



Centrum voor Wiskunde en Informatica

# REPORTRAPPORT

---

The Tree Identify Protocol of IEEE 1394 in muCRL

C. Shankland, M.B. van der Zwaag

Software Engineering (SEN)

**SEN-R9831 November 1998**

Report SEN-R9831  
ISSN 1386-369X

CWI  
P.O. Box 94079  
1090 GB Amsterdam  
The Netherlands

CWI is the National Research Institute for Mathematics and Computer Science. CWI is part of the Stichting Mathematisch Centrum (SMC), the Dutch foundation for promotion of mathematics and computer science and their applications.

SMC is sponsored by the Netherlands Organization for Scientific Research (NWO). CWI is a member of ERCIM, the European Research Consortium for Informatics and Mathematics.

Copyright © Stichting Mathematisch Centrum  
P.O. Box 94079, 1090 GB Amsterdam (NL)  
Kruislaan 413, 1098 SJ Amsterdam (NL)  
Telephone +31 20 592 9333  
Telefax +31 20 592 4199

# The Tree Identify Protocol of IEEE 1394 in $\mu$ CRL

Carron Shankland<sup>1</sup> & Mark van der Zwaag<sup>2</sup>

1. University of Stirling

Department of Computing Science and Mathematics

Stirling FK9 4LA, UK

E-mail: ces@cs.stir.ac.uk

2. CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

E-mail: Mark.van.der.Zwaag@cwi.nl

## ABSTRACT

We specify the tree identify protocol of the IEEE 1394 high performance serial multimedia bus at three different levels of detail using  $\mu$ CRL. We use the cones and foci verification technique of Groote and Springintveld to show that the descriptions are equivalent under branching bisimulation, thereby demonstrating that the protocol behaves as expected.

1991 *Mathematics Subject Classification*: 68Q22: Parallel and distributed algorithms; 68Q45: Formal languages; 68Q60: Specification and verification of programmes

1991 *Computing Reviews Classification System*: D.2.1, D.2.4, D.3.3, F.3.1

*Keywords & Phrases*: process algebra, verification of distributed systems, leader election protocols

## 1 Introduction

Much time and effort is expended in the development of new techniques for description and analysis of (computer) systems; however, many of these techniques remain the preserve only of their inventors, and are never widely used. This is often due to the sharp learning curve required to adopt them; many verification techniques have complex theoretical underpinnings, and require sophisticated mathematical skills to apply them. Case studies therefore have a valuable role to play both in promoting and demonstrating particular verification techniques, and providing practical examples of their application. This paper presents one such case study. We apply the cones and foci technique of Groote and Springintveld [GS95] to a fragment of the software for a high performance serial multimedia bus, the IEEE standard 1394 [IEE96], also known as “Firewire”.

Briefly, IEEE 1394 connects together a collection of systems and devices in order to carry all forms of digitized video and audio quickly, reliably, and inexpensively. Its architecture is scalable, and it is “hot-pluggable”, so a designer or user can add or remove systems and peripherals easily at any time. The only requirement is that the form of the network should be a tree (other configurations lead to errors).

The protocol is subdivided into layers, in the manner of OSI, and further into phases, corresponding to particular tasks, e.g. data transmission or bus master identification. Much effort has been expended on the description and verification of various parts of the standard, using several different formalisms and proof techniques. For example, the operation of sending packets of information across the network is described using  $\mu$ CRL in [Lut97] and using E-LOTOS in [SM97]. The former is essentially a description only, with five correctness properties stated informally, but not formalised or proved. The

exercise of [SM97] is based on the  $\mu$ CRL description, adding another layer of the protocol and carrying out the verification suggested, using the tool CADP [FGK<sup>+</sup>96].

In this paper we concentrate on the tree identify phase of the physical layer which occurs after a bus reset in the system, e.g. when a node is added to or removed from the network. The purpose of the tree identify protocol is to assign a (new) root, or leader, to the network. Essentially, the protocol consists of a set of negotiations between nodes to establish the direction of the parent-child relationship. Another way to look at this is that from a general graph a spanning tree is created (where possible). Potentially, a node can be a parent to many nodes, but a child of at most one node. A node with no parent (after the negotiations are complete) is the leader. The tree identify protocol must ensure that a leader is chosen, and that it is the only leader chosen.

This part of the 1394 is described using I/O automata in [DGRV97]. Verification is by (manual) manipulation of a number of invariants, phrased in predicate calculus. Also discussed is the mechanisation of this verification in the theorem prover PVS.

There are three descriptions of the protocol, written using  $\mu$ CRL [GP95], in this paper:

**Specification** The specification of the external behaviour of the protocol merely announces a single leader has been chosen;

**Implementation A** In Implementation A nodes are specified individually and negotiate with their neighbours to determine the parent-child relationship. Communication is by hand shaking;

**Implementation B** This implementation has negotiation as above, but communication between nodes occurs via two unidirectional channels (therefore messages may pass each other, causing conflicts in assigning the leader).

These descriptions may be found in Sections 2.1, 2.2 and 2.3 respectively. They were derived with reference to the transition diagram in Section 4.4.2.2 of the standard [IEE96]. Section 3 gives an informal overview of the cones and foci technique of [GS95], together with some common definitions. The formal details of this technique are repeated in the appendix for convenience.

We prove, using the cones and foci technique, that Implementation A and Implementation B have the same behaviour with respect to branching bisimulation as the simple specification, therefore showing that these descriptions behave as required, i.e. a single leader is chosen. The proofs may be found in Section 4 and Section 5, respectively.

We conclude with some remarks about the success of this case study and about verification using the technique of [GS95] in general.

## 2 Description of the Tree Identify Protocol

The descriptions are given in  $\mu$ CRL, which is roughly ACP [BW90] extended with a formal treatment of data. Familiarity is assumed with this formalism; an introduction may be found in [GP95].

Briefly, the main features of the formalism are as follows:  $\delta$  represents deadlock,  $p \cdot q$  indicates sequential composition and  $p + q$  indicates alternative composition. The process  $\sum_{x:D} p(x)$  behaves as the possibly infinite choice between processes  $p(d)$  where  $d$  is any data term of sort  $D$ . The parallel composition of processes  $p$  and  $q$  is written  $p \parallel q$ . We have a sort  $\mathbb{B}$  of booleans with two elements  $t$  and  $f$  and the usual boolean operators. Conditionals are written  $p \triangleleft b \triangleright q$ , meaning if  $b$  holds behave as  $p$ , otherwise behave as  $q$ . The operator  $\tau_I$  hides all those actions in the set  $I$ , by converting them to silent  $\tau$  actions, and  $\partial_H$  restricts enabled actions, by renaming actions in  $H$  to  $\delta$ . We choose  $H$  such that the  $\partial_H$  operator forces the enclosed processes to communicate with each other. For booleans we assume the following binding conventions:  $\neg$  binds stronger than  $\wedge, \vee$ , which bind stronger than  $\rightarrow$ .

The  $\mu$ CRL data definitions used (e.g.  $\mathbb{N}$ ,  $\mathbb{N}Set$ ,  $\mathbb{N}SetList$ ) are assumed and not presented here; these are straightforward and examples of the appropriate types or similar may be found in [GP95, Lut97].

## 2.1 Specification

The most abstract specification of the tree identify protocol is the one which merely reports that a leader has been found. The network is viewed as a whole, and no communications between nodes are specified. We define

$$Spec = \text{leader} \cdot \delta.$$

## 2.2 Implementation A

A more fine grained model is given by representing each node in the network by a separate process. Individual nodes are specified below as processes *NodeA*. Nodes are described by three parameters:

- a natural number  $i$ : the identification number of the node. This is used to parameterise communications between nodes, and is not changed during the protocol;
- a set of natural numbers  $p$ : the set of node identifiers of potential parents of the node. The initial value is the set of all neighbours, decreasing to either a singleton (containing the parent node) or the empty set (indicating that the node is the elected leader);
- a natural number  $s$ : the current state of the node. We use two state values: 0 corresponds to “still working” and 1 to “finished”. The initial value is 0.

The identification number of nodes has been introduced to aid specification and does not appear in [IEE96]. In reality a device has a number of ports and knows whether or not a port is connected to another node; there is no need for node identifiers.

A node can send and receive messages: an action  $s(i, j, par)$  is the sending of the parent request  $par$  by node  $i$  to node  $j$ , and an action  $r(i, j, par)$  is the receiving of a parent request from node  $i$  by node  $j$ . When the nodes of the network are composed in parallel, these two actions synchronise with each other to produce  $c$  actions. An action  $c(i, j, par)$  is the establishment of a child-parent relation between node  $i$  and node  $j$ , where  $i$  is the child and  $j$  is the parent. In this case, the type  $\mathbb{M}$  of messages has only one element, i.e. the parent request message  $par$ .

We define the set of actions

$$Act = \{s, r, c : \mathbb{N} \times \mathbb{N} \times \mathbb{M}, \text{leader}\}$$

and the communication  $s|r = c$ . There are no other communications defined.

If a node is still active and its set of potential parents is empty, it declares itself leader by the execution of the *leader* action. By definition, nodes in state 1 are equivalent to deadlock. Individual nodes are defined in Figure 1 as processes *NodeA*.

---


$$\begin{aligned} NodeA(i:\mathbb{N}, p:\mathbb{N}Set, s:\mathbb{N}) = \\ \text{leader} \cdot NodeA(i, p, 1) \triangleleft s = 0 \wedge \text{empty}(p) \triangleright \delta + \\ \sum_{j:\mathbb{N}} r(j, i, par) \cdot NodeA(i, p \setminus \{j\}, s) \triangleleft s = 0 \wedge j \in p \triangleright \delta + \\ \sum_{j:\mathbb{N}} s(i, j, par) \cdot NodeA(i, p, 1) \triangleleft s = 0 \wedge p = \{j\} \triangleright \delta. \end{aligned}$$


---

Figure 1: The process *NodeA*

The process  $ImpA(n, P_0)$  is the parallel composition of  $n + 1$  nodes, with  $P_0$  describing the configuration of the network:

$$ImpA(n:\mathbb{N}, P_0:\mathbb{N}SetList) = \partial_H(NodesA(n, P_0)),$$

where  $H = \{s, r\}$  and

$$NodesA(n, P_0) = NodeA(0, P_0[0], 0) \triangleleft n = 0 \triangleright (NodeA(n, P_0[n], 0) \parallel NodesA(n - 1, P_0)).$$

$P_0$  is a list of sets of connections for all nodes indexed by node number; it gives the initial values for the sets of potential parents. Initially all nodes are in state 0.

### 2.3 Implementation B

Implementation A assumed hand-shaking communication between nodes; in reality messages are sent by variations in voltage along wires of various lengths and are therefore not received instantaneously, i.e. they are asynchronous communications. This means a node may ask to be a child of its neighbour, while that neighbour has already sent out a message asking to be *its* child (but the messages have crossed in transmission). That contention has to be resolved, and one node assigned to be the parent and the other the child.

In Implementation B unidirectional one-element buffers are introduced to model communication between nodes; there are two buffers for each pair of nodes. The communication events also become more complex: in addition to the parent requests, nodes must also send acknowledgements (since a node cannot assume its parent request is successful until an acknowledgement is received). Therefore we introduce the acknowledgement message *ack*. Let  $\mathbf{M}$  be the sort of messages with two elements *par* and *ack*.

The parallel composition of all buffers is defined in Figure 2 as the process *Buffers*. The names of actions in this definition may be confusing; for a buffer an *s* action is a *read* action and a  $\bar{s}$  action is a *send* action. This is a consequence of the names used in the specification of nodes defined also in Figure 2.

Again individual nodes of the network are specified by separate processes. The parameters are similar to those for Implementation A, except there are now three more states, and there is an extra parameter: a set of naturals  $c$  that is used to keep track of children that have to be acknowledged.

In state 0, a node receives parent requests setting up the parent-child relationship. When it has received requests from all or all but one of its neighbours, it moves into state 1. In state 1 a node acknowledges its children. A node can leave state 1 by sending a parent request to its only remaining potential parent (if any). Leaf nodes can skip state 1, and go to state 2 immediately. In state 2, if a node has an empty potential parent set it is the leader and it can do a *leader* action. If not, a node waits for an acknowledgement from its parent. In state 2, a node may receive a parent request instead of an acknowledgement from its requested parent; it then moves into state 3, attempting to resolve contention.

In the standard, contention is resolved by waiting a randomly chosen time before checking for a offer to be a child from the other node, and, if there is none, resending its own parent request. There is no time in  $\mu$ CRL so here there is a choice between sending the parent request again and waiting to receive a child request. Note that there is the possibility of an internal loop if the nodes in contention keep sending each other parent requests. Contention is resolved if in the state where both nodes are in state 3, one of the nodes sends a parent request and the other node does not retransmit its own request, but waits to receive the request from the other node. After the contention has been resolved one of the nodes returns to state 1; this node has received a parent request from the other node and it has to acknowledge this new child. The other node moves into state 2 and waits to be acknowledged. State 4 corresponds to finished.

As for Implementation A, there is the special case where  $n = 0$ , i.e. there is only one node in the network. In this case this one node can do the leader action immediately.

An action  $\bar{s}(i, j, par)$  is the sending of a parent request from  $i$  to  $j$ . Through the buffer, the  $\bar{s}$  action is transformed into a  $\bar{r}$  action, synchronising with  $r$  actions in other nodes. An action  $r(j, i, par)$  is therefore the receiving of a parent request from  $j$  by  $i$ . Acknowledgements  $\bar{s}(i, j, ack)$  from  $i$  to  $j$  acknowledge that  $i$  will be  $j$ 's parent.

---


$$\begin{aligned}
Buffer(i:\mathbb{N}, j:\mathbb{N}) &= \sum_{m:\mathbb{M}} s(i, j, m) \cdot \bar{r}(i, j, m) \cdot Buffer(i, j) \\
Buffers(index:\mathbb{N}, n:\mathbb{N}) &= \\
&\quad BList(0, n) \triangleleft index = 0 \triangleright (BList(index, n) \parallel Buffers(index - 1, n)) \\
&\quad BList(row:\mathbb{N}, col:\mathbb{N}) = \\
&\quad \quad Buffer(row, 0) \triangleleft col = 0 \triangleright (Buffer(row, col)) \parallel BList(row, col - 1)) \\
NodeB(i:\mathbb{N}, p:\mathbb{N}Set, c:\mathbb{N}Set, s:\mathbb{N}) &= \\
&\quad leader \cdot NodeB(i, p, c, 4) \triangleleft (s = 0 \vee s = 2) \wedge empty(p) \triangleright \delta + \\
&\quad \sum_{j:\mathbb{N}} r(j, i, par) \cdot NodeB(i, p \setminus \{j\}, c \cup \{j\}, if(singleton(p), 1, 0)) \\
&\quad \quad \triangleleft s = 0 \wedge j \in p \triangleright \delta + \\
&\quad \sum_{j:\mathbb{N}} \bar{s}(i, j, ack) \cdot NodeB(i, p, c \setminus \{j\}, 1) \\
&\quad \quad \triangleleft s = 0 \wedge singleton(p) \wedge j \in c \triangleright \delta + \\
&\quad \sum_{j:\mathbb{N}} \bar{s}(i, j, par) \cdot NodeB(i, p, c, 2) \\
&\quad \quad \triangleleft s = 0 \wedge p = \{j\} \wedge empty(c) \triangleright \delta + \\
&\quad \sum_{j:\mathbb{N}} \bar{s}(i, j, ack) \cdot NodeB(i, p, c \setminus \{j\}, if(empty(p) \wedge singleton(c), 2, 1)) \\
&\quad \quad \triangleleft s = 1 \wedge j \in c \triangleright \delta + \\
&\quad \sum_{j:\mathbb{N}} \bar{s}(i, j, par) \cdot NodeB(i, p, c, 2) \triangleleft s = 1 \wedge p = \{j\} \wedge empty(c) \triangleright \delta + \\
&\quad \sum_{j:\mathbb{N}} r(j, i, ack) \cdot NodeB(i, p, c, 4) \triangleleft s = 2 \wedge p = \{j\} \triangleright \delta + \\
&\quad \sum_{j:\mathbb{N}} r(j, i, par) \cdot NodeB(i, p, c, 3) \triangleleft s = 2 \wedge p = \{j\} \triangleright \delta + \\
&\quad \sum_{j:\mathbb{N}} r(j, i, par) \cdot NodeB(i, p \setminus \{j\}, c \cup \{j\}, 1) \triangleleft s = 3 \wedge p = \{j\} \triangleright \delta + \\
&\quad \sum_{j:\mathbb{N}} \bar{s}(j, i, par) \cdot NodeB(i, p, c, 2) \triangleleft s = 3 \wedge p = \{j\} \triangleright \delta
\end{aligned}$$


---

Figure 2: The processes *NodeB* and *Buffers*

We define the set of actions

$$Act = \{r, \bar{r}, r^*, s, \bar{s}, s^* : \mathbb{N} \times \mathbb{N} \times \mathbb{M}, leader\}$$

and the communications  $r|\bar{r} = r^*$  and  $s|\bar{s} = s^*$ . There are no other communications defined.

Individual nodes *NodeB* are specified in Figure 2. The complete process *ImpB* is the parallel composition of all nodes and buffers. Note that buffers not required for communication will simply not be used because of the requirement for synchronisation between *NodesB* and *Buffers*. We define

$$ImpB(n:\mathbb{N}, P_0:\mathbb{N}SetList) = \partial_H(NodesB(n, P_0) \parallel Buffers(n, n)),$$

where  $H = \{r, \bar{r}, s, \bar{s}\}$  and

$$NodesB(n, P_0) = NodeB(0, P_0[0], \emptyset, 0) \triangleleft n = 0 \triangleright (NodeB(n, P_0[n], \emptyset, 0) \parallel NodesB(n - 1, P_0)).$$

### 3 Correctness

**Cones and Foci** In process algebra it is common to verify the correctness of a description (the implementation) by proving it equivalent in some sense, e.g. with respect to strong bisimulation, to a more abstract specification. When data is introduced to the descriptions proving equivalence is

more complex since data can considerably alter the flow of control in the process. The cones and foci technique of [GS95] addresses this problem. The main idea of this technique is that there are usually many internal events in the implementation, but they are only significant in that they must progress somehow towards producing a visible event which can be matched with a visible event in the specification. A state of the implementation where no internal actions are enabled is called a *focus point*, and there may be several such points in the implementation. In Implementation A a focus comes when the implementation can perform a *leader* action, because the *leader* action is always the last action to be performed. In Implementation B there may be internal actions enabled in states where the *leader* action is enabled, and a focus comes when the *leader* action is the only enabled action. Focus points are characterised by a boolean condition on the data of the process called the *focus condition*. The focus condition is the negation of the condition which allows  $\tau$  actions to occur. The *cone* belonging to a focus point is the part of the state space from which the focus can be reached by internal actions; imagine the transition system forming a cone or funnel pointing towards the focus. There may also be unreachable states in the implementation; these can be excluded by use of a data invariant.

The final element in the technique is a mapping between the data states of the implementation and the data states of the specification. This mapping is surjective, but almost certainly not injective, since the data of the specification is likely to be simpler than that of the implementation. So in this respect we have a refinement, but in terms of actions we have an equivalence.

Equivalence between the two systems can then be shown by proving six *matching criteria* to hold. Informally, these say

1. The implementation must be convergent.
2. Internal actions in the implementation preserve the mapping.
3. If the implementation can do a visible action then so can the specification.
4. If the specification can do a visible action and the focus condition holds, then so can the implementation.
5. The implementation and the specification have the same data on visible actions.
6. If the implementation does a visible action then the mapping is preserved afterwards.

If these six criteria are satisfied then the specification and the implementation can be said to be branching bisimilar under the General Equality Theorem of [GS95] (repeated in the appendix here as Theorem A.1). The general forms of the matching criteria are given in Definition A.3. Given the particular actions, conditions and mapping for a system, the matching criteria can be mechanically derived. Of course, the choice of mapping requires some thought, as does the subsequent proof of the criteria.

In Section 5 we will see that for Implementation B, the procedure is more complicated. In this case contention results in internal loops within the cone (therefore the implementation is not convergent). Fortunately, [GS95] has, in addition to the General Equality Theorem, a version which is extended by notions of progression and fairness to counteract the problem of implementations with internal loops (this is Theorem A.2). Fairness allows that we define convergence with respect to progressing internal actions only, i.e. those which are somehow moving towards the focus point. A measure of progression is defined which allows us to formalise this notion of distance from a focus point. The abstraction from progressing internal actions is obtained by the application of a *pre-abstraction* function. We will use a focus condition and matching criteria relative to this pre-abstraction (Definitions A.4 and A.6).

A requirement of the cones and foci proof method is that the process be defined by a linear equation (Definition A.1). The linearisation of process terms is a common transformation in process algebra. Informally, all operators other than  $\cdot$ ,  $+$  and the conditional are eliminated. The linearisation technique

of [Gro96] provides rules for the transformation in the special case that the process is the parallel composition of similar processes (as in  $ImpA$  and  $NodesB$ ).

**Good Topology** As mentioned earlier, the protocol operates correctly only on tree networks, i.e. assuming the network has a good topology. Networks with loops will cause a timeout in the real protocol, and unconnected nodes will simply be regarded as another network. The property of *GoodTopology* is formalised below.

**Definition 3.1** Given  $n:\mathbb{N}$ , the maximal node identifier in the network, and a list  $P_0:\mathbb{N}SetList$  giving a set of neighbours for all nodes in the network, the conjunction of the following properties is called  $GoodTopology(n, P_0)$ :

- $P_0$  is symmetric:  $\forall i, j. (i \in P_0[j] \Leftrightarrow j \in P_0[i])$ .
- $P_0$  is a tree, i.e. it is a connected graph with no loops.
  - connected: there exists a path<sup>1</sup>  $s$  between every pair of nodes.  
 $\forall k, j \leq n. \exists s = i_0 \dots i_m. (i_0 = k \wedge i_m = j)$
  - no loops :  $\forall i. \neg \exists$  direct path  $s = i_0 i_1 \dots i_m. (i = i_0 \wedge i = i_m)$ .

**Linearisation of the specification** As a preliminary step to applying the cones and foci proof method for either Implementation A or Implementation B, the process  $Spec$  defined in Section 2.1 must be translated into linear form. Additionally, a data parameter must be added on which to base a mapping from the data of process  $ImpA$  or  $ImpB$ . We define

$$L-Spec(b:\mathbb{B}) = leader \cdot L-Spec(f) \lhd b \rhd \delta.$$

Clearly  $L-Spec(t) = Spec$ .

## 4 Correctness of Implementation A

### 4.1 Linearisation

The linearisation of  $ImpA$  is given in Figure 3 as  $L-ImpA$ . For recursive calls of  $L-ImpA$  only those arguments which are updated are given, e.g.  $L-ImpA(1/S[i])$  means replace the  $i$ th element of  $S$  by 1, leaving all other elements as they are. This linearisation can be derived straightforwardly from the definition of individual nodes using the linearisation technique of [Gro96]. We assert

$$ImpA(n, P_0) = L-ImpA(n, P_0, S_0),$$

where  $S_0$  is the list of initial state values for the nodes, so  $\forall i. S_0[i] = 0$ .

### 4.2 Invariants

The proof of correctness also requires an invariant on the data states of the implementation. The invariant  $I(n, P, S)$  is the conjunction of the invariants listed below. These invariants hold in every state that can be reached from the initial state  $(n, P_0, S_0)$ . The variables  $i$  and  $j$  are universally quantified over  $\{0, \dots, n\}$ .

$$I_1 : S[i] = 0 \vee S[i] = 1$$

---

<sup>1</sup>Define *paths*,  $s:\mathbb{N}List = i_0 i_1 \dots i_m$  such that  $\forall k < m. i_{k+1} \in P_0[i_k]$ . *Direct paths* are paths which do not backtrack down an edge already followed (remember  $P_0$  is symmetric). A path  $s$  is direct if  $\forall k < m. (i_k i_{k+1} \in s) \Rightarrow (i_{k+1} i_k \notin s)$ .

---


$$\begin{aligned}
L\text{-}ImpA(n:\mathbb{N}, P:\mathbb{N}\text{SetList}, S:\mathbb{N}\text{List}) = \\
\sum_{i:\mathbb{N}} \text{leader} \cdot L\text{-}ImpA(1/S[i]) \triangleleft S[i] = 0 \wedge \text{empty}(P[i]) \wedge i \leq n \triangleright \delta + \\
\sum_{i,j:\mathbb{N}} c(j, i, \text{par}) \cdot L\text{-}ImpA((P[i] \setminus \{j\})/P[i], 1/S[j]) \\
\triangleleft S[j] = 0 \wedge P[j] = \{i\} \wedge S[i] = 0 \wedge j \in P[i] \wedge \\
i \neq j \wedge i, j \leq n \triangleright \delta
\end{aligned}$$


---

Figure 3: The linearisation of  $ImpA$ 

$$\begin{aligned}
I_2 : j \in P_0[i] \leftrightarrow j \in P[i] \vee i \in P[j] \\
I_3 : j \in P_0[i] \wedge j \notin P[i] \rightarrow S[j] = 1 \\
I_4 : S[i] = 1 \rightarrow \text{singleton}(P[i]) \vee \text{empty}(P[i]) \\
I_5 : j \in P[i] \wedge S[i] = 0 \rightarrow S[j] = 0 \wedge i \in P[j]
\end{aligned}$$

The proofs of these are straightforward, and omitted here.

### 4.3 Some intermediate steps

The linearisation  $L\text{-}ImpA$  is not sufficient to allow us to apply Theorem A.1. A prerequisite for applying the cones and foci technique is that the indices of the sums preceding any visible actions must be the same in both the specification and the implementation; clearly this is not the case. The summation over the node identifiers preceding the *leader* action in  $L\text{-}ImpA$  correctly reflects that any node can be the root, i.e. there are multiple foci. However, it is not important *which* node is the root, only that one is chosen, and the boolean condition guarding the *leader* action in  $L\text{-}ImpA$  ensures that this is the case, summed up in Lemma 4.1. This lemma says that if a node can do the *leader* action, then all other nodes are in state 1. So if a node declares itself leader then it is the first one to do so, and because after this action all nodes will be in state 1, there will be no *leader* action, or any other action, after it.

**Lemma 4.1** (*Uniqueness of Root*)

$$\forall i \leq n. \text{empty}(P[i]) \rightarrow \forall j \leq n. j \neq i \rightarrow S[j] = 1$$

*Proof.* We assume nodes  $i, j \leq n$  such that  $i \neq j \wedge \text{empty}(P[i]) \wedge S[j] = 0$ , and derive a contradiction. By *GoodTopology* there is a path of distinct nodes  $i = i_0 \dots i_m = j$ , such that  $\forall k < m. i_{k+1} \in P_0[i_k]$ . By  $I_2$  and  $\text{empty}(P[i_0])$  we see that  $i_0 \in P[i_1]$ . Then by  $I_3$   $S[i_1] = 1$ , and by  $I_4$   $\text{singleton}(P[i_1])$ . In a similar way we derive for all  $0 < k \leq m$  that  $P[i_k] = \{i_{k-1}\}$  and  $S[i_k] = 1$ . So in particular  $S[j] = 1$ .  $\square$

The information that the root node chosen is unique (once a certain point in the cone is reached) can be exploited to give a new definition of the linearisation of  $ImpA$ . We introduce the function  $pr$  on data states of the implementation, which is the minimal node identifier of nodes in state 0, and if there is no node in state 0 it is defined to be 0:

$$pr(n, P, S) = \text{if}(\exists i \leq n. S[i] = 0, \min(\{i \leq n \mid S[i] = 0\}), 0).$$

By Lemma 4.1, only one node will perform the *leader* action. We see that if a node  $i$  can perform the *leader* action, i.e. if it satisfies  $S[i] = 0 \wedge \text{empty}(P[i])$ , then it will be the value of  $pr$ . So it is safe to eliminate the summation over  $i$  in the first summand of  $L\text{-Imp}A$  by instantiating it with  $\text{pr}(n, P, S)$ . This elimination yields the redefinition of process  $L\text{-Imp}A$  defined in Figure 4. We often write  $\text{pr}$  to denote the value of  $pr$  in the current state.

---


$$\begin{aligned} L\text{-Imp}A(n:\mathbb{N}, P:\mathbb{N}\text{SetList}, S:\mathbb{N}\text{List}) = \\ \text{leader} \cdot L\text{-Imp}A(1/S[pr]) \triangleleft S[pr] = 0 \wedge \text{empty}(P[pr]) \triangleright \delta + \\ \sum_{i,j:\mathbb{N}} c(j, i, \text{par}) \cdot L\text{-Imp}A((P[i] \setminus \{j\})/P[i], 1/S[j]) \\ \triangleleft S[j] = 0 \wedge P[j] = \{i\} \wedge S[i] = 0 \wedge j \in P[i] \wedge i \neq j \wedge i, j \leq n \triangleright \delta \end{aligned}$$


---

Figure 4: The linearisation of  $\text{Imp}A$  (redefined)

#### 4.4 Verification

The theorem to be demonstrated can now be stated as:

**Theorem 4.1** *Under the assumption of  $\text{GoodTopology}(n, P_0)$  and  $I(n, P_0, S_0)$  it holds that*

$$\tau \cdot L\text{-Spec}(\mathbf{t}) = \tau \cdot \tau_{\{\mathbf{c}\}} L\text{-Imp}A(n, P_0, S_0).$$

*In the special case where  $n = 0$  (there is only one node in the network) we have*

$$L\text{-Spec}(\mathbf{t}) = \tau_{\{\mathbf{c}\}} L\text{-Imp}A(n, P_0, S_0).$$

This is a direct instantiation of Theorem A.1 with the initial state, because in the initial state the focus condition (defined below) is true if and only if  $n = 0$ . In order to prove Theorem 4.1 the matching criteria must be satisfied. To show that the matching criteria hold we first define the focus condition and the state mapping for  $\tau_{\{\mathbf{c}\}} L\text{-Imp}A$ . The focus condition  $FC$  is the condition under which no more  $\tau$  steps can be made, i.e. it is the negation of the condition for making a  $\tau$  step:

$$FC(n, P, S) = \forall i, j \leq n. S[i] = 1 \vee P[i] \neq \{j\} \vee S[j] = 1 \vee i \notin P[j] \vee i = j.$$

The state mapping  $h$  is a function mapping data states of the implementation into data states of the simple specification. In this case  $h$  is defined so that it is  $\mathbf{t}$  before the visible *leader* action occurs and  $\mathbf{f}$  afterwards:

$$h(n, P, S) = (S[pr] < 1).$$

Intuitively  $h$  says that as long as the possible root,  $pr$ , introduced in the last section, has not moved to state 1, then the *leader* action has not yet occurred.

**The matching criteria** Given the particulars of  $L\text{-Imp}A$ ,  $L\text{-Spec}$ ,  $FC$  and  $h$ , the matching criteria are mechanically derived from the general forms of Definition A.3. The instantiated matching criteria are stated below, together with the proofs that they hold.

- 1.. The implementation is convergent.

Using the number of nodes  $i$  for which  $S[i] = 0$  as a measure, then each  $\tau$  step decreases that measure by one.

- 2.. In any data state  $d = (n, P, S)$  of the implementation, the execution of an internal step leads to a state with the same  $h$ -image.

Suppose an internal action is possible, i.e. there are nodes  $i, j \leq n$  such that

$$S[i] = 0 \wedge P[i] = \{j\} \wedge S[j] = 0 \wedge i \in P[j] \wedge i \neq j.$$

We see that  $S[pr] = 0$ . We have to show that if we reach a state  $d' = (n, P', S')$  by the communication between nodes  $i$  and  $j$ , then  $S'[pr'] = 0$ , where  $pr'$  is the value of  $pr$  in state  $d'$ . It holds that  $S = S'$  except that  $S'[i] = 1$ . By definition of  $pr$ ,  $pr' \neq i$  because there is at least one node, i.e.  $j$ , with a state value equal to 0.

- 3.. If the implementation can do the *leader* action, then so can the specification:

$$S[pr] = 0 \wedge \text{empty}(P[pr]) \rightarrow S[pr] < 1.$$

Trivial.

- 4.. If the specification can do the *leader* action and the implementation cannot do an internal action, then the implementation must be able to do the *leader* action:

$$S[pr] < 1 \wedge FC \rightarrow S[pr] = 0 \wedge \text{empty}(P[pr]).$$

Assume  $S[pr] < 1 \wedge FC$ . Then trivially  $S[pr] = 0$ . We prove  $\text{empty}(P[pr])$  by assuming  $\neg \text{empty}(P[pr])$  and deriving a contradiction. Let  $i_1 \in P[pr]$ . By I<sub>5</sub> we have  $S[i_1] = 0$  and  $pr \in P[i_1]$ . By  $FC$  we see that  $\neg \text{singleton}(P[i_1])$ , so there is a  $i_2 \neq pr$  in  $P[i_1]$  such that  $S[i_2] = 0$  and  $i_1 \in P[i_2]$ . We see that proceeding in this way we can construct an infinite path  $i_0 i_1 i_2 \dots$ , where  $pr = i_0$ , such that for all  $k$  it holds that  $S[i_k] = 0$ ,  $i_k \in P[i_{k+1}]$  and  $i_k \neq i_{k+2}$ . By I<sub>2</sub> we see that this infinite path is also a path in  $P_0$ . This contradicts *GoodTopology*.

- 5.. The implementation and the specification perform external actions with the same parameter. Trivial; the action *leader* involves no data.

- 6.. After the implementation and the specification perform the *leader* action, the mapping  $h$  still holds: if the implementation can reach data state  $d'$  by the execution of the *leader* action, then  $h(d') = f$ .

Assume  $S[pr] = 0 \wedge \text{empty}(P[pr])$  (the *leader* action can be executed).

Then by Lemma 4.1 we see that all nodes other than  $pr$  are in state 1. We also see that by the execution of the *leader* action the state of the node that is the value of  $pr$  becomes 1. So after the action all nodes are in state 1, so then the value of  $h$  will be  $f$ .

By Theorem A.1 it follows that Theorem 4.1 holds.

## 5 Correctness of Implementation B

In Figure 5 we give a new definition for individual nodes *NodeB*. The definition in Figure 2 is easier to read, but we will use the new definition because it is more compact and therefore easier to reason about. Using  $s = 3 \rightarrow \text{empty}(c)$  and  $s > 0 \rightarrow \text{empty}(p) \vee \text{singleton}(p)$ , that hold in every state reachable from the initial state, it is easy to check that these definitions are equivalent (cf. I<sub>4</sub> and I<sub>8</sub> of Section 5.2).

---


$$\begin{aligned}
NodeB(i:\mathbb{N}, p:\mathbb{N}Set, c:\mathbb{N}Set, s:\mathbb{N}) = \\
& leader \cdot NodeB(i, p, c, 4) \triangleleft (s = 0 \vee s = 2) \wedge empty(p) \triangleright \delta + \\
& \sum_{j:\mathbb{N}} r(j, i, par) \cdot NodeB(i, if(s = 2, p, p \setminus \{j\}), if(s = 2, c, c \cup \{j\}), \\
& \quad if(s = 2, 3, if(singleton(p), 1, 0))) \\
& \quad \triangleleft (s = 0 \vee s = 2 \vee s = 3) \wedge j \in p \triangleright \delta + \\
& \sum_{j:\mathbb{N}} r(j, i, ack) \cdot NodeB(i, p, c, 4) \triangleleft s = 2 \wedge p = \{j\} \triangleright \delta + \\
& \sum_{j:\mathbb{N}} \bar{s}(i, j, par) \cdot NodeB(i, p, c, 2) \\
& \quad \triangleleft (s = 0 \vee s = 1 \vee s = 3) \wedge p = \{j\} \wedge empty(c) \triangleright \delta + \\
& \sum_{j:\mathbb{N}} \bar{s}(i, j, ack) \cdot NodeB(i, p, c \setminus \{j\}, if(empty(p) \wedge singleton(c), 2, 1)) \\
& \quad \triangleleft ((s = 0 \wedge singleton(p)) \vee s = 1) \wedge j \in c \triangleright \delta
\end{aligned}$$


---

Figure 5: New definition of process *NodeB*

### 5.1 Linearisation

The linearisation of the process *Buffers* is the process *L-Buffers* defined in Figure 6. We left out the linearisation of the process *Buffer*. Individual buffers are modelled by the identifiers of their source and target nodes, a natural 0 or 1 giving the *state* of the buffer – where 0 means the buffer is empty and 1 means the buffer is full, and a message value of type  $\mathbb{M}$ . The parameters *BS* and *BM* in the definition of *L-Buffers* are tables containing entries for pairs of naturals: for all naturals *i* and *j*, *BS*[*i*, *j*] of type  $\mathbb{N}$  is the state value of the buffer from *i* to *j* and *BM*[*i*, *j*] of type  $\mathbb{M}$  is the message value of the buffer from *i* to *j*.

---


$$\begin{aligned}
L\text{-}Buffers(n:\mathbb{N}, BS:\mathbb{N}Table, BM:\mathbb{M}Table) = \\
& \sum_{i,j:\mathbb{N}} \sum_{m:\mathbb{M}} s(i, j, m) \cdot L\text{-}Buffers(m/BM[i, j], 1/BS[i, j]) \triangleleft BS[i, j] = 0 \wedge i, j \leq n \triangleright \delta + \\
& \sum_{i,j:\mathbb{N}} \bar{r}(i, j, BM[i, j]) \cdot L\text{-}Buffers(0/BS[i, j]) \triangleleft BS[i, j] = 1 \wedge i, j \leq n \triangleright \delta
\end{aligned}$$


---

Figure 6: The linearisation of *Buffers*

The linearisation of the process *NodesB* is defined in Figure 7 as process *L-NodesB*. Let the initial values of the parameters be such that *GoodTopology*(*n*, *P*<sub>0</sub>) and

$$\forall i, j. C_0[i] = \emptyset \wedge S_0[i] = 0 \wedge BS_0[i, j] = 0 \wedge BM_0[i, j] = ack.$$

We took the initial message values to be acknowledgements for convenience; this is not essential. The implementation *ImpB* is given by

$$ImpB(n:\mathbb{N}, P_0:\mathbb{N}SetList) = \partial_H(L\text{-}NodesB(n, P_0, C_0, S_0) \parallel L\text{-}Buffers(n, BS_0, BM_0)).$$

Linearisation of *ImpB* is the process *L-ImpB* defined in Figure 7.

### 5.2 Invariants

The invariant *I* on data states (*n*, *P*, *C*, *S*, *BS*, *BM*) is the conjunction of the invariants listed below. It holds in every state that is reachable from the initial state. The variables *i* and *j* are universally quantified over  $\{0, \dots, n\}$ .

---


$$\begin{aligned}
L\text{-}NodesB(n:\mathbb{N}, P:\mathbb{N}\text{SetList}, C:\mathbb{N}\text{SetList}, S:\mathbb{N}\text{List}) = & \\
\sum_{i:\mathbb{N}} leader \cdot L\text{-}NodesB(4/S[i]) & \\
\quad \lhd (S[i] = 0 \vee S[i] = 2) \wedge empty(P[i]) \wedge i \leq n \triangleright \delta + & \\
\sum_{i,j:\mathbb{N}} r(j, i, par) \cdot L\text{-}NodesB(if(S[i] = 2, P[i], P[i] \setminus \{j\})/P[i], & \\
\quad if(S[i] = 2, C[i], C[i] \cup \{j\})/C[i], & \\
\quad if(S[i] = 2, 3, if(singleton(P[i]), 1, 0))/S[i]) & \\
\quad \lhd (S[i] = 0 \vee S[i] = 2 \vee S[i] = 3) \wedge j \in P[i] \wedge i, j \leq n \wedge i \neq j \triangleright \delta + & \\
\sum_{i,j:\mathbb{N}} r(j, i, ack) \cdot L\text{-}NodesB(4/S[i]) & \\
\quad \lhd S[i] = 2 \wedge P[i] = \{j\} \wedge i, j \leq n \wedge i \neq j \triangleright \delta + & \\
\sum_{i,j:\mathbb{N}} \bar{s}(i, j, par) \cdot L\text{-}NodesB(2/S[i]) & \\
\quad \lhd (S[i] = 0 \vee S[i] = 1 \vee S[i] = 3) \wedge P[i] = \{j\} \wedge & \\
\quad empty(C[i]) \wedge i, j \leq n \wedge i \neq j \triangleright \delta + & \\
\sum_{i,j:\mathbb{N}} \bar{s}(i, j, ack) \cdot L\text{-}NodesB((C[i] \setminus \{j\})/C[i], & \\
\quad if(empty(P[i]) \wedge singleton(C[i]), 2, 1)/S[i]) & \\
\quad \lhd ((S[i] = 0 \wedge singleton(P[i])) \vee S[i] = 1) \wedge j \in C[i] \wedge i, j \leq n \wedge i \neq j \triangleright \delta & \\
\\
L\text{-}ImpB(n:\mathbb{N}, P:\mathbb{N}\text{SetList}, C:\mathbb{N}\text{SetList}, S:\mathbb{N}\text{List}, BS:\mathbb{N}\text{Table}, BM:\mathbb{M}\text{Table}) = & \\
\sum_{i:\mathbb{N}} leader \cdot L\text{-}ImpB(4/S[i]) & \\
\quad \lhd (S[i] = 0 \vee S[i] = 2) \wedge empty(P[i]) \wedge i \leq n \triangleright \delta + & \\
\sum_{i,j:\mathbb{N}} r^*(j, i, par) \cdot L\text{-}ImpB(if(S[i] = 2, P[i], P[i] \setminus \{j\})/P[i], & \\
\quad if(S[i] = 2, C[i], C[i] \cup \{j\})/C[i], & \\
\quad if(S[i] = 2, 3, if(singleton(P[i]), 1, 0))/S[i], & \\
\quad 0/BS[j, i]) & \\
\quad \lhd (S[i] = 0 \vee S[i] = 2 \vee S[i] = 3) \wedge j \in P[i] \wedge i, j \leq n \wedge i \neq j \wedge & \\
\quad BS[j, i] = 1 \wedge BM[j, i] = par \triangleright \delta + & \\
\sum_{i,j:\mathbb{N}} r^*(j, i, ack) \cdot L\text{-}ImpB(4/S[i], 0/BS[j, i]) & \\
\quad \lhd S[i] = 2 \wedge P[i] = \{j\} \wedge i, j \leq n \wedge i \neq j \wedge & \\
\quad BS[j, i] = 1 \wedge BM[j, i] = ack \triangleright \delta + & \\
\sum_{i,j:\mathbb{N}} s^*(i, j, par) \cdot L\text{-}ImpB(2/S[i], 1/BS[i, j], par/BM[i, j]) & \\
\quad \lhd (S[i] = 0 \vee S[i] = 1 \vee S[i] = 3) \wedge P[i] = \{j\} \wedge empty(C[i]) \wedge & \\
\quad i, j \leq n \wedge i \neq j \wedge BS[i, j] = 0 \triangleright \delta + & \\
\sum_{i,j:\mathbb{N}} s^*(i, j, ack) \cdot L\text{-}ImpB((C[i] \setminus \{j\})/C[i], & \\
\quad if(empty(P[i]) \wedge singleton(C[i]), 2, 1)/S[i], & \\
\quad 1/BS[i, j], ack/BM[i, j]) & \\
\quad \lhd ((S[i] = 0 \wedge singleton(P[i])) \vee S[i] = 1) \wedge j \in C[i] \wedge & \\
\quad i, j \leq n \wedge i \neq j \wedge BS[i, j] = 0 \triangleright \delta &
\end{aligned}$$


---

Figure 7: The linearisations of *NodesB* and *ImpB*

- I<sub>1</sub> :  $S[i] \leq 4$
- I<sub>2</sub> :  $j \in P_0[i] \leftrightarrow j \in P[i] \vee i \in P[j]$
- I<sub>3</sub> :  $S[i] = 0 \wedge \text{empty}(P[i]) \rightarrow \text{empty}(P_0[i])$
- I<sub>4</sub> :  $S[i] > 0 \rightarrow \text{empty}(P[i]) \vee \text{singleton}(P[i])$
- I<sub>5</sub> :  $S[i] = 0 \wedge j \in P[i] \rightarrow (BS[j, i] = 0 \leftrightarrow BM[j, i] = \text{ack})$
- I<sub>6</sub> :  $S[i] \leq 1 \wedge (j \in P[i] \vee j \in C[i]) \rightarrow BS[i, j] = 0 \wedge BM[i, j] = \text{ack}$
- I<sub>7</sub> :  $S[i] = 1 \rightarrow \neg(\text{empty}(P[i]) \wedge \text{empty}(C[i]))$
- I<sub>8</sub> :  $S[i] = 3 \rightarrow \text{empty}(C[i]) \wedge \text{singleton}(P[i])$
- I<sub>9</sub> :  $S[i] = 3 \wedge P[i] = \{j\} \rightarrow BM[j, i] = \text{par}$
- I<sub>10</sub> :  $S[i] = 3 \wedge P[i] = \{j\} \rightarrow P[j] = \{i\} \wedge (S[j] = 2 \vee S[j] = 3)$
- I<sub>11</sub> :  $S[i] > 0 \wedge j \in P_0[i] \rightarrow P[i] = \{j\} \vee (S[j] > 0 \wedge P[j] = \{i\})$
- I<sub>12</sub> :  $S[i] = 4 \wedge P[i] = \{j\} \rightarrow i \notin P[j]$
- I<sub>13</sub> :  $S[i] = 0 \wedge j \in P[i] \rightarrow i \in P[j] \wedge (S[j] = 0 \vee S[j] = 1 \vee (S[j] = 2 \wedge BS[j, i] = 1))$
- I<sub>14</sub> :  $S[i] = 3 \wedge P[i] = \{j\} \wedge S[j] = 3 \rightarrow BS[i, j] = 0$
- I<sub>15</sub> :  $S[i] = 2 \wedge S[j] = 2 \wedge P[i] = \{j\} \wedge P[j] = \{i\} \rightarrow BS[i, j] = 1$
- I<sub>16</sub> :  $S[i] = 2 \wedge S[j] = 3 \wedge P[i] = \{j\} \wedge P[j] = \{i\} \rightarrow (BS[j, i] = 0 \rightarrow BS[i, j] = 1)$

Most of these invariants are easy to check. The last three invariants relate to contention in the system; they are illustrated by the picture in Figure 8. We hope the picture is self-explanatory. It shows nodes  $i$  and  $j$ , and the buffers between them. A thick box indicates a buffer is in state 1, i.e. it holds a message that is to be transmitted.

### 5.3 Some intermediate steps

Linearisation of  $\text{ImpB}$  yields an expression where the summand starting with the external *leader* action is preceded by a summation over the node identifiers. We eliminate this summation in the same way as in Section 4. Here the function *pr* on data states of the implementation is defined as taking the minimum of the set

$$\text{if } (\exists i. \text{empty}(P[i]), \{i \mid \text{empty}(P[i])\}, \{i \mid \neg \exists j. S[j] < S[i]\}),$$

where variables  $i$  and  $j$  are quantified over  $\{0, \dots, n\}$ . We again need a “uniqueness of root” lemma. Lemma 5.1 says that if a node  $i$  can declare itself leader or has declared itself leader, then there cannot be another node that can do the *leader* action. We also see that this  $i$  will then be the value of the function *pr*. Given the function *pr*, the new linearisation of  $\text{ImpB}$  is as presented in Figure 9.

**Lemma 5.1**  $\forall i \leq n. \text{empty}(P[i]) \rightarrow \neg \exists j \leq n. j \neq i \wedge \text{empty}(P[j])$

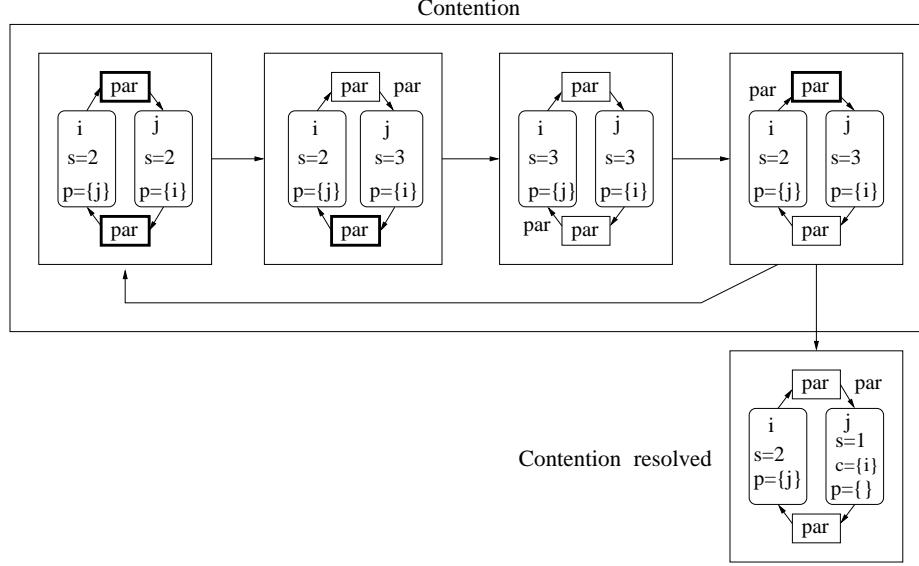


Figure 8: Contention illustrated

*Proof.* By I<sub>1</sub>  $S[i] \leq 4$ . If  $S[i] = 0 \wedge \text{empty}(P[i])$ , then  $\text{empty}(P_0[i])$  by I<sub>3</sub>, and by *GoodTopology* there is only one node in the network, so the lemma trivially holds.

Now assume  $S[i] > 0 \wedge \text{empty}(P[i])$ . Take a  $j \leq n$  such that  $j \neq i$ . By *GoodTopology*, there is a path of distinct nodes  $i_0 i_1 \dots i_m$  with  $i = i_0$ ,  $j = i_m$  and  $\forall k < m. i_{k+1} \in P_0[i_k]$ . By I<sub>11</sub> we see that since  $P[i_0] \neq \{i_1\}$ , it holds that  $S[i_1] > 0$  and  $P[i_1] = \{i_0\}$ . Also by I<sub>11</sub> it now holds that  $P[i_k] = \{i_{k-1}\}$  for all  $0 < k \leq m$ . So  $\neg \text{empty}(P[j])$ .  $\square$

Let *contention* abbreviate the existence of  $i, j \leq n$  such that

$$(S[i] = 2 \vee S[i] = 3) \wedge (S[j] = 2 \vee S[j] = 3) \wedge P[i] = \{j\} \wedge P[j] = \{i\}.$$

The following lemma says that if all nodes are in state 2 or higher and none has an empty parent set, then there must be a case of contention.

**Lemma 5.2**  $S[pr] > 1 \wedge \neg \text{empty}(P[pr]) \rightarrow \text{contention}$

*Proof.* Suppose  $S[pr] > 1$  and  $\neg \text{empty}(P[pr])$ . Since  $\neg \text{empty}(P[pr])$ , there are at least two nodes. By definition of *pr* all nodes *i* have  $S[i] > 1$  and  $\neg \text{empty}(P[i])$ . Then by I<sub>4</sub>  $\text{singleton}(P[i])$  for all nodes *i*. Now supposing there is no pair of nodes that have each other as potential parent leads to a contradiction: Take any node *i*. Construct a path  $i_0 i_1 \dots$  such that  $\forall k. P[i_k] = \{i_{k+1}\}$ . By assumption  $\neg \exists k. P[i_{k+1}] = \{i_k\}$ . Now *GoodTopology* and I<sub>2</sub> tell us  $\forall k. i_k \notin \{i_0, \dots, i_{k-1}\}$ . So this path must visit infinitely many nodes. This contradicts *GoodTopology*.

So there are nodes *i, j* such that  $S[i] > 1 \wedge S[j] > 1 \wedge P[i] = \{j\} \wedge P[j] = \{i\}$ . By I<sub>12</sub> we know that  $S[i] \neq 4$  and  $S[j] \neq 4$ . The lemma follows by I<sub>1</sub>.  $\square$

**Corollary 5.3**  $S[pr] = 4 \rightarrow \text{empty}(P[pr])$

*Proof.* Suppose  $S[pr] = 4$  and  $\neg \text{empty}(P[pr])$ . By the definition of *pr* it holds that  $S[i] = 4$  for all nodes *i*. So  $\neg \text{contention}$ , contradicting Lemma 5.2.  $\square$

---


$$\begin{aligned}
L\text{-}ImpB(n:\mathbb{N}, P:\mathbb{N} \text{SetList}, C:\mathbb{N} \text{SetList}, S:\mathbb{N} \text{List}, LS:\mathbb{N} \text{Table}, LM:\mathbb{M} \text{Table}) = \\
& leader \cdot L\text{-}ImpB(4/S[pr]) \triangleleft (S[pr] = 0 \vee S[pr] = 2) \wedge empty(P[pr]) \triangleright \delta + \\
& \sum_{i,j:\mathbb{N}} r^*(j, i, par) \cdot L\text{-}ImpB(if(S[i] = 2, P[i], P[i] \setminus \{j\})/P[i], \\
& \quad if(S[i] = 2, C[i], C[i] \cup \{j\})/C[i], \\
& \quad if(S[i] = 2, 3, if(singleton(P[i]), 1, 0))/S[i], \\
& \quad 0/LS[j, i]) \\
& \triangleleft (S[i] = 0 \vee S[i] = 2 \vee S[i] = 3) \wedge j \in P[i] \wedge \\
& \quad i, j \leq n \wedge i \neq j \wedge LS[j, i] = 1 \wedge LM[j, i] = par \triangleright \delta + \\
& \sum_{i,j:\mathbb{N}} r^*(j, i, ack) \cdot L\text{-}ImpB(4/S[i], 0/LS[j, i]) \\
& \triangleleft S[i] = 2 \wedge P[i] = \{j\} \wedge i, j \leq n \wedge i \neq j \wedge \\
& \quad LS[j, i] = 1 \wedge LM[j, i] = ack \triangleright \delta + \\
& \sum_{i,j:\mathbb{N}} s^*(i, j, par) \cdot L\text{-}ImpB(2/S[i], 1/LS[i, j], par/LM[i, j]) \\
& \triangleleft (S[i] = 0 \vee S[i] = 1 \vee S[i] = 3) \wedge P[i] = \{j\} \wedge empty(C[i]) \\
& \quad \wedge i, j \leq n \wedge i \neq j \wedge LS[i, j] = 0 \triangleright \delta + \\
& \sum_{i,j:\mathbb{N}} s^*(i, j, ack) \cdot L\text{-}ImpB((C[i] \setminus \{j\})/C[i], \\
& \quad if(empty(P[i]) \wedge singleton(C[i]), 2, 1)/S[i], \\
& \quad 1/LS[i, j], ack/LM[i, j]) \\
& \triangleleft ((S[i] = 0 \wedge singleton(P[i])) \vee S[i] = 1) \wedge j \in C[i] \wedge \\
& \quad i, j \leq n \wedge i \neq j \wedge LS[i, j] = 0 \triangleright \delta
\end{aligned}$$


---

Figure 9: The process  $L\text{-}ImpB$  (redefined)

## 5.4 Verification

The correctness of Implementation B is stated by the following theorem.

**Theorem 5.1** *GoodTopology( $n, P_0$ ) and  $I(n, P_0, C_0, S_0, BS_0, BM_0)$  imply*

$$\tau \cdot L\text{-}Spec(t) = \tau \cdot \tau_{\{r^*, s^*\}} L\text{-}ImpB(n, P_0, C_0, S_0, BS_0, BM_0).$$

We will prove this theorem by application of Theorem A.2 (taking  $Int = \{r^*, s^*\}$  and  $Ext = \{leader\}$ ).

**Pre-abstraction, State Mapping and Focus Condition** As explained in Section 2.3, the process  $ImpB$  is not convergent. Theorem A.2 requires that we distinguish between progressing and non-progressing internal actions. We define a pre-abstraction function on actions and their data, that yields  $f$  on non-progressing internal actions only. In this case, non-progressing actions occur when two nodes that are in contention send each other a parent request. More precisely: if one of the nodes has sent a parent request, and has moved into state 2, then the sending of a parent request by the other node is non-progressing. The pre-abstraction function  $\xi$  is defined by

$$\xi(a) = \begin{cases} \neg(S[i] = 3 \wedge S[j] = 2) & \text{if } a = s^*(i, j, par), \\ t & \text{otherwise.} \end{cases}$$

The focus condition of  $L\text{-}ImpB$  relative to  $\xi$  is the conjunction of the negations of the conditions

for performing a progressing internal action (cf. Definition A.4):

$$\begin{aligned}
FC_\xi(n, P, C, S, BS, BM) = & \forall i, j \leq n. i \neq j \rightarrow \\
& \neg((S[i] = 0 \vee S[i] = 2 \vee S[i] = 3) \wedge j \in P[i] \wedge BS[j, i] = 1 \wedge BM[j, i] = par) \\
& \wedge \\
& \neg(S[i] = 2 \wedge P[i] = \{j\} \wedge BS[j, i] = 1 \wedge BM[j, i] = ack) \\
& \wedge \\
& \neg((S[i] = 0 \vee S[i] = 1 \vee S[i] = 3) \wedge P[i] = \{j\} \wedge empty(C[i]) \wedge \\
& \quad BS[i, j] = 0 \wedge \neg(S[i] = 3 \wedge S[j] = 2)) \\
& \wedge \\
& \neg(((S[i] = 0 \wedge singleton(P[i])) \vee S[i] = 1) \wedge j \in C[i] \wedge BS[i, j] = 0).
\end{aligned}$$

Using Invariants 4–9 we can simplify this formula to

$$\begin{aligned}
FC_\xi(n, P, C, S, BS, BM) = & \forall i, j \leq n. i \neq j \rightarrow \\
& (S[i] = 0 \rightarrow (j \in P[i] \rightarrow BS[j, i] = 0) \wedge \neg singleton(P[i])) \\
& \wedge \quad S[i] \neq 1 \\
& \wedge \quad (S[i] = 2 \wedge P[i] = \{j\} \rightarrow BS[j, i] = 0) \\
& \wedge \quad (S[i] = 3 \wedge P[i] = \{j\} \rightarrow BS[j, i] = 0 \wedge (BS[i, j] = 1 \vee S[j] = 2)).
\end{aligned}$$

We define a state mapping  $h$  from data states of the implementation to data states of the specification. As before, this mapping is only concerned with values of states:

$$h(n, P, C, S, BS, BM) = (S[pr] < 4).$$

Before we prove the matching criteria, we add the following lemma.

**Lemma 5.4** *contention*  $\rightarrow \neg FC_\xi$

*Proof.* Suppose *contention*. So there are nodes  $i, j \leq n$  such that

$$(S[i] = 2 \vee S[i] = 3) \wedge (S[j] = 2 \vee S[j] = 3) \wedge P[i] = \{j\} \wedge P[j] = \{i\}.$$

If one of these nodes, say  $i'$  – call the other node  $j'$ , is in state 2, then we distinguish cases

- $BS[j', i'] = 1$ . This contradicts the third conjunct of  $FC_\xi$ .
- $BS[j', i'] = 0$ . Now by I<sub>15</sub> it must be the case that  $S[j'] = 3$ . By I<sub>16</sub> we see that  $BS[i', j'] = 1$ . This contradicts the last conjunct of  $FC_\xi$ .

So both nodes are in state 3. Then by I<sub>14</sub> it holds that  $BS[i', j'] = BS[j', i'] = 0$ . This contradicts the last conjunct of  $FC_\xi$ .  $\square$

**The Matching Criteria** We instantiate Definition A.6 with the processes *L-ImpB* and *L-Spec*, the state mapping  $h$  and the pre-abstraction  $\xi$ .

1.. The process *L-ImpB* is convergent w.r.t.  $\xi$ .

Let  $rq$  be  $\sum_{i \leq n} |P[i]|$ ;  $ac$  be  $\sum_{i \leq n} |C[i]|$ ;  $s_k$  be the number of nodes in state  $k$ ; and  $l_2$  be the number of requests sent to nodes in state 2, but not received yet. In other words: the number of lines such that its state equals 1 and the receiving node is in state 2.

We define the following measure on data states:

$$\text{Measure} = \langle rq, ac, s_0, s_1, l_2, s_3, s_2 \rangle.$$

Let  $\prec$  be the lexicographical ordering on  $\mathbb{N}^7$ . Now  $\prec$  is a well-founded ordering on the data states of  $L\text{-ImpB}$  such that the measure decreases at every execution of a progressing internal step.

- 2.. In any state  $d$  of the implementation, the execution of an internal step leads to a state with the same  $h$ -image.

Suppose  $S[pr] < 4$ . The only internal action that can change the state of a node  $i$  to 4, is the receiving of an acknowledgement by  $i$ , where  $S[i] = 2$  and  $\text{singleton}(P[i])$ .

Suppose in the state  $d'$  reached by this action,  $i$  becomes the value of  $pr$ , then  $S'[pr'] = 4 \wedge \text{singleton}(P'[pr'])$ . This contradicts Corollary 5.3.

So in every state  $d'$  reachable by an internal action  $S'[pr'] < 4$ .

Suppose  $S[pr] \not< 4$ . By I<sub>1</sub> and Corollary 5.3 it holds that  $\text{empty}(P[pr])$ . Now we see by Lemma 5.1 that  $pr$  will keep the same value.

- 3.. If the implementation can do the *leader* action, then so can the specification:

$$(S[pr] = 0 \vee S[pr] = 2) \wedge \text{empty}(P[pr]) \rightarrow S[pr] < 4.$$

Trivial.

- 4.. If the specification can do the *leader* action and the implementation cannot do a progressing internal action, then the implementation must be able to do the *leader* action:

$$FC_\xi \wedge S[pr] < 4 \rightarrow (S[pr] = 0 \vee S[pr] = 2) \wedge \text{empty}(P[pr]).$$

Suppose  $FC_\xi$  and  $S[pr] < 4$ .  $S[pr] \neq 1$  by  $FC_\xi$ . If  $S[pr] = 3$ , then we have by I<sub>8</sub> and I<sub>10</sub> that *contention*, contradicting the assumption  $FC_\xi$  by Lemma 5.4.

So  $(S[pr] = 0 \vee S[pr] = 2)$ . We have to show  $\text{empty}(P[pr])$ . We distinguish cases  $S[pr] = 0$  and  $S[pr] = 2$  and show that the assumption  $\neg\text{empty}(P[pr])$  leads to a contradiction.

- $S[pr] = 0$ . Assume  $\neg\text{empty}(P[pr])$ . Let  $pr = i_0$  and  $i_1 \in P[i_0]$ . By I<sub>13</sub> we can make the following case distinction, where  $S[i_1] \neq 1$  by  $FC_\xi$ :

$$S[i_1] = 0 \text{ or } S[i_1] = 2 \wedge BS[i_1, i_0] = 1.$$

In the second case  $\neg FC_\xi$  because  $\neg(i_1 \in P[i_0] \rightarrow BS[i_1, i_0] = 0)$  and  $S[i_0] = 0$ . Contradiction.

In the first case we see by  $FC_\xi$  that  $\neg\text{singleton}(P[i_1])$ , so there is a  $i_2 \neq i_0$  in  $P[i_1]$ . We can repeat the argument above for  $i_1$  and  $i_2$ . But we cannot construct an infinite path  $i_0 i_1 i_2 \dots$  where  $\forall k. S[i_k] = 0 \wedge i_{k+1} \in P[i_k] \wedge i_k \neq i_{k+2}$ , as this would violate *GoodTopology* by I<sub>2</sub>. So for some  $k$  we get  $S[i_k] = 0$  and  $\neg(i_{k+1} \in P[i_k] \rightarrow BS[i_{k+1}, i_k] = 0)$ , contradicting  $FC_\xi$  as above.

- $S[pr] = 2$ . Suppose  $\neg\text{empty}(P[pr])$ . Then we find  $\neg FC_\xi$  by Lemma 5.2 and Lemma 5.4. Contradiction.

- 5.. The implementation and the specification perform external actions with the same parameter. Trivial; the action *leader* involves no data.

- 6.. If from a data state  $d$ , the implementation reaches state  $d'$  by the execution of the *leader* action, then  $h(d') = f$ .

We see by Lemma 5.1 that the value of  $pr$  will be the same for  $d$  and  $d'$ . It holds that  $S = S'$  except that  $S'[pr] = 4$ . So  $h(d') = 4 < 4$ , which is false.

Now Theorem 5.1 follows by Theorem A.2.

## 6 Conclusions

We have described the tree identify protocol of the 1394 multimedia serial bus. This was an exercise in specification using  $\mu$ CRL and in verification using the cones and foci technique. While no errors were identified in this view of the system, the exercise has been worthwhile for a number of reasons.

One of our original goals was to “test” the verification technique. We mentioned at the beginning that uptake of verification techniques is often slow due to their complexity. The cones and foci technique has a simple and appealing principle at its heart, and provides a useful structure for the verification, but, as has been seen here, is complex to apply. In particular it relies on expertise in the domain, experience in applying the technique to other examples, and some creativity! This is true of many formal methods.

To aid the verification process it is essential to have good tool support. It should be straightforward to automate parts of the technique of [GS95] used here. In particular, the initial linearisation can be generated automatically, and some development in this area is underway. In fact, computer checked proofs using this technique are described in [KS96]. Note, however, that in the study described here the proof process fed back into the description, in that it was impossible to prove the matching criteria held with the original linearisation of *ImpA*. At that point experience and creativity stepped in and the function  $pr$  was introduced, altering the description of the system and therefore the matching criteria and making the proof possible.

The matching criteria can be automatically generated given the linear specification and implementation, and the state mapping  $h$ . Automation of this and linearisation would leave the verifier free to consider the more tricky questions of the definition of  $h$  and the proofs of the matching criteria. Several proof assistants exist which could be used to computer check such proofs, eliminating the possibility of manually introduced errors. If a more powerful tool such as HOL [GM93] were used then it may also be possible to use higher level tactics to aid the proof process. An interesting problem might be to examine a number of case studies using this verification technique to try to extract some general principles which could be coded in some specialised tactics. In order for this to be possible, a number of studies must be carried out.

Our second achievement is that our study is one example, and adds to the body of experience in applying formal methods; however, at present there are too few examples of the application of [GS95] to allow us to draw any useful conclusions. From the limited set of examples available, we note that the verification of a distributed summation algorithm presented in [GS96] does have similar features (the use of similar processes to describe the system, state-based descriptions, the use of the state parameter to define the mapping function, a simple boolean in the specification and an invariant on the topology of the network). With more case studies it may turn out that these are all common features of specification and verification of distributed systems in  $\mu$ CRL.

This proof technique compares favourably with earlier proofs in  $\mu$ CRL, e.g. [GK95, FGK97], which relied on much lower level proof rules (the usual rules for manipulating process algebra expressions), although we note that the proof given in [FGK97] contains some similar features to the specifications here and in [GS96] (state based specification, n similar processes). The cones and foci technique allows the verifier to concentrate on features of the data, and the structure of the proof technique takes care of the process algebra part.

This proof technique also contrasts with the approaches of [GM97] in which automated proofs of branching bisimulation are carried out using the CADP toolbox, and [SM97] which again uses

the CADP toolbox, but this time to check the validity of modal formulae with respect to labelled transition systems generated from the descriptions. In both cases the size of the system must be restricted in order to allow automated checking. These may then be useful as a prototype stage; automated verification on a small number of nodes, followed by assisted verification on a bounded but undetermined number of nodes using techniques such as cones and foci.

## Acknowledgements

Thanks are due to Jan Friso Groote, who instigated this case study, for many helpful discussions regarding the application of the cones and foci verification technique. Thanks also to Judi Romijn and David Griffioen for discussions regarding the operation of the 1394 tree identify protocol. The first author thanks the Programming Research Group at the University of Amsterdam, EXPRESS project partners for providing a pleasant working environment, and the EC HCM Fellowship scheme for funding her visit. The second author was supported by the Netherlands Organization for Scientific Research (NWO) under contract SION-2854/612-61-002.

## References

- [BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.
- [DGRV97] M.C.A. Devillers, W.O.D. Griffioen, J.M.T Romijn, and F.W. Vaandrager. Verification of a leader election protocol — formal methods applied to IEEE 1394. Technical report, Computing Science Institute, University of Nijmegen, December 1997.
- [FGK<sup>+</sup>96] J-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP (CAESAR/ALDEBARAN Development Package): A protocol validation and verification toolbox. In R. Alur and T.A. Henzinger, editors, *Proceedings of CAV'96*, number 1102 in LNCS, pages 437–440. Springer-Verlag, 1996.
- [FGK97] L. Fredlund, J.F. Groote, and H. Korver. Formal verification of a leader election protocol in process algebra. *Theoretical Computer Science*, 177(2):237–440, 1997.
- [GK95] J.F. Groote and H. Korver. Correctness proof of the bakery protocol in  $\mu$ CRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes '94*, Workshops in Computing, pages 63–86. Springer-Verlag, 1995.
- [GM93] M.J.C. Gordon and T.F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.
- [GM97] H. Garavel and L. Mounier. Specification and verification of various distributed leader election algorithms for unidirectional ring networks. *Science of Computer Programming*, 29(1–2):171–197, 1997.
- [GP95] J.F. Groote and A. Ponse. The syntax and semantics of  $\mu$ CRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes '94*, Workshops in Computing. Springer-Verlag, 1995.
- [Gro96] J.F. Groote. A note on  $n$  similar parallel processes. Technical Report CS-R9626, Centrum voor Wiskunde en Informatica, Amsterdam, 1996.
- [GS95] J.F. Groote and J. Springintveld. Focus points and convergent process operators. Technical Report 142, University of Utrecht, Logic Group Preprint Series, 1995.

- [GS96] J.F. Groote and J. Springintveld. Algebraic verification of a distributed summation algorithm. Technical Report CS-R9640, Centrum voor Wiskunde en Informatica, Amsterdam, 1996.
- [IEE96] IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Std 1394-1995, August 1996.
- [KS96] H.P. Korver and M.P.A. Sellink. On automating process algebra proofs. In V. Atalay et al, editor, *Proceedings of the 11-th International Symposium on Computer and Information Sciences, ISCIS XI Antalya, Turkey*, volume 2, pages 815–826, 1996.
- [Lut97] S.P. Luttik. Description and formal specification of the link layer of P1394. Technical Report SEN-R9706, Centrum voor Wiskunde en Informatica, Amsterdam, 1997.
- [SM97] M. Sighireanu and R. Mateescu. Validation of the link layer protocol of the IEEE-1394 serial bus (FireWire): an experiment with E-LOTOS. Technical Report 3172, INRIA, 1997.

## A Theorems and Definitions

We repeat here the most important definitions and theorems from [GS95]. For the formulation we rely in part on the appendix of [GS96].

### A.1 General Equality Theorem

**Definition A.1** Let  $A \subseteq \text{Act} \cup \{\tau\}$  be a finite set of actions, and let  $D$  be a data type. A *linear process equation* (LPE) over  $\text{Act}$  and  $D$  is an equation of the form

$$X(d:D) = \sum_{a \in A} \sum_{e:E_a} a(f_a(d, e)) \cdot X(g_a(d, e)) \triangleleft b_a(d, e) \triangleright \delta$$

for some data types  $E_a$ ,  $D_a$ , and functions  $f_a:D \rightarrow E_a \rightarrow D_a$ ,  $g_a:D \rightarrow E_a \rightarrow D$ ,  $b_a:D \rightarrow E_a \rightarrow \mathbb{B}$ . (We assume that  $\tau$  has no parameter.)

A summand  $a(f_a(d, e)) \cdot X(g_a(d, e)) \triangleleft b_a(d, e) \triangleright \delta$  means that if for some  $e$  of type  $E_a$  the guard  $b_a(d, e)$  is satisfied, the action  $a$  can be performed with parameter  $f_a(d, e)$ , followed by a recursive call of  $X$  with new value  $g_a(d, e)$ . The main feature of LPEs is that for each action  $a$  there is at most one summand in the alternative composition. Note that therefore the definition of process  $L\text{-Imp}B$  in Figure 9 does not directly fit into this format. We made sure that theorems were applied correctly.

**Definition A.2** An LPE  $X$  written as in Definition A.1 is called *convergent* if it does not admit infinite  $\tau$ -paths, i.e. there is a well-founded ordering  $<$  on  $D$  such that for all  $e:E_\tau$  and  $d:D$  we have that  $b_\tau(d, e)$  implies  $g_\tau(d, e) < d$ .

An *invariant* of an LPE  $X$  written as in Definition A.1 is a function  $I:D \rightarrow \mathbb{B}$  such that for all  $a \in A$ ,  $e:E_a$ , and  $d:D$  we have  $b_a(d, e) \wedge I(d) \rightarrow I(g_a(d, e))$ .

**Definition A.3** Let  $X$  and  $Y$  be LPEs given as follows:

$$\begin{aligned} X(d:D_X) &= \sum_{a \in A} \sum_{e:E_a} a(f_a(d, e)) \cdot X(g_a(d, e)) \triangleleft b_a(d, e) \triangleright \delta \\ Y(d:D_Y) &= \sum_{a \in A \setminus \{\tau\}} \sum_{e:E_a} a(f'_a(d, e)) \cdot Y(g'_a(d, e)) \triangleleft b'_a(d, e) \triangleright \delta \end{aligned}$$

Let  $FC_X$  be a formula over  $d:D_X$  describing exactly the states of  $X$  from which no  $\tau$ -action is enabled (i.e. equivalent to  $\neg \exists e:E_\tau b_\tau(d, e)$ ). Let  $h:D_X \rightarrow D_Y$  be a state mapping. The following 6 conditions are called the *matching criteria* and their conjunction is denoted by  $C_{X,Y,h}(d)$ .

1.  $X$  is convergent
2.  $\forall e:E_\tau (b_\tau(d, e) \rightarrow h(d) = h(g_\tau(d, e)))$
3.  $\forall a \in A \setminus \{\tau\} \forall e:E_a (b_a(d, e) \rightarrow b'_a(h(d), e))$
4.  $\forall a \in A \setminus \{\tau\} \forall e:E_a (FC_X(d) \wedge b'_a(h(d), e) \rightarrow b_a(d, e))$
5.  $\forall a \in A \setminus \{\tau\} \forall e:E_a (b_a(d, e) \rightarrow f_a(d, e) = f'_a(h(d), e))$
6.  $\forall a \in A \setminus \{\tau\} \forall e:E_a (b_a(d, e) \rightarrow h(g_a(d, e)) = g'_a(h(d), e))$

**Theorem A.1 (General Equality Theorem)** *Let  $X$ ,  $Y$ ,  $FC_X$ , and  $h$  be as above. Suppose  $I$  is an invariant of  $X$  and, for all  $d:D_X$ ,  $I(d) \rightarrow C_{X,Y,h}(d)$ . Assume that  $r$  and  $q$  are solutions of  $X$  and  $Y$ , respectively, then*

$$\forall d:D_X I(d) \rightarrow r(d) \triangleleft FC_X(d) \triangleright \tau r(d) = q(h(d)) \triangleleft FC_X(d) \triangleright \tau q(h(d)).$$

## A.2 Abstraction and idle loops

Let  $X$  and  $Y$  be LPEs given as follows:

$$\begin{aligned} X(d:D_X) &= \sum_{a \in Ext \cup Int \cup \{\tau\}} \sum_{e:E_a} a(f_a(d, e)) \cdot X(g_a(d, e)) \triangleleft b_a(d, e) \triangleright \delta \\ Y(d:D_Y) &= \sum_{a \in Ext} \sum_{e:E_a} a(f'_a(d, e)) \cdot Y(g'_a(d, e)) \triangleleft b'_a(d, e) \triangleright \delta \end{aligned}$$

where  $Ext$ ,  $Int$  and  $\{\tau\}$  are mutually disjoint.

**Definition A.4** Let  $\xi$  be a pre-abstraction function. The *focus condition* of  $X$  relative to  $\xi$  is defined by:

$$FC_{X,Int,\xi}(d) = \forall a \in Int \cup \{\tau\} \forall e:E_a \neg(b_a(d, e) \wedge \xi(a)(d, e))$$

**Definition A.5**  $X$  is *convergent w.r.t.  $\xi$*  iff there is a well-founded ordering  $<$  on  $D_X$  such that for all  $a \in Int \cup \{\tau\}$ ,  $d:D_X$  and all  $e:E_a$  we have that  $b_a(d, e)$  and  $\xi(a)(d, e)$  imply  $g_a(d, e) < d$ .

**Definition A.6** Let  $X, Y$  be as above. Let  $h:D_X \rightarrow D_Y$  and let  $\xi$  be a pre-abstraction function. The following 6 conditions are called the *matching criteria for idle loops* and their conjunction is denoted by  $CI_{X,Y,\xi,h}(d)$ .

1.  $X$  is convergent w.r.t.  $\xi$
2.  $\forall a \in Int \cup \{\tau\} \forall e:E_a (b_a(d, e) \rightarrow h(d) = h(g_a(d, e)))$
3.  $\forall a \in Ext \forall e:E_a (b_a(d, e) \rightarrow b'_a(h(d), e))$
4.  $\forall a \in Ext \forall e:E_a (FC_{X,Int,\xi} \wedge b'_a(h(d), e) \rightarrow b_a(d, e))$
5.  $\forall a \in Ext \forall e:E_a (b_a(d, e) \rightarrow f_a(d, e) = f'_a(h(d), e))$
6.  $\forall a \in Ext \forall e:E_a (b_a(d, e) \rightarrow h(g_a(d, e)) = g'_a(h(d), e))$

**Theorem A.2** Let  $X, Y, \xi$  and  $h$  be as above. Let  $p$  and  $q$  be solutions of  $X$  and  $Y$ , respectively. If  $I$  is an invariant of  $X$  and  $\forall d:D_X (I(d) \rightarrow CI_{X,Y,\xi,h}(d))$ , then

$$\forall d:D_X I(d) \rightarrow \tau\tau_{Int}(p(d)) = \tau q(h(d)).$$