc, in the sense that one has to develop the entire proof structure from scratch. Inventing such a structure takes a lot of effort, and makes that in general such a proof cannot be readily adapted to other protocols. Groote and Springintveld [25] proposed a general proof framework for protocol verification, which they named the *cones and foci* method. In this paper we introduce some significant improvements for this framework. Furthermore, we have cast the framework in the interactive theorem prover PVS [34].

We present our work in the setting of μCRL [23] (see also [24]), which combines the process algebra ACP [3] with equational abstract data types [30]. 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. A special action τ [5] represents hidden internal activity. A labeled transition system is associated to each μCRL specification. Two μCRL specifications are considered equivalent if the initial states of their labeled transition systems are branching bisimilar [18]. 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.

For finite labeled transition systems, checking whether two states are branching bisimilar can be performed efficiently [26]. The μCRL tool set [8] supports the generation of labeled transition systems, together with reduction modulo branching bisimulation equivalence, and allows model checking of temporal logic formulas [11] via a back-end to the CADP tool set [13]. This approach to verify system correctness has three important drawbacks. First, the labeled transition systems of the μCRL specifications involved must be 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.

Linear process equations [6] constitute a restricted class of μCRL specifications in a so-called linear format. Algorithms have been developed to transform μCRL specifications into this linear format [21, 27, 38]. In a linear process equation, the states of the associated labeled transition system are data objects.

The cones and foci method from [25] rephrases the question whether two linear process equations 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., [17, 22, 37]), often with the support of a theorem prover or proof checker.



The main idea of the cones and foci method is that quite often in the implementation of a system, τ-transitions progress inertly towards a state in which no τ 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 τ-transitions. In the absence of infinite sequences of τ-transitions, each state belongs to some 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 τ-transitions in the cone would not be inert.

The starting point of the cones and foci method are two linear process equations, expressing the implementation and the desired external behavior of a system. A *state mapping* ϕ relates each state of the implementation to a state of the desired external behavior. Groote and Springintveld [25] 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, gives rise to infinite sequences of τ-actions, then Groote and Springintveld [25] distinguish between *progressing* and *non-progressing* τ's, where the latter are treated in the same way as external actions. They require that there is no infinite sequence of progressing τ's, and define focus points as the states that cannot perform progressing τ's. A *pre-abstraction function* divides occurrences of τ in the implementation into progressing and non-progressing ones; 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) τ's.

In this paper, we propose an adaptation of the cones and foci method, in which the cumbersome treatment of infinite sequences of τ-transitions is no longer necessary. This improvement of the cones and foci method was conceived during the verification of a sliding window protocol [14], 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 τ-actions can be performed), as long as each state is in the cone of some focus point. We do allow infinite sequences of τ-transitions. No distinction between progressing and non-progressing τ's is needed, and τ-loops are eliminated without having to resort explicitly to a fair abstraction rule. We prove that our method is sound modulo branching bisimulation equivalence.

Compared to the original cones and foci method [25], our method is more generally applicable. As expected, some extra price may have to be paid for this generalization. Groote and Springintveld must prove strong termination of progressing τ-transitions. They use a standard approach to prove strong termination: provide a well-founded ordering on states such that for each progressing τ-transition $s \xrightarrow{\tau} s'$ one has $s > s'$. Here we must prove that each state can reach a focus point by a series of τ-transitions. This means that in principle we have a weaker proof obligation, but for a larger class of τ-transitions. We develop a set of rules to prove the reachability of focus points. These rules have been formalized and proved in PVS.

We formalize the cones and foci method in PVS. The intent is to provide a common framework for mechanical verification of protocols using our approach. PVS theories are developed to represent basic notions like labeled transition systems, branching bisimulation, linear process equations, and then the cones and foci method itself. The proof of soundness for the method has been mechanically checked by PVS within this framework. Once we had the linear process equations, the state mapping and the focus condition encoded in PVS, the PVS theorem prover and its type-checking condition system were then used to generate and verify all correctness conditions to ensure that the implementation and the external behavior of a system are branching bisimilar.

We apply our mechanical proof framework to the Concurrent Alternating Bit Protocol [29], which served as the main example in [25]. Our aims are to compare our method with the one from [25], and to illustrate our mechanical proof framework and our approach to the reachability analysis of focus points. While the old cones and foci method required a typical cumbersome treatment of τ-loops, here we can take these τ-loops in our stride. Thanks to the mechanical proof framework we detected a bug in one of the invariants of our original manual proof. The reachability analysis of focus points is quite crisp.

This paper is organized as follows. In Section 2, we present the preliminaries of our cones and foci method. In Section 3, we present the main theorem and prove that our method is sound modulo branching bisimulation equivalence. A proof theory for reachability of focus points is also presented. In Section 4, the cones and foci method is formalized in PVS, and a mechanical proof framework is set up. In Section 5, we illustrate the method by verifying the Concurrent Alternating Bit Protocol. Part of the verification within the mechanical proof framework in PVS is presented in Section 5.4.

An earlier version of this paper (lacking the formalization in PVS and the methodology for reachability analysis) appeared as [15].

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., [10, 19, 33]). The methodology surrounding cones and foci incorporates well-known and useful concepts such as the precondition/effect notation [28, 31], invariants and simulations. Linear process equations resemble the UNITY format [9] and recursive applicative program schemes [12]; state mappings are comparable to refinement mappings [32, 35] and simulation [16]. Van der Zwaag [39] gave an adaptation of the cones and foci method from [25] to a timed setting, modulo timed branching bisimulation equivalence.

2 Preliminaries

2.1 μCRL

μCRL [23] 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 μCRL 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 μCRL 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 T and 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 actions, recursion variables and process algebraic operators. Actions and recursion variables carry zero or more data parameters. There are two predefined processes in μCRL: δ represents deadlock, and τ a hidden action. These two processes never carry data parameters. $p{\cdot}q$ denotes sequential composition and $p + q$ non-deterministic choice, summation $\sum_{d:D} p(d)$ provides the possibly infinite choice over a data type D, and the conditional construct $p \lhd b \rhd q$ with b a data term of sort *Bool* behaves as p if b and as q if $\neg b$. Parallel composition $p \parallel q$ interleaves the actions of p and q; moreover, actions from p and q may also 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 action names from the set H into δ, 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 τ. The syntax and semantics of μCRL are given in [23].

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 μCRL specification belongs an LTS, defined by the structural operational semantics for μCRL in [23].

Definition 2.1 (Labeled transition system) A *labeled transition system* is a tuple $(S, \mathit{Lab}, \rightarrow, s_0)$, where S is a set of states, Lab a set of transition labels, $\rightarrow \subseteq S \times \mathit{Lab} \times S$ a transition relation, and s_0 the initial state. A transition (s, ℓ, s') is denoted by $s \xrightarrow{\ell} s'$.

Here, S consists of μCRL specifications, and Lab consists of actions from $\mathit{Act} \cup \{\tau\}$ parametrized by data. We define *branching bisimilarity* [18] between states in LTSs. Branching bisimulation is an equivalence relation [4].

Definition 2.2 (Branching bisimulation) Assume an LTS. A *branching bisimulation relation* B is a symmetric binary relation on states such that if sBt and $s \xrightarrow{\ell} s'$, then

- either $\ell = \tau$ and $s' \mathcal{B}\, t$;

- or there is a sequence of (zero or more) τ-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 $s \underline{\leftrightarrow}_b t$, if there is a branching bisimulation relation $\mathcal{B}$ such that $s\,\mathcal{B}\,t$.

The μCRL tool set [8] supports the generation of labeled transition systems of μCRL specifications, together with reduction modulo branching bisimulation equivalence and model checking of temporal logic formulas [36, 7, 20]. This approach has been used to analyze a wide range of protocols and distributed systems.

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 μCRL 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 μCRL specification that does not include successful termination can be transformed into an LPE [38].[1]

Definition 2.3 (Linear process equation) A *linear process equation* is a μCRL specification of the form

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

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

The LPE in Definition 2.3 has exactly one LTS as its solution (modulo strong bisimulation).[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_a gives LPEs a more general form, as not only the data parameter $d{:}D$ but also the data parameter $e{:}E_a$ can influence the parameter of action a, the condition h_a and the resulting state g_a.

[1] To cover μCRL specifications with successful termination, LPEs should include a summand $\sum_{a \in Act \cup \{\tau\}} \sum_{e:E_a} a(f_a(d,e)) \lhd h_a(d,e) \rhd \delta$. The cones and foci method extends to this setting without any complication. However, this extension would complicate the matching criteria in Definition 3.3. For the sake of presentation, successful termination is not taken into account in this paper.

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

Definition 2.4 (Invariant) A mapping $\mathcal{I} : D \to Bool$ is an invariant for an LPE, written as in Definition 2.3, 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 approximates the set of reachable states of an LPE. That is, if $\mathcal{I}(d)$, and if one can evolve 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 from [25]. Suppose that we have an LPE $X(d{:}D)$ specifying the implementation of a system, and an LPE $Y(d'{:}D')$ (without occurrences of τ) specifying the desired input/output behavior of this system. 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 [25], we allow to freely designate focus points, as long as each state $d{:}D$ of X with $\mathcal{I}(d)$ can reach a focus point by a sequence of τ-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) all τ-transitions at d are inert, (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.

In Section 3.1, we present the general theorem underlying our method. Then we introduce proof rules for the reachability of focus points in Section 3.2.

3.1 The general theorem

Let the LPE X be of the form

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

Furthermore, let the LPE Y be of the form

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

Note that Y is not allowed to have τ-steps. 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.

Definition 3.1 (Focus point) A *focus condition* is a mapping $FC : D \to Bool$. If $FC(d)$, then d is called a *focus point*.

Definition 3.2 (State mapping) A *state mapping* is of the form $\phi : D \to D'$.

Definition 3.3 (Matching criteria) A state mapping $\phi : D \to D'$ satisfies the *matching criteria* for $d{:}D$ if for all $a \in Act$:

I $\quad \forall e{:}E_a\, (h_\tau(d,e) \Rightarrow \phi(d) = \phi(g_\tau(d,e)))$;

II $\quad \forall e{:}E_a\, (h_a(d,e) \Rightarrow h'_a(\phi(d),e))$;

III $\quad FC(d) \Rightarrow \forall e{:}E_a\, (h'_a(\phi(d),e) \Rightarrow h_a(d,e))$;

IV $\quad \forall e{:}E_a\, (h_a(d,e) \Rightarrow f_a(d,e) = f'_a(\phi(d),e))$;

V $\quad \forall e{:}E_a\, (h_a(d,e) \Rightarrow \phi(g_a(d,e)) = g'_a(\phi(d),e))$.

Matching criterion I requires that the τ-transitions at d are inert, meaning that d and $g_\tau(d,e)$ 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 3.4 Assume LPEs $X(d{:}D)$ and $Y(d'{:}D')$ written as before Definition 3.1. 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 $d \xrightarrow{\tau} \cdots \xrightarrow{\tau} \hat{d}$ in the LTS for X.

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

$$X(d) \underline{\leftrightarrow}_b Y(\phi(d)).$$

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 $h_\tau(s,e)$ and $s' = g_\tau(s,e)$ for some $e{:}E$. By matching criterion I, $\phi(g_\tau(s,e)) = t$. Moreover, $\mathcal{I}(s)$ and $h_\tau(s,e)$ together imply $\mathcal{I}(g_\tau(s,e))$. Hence, $g_\tau(s,e)Bt$.
2. If $\ell \neq \tau$, then $h_a(s,e)$, $s' = g_a(s,e)$ and $\ell = a(f_a(s,e))$ for some $a \in Act$ and $e{:}E$. By matching criteria II and IV, $h'_a(t,e)$ and $f_a(s,e) = f'_a(t,e)$. Hence, $t \xrightarrow{a(f_a(s,e))} g'_a(t,e)$. 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 $s \xrightarrow{\ell} s'$, for some $a \in Act$ 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 $\mathcal{I}(t)$. By assumption 2 of the Theorem, there is a $\hat{t}{:}D$ with $FC(\hat{t})$ such that $t \xrightarrow{\tau} ... \xrightarrow{\tau} \hat{t}$ in the LTS for X. Invariant $\mathcal{I}$, so also the matching criteria, hold for all states on this τ-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 $t \xrightarrow{\tau} ... \xrightarrow{\tau} \hat{t} \xrightarrow{a(f'_a(s,e))} g_a(\hat{t},e)$. 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 relation. ⊠

We note that Groote and Springintveld [25] proved for their version of the cones and foci method that it can be derived from the axioms of μCRL, 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 μCRL.

3.2 Proof rules for reachability

The cones and foci method requires as input a state mapping and a focus condition. It generates two kinds of proof obligations: matching criteria, and a reachability criterion. The latter states that from all reachable states, a state satisfying the focus condition must be reachable. Note that it suffices to prove that from any state satisfying a given set of invariants, a state satisfying the focus conditions is reachable. In this section we develop proof rules, in order to establish this condition. First we introduce some notation.

Definition 3.5 (τ-Reachability) Given an LTS $(S, Lab, \rightarrow, s_0)$ and $\phi, \psi \subseteq S$. ψ is τ-reachable from ϕ, written as $\phi \twoheadrightarrow \psi$, if and only if for all $x \in \phi$ their exists a $y \in \psi$ such that $x \xrightarrow{\tau} \cdots \xrightarrow{\tau} y$.

The above mentioned reachability criterion can now be expressed as $Inv \twoheadrightarrow FC$, where *Inv* denotes a set invariants, and *FC* denotes the focus condition. Here and in the sequel, we use predicates with variables from the state vector to denote sets of states.

Definition 3.6 (Reachability in one τ-step) Let an LPE $X(d{:}D)$ written as before (see Definition 3.1). The set of states $Pre_X(\psi)$, that can reach the set of states ψ in one τ-step, is defined as:

$$Pre_X(\psi)(d) = \exists e{:}E.h_\tau(d,e) \wedge \psi(g_\tau(d,e))$$

Lemma 3.7 (Proof rules for reachability) We give a list of rules for proving $\twoheadrightarrow$ with respect to an LPE X as follows:

- (precondition) $Pre_L(\phi) \twoheadrightarrow \phi$

- (implication) If $\phi \Rightarrow \psi$ then $\phi \twoheadrightarrow \psi$.
- (transitivity) If $\phi \twoheadrightarrow \psi$ and $\psi \twoheadrightarrow \chi$ then $\phi \twoheadrightarrow \chi$.
- (disjunction) If $\phi \twoheadrightarrow \chi$ and $\psi \twoheadrightarrow \chi$, then $\{\phi \vee \psi\} \twoheadrightarrow \chi$.
- (invariant) If $\phi \twoheadrightarrow \psi$ and $\mathcal{I}$ is an invariant, then $\{\phi \wedge \mathcal{I}\} \twoheadrightarrow \{\psi \wedge \mathcal{I}\}$.
- (induction) If for all $n > 0$, $\{\phi \wedge (t = n)\} \twoheadrightarrow \{\phi \wedge (t < n)\}$, then $\phi \twoheadrightarrow \{\phi \wedge (t = 0)\}$, where t is any term containing state variables from D.

Proof. These rules can be easily proved. In the precondition rule we obtain a one step reduction from the semantics of LPEs. The implication rule is obtained by an empty reduction sequence; for transitivity we can concatenate the reduction sequences. The disjunction rule can be proved by case distinction. For the invariant rule, assume that $\phi(d)$ and $\mathcal{I}(d)$ hold. By the assumption $\phi \twoheadrightarrow \psi$, we obtain a sequence $d \xrightarrow{\tau} \cdots \xrightarrow{\tau} d'$, such that $\psi(d')$. Because $\mathcal{I}$ is an invariant, we have $\mathcal{I}(d')$ (by induction on the length of that reduction). So indeed $\{\psi \wedge \mathcal{I}\}(d')$. Finally, for the induction rule we first prove with well-founded induction over n and using the transitivity rule that $\forall n.\{\phi \wedge (t = n)\} \twoheadrightarrow \{\phi \wedge (t = 0)\}$. Then observe that $\phi \Rightarrow \{\phi \wedge (t = t)\}$, and use the implication and transitivity rule to conclude that $\phi \twoheadrightarrow \{\phi \wedge (t = 0)\}$. ⊠

The proof rules for reachability were proved correct in PVS, and they were used in the verification of the reachability criterion for the CABP in PVS, which we will present in Section 5.4.

4 A mechanical proof framework

In this section, our method is formalized in the language of the interactive theorem prover PVS [34]. This formalism enables computer aided protocol verification using the cones and foci method. PVS is chosen for the following main reasons. First, the specification language of PVS is based on simply typed higher-order logics. PVS provides a rich set of types and the ability to define subtypes and dependent types. Second, PVS constitutes a powerful, extensible system for verifying obligations. It has a tool set consisting of a type checker, an interactive theorem prover, and a model checker. Third, PVS includes high level proof strategies and decision procedures that take care of many of the low level details associated with computer aided theorem proving. In addition, PVS has useful proof management facilities, such as a graphical display of the proof tree, and proof stepping and editing.

The type system of PVS contains basic types such as *boolean, natural, integer, real,* etc. and type constructors such as *set, tuple, record,* and *function.* Tuple, record, and type constructors are extensively used in the following sections to formalize the cones and foci method. Tuple types have the form `[T1,...,Tn]`, where the `Ti` are type expressions. A record is a finite list of fields of the form `R:TYPE=[# E1:T1, ...,En:Tn #]`, where the `Ei` are *record* *accessor* functions. Associated with every tuple type or record is a set of projection functions: `‘1,‘2,...,` (or `proj_1,proj_2,...`). A function constructor has the form `F:TYPE=[T1,...,Tn->R]`, where `F` is a function with domain `T1`×`T2`×`...`×`Tn` and range `R`.

A PVS specification can be structured through a hierarchy of *theories*. Each theory consists of a *signature* for the type names, constants introduced in the theory, axioms, definitions, and theorems associated with the signature. A PVS theory can be parametric in certain specified types and values, which are placed between `[ ]` after the theory name. A theory can build on other theories. To import a theory, PVS uses the notation `IMPORTING` followed by the theory name. For example, we can give part of the theory of abstract reduction systems [1] in PVS as follows:

```
ARS[A:TYPE]: THEORY BEGIN
      x,y,z:VAR A   n:VAR nat   R:VAR pred[[A,A]]
      iterate(R,n)(x,y):RECURSIVE bool=
         IF n=0 THEN x=y
         ELSE EXISTS z:  iterate(R,n-1)(x,z) AND R(z,y)
         ENDIF MEASURE n
      star(R)(x,y):bool= EXISTS n:  iterate(R,n)(x,y)
      ...
END ARS
```

Theory `ARS` contains the basic notations, like transitive closure of a relation, and theorems for abstract reduction systems. The rest of this section gives the main part of the PVS formalism of our approach. We will explain PVS notation throughout this section, when necessary.

4.1 LTSs and branching bisimulation

In this section, we formalize the preliminaries from Section 2 in PVS. An LTS (see Definition 2.1) is parameterized by a set of states `D`, a set of actions `Act` and a special action `tau`. The type `LTS` is then defined as a record containing an initial state, and a ternary step relation. The relation `step_01` extends `step` with the reflexive closure of the `tau`-steps. We also abbreviate the reflexive transitive closure of `tau`-steps `tau_star`. Finally, the set `reachable` of states reachable from the initial state can be easily characterized using an inductive definition.

```
LTS[D,Act:TYPE,tau:Act]:  THEORY BEGIN
      IMPORTING ARS[D]
      LTS: TYPE = [# init:D, step:[D,Act,D->bool] #]
      x,y:VAR D   a:VAR Act   lts:VAR LTS
      step(lts,a)(x,y):bool= lts‘step(x,a,y)
      step_01(lts)(x,a,y):bool = lts‘step(x,a,y) OR (a=tau AND x=y)
      tau_star(lts)(x,y):bool = star(step(lts,tau))(x,y)
      reachable(lts)(x):  INDUCTIVE bool =
         x=lts‘init OR EXISTS y,a:  reachable(lts)(y) AND lts‘step(y,a,x)
END LTS
```

To define a branching bisimulation relation (see Definition 2.2) between two labeled transition systems in PVS, we first introduce a formalization of a branching simulation relation in PVS. A relation is a branching bisimulation if and only if both itself and its inverse are a branching simulation relation.

```
BRANCHING_SIMULATION [D,E,Act:TYPE,tau:Act]:  THEORY BEGIN
      IMPORTING LTS[D,Act,tau], LTS[E,Act,tau]
      x1,y1,z1:VAR D   x2,y2,z2:VAR E
      lts1:VAR LTS[D,Act,tau]   lts2:VAR LTS[E,Act,tau]
      a:VAR Act   R:VAR [D,E->bool]
      brsim(lts1,lts2)(R):bool=
         FORALL x1,a,z1,x2:  lts1`step(x1,a,z1) AND R(x1,x2) IMPLIES
         EXISTS y2,z2:  tau_star(lts2)(x2,y2) AND step_01(lts2)(y2,a,z2)
            AND R(x1,y2) AND R(z1,z2)
END BRANCHING_SIMULATION
BRANCHING_BISIMULATION [D,E,Act:TYPE,tau:Act]:  THEORY BEGIN
      IMPORTING BRANCHING_SIMULATION[D,E,Act,tau],
         BRANCHING_SIMULATION[E,D,Act,tau]
      x1:VAR D   x2:VAR E
      lts1:VAR LTS[D,Act,tau]   lts2:VAR LTS[E,Act,tau]
      a:VAR Act   R:VAR [D,E->bool]
      brbisim(lts1,lts2)(R):bool=
         brsim(lts1,lts2)(R) AND brsim(lts2,lts1)(converse(R))
      brbisimilar(lts1,lts2)(x1,x2):bool=
         EXISTS R : brbisim(lts1,lts2)(R) AND R(x1,x2)
      brbisimilar(lts1,lts2):bool= brbisimilar(lts1,lts2)(lts1`init,lts2`init)
END BRANCHING_BISIMULATION
```

In our actual PVS theory of branching bisimulation, we also defined a semi-branching bisimulation relation [18]. In [4], this notion was used to show that branching bisimilarity is an equivalence. Basten showed that the relation composition of two branching bisimulation relations is not necessarily again a branching bisimulation relation, while the relation composition of two semi-branching bisimulation relations is again a semi-branching bisimulation relation. It is easy to see that semi-branching bisimilarity is reflexive and symmetric. Hence, semi-branching bisimilarity is an equivalence relation. Basten also proved that semi-branching bisimilarity and branching bisimilarity coincide, that means two states in an LTS are related by a branching bisimulation relation if and only if they are related by a semi-branching bisimulation relation. Thus, he proved that branching bisimilarity is an equivalence relation. We have checked these facts in PVS.

4.2 Representing LPEs and invariants

We now show how an LPE (see Definition 2.3) can be represented in PVS. The formal definitions will slightly deviate from the mathematical presentation before. A first decision was to represent μCRL abstract data types directly by PVS types. This enables one to reuse the PVS library for definitions and theorems of "standard" data types, and to focus on the behavioral part.

A second distinction will be that we assumed so far that LPEs are *clustered.*

This means that each action name occurs in at most one summand, so that the set of summands can be indexed by the set of action names *Act*. This is no real limitation, because any LPE can be transformed into clustered form, basically by replacing + by $\sum$ over finite types. Clustered LPEs enable a notationally smoother presentation of the theory. However, when working with concrete LPEs this restriction is not convenient, so we avoid it in the PVS framework: an arbitrarily sized index set $\{0, \ldots, n-1\}$ will be used, represented by the PVS type `below(n)`. A third deviation is that we will assume from now on that every summand has the same set E of local variables (instead of E_a before). Again this is no limitation, because void summations can always be added (i.e.: $p = \sum_{e:E} p$, when e doesn't occur in p). This restriction is needed to avoid the use of polymorphism, which doesn't exist in PVS. A fourth deviation is that we do not distinguish action names from action data parameters. We simply work with one type *Act* of expressions for actions. Note that this is a real extension. Namely, in our PVS formalization, each LPE summand is a function from $D \times E$ (with D the set of states) to $Act \times Bool \times D$, so one summand may now generate steps with various action names, possibly visible as well as invisible.

So an LPE is parameterized by a set of actions (`Act`), a global parameter (`State`) and a local variable (`Local`), and by the size of its index set (`n`) and the special action τ (`tau`). Note that the guard, action and next-state of a summand depend on the global parameter `d:State` and on the local variable `e:Local`. This dependency is represented in the definition `SUMMAND` by a PVS function type. An LPE consists of an initial state and a list of summands indexed by `below(n)`. Finally, the function `lpe2lts` provides the LTS semantics of an LPE, `Step(L,a)` provides the corresponding binary relation on states, and the set of `Reachable` states is lifted from LTS to LPE level.

```
LPE[Act,State,Local:TYPE,n:nat,tau:Act]:  THEORY BEGIN
      IMPORTING LTS[State,Act,tau]
      SUMMAND:TYPE= [State,Local-> [#act:Act,guard:bool,next:State#] ]
      LPE:TYPE= [#init:State,sums:[below(n)->SUMMAND]#]
      L:VAR LPE   i:VAR below(n)   d,d1,d2:VAR State
      a:VAR Act   e:VAR Local   s:VAR SUMMAND
      step(s)(d1,a,d2):bool=
         EXISTS e:  s(d1,e)`guard AND a=s(d1,e)`act AND d2=s(d1,e)`next
      lpe2lts(L):LTS= (#init:= init(L),
         step:= LAMBDA d1,a,d2:  EXISTS i:  step(L`sums(i))(d1,a,d2) #)
      Step(L,a)(d1,d2):bool = step(lpe2lts(L),a)(d1,d2)
      Reachable(L)(d):bool = reachable(lpe2lts(L))(d)
END LPE
```

We define an invariant (see Definition 2.4) of an LPE in PVS by a theory `INVARIANT` as follows, where `p` is a predicate over states. `p` is an invariant of an LPE if and only if it holds initially and it is preserved by the execution of every summand. Note that we only require preservation for reachable states. This allows that previously proved invariants can be used in proving that `p` is invariant, which occurs frequently in practice. The abstract notion of reachability can itself be proved to be the strongest invariant (`reachable_inv1` and reachable_inv2).

```
INVARIANT[Act,State,Local:TYPE,n:nat,tau:Act]:  THEORY BEGIN
      IMPORTING LPE[Act,State,Local,n,tau]
      L:VAR LPE   p:VAR [State->bool]
      d:VAR State   a:VAR Act   e:VAR Local   i:VAR below(n)
      preserves(L,i)(p):bool=
         FORALL d,e:  Reachable(L)(d) AND p(d) AND L`sums(i)(d,e)`guard
         IMPLIES p(L`sums(i)(d,e)`next)
      invariant(L)(p):bool = p(L`init) AND FORALL i:  preserves(L,i)(p)
      reachable_inv1:  LEMMA invariant(L)(Reachable(L))
      reachable_inv2:  LEMMA invariant(L)(p) IMPLIES subset?(Reachable(L),p)
END INVARIANT
```

4.3 Formalizing the cones and foci method

In this section, we give the PVS development of the cones and foci method. Compared to the mathematical definitions in Section 3 we make two adaptations. First, we use the abstract reachability predicate instead of invariants; by the previous lemmas we can always switch back to invariants. Second, we have to reformulate the matching criteria in the setting of our slightly extended notion of LPEs, allowing arbitrary index sets, and more action names per summand.

We start with two LPEs, for the implementation and the desired external behavior of a system, `X:LPE[Act,D,L,m,tau]` and `Y:LPE[Act,E,L,n,tau]` respectively. Both LPE `X` and LPE `Y` have the same set of actions and the same set of local variables. However, the type of global parameters (D and E, respectively) and the number of summands (`m` and `n`, respectively) may be different. Note that here we do not exclude the presence of `tau` in the LPE `Y`. For the correctness proof this restriction is not needed, and by lifting this restriction we avoid the use of subtypes in PVS. However it does not really extend the method, because the matching criteria enforce that all `tau`-steps in `Y` are `tau`-loops.

The next ingredients are the state mapping function `h:[D->E]` and a focus condition `fc:pred[D]`. But, as summands are no longer indexed by action names, we also need a mapping of the summands `k:[below(m)->below(n)]`. The idea is that summand $i : \mathit{below}(m)$ of LPE X is mapped to summand $k(i) : \mathit{below}(n)$ of LPE Y. Having these ingredients, we can subsequently define the matching criteria (MC) and the reachability criterion (RC). The individual matching criteria (MC1–MC5) are displayed separately.

```
CONESFOCI_METHOD [D,E,L,Act:TYPE,tau:Act,m,n:nat]:  THEORY BEGIN
      IMPORTING BRANCHING_BISIMULATION [D,E,Act,tau],
         LPE[Act,D,L,m,tau], LPE [Act,E,L,n,tau]
      X:VAR LPE[Act,D,L,m,tau]   Y:VAR LPE[Act,E,L,n,tau]
      h:VAR [D->E]   fc:VAR pred[D]   k:VAR [below(m)->below(n)]
      d,d1:VAR D
      ...
      MC(X,Y,k,h,fc)(d):bool=
         MC1(X,h)(d) AND MC2(X,Y,k,h)(d) AND MC3(X,Y,k,h,fc)(d)
         AND MC4(X,Y,k,h)(d) AND MC5(X,Y,k,h)(d)
      RC(X,fc)(d):bool=
         EXISTS d1 :  fc(d1) AND tau_star(lpe2lts(X))(d,d1)
      CONESFOCI: THEOREM
         h(X`init)=Y`init AND (FORALL d:  Reachable(X)(d)
            IMPLIES MC(X,Y,k,h,fc)(d) AND RC(X,fc)(d))
         IMPLIES brbisimilar(lpe2lts(X),lpe2lts(Y))
END CONESFOCI_METHOD
```

```
x:VAR L   i:VAR [below(m)]   j:VAR [below(n)]
MC1(X,h)(d):bool= FORALL i :  FORALL x:
      X`sums(i)(d,x)`act=tau AND X`sums(i)(d,x)`guard
      IMPLIES h(d)=h(X`sums(i)(d,x)`next)
MC2(X,Y,k,h)(d):bool= FORALL i :  FORALL x:
      NOT X`sums(i)(d,x)`act=tau AND X`sums(i)(d,x)`guard
      IMPLIES Y`sums(k(i))(h(d),x)`guard
MC3(X,Y,k,h,fc)(d):bool= FORALL j :  FORALL x:
      fc(d) AND Y`sums(j)(h(d),x)`guard
      IMPLIES EXISTS i :
      k(i)=j AND X`sums(i)(d,x)`guard AND NOT X`sums(i)(d,x)`act=tau
MC4(X,Y,k,h)(d):bool= FORALL i :  FORALL x:
      NOT X`sums(i)(d,x)`act=tau AND X`sums(i)(d,x)`guard
      IMPLIES X`sums(i)(d,x)`act = Y`sums(k(i))(h(d),x)`act
MC5(X,Y,k,h)(d):bool= FORALL i :  FORALL x:
      NOT X`sums(i)(d,x)`act=tau AND X`sums(i)(d,x)`guard
      IMPLIES h(X`sums(i)(d,x)`next) = Y`sums(k(i))(h(d),x)`next
```

The theorem `CONESFOCI` was proved in PVS along the lines of Section 3.

4.4 The symbolic reachability criterion

The last part of the formalization of the framework in PVS is on the proof rules for the reachability criterion. We start on the level of abstract reduction systems (`ARS[S]`), which talks about binary relations, formalized in PVS as `pred[S,S]`. First, we have to lift conjunction (`AND`) and disjunction (`OR`) to predicates on `S` (overloading is allowed in PVS). We use `Reach` to denote $\twoheadrightarrow$. Next, several proof rules can be expressed and proved in PVS. Here we only show the rules for disjunction and induction; the latter depends on a measure function `f:[S->nat]` (this rule is not used in the verification of Concurrent Alternating Bit Protocol later, but it was essential in the verification of the Sliding Window Protocol [14]).

```
REACH_CONDITION [S:TYPE]: THEORY BEGIN
      IMPORTING ARS[S]
      X,Y,Z:VAR pred[S]   x,y:VAR S   R:VAR pred[[S,S]]
      AND(X,Y)(x):bool = X(x) AND Y(x) ;
      OR(X,Y)(x) :bool = X(x) OR Y(x) ;
      Reach(R)(X,Y):bool= FORALL x :  X(x)
         IMPLIES EXISTS y :  Y(y) AND star(R)(x,y)
      reach_disjunction:  LEMMA % Disjunction rule
         Reach(R)(X,Z) AND Reach(R)(Y,Z) IMPLIES Reach(R)(X OR Y,Z)
      f:VAR [S->nat]   n:VAR nat
      reach_induction:  LEMMA % Induction rule
         (FORALL n:n>0 IMPLIES
            Reach(R)( X AND LAMBDA x:  f(x)=n, X AND LAMBDA x:  f(x)<n))
         IMPLIES Reach(R)( X , X AND LAMBDA x:  f(x)=0 )
END REACH_CONDITION
```

Finally, the *precondition* and *invariant* rules depend on the LPE under scrutiny, so we define them in a separate theory:

```
PRECONDITION [Act,State,Local:TYPE,n:nat,tau:Act]:  THEORY BEGIN
      IMPORTING INVARIANT[Act,State,Local,n,tau], REACH_CONDITION[State]
      L:VAR LPE   X,Y:VAR pred[State]   i:VAR below(n)
      d:VAR State   e:VAR Local   I:VAR [State->bool]
      precondition(L,X)(d):bool=
         EXISTS i:  EXISTS e:  L`sums(i)(d,e)`act=tau
         AND L`sums(i)(d,e)`guard AND X(L`sums(i)(d,e)`next)
      reach_precondition:  LEMMA % Precondition rule
         Reach(Step(L,tau))(precondition(L,X),X)
      reach_invariant:  LEMMA % Invariant rule
         Reach(Step(L,tau))(X,Y) AND invariant(L)(I)
         IMPLIES Reach(Step(L,tau))(X AND I, Y AND I)
END PRECONDITION
```

To connect the proof rules on the `Reach` predicate with the reachability condition of the previous section, we proved the following theorem in PVS:

```
reachability[D,E,L,Act:TYPE, tau:Act, m,n:nat]:  THEORY BEGIN
      IMPORTING CONESFOCI_METHOD[D,E,L,Act,tau,m,n],
         PRECONDITION[Act,D,L,m,tau]
      I,fc:  VAR [D->bool] X: VAR LPE[Act,D,L,m,tau] d:  VAR D
      REACH_CRIT: LEMMA invariant(L)(I) AND Reach(Step(L,tau))(I,fc) IMPLIES
         (FORALL d:  Reachable(L)(d) IMPLIES RC(L,fc)(d))
END reachability
```

This finishes the formalization of the cones and foci method in PVS. We view this as an important step. First of all, this part is protocol independent, so it can be reused in different protocol verifications. Second, it provides a rigorous formalization of the meta-theory. For a concrete protocol specification and implementation, and given invariants, mapping functions and focus condition, all proof obligations can be generated automatically and proved with relatively little effort. The theorem `CONESFOCI` in Section 4.3 states that this is sufficient to prove that the implementation is correct w.r.t. the specification modulo branching bisimulation. No additional axioms are used besides the standard PVS library. The complete dump files of the PVS formalization of the cones and foci method can be found at `http://www.cwi.nl/~vdpol/conesfoci/`.

5 Application to the CABP

Groote and Springintveld [25] proved correctness of the Concurrent Alternating Bit Protocol (CABP) [29] 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 [25] we can take τ-loops in our stride. We also illustrate our mechanical proof framework and our approach to the reachability analysis of focus points by this case study.

5.1 Informal description

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 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.

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 succeeds in sending 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 [25] defined a pre-abstraction function to hide all τ's except those that are executed in focus points, and used Koomen's fair abstraction rule [2] to eliminate the remaining τ-loops. In our adaptation of the cones and foci method, neither pre-abstraction nor Koomen's fair abstraction rule are needed.

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 a datum 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:

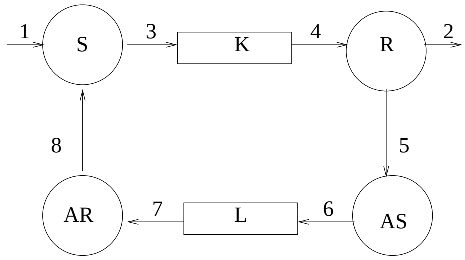


Figure 1: The structure of the CABP

lose the datum or change it into a checksum error *ce*.

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 an acknowledgment error *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 $c_n(...)$.

5.2 μCRL specification

We give descriptions of the data types and each component's specification in μCRL, which were originally presented in [25]. For convenience of notation, in each summand of the μCRL specifications below, we only present the parameters whose values are changed.

We use the sort *Nat* of natural numbers, and the sort *Bit* with elements b_0 and b_1 with an inversion function $inv : Bit \to Bit$ to model the alternating bit. The sort D contains the data elements to be transferred. The sort *Frame* consists of pairs $\langle d, b\rangle$ with $d{:}D$ and $b{:}Bit$. *Frame* also contains two error messages, *ce* for checksum error and *ae* for acknowledgment error. $eq : S \times S \to Bool$ coincides with the equality relation between elements of the sort S.

The data transmitter S reads a datum at port 1 and repeatedly transmits the datum with a bit b_s attached at port 3 until it receives an acknowledgment *ac* through port 8; after that, the bit-to-be-attached is inverted. The parameter i_s is used to model the state of the data transmitter.

Definition 5.1 (Data transmitter)

$$
\begin{array}{ll}
 & S(d_s{:}D, b_s{:}Bit, i_s{:}Nat) \\
= & \sum_{d:D} r_1(d)\cdot S(d/d_s, 2/i_s) \lhd eq(i_s, 1) \rhd \delta \\
+ & (s_3(\langle d_s, b_s\rangle)\cdot S() + r_8(ac)\cdot S(inv(b_s)/b_s, 1/i_s)) \lhd eq(i_s, 2) \rhd \delta
\end{array}
$$

The data transmission channel K reads a datum at port 3. It can do one of three things: it can deliver the datum correctly via port 4, lose the datum, or corrupt the datum by changing it into *ce*. The non-deterministic choice between the three options is modeled by the action j. b_k is the attached alternating bit for K. And its state is modeled by the parameter i_k.

Definition 5.2 (Data transmission channel)

$$
\begin{array}{ll}
 & K(d_k{:}D, b_k{:}Bit, i_k{:}Nat) \\
= & \sum_{d:D}\sum_{b:Bit} r_3(\langle d, b\rangle)\cdot K(d/d_k, b/b_k, 2/i_k) \lhd eq(i_k, 1) \rhd \delta \\
+ & (j\cdot K(1/i_k) + j\cdot K(3/i_k) + j\cdot K(4/i_k)) \lhd eq(i_k, 2) \rhd \delta \\
+ & s_4(\langle d_k, b_k\rangle)\cdot K(1/i_k) \lhd eq(i_k, 3) \rhd \delta \\
+ & s_4(ce)\cdot K(1/i_k) \lhd eq(i_k, 4) \rhd \delta
\end{array}
$$

The data receiver R reads a datum at port 4. If the datum is not a checksum *ce* and if the bit attached is the expected bit, it sends the received datum into port 2, sends an acknowledgment *ac* via port 5, and inverts the bit-to-be-expected is inverted. If the datum is *ce* or the bit attached is not the expected one, the datum is simply ignored. The parameter i_r is used to model the state of the data receiver.

Definition 5.3 (Data receiver)

$$
\begin{array}{ll}
 & R(d_r{:}D, b_r{:}Bit, i_r{:}Nat) \\
= & \sum_{d:D} r_4(\langle d, b_r\rangle)\cdot R(d/d_r, 2/i_r) \lhd eq(i_r, 1) \rhd \delta \\
+ & (r_4(ce) + \sum_{d:D} r_4(\langle d, inv(b_r)\rangle))\cdot R() \lhd eq(i_r, 1) \rhd \delta \\
+ & s_2(d_r)\cdot R(3/i_r) \lhd eq(i_r, 2) \rhd \delta \\
+ & s_5(ac)\cdot R(inv(b_r)/b_r, 1/i_r) \lhd eq(i_r, 3) \rhd \delta
\end{array}
$$

The acknowledgment transmitter AS repeats sending its acknowledgment bit b'_r via port 6, until it receives an acknowledgment *ac* from port 5, after which the acknowledgment bit is inverted.

Definition 5.4 (Acknowledgment transmitter)

$$AS(b'_r{:}Bit) = r_5(ac)\cdot AS(inv(b'_r)) + s_6(b'_r)\cdot AS()$$

The acknowledgment transmission channel L reads an acknowledgment bit from port 6. It non-deterministically does one of three things: deliver it correctly via port 7, lose the acknowledgment, or corrupt the acknowledgment by changing it to *ae*. The non-deterministic choice between the three options is modeled by the action j. b_l is the attached alternating bit for L. And its state is modeled by the parameter i_l.

Definition 5.5 (Acknowledgment transmission channel)

$$
\begin{array}{rl}
& L(b_l{:}Bit, i_l{:}Nat) \\
= & \sum_{b:Bit} r_6(b)\cdot L(b/b_l, 2/i_l) \lhd eq(i_l, 1) \rhd \delta \\
+ & (j\cdot L(1/i_l) + j\cdot L(3/i_l) + j\cdot L(4/i_l)) \lhd eq(i_l, 2) \rhd \delta \\
+ & s_7(b_l)\cdot L(1/i_l) \lhd eq(i_l, 3) \rhd \delta \\
+ & s_7(ae)\cdot L(1/i_l) \lhd eq(i_l, 4) \rhd \delta
\end{array}
$$

The acknowledgment receiver AR reads an acknowledgment bit from port 7. If the bit is the expected one, it sends an acknowledgment ac to the data transmitter S via port 8, after which the bit-to-be-expected is inverted. Acknowledgments errors ae or unexpected bits are ignored.

Definition 5.6 (Acknowledgment receiver)

$$
\begin{array}{rl}
& AR(b'_s{:}Bit, i'_s{:}Nat) \\
= & r_7(b'_s)\cdot AR(2/i'_s) \lhd eq(i'_s, 1) \rhd \delta \\
+ & (r_7(ae) + r_7(inv(b'_s)))\cdot AR() \lhd eq(i'_s, 1) \rhd \delta \\
+ & s_8(ac)\cdot AR(inv(b'_s)/b'_s, 1/i'_s) \lhd eq(i'_s, 2) \rhd \delta
\end{array}
$$

The μCRL specification of the CABP is obtained by putting the six components in parallel and encapsulating the internal actions at ports $\{3, 4, 5, 6, 7, 8\}$. Synchronization between the components is modeled by communication actions at connecting ports.

Definition 5.7 Let H denote $\{s_3, r_3, s_4, r_4, s_5, r_5, s_6, r_6, s_7, r_7, s_8, r_8\}$, and I denote $\{c_3, c_4, c_5, c_6, c_7, c_8, j\}$.

$$
\begin{array}{rl}
& CABP(d{:}D) \\
= & \tau_I(\partial_H(S(d, b_0, 1) \parallel AR(b_0, 1) \parallel K(d, b_1, 1) \parallel L(b_1, 1) \parallel R(d, b_0, 1) \parallel AS(b_1)))
\end{array}
$$

Next the CABP is expanded to an LPE *Sys*. Note that the parameters b'_s (of AR) and b'_r (of AS) are missing. The reason for this is that during the linearization the communications at ports 6 and 7 enforce $eq(b'_s, b_l)$ and $eq(b'_r, b_l)$.

Lemma 5.8 For all $d{:}D$ we have

$$
CABP(d) \quad = \quad Sys(d, b_0, 1, 1, d, b_0, 1, d, b_1, 1, b_1, 1)
$$

where

$$
\begin{array}{rll}
& Sys(d_s{:}D, b_s{:}Bit, i_s{:}Nat, i'_s{:}Nat, d_r{:}D, b_r{:}Bit, i_r{:}Nat, & \\
& \quad d_k{:}D, b_k{:}Bit, i_k{:}Nat, b_l{:}Bit, i_l{:}Nat) & \\
= & \sum_{d:D} r_1(d)\cdot Sys(d/d_s, 2/i_s) \lhd eq(i_s,1) \rhd \delta & (1) \\
+ & \tau\cdot Sys(d_s/d_k, b_s/b_k, 2/i_k) \lhd eq(i_s,2) \wedge eq(i_k,1) \rhd \delta & (2) \\
+ & (\tau\cdot Sys(1/i_k) + \tau\cdot Sys(3/i_k) + \tau\cdot Sys(4/i_k)) \lhd eq(i_k,2) \rhd \delta & (3) \\
+ & \tau\cdot Sys(d_k/d_r, 2/i_r, 1/i_k) \lhd eq(i_r,1) \wedge eq(b_r,b_k) \wedge eq(i_k,3) \rhd \delta & (4) \\
+ & \tau\cdot Sys(1/i_k) \lhd eq(i_r,1) \wedge eq(b_r, inv(b_k)) \wedge eq(i_k,3) \rhd \delta & (5) \\
+ & \tau\cdot Sys(1/i_k) \lhd eq(i_r,1) \wedge eq(i_k,4) \rhd \delta & (6) \\
+ & s_2(d_r)\cdot Sys(3/i_r) \lhd eq(i_r,2) \rhd \delta & (7) \\
+ & \tau\cdot Sys(inv(b_r)/b_r, 1/i_r) \lhd eq(i_r,3) \rhd \delta & (8) \\
+ & \tau\cdot Sys(inv(b_r)/b_l, 2/i_l) \lhd eq(i_l,1) \rhd \delta & (9) \\
+ & (\tau\cdot Sys(1/i_l) + \tau\cdot Sys(3/i_l) + \tau\cdot Sys(4/i_l)) \lhd eq(i_l,2) \rhd \delta & (10) \\
+ & \tau\cdot Sys(1/i_l, 2/i'_s) \lhd eq(i'_s,1) \wedge eq(b_l,b_s) \wedge eq(i_l,3) \rhd \delta & (11) \\
+ & \tau\cdot Sys(1/i_l) \lhd eq(i'_s,1) \wedge eq(b_l, inv(b_s)) \wedge eq(i_l,3) \rhd \delta & (12) \\
+ & \tau\cdot Sys(1/i_l) \lhd eq(i'_s,1) \wedge eq(i_l,4) \rhd \delta & (13) \\
+ & \tau\cdot Sys(inv(b_s)/b_s, 1/i_s, 1/i'_s) \lhd eq(i_s,2) \wedge eq(i'_s,2) \rhd \delta & (14)
\end{array}
$$

Proof. See [25]. ⊠

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

Definition 5.9 The LPE of the external behavior of the CABP is

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

5.3 Verification using cones and foci

We apply our version of the cones and foci method to verify the CABP. Let Ξ 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 $(d_s, b_s, i_s, i'_s, d_r, b_r, i_r, d_k, b_k, i_k, b_l, i_l)$. We list six invariants for the CABP, which are taken from [25].

Definition 5.10

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

$\mathcal{I}_1 \sim \mathcal{I}_5$ describe the range of the data parameters i_s, i'_s, i_k, i_r, and i_l, respectively. $\mathcal{I}_6$ 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.

Lemma 5.11 $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{I}_3$, $\mathcal{I}_4$, $\mathcal{I}_5$ and $\mathcal{I}_6$ are invariants of *Sys*.

Proof. We need to show that the invariants are preserved by each of the summands (1) − (14) in the specification of *Sys*. Invariants $\mathcal{I}_1 - \mathcal{I}_5$ are trivial to prove. To prove $\mathcal{I}_6$, we divide $\mathcal{I}_6$ into its six parts:

$$\begin{array}{rcl}
\mathcal{I}_{61}(\xi) & \equiv & (eq(i_s,1) \Rightarrow eq(b_s, inv(b_k)) \wedge eq(b_s,b_r) \wedge eq(d_s,d_k) \\
& & \qquad\qquad\qquad \wedge\, eq(d_s,d_r) \wedge eq(i'_s,1) \wedge eq(i_r,1)) \\
\mathcal{I}_{62}(\xi) & \equiv & eq(b_s,b_k) \Rightarrow eq(d_s,d_k) \\
\mathcal{I}_{63}(\xi) & \equiv & eq(i_r,2) \vee eq(i_r,3) \Rightarrow eq(d_s,d_r) \wedge eq(b_s,b_r) \wedge eq(b_s,b_k) \\
\mathcal{I}_{64}(\xi) & \equiv & eq(b_s, inv(b_r)) \Rightarrow eq(d_s,d_r) \wedge eq(b_s,b_k) \\
\mathcal{I}_{65}(\xi) & \equiv & eq(b_s,b_l) \Rightarrow eq(b_s, inv(b_r)) \\
\mathcal{I}_{66}(\xi) & \equiv & eq(i'_s,2) \Rightarrow eq(b_s,b_l).
\end{array}$$

We consider only seven summands in the specification of *Sys*; the other summands trivially preserve $\mathcal{I}_6$. For the sake of presentation, we represent $eq(b_1, inv(b_2))$ as $\neg eq(b_1,b_2)$, where b_1 and b_2 range over the sort *Bit*.

1. Summand (1): $\mathcal{I}_6 \wedge eq(i_s,1) \Rightarrow \mathcal{I}_6(d/d_s, 2/i_s)$.

   $\mathcal{I}_{61}(d/d_s,2/i_s)$ is straightforward. By $eq(i_s,1)$ and $\mathcal{I}_{61}$, we have $\neg eq(b_s,b_k)$, $eq(i_r,1)$, and $eq(b_s,b_r)$. By $\neg eq(b_s,b_k)$, $\mathcal{I}_{62}(d/d_s,2/i_s)$. By $eq(i_r,1)$, $\mathcal{I}_{63}(d/d_s,2/i_s)$. $eq(b_s,b_r)$ implies $\mathcal{I}_{64}(d/d_s,2/i_s)$. $\mathcal{I}_{65}$, $\mathcal{I}_{66}(d/d_s,2/i_s)$ are trivial.

2. Summand (2): $\mathcal{I}_6 \wedge eq(i_s,2) \wedge eq(i_k,1) \Rightarrow \mathcal{I}_6(d_s/d_k, b_s/b_k, 2/i_k)$.

   $eq(i_s,2)$ implies $\mathcal{I}_{61}(d_s/d_k,b_s/b_k,2/i_k)$. $\mathcal{I}_{62}(d_s/d_k,b_s/b_k,2/i_k)$ is straightforward. $\mathcal{I}_{63}(d_s/d_k,b_s/b_k,2/i_k)$ and $\mathcal{I}_{64}(d_s/d_k,b_s/b_k,2/i_k)$ follows immediately from $\mathcal{I}_{63}$ and $\mathcal{I}_{64}$, respectively. $\mathcal{I}_{65}$, $\mathcal{I}_{66}(d_s/d_k,b_s/b_k,2/i_k)$ are trivial.

3. Summand (4): $\mathcal{I}_6 \wedge eq(i_r,1) \wedge eq(b_r,b_k) \wedge eq(i_k,3) \Rightarrow \mathcal{I}_6(d_k/d_r, 2/i_r, 1/i_k)$.

   Assuming $eq(i_s,1)$, by $\mathcal{I}_{61}$, it follows that $\neg eq(b_s,b_k)$ and $eq(b_s,b_r)$. Hence, $\neg eq(b_r,b_k)$. This contradicts the condition $eq(b_r,b_k)$. So $\mathcal{I}_{61}(d_k/d_r,2/i_r,1/i_k)$. $\mathcal{I}_{64}$ implies $eq(b_s,b_r) \vee eq(b_s,b_k)$, which together with the condition $eq(b_r,b_k)$ yields $eq(b_s,b_r) \wedge eq(b_s,b_k)$. So $\mathcal{I}_{62}$ implies $eq(d_s,d_k)$. Hence, $\mathcal{I}_{63}(d_k/d_r,2/i_r,1/i_k)$. By $eq(b_s,b_r)$, $\mathcal{I}_{64}(d_k/d_r,2/i_r,1/i_k)$. $\mathcal{I}_{62}$, $\mathcal{I}_{65}$, $\mathcal{I}_{66}(d_k/d_r,2/i_r,1/i_k)$ are trivial.

4. Summand (8): $\mathcal{I}_6 \wedge eq(i_r,3) \Rightarrow \mathcal{I}_6(inv(b_r)/b_r, 1/i_r)$.

   Assuming $eq(i_s,1)$, by $\mathcal{I}_{61}$, we have $eq(i_r,1)$, which contradicts the condition $eq(i_r,3)$. So $\mathcal{I}_{61}(inv(b_r)/b_r,1/i_r)$. $\mathcal{I}_{63}(inv(b_r)/b_r,1/i_r)$ is straightforward. By $eq(i_r,3)$ and $\mathcal{I}_{63}$, we have $eq(d_s,d_r)$ and $eq(b_s,b_k)$. Hence,

$\mathcal{I}_{64}(inv(b_r)/b_r, 1/i_r)$. By $eq(i_r, 3)$ and $\mathcal{I}_{63}$, we have $eq(b_s, b_r)$, so $\mathcal{I}_{65}$ implies $\neg eq(b_s, b_l)$. Hence, $\mathcal{I}_{65}(inv(b_r)/b_r, 1/i_r)$. $\mathcal{I}_{62}$, $\mathcal{I}_{66}(inv(b_r)/b_r, 1/i_r)$ are trivial.

5. Summand (9): $\mathcal{I}_6 \wedge eq(i_l, 1) \Rightarrow \mathcal{I}_6(inv(b_r)/b_l, 2/i_l)$,

   $\mathcal{I}_{65}(inv(b_r)/b_l, 2/i_l)$ is straightforward. If $eq(i'_s, 2)$, by $\mathcal{I}_{66}$ we have $eq(b_s, b_l)$, so by $\mathcal{I}_{65}$ we have $\neg eq(b_l, b_r)$. Hence, $\mathcal{I}_{66}(inv(b_r)/b_l, 2/i_l)$. $\mathcal{I}_{61} \sim \mathcal{I}_{64}(inv(b_r)/b_l, 2/i_l)$ are trivial.

6. Summand (11): $\mathcal{I}_6 \wedge eq(i'_s, 1) \wedge eq(b_l, b_s) \wedge eq(i_l, 3) \Rightarrow \mathcal{I}_6(1/i_l, 2/i'_s)$.

   By $eq(b_l, b_s)$ and $\mathcal{I}_{65}$, we have $\neg eq(b_s, b_r)$. So by $\mathcal{I}_{61}$, $\neg eq(i_s, 1)$. Hence, $\mathcal{I}_{61}(1/i_l, 2/i'_s)$. $eq(b_l, b_s)$ implies $\mathcal{I}_{66}(1/i_l, 2/i'_s)$. $\mathcal{I}_{62} \sim \mathcal{I}_{65}(1/i_l, 2/i'_s)$ are trivial.

7. Summand (14): $\mathcal{I}_6 \wedge eq(i_s, 2) \wedge eq(i'_s, 2) \Rightarrow \mathcal{I}_6(inv(b_s)/b_s, 1/i_s, 1/i'_s)$.

   To prove $\mathcal{I}_{61}(inv(b_s)/b_s, 1/i_s, 1/i'_s)$, we need to show $eq(b_s, b_k) \wedge \neg eq(b_r, b_s) \wedge eq(d_s, d_k) \wedge eq(d_s, d_r) \wedge eq(i_r, 1)$. As $eq(i'_s, 2)$, by $\mathcal{I}_{66}$ we have $eq(b_s, b_l)$, so by $\mathcal{I}_{65}$, we have $\neg eq(b_s, b_r)$. By $\mathcal{I}_{64}$, it follows that $eq(d_s, d_r) \wedge eq(b_s, b_k)$. As $eq(b_s, b_k)$, by $\mathcal{I}_{62}$, $eq(d_s, d_k)$. By $\mathcal{I}_{63}$ and $\mathcal{I}_4$, $\neg eq(b_s, b_r)$ implies $eq(i_r, 1)$. Hence, $\mathcal{I}_{61}(inv(b_s)/b_s, 1/i_s, 1/i'_s)$. $\mathcal{I}_{62} \sim \mathcal{I}_{66}(inv(b_s)/b_s, 1/i_s, 1/i'_s)$ are trivial.

$\boxtimes$

We define the focus condition (see Definition 3.1) for *Sys* as the disjunction of the conditions of summands in the LPE in Definition 5.8 that deal with an external action; these summands are (1) and (7). (Note that this differs from the prescribed focus condition in [25], which would be the negation of the disjunction of conditions of the summands that deal with a τ.)

Definition 5.12 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 $\mathcal{I}_1 - \mathcal{I}_6$ can reach a focus point (see Definition 3.1) by a sequence of τ-transitions.

Lemma 5.13 (Reachability of focus points) For each $\xi{:}\Xi$ with $\bigwedge_{n=1}^{6} \mathcal{I}_n(\xi)$, there is a $\hat{\xi}{:}\Xi$ such that $FC(\hat{\xi})$ and $\xi \xrightarrow{\tau} \cdots \xrightarrow{\tau} \hat{\xi}$ in *Sys*.

Proof. The case $FC(\xi)$ is trivial. 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 $eq(i_s, 2) \wedge eq(i_r, 3)$, by summand (8) we can reach a state with $eq(i_s, 2) \wedge eq(i_r, 1)$. From a state with $eq(i_s, 2) \wedge eq(i_r, 1)$, by $\mathcal{I}_3$ and summands (2), (3) and (6), we can reach a state where $eq(i_s, 2) \wedge eq(i_r, 1) \wedge eq(i_k, 3)$. We distinguish two cases.

1. $eq(b_r, b_k)$.

   By summand (4) we can reach a focus point.

2. $eq(b_r, inv(b_k))$.

   If $i'_s = 2$, then by summand (14) we can reach a focus point. So by $\mathcal{I}_2$ we can assume that $i'_s = 1$. By summands (5), (2) and (3), we can reach a state where $eq(i_s, 2) \wedge eq(i'_s, 1) \wedge eq(i_r, 1) \wedge eq(i_k, 3) \wedge eq(b_r, inv(b_k)) \wedge eq(b_k, b_s)$. By $\mathcal{I}_5$ and summands (10), (9) and (13) we can reach a state where $eq(i_s, 2) \wedge eq(i'_s, 1) \wedge eq(i_r, 1) \wedge eq(i_k, 3) \wedge eq(b_r, inv(b_k)) \wedge eq(b_k, b_s) \wedge eq(i_l, 3)$. If $eq(b_l, b_s)$, then by summands (11) and (14) we can reach a focus point. Otherwise, $eq(b_l, inv(b_s))$. Since $eq(b_k, b_s)$ and $eq(b_r, inv(b_k))$, we have $eq(b_l, b_r)$. By summand (12), we can reach a state where $eq(i_s, 2) \wedge eq(i'_s, 1) \wedge eq(i_r, 1) \wedge eq(i_k, 3) \wedge eq(b_r, inv(b_k)) \wedge eq(b_k, b_s) \wedge eq(i_l, 1) \wedge eq(b_l, inv(b_s)) \wedge eq(b_l, b_r)$. Then by summand (9) we can reach a state where $eq(b_l, b_s)$, since b_l is replaced by $inv(b_r)$. Then by summands (10), (11) and (14), we can reach a focus point.

Our completely formal proof in PVS has many more steps. The main steps of the proof using the rules in Definition 3.7 can be found in Section 5.4.

$\boxtimes$

We define the state mapping $\phi : \Xi \to D \times Bool$ (see Definition 3.2) by

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

Intuitively, ϕ maps those states to T in which R is awaiting a datum that still has to be received by S. This is the case if either S is awaiting a datum ($eq(i_s, 1)$), or R has sent out a datum that was not yet acknowledged to S ($eq(i_r, 3) \vee \neg eq(b_s, b_r)$). Note that ϕ 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 5.14 For all $d{:}D$ and $b_0, b_1{:}Bit$,

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

Proof. It is easy to check that $\wedge_{n=1}^{6} \mathcal{I}_n(d, b_0, 1, 1, d, b_0, 1, d, b_1, 1, b_1, 1)$.

We obtain the following matching criteria (see Definition 3.3). 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

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

I.2 Let $eq(i_r, 3)$. Then by $\mathcal{I}_6$, $eq(b_s, b_r)$. Hence,

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

I.3 Let $eq(i'_s, 2)$. $\mathcal{I}_6$, $eq(b_s, b_l)$ together with $\mathcal{I}_6$ yield $eq(b_s, inv(b_r))$. Hence,

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

II.1 Trivial.

II.2 Let $eq(i_r, 2)$. Then clearly $\neg eq(i_r, 3)$, and by $\mathcal{I}_6$, $eq(b_s, b_r)$. Furthermore, according to $\mathcal{I}_6$, $eq(i_s, 1) \Rightarrow eq(i_r, 1)$, 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 $\mathcal{I}_6$, $eq(b_s, b_r)$, 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 $\neg(eq(i_s, 1) \vee eq(i_r, 3) \vee \neg eq(b_s, b_r))$ is false, so that it implies $eq(i_r, 2)$.

IV.1 Trivial.

IV.2 Let $eq(i_r, 2)$. Then by $\mathcal{I}_6$, $eq(d_r, d_s)$.

V.1 Let $eq(i_s, 1)$. Then by $\mathcal{I}_6$, $eq(i_r, 1)$ and $eq(b_s, b_r)$. So for any $d{:}D$,

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

V.2

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

Note that $\phi(d, b_0, 1, b_0, 1) = \langle d, \mathsf{T}\rangle$. So by Theorem 3.4 and Lemma 5.13,

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

$\boxtimes$

5.4 Illustration of the proof framework

Let us illustrate the mechanical proof framework set up in Section 4 on the verification of the CABP as it was described in Section 5.3. The purpose of this section is to show how the mechanical framework can be instantiated with a concrete protocol. A second goal is to illustrate in more detail how we can use the proof rules (see Lemma 3.7) for reachability, to formally prove in PVS that focus points are always reachable.

To apply the generic theory, we use the PVS mechanism of theory instantiation. For instance, the theory LPE was parameterized by sets of actions, states, etc. This theory will be imported, using the set of actions, states etc. from the linearized version of CABP, which we have to define first. To this end we start a new theory, parameterized by an arbitrary type of data elements (D, with special element $d_0 : D$).

Defining the LPEs. The starting point will be the linearized version of the CABP, represented by Sys in Lemma 5.8. The type `cabp_state` is defined as a record of all state parameters. Note that we use the predefined PVS-types `nat` and `bool` (`bool` is also used to represent sort Bit). The type `cabp_act` is defined as an abstract data type. The syntax below introduces constructors (`r1,s2:[D->cabp_act]` and `tau:cabp_act`), recognizer predicates (`r1?,s2?,tau?:[cabp_act->bool]`), and destructors (`d:[(r1?)->D]` and `d:[(r2?)->D]`). Subsequently we import the theory LPE with the corresponding parameters. The LPE for the implementation of the CABP contains 18 summands (note that summands (3) and (10) in Lemma 5.8 each represent three summands). Note that the only local parameter in this LPE that is bound by $\sum$ has type `D`.

```
CABP[D:TYPE+,d0:D]: THEORY BEGIN
      cabp_state:TYPE= [#ds:D,bs:bool,is:nat,i1s:nat,dr:D,br:bool,
         ir:nat,dk:D,bk:bool,ik:nat,bl:bool,il:nat#]
      cabp_act:DATATYPE BEGIN
            r1(d:D):r1?
            s2(d:D):s2?
            tau:tau?
         END cabp_act
      IMPORTING LPE[cabp_act,cabp_state,D,18,tau]
```

The next step is to define the implementation of the CABP as an LPE in PVS. It consists of an initial vector, and a list of summands, indexed by `LAMBDA i`. The `LAMBDA (S,d)` indicates the dependency of each summand on the state and the local variables. Note that given state `S`, ``S`x`` denotes the value of parameter `x` in `S`. The notation `S WITH [x := v]` denotes the same state as `S` except the value of field `x` which is set to `v`. We only display the summands corresponding to summand (1) and (14) of Sys.

```
      i:VAR below(18)   S:VAR cabp_state   d:VAR D
      cabp:  LPE= (#
         init:= (#ds:=d0,bs:=FALSE,is:=1,i1s:=1,dr:=d0,
            br:=FALSE,ir:=1,dk:=d0,bk:=TRUE,ik:=1,bl:=TRUE,il:=1#),
         sums:=LAMBDA i:  LAMBDA (S,d):  COND
            i=0->(#act:=r1(d),guard:=S`is=1,next:=S WITH [ds:=d,is:=2]#),
            ...
            i=17->(#act:=tau,guard:=S`is=2 AND S`i1s=2,
               next:=S WITH [bs:=NOT S`bs,is:=1,i1s:=1]#)
            ENDCOND#)
```

In a similar way, the desired external behavior of the CABP is presented as a one-datum buffer. The representation of the LPE B from Definition 5.9 in PVS is:

```
      buf_state:TYPE=[#d:D,b:bool#]
      B:VAR buf_state d1:VAR D j:VAR below(2)
      IMPORTING LPE[cabp_act,buf_state,D,2,tau]
      buffer:  LPE =
         (#init:=(#d:=d0,b:=TRUE#),
         sums:=LAMBDA j:  LAMBDA (B,d1):  COND
            j=0->(#act:=r1(d1),guard:=B`b,next:=(#d:=d1,b:=FALSE#)#),
            j=1->(#act:=s2(B`d),guard:=NOT B`b,next:=B WITH [b:=TRUE]#)
         ENDCOND#)
```

Invariants, state mapping, focus points. The next step is to define the ingredients for the cones and foci method. We need to define invariants, a state mapping and focus points. In PVS these are all functions that take state vectors as input. We only show a snapshot:

```
IMPORTING invariant[cabp_act,cabp_state,D,18]
   I1(S):bool = S`is=1 OR S`is=2
   ...
   I64(S):bool = (S`bs = NOT S`br) IMPLIES S`ds=S`dr AND S`bs=S`bk
   I6(S):bool=I61(S) AND ...  AND I66(S)
IMPORTING CONESFOCI_METHOD[cabp_state,buf_state,D,cabp_act,tau,18,2]
   FC(S):bool= S`is=1 OR S`ir=2
   h(S):buf_state=(#d:=S`ds,b:=S`is=1 OR S`ir=3 OR NOT S`bs=S`br#)
cabp_inv:  LEMMA invariant(cabp)(I1 AND I2 AND I3 AND I4 AND I5 AND I6)
matching:  LEMMA Reachable(cabp)(S) IMPLIES MC(cabp,buffer,k,h,FC)(S)
```

The proof of the reachability criterion will be discussed in the next paragraph. The correctness of the invariants and the matching criteria were proved already (see Section 5). These proofs could be formalized in PVS in a straightforward fashion. The proof script follows a fixed pattern: first we unfold the definitions of LPE and invariants or matching criteria. Then we use rewriting to generate a finite conjunction from the quantification `FORALL i:below(n)`. Subsequently (using the PVS tactic `THEN*`), we apply the powerful PVS tactic `(GRIND)` to the subgoals. Sometimes a few subgoals remain, which are then proved manually.

Reachability of focus points. We formally prove Lemma 5.13, which states that each reachable state of the CABP can reach a focus point by a sequence of τ-transitions using the rules in Lemma 3.7. This corresponds to the theorem `CABP_RC` in the PVS part below. Using the general theorems `CONESFOCI` and `REACH_CRIT`, we can conclude from the specific theorems `cabp_inv`, `matching` and `CABP_RC` that `CABP` is indeed `CORRECT` w.r.t. the one-datum buffer specification.

```
     IMPORTING PRECONDITION[cabp_act,cabp_state,D,18]
        ...
        CABP_RC:LEMMA Reach(step(cabp,tau))(I1 AND I2 AND I3 AND I4 AND I5,FC)
        CABP_CORRECT: THEOREM brbisimilar(lpe2lts(cabp),lpe2lts(buffer))
END CABP
```

We now explain the structure of the proof of `CABP_RC`. This proof is based on the proof rules for reachability, introduced in Sections 3.2 and 4.4. It requires some manual work, viz. the identification of the intermediate predicates, and characterizing the reachable set of states after a number of steps. Each step corresponds to a separate lemma in PVS. The atomic steps are proved by the precondition-rule (semi-automatically). An example of such a lemma in PVS is:

```
Q2(S):bool = S`ir=1 AND S`is=2 AND S`ik=2 AND S`i1s=1 AND S`bk = S`bs
Q3(S):bool = S`ir=1 AND S`is=2 AND S`ik=3 AND S`i1s=1 AND S`bk = S`bs
Q2_to_Q3:  LEMMA Reach(Tau)(Q2,Q3)
```

These basic steps are combined by using mainly the transitivity rule and the disjunction-rule. We now provide the complete list of the intermediate predicates, together with the used proof rules. We do not display the use of implication and invariant rules, but of course the PVS proofs contain all details. The fragment before corresponds to the third step of item (5) below, where summand (3) is used to increase i_k.

1. $\{i_r = 1 \wedge i_s = 2 \wedge i_k = 4\} \twoheadrightarrow \{i_r = 1 \wedge i_s = 2 \wedge i_k = 1\} \twoheadrightarrow$
   $\{i_r = 1 \wedge i_s = 2 \wedge i_k = 2\} \twoheadrightarrow \{i_r = 1 \wedge i_s = 2 \wedge i_k = 3\}$
   Using the precondition rule, on summands (6), (2) and (3), respectively.

2. $\{\mathcal{I}_3 \wedge i_r = 1 \wedge i_s = 2\} \twoheadrightarrow \{i_r = 1 \wedge i_s = 2 \wedge i_k = 3\}$
   Using the disjunction rule with $i_k = 1 \vee i_k = 2 \vee i_k = 3 \vee i_k = 4$, and the transitivity rule on the results of step 1.

3. $\{i_r = 1 \wedge i_s = 2 \wedge i_k = 3 \wedge b_r = b_k\} \twoheadrightarrow FC$
   Using the precondition rule on summand (4).

4. $\{i_r = 1 \wedge i_s = 2 \wedge i_k = 3 \wedge i'_s = 2\} \twoheadrightarrow FC$
   Using the precondition rule on summand (14).

5. $\{i_r = 1 \wedge i_s = 2 \wedge i_k = 3 \wedge i'_s = 1 \wedge b_r \neq b_k\} \twoheadrightarrow$
   $\{i_r = 1 \wedge i_s = 2 \wedge i_k = 1 \wedge i'_s = 1\} \twoheadrightarrow$
   $\{i_r = 1 \wedge i_s = 2 \wedge i_k = 2 \wedge i'_s = 1 \wedge b_k = b_s\} \twoheadrightarrow$
   $\{i_r = 1 \wedge i_s = 2 \wedge i_k = 3 \wedge i'_s = 1 \wedge b_k = b_s\} =: Q$
   Using the precondition rule on summands (5), (2) and (3).

6. $\{Q \wedge i_l = 2\} \twoheadrightarrow \{Q \wedge i_l = 1\}$;
   $\{Q \wedge i_l = 4\} \twoheadrightarrow \{Q \wedge i_l = 1\}$;
   $\{Q \wedge i_l = 3 \wedge b_l \neq b_s\} \twoheadrightarrow \{Q \wedge i_l = 1\} \twoheadrightarrow \{Q \wedge i_l = 2 \wedge b_l \neq b_r\} \twoheadrightarrow$
   $\{Q \wedge i_l = 3 \wedge b_l \neq b_r\}$
   Using the precondition rule on summands (10), (13), (12), (9) and (10), respectively.

7. $\{Q \wedge (i_l \in \{1, 2, 4\} \vee (i_l = 3 \wedge b_l \neq b_s))\} \twoheadrightarrow \{Q \wedge i_l = 3 \wedge b_l \neq b_r\}$.
   Using the disjunction rule and the transitivity rule on the results of step 6.

8. $\{Q \wedge i_l = 3 \wedge b_l = b_s\} \twoheadrightarrow \{i_r = 1 \wedge i_s = 2 \wedge i_k = 3 \wedge i'_s = 2\} \twoheadrightarrow FC$.
   Using the precondition rule on summand (11), and then the transitivity rule with step 4.

9. $\{Q \wedge \mathcal{I}_5\} \twoheadrightarrow FC$.
   By $\mathcal{I}_5$, $i_l \in \{1, 2, 3, 4\}$. So we can distinguish the cases $i_l \in \{1, 2, 4\}$, $i_l = 3 \wedge b_l \neq b_s$ and $i_l = 3 \wedge b_l = b_s$. In all but the last case, we arrive at a situation where $b_k = b_s \wedge b_l \neq b_r$ (by step 7). Note that this implies $b_k = b_r \vee b_l = b_s$. So we can use case distinction again, and reach the focus condition via step 3 or step 8.

10. $\{i_r = 1 \wedge i_s = 2 \wedge i_k = 3 \wedge \mathcal{I}_2 \wedge \mathcal{I}_5\} \twoheadrightarrow FC$.
    From $\mathcal{I}_2$ and the disjunction rule we can distinguish the cases $b_r = b_k$,

$i'_s = 2$ and $i'_s = 1 \wedge b_r \neq b_k$. We solve them by the results of step 3, step 4, and transitivity of 5 and 9, respectively.

11. $\{i_r = 3 \wedge i_s = 2\} \twoheadrightarrow \{i_r = 1 \wedge i_s = 2\}$.
Using the precondition rule on summand (8).

12. $\mathcal{I}_1 \wedge \mathcal{I}_2 \wedge \mathcal{I}_3 \wedge \mathcal{I}_4 \wedge \mathcal{I}_5 \twoheadrightarrow FC$.
Using the invariants $\mathcal{I}_1$ and $\mathcal{I}_4$, we can distinguish the following cases:

    - $i_s = 1$ or $i_s = 2 \wedge i_r = 2$ (both reach FC in zero steps);
    - $i_s = 2 \wedge i_r = 3$ (leads to the next case by step 11);
    - $i_s = 2 \wedge i_r = 1$. This leads to $i_s = 2 \wedge i_r = 1 \wedge i_k = 3$ by step 2 and then to FC by step 10.

This finishes the complete mechanical verification of the CABP in PVS using the cones and foci method. The dump files of the verification of the CABP in PVS can be found at `http://www.cwi.nl/~vdpol/conesfoci/cabp/`.

6 Concluding Remarks

In this paper, we have developed a mechanical framework for protocol verification, based on the cones and foci method. We summarize our main contribution as follows:

- We generalized the original cones and foci method [25]. Compared to the original one, our method is more generally applicable, in the sense that it can deal with τ-loops without requiring a cumbersome treatment to eliminate them.
- We presented a set of rules to support the reachability analysis of focus points. They have been proved to be quite powerful in two case studies.
- We formalized the complete cones and foci method in PVS.

The feasibility of this mechanical framework has been illustrated by the verification of the CABP. We are confident that the framework forms a solid basis for mechanical protocol verification. For instance, the same framework has been applied to the verification of a sliding window protocol in μCRL [14], which we consider a true milestone in verification efforts using process algebra.

The foci and cones method provides a systematic approach to protocol verification. It allows for fully rigorous correctness proofs in a general setting with possibly infinite state spaces (i.e. with arbitrary data, arbitrary window size, etc.). The method requires intelligent manual steps, such as the invention of invariants, a state mapping, and the focus criterion. However, the method is such that after these creative parts a number of verification conditions can be generated and proved (semi-)automatically. So the strength of the mechanical framework is that one can focus on the creative steps, and check the tedious parts by a theorem prover. Yet, a complete machine-checked proof is obtained, because the meta-theory has also been proof-checked in a generic manner. We experienced that many proofs and proof scripts can be reused after small changes in the protocol, or after a change in the invariants. Actually, in some cases the PVS theorem prover assisted in finding the correct invariants.

Acknowledgments Jan Friso Groote is thanked for valuable discussions. 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.

References

[1] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[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 and J.C. van de Pol. State space reduction by proving confluence. In *Proc. 14th Conference on Computer Aided Verification*, LNCS 2404, pp. 596–609. Springer, 2002.

[8] S.C.C. Blom, W.J. Fokkink, J.F. Groote, I.A. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: A toolset for analysing algebraic specifications. In *Proc. 13th Conference on Computer Aided Verification*, LNCS 2102, pp. 250–254. Springer, 2001.

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

[10] 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.

[11] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking.* MIT Press, 2000.

[12] B. Courcelle. Recursive applicative program schemes. In *Handbook of Theoretical Computer Science, Volume B, Formal Methods and Semantics*, pp. 459–492. Elsevier, 1990.

[13] 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.

[14] W.J. Fokkink, J.F. Groote, J. Pang, B. Badban, and J.C. van de Pol. Verifying a sliding window protocol in μCRL. Technical Report SEN-R0308, CWI, 2003.

[15] W.J. Fokkink and J. Pang. Cones and foci for protocol verification revisited. In *Proc. 6th Conference on Foundations of Software Science and Computation Structures*, LNCS 2620, pp. 267–281, Springer, 2003.

[16] 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.

[17] 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.

[18] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43:555–600, 1996.

[19] 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.

[20] J. F. Groote and B. Lisser. Computer assisted manipulation of algebraic process specifications. In *Proc. 3rd Workshop on Verification and Computational Logic*, Technical Report DSSE-TR-2002-5. Department of Electronics and Computer Science, University of Southampton, 2002.

[21] J. F. Groote, A. Ponse, and Y.S. Usenko. Linearization in parallel pCRL. *Journal of Logic and Algebraic Programming*, 48:39–72, 2001.

[22] 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.

[23] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In *Proc. 1st Workshop on the Algebra of Communicating Processes*, Workshops in Computing Series, pp. 26–62. Springer, 1995.

[24] J.F. Groote and M. Reniers. Algebraic process verification. In J.A. Bergstra, A. Ponse, and S.A. Smolka, eds, *Handbook of Process Algebra*, pp. 1151–1208. Elsevier, 2001.

[25] 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.

[26] 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.

[27] J.F. Groote and J.J. van Wamel. The parallel composition of uniform processes with data. *Theoretical Computer Science*, 266:631–652, 2001.

[28] B. Jonsson. *Compositional Verification of Distributed Systems*. PhD thesis, Department of Computer Science, Uppsala University, 1987.

[29] 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.

[30] J. Loeckx, H.-D. Ehrich, and M. Wolf. *Specification of Abstract Data Types*. Wiley/Teubner, 1996.

[31] 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.

[32] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations. Part I: Untimed systems. *Information and Computation*, 121:214–233, 1995.

[33] G. Necula. Translation validation for an optimizing compiler. In *Proc. 2000 ACM SIGPLAN Conference on Programming Language Design and Implementation*, SIGPLAN Notices 35:83–94. ACM, 2000.

[34] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In *Proc. 8th Conference on Computer-Aided Verification*, LNCS 1102, pp. 411-414. Springer, 1996.

[35] 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.

[36] J.C. van de Pol. A prover for the μCRL toolset with applications – version 0.1. Technical Report SEN-R0106, CWI Amsterdam, 2001.

[37] C. Shankland and M.B. van der Zwaag. The tree identify protocol of IEEE 1394 in μCRL. *Formal Aspects of Computing*, 10:509–531, 1998.

[38] Y.S. Usenko. Linearization of μCRL specifications (extended abstract). In *Proc. 3rd Workshop on Verification and Computational Logic*, Technical Report DSSE-TR-2002-5. Department of Electronics and Computer Science, University of Southampton, 2002.

[39] M.B. van der Zwaag. The cones and foci proof technique for timed transition systems. *Information Processing Letters*, 80(1):33–40, 2001.