Models and logics for process algebra
van der Zwaag, M.B.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
The Tree Identify Protocol of IEEE 1394

With Carron Shankland

We specify the tree identify protocol of the IEEE 1394 high performance serial multimedia bus at three different levels of detail using \( \mu \text{CRL} \). 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.

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 [53] to a fragment of the software for a high performance serial multimedia bus, the IEEE standard 1394 [58], 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: 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 \( \mu \text{CRL} \) in [65] and using E-LOTOS in [79]. The former is essentially a description only, with five correctness properties stated informally, but not formalized or proved. The exercise of [79] is based on the \( \mu \text{CRL} \) description, adding another layer of the protocol and carrying out the verification suggested, using the tool CADP [33].

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 [31]. Verification is by (manual) manipulation of a number of invariants, phrased in predicate calculus. Also discussed is the mechanization of this verification in the theorem prover PVS.

There are three descriptions of the protocol, written using \( \mu \text{CRL} \) [52], in this paper.

- **Specification.** The external behavior of the protocol is specified as the mere announcement that 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 handshaking.
- **Implementation B.** In this implementation communication between nodes occurs via unidirectional channels (so that 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 [58]. Section 3 gives an informal overview of the cones and foci technique of [53], 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 the implementations A and B have the same behavior with respect to branching bisimulation as the simple specification, therefore showing that these descriptions behave as required, that is, that 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 [53] in general.
2. Description of the Protocol

The descriptions are given in μCRL, which is roughly the process algebra ACP [11] extended with a formal treatment of data. Familiarity is assumed with this formalism; an introduction may be found in [52].

Briefly, the main features of the formalism are as follows: the constant δ represents deadlock, \( p \cdot q \) is the sequential composition of processes \( p \) and \( q \), and \( p + q \) is the alternative composition of \( p \) and \( q \). The process \( \sum_{v:D} p \) behaves as the possibly infinite choice between processes \( p[t/v] \) where \( t \) is any data term of sort \( D \). The parallel composition of processes \( p \) and \( q \) is written \( p \parallel q \). We have booleans with two elements \( \top \) (true) and \( \bot \) (false) and the usual boolean operators. Conditionals are written \( p < b > q \), meaning if boolean condition \( b \) is true, then behave as \( p \), otherwise behave as \( q \). For booleans we assume the following binding conventions: \( \neg \) binds stronger than \( \wedge, \vee \), which bind stronger than \( \rightarrow \).

The abstraction operator \( r \) hides all those actions in the set \( I \), by converting them to silent actions \( \tau \), and the encapsulation operator \( \partial_H \) restricts enabled actions, by renaming actions in the set \( H \) to \( \delta \). We choose \( H \) such that the \( \partial_H \) operator forces enclosed parallel processes to communicate with each other.

The μCRL data definitions used, such as \( Nat, NatList \), and \( NatSetList \), are assumed and not presented here; these are straightforward and examples of the appropriate types or similar may be found in [52, 65].

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 \( NodeA \); their data states are described by the following three parameters.

- An identification number \( i \) for the node. This number is used to parameterize communications between nodes, and is not changed during the protocol.
- A set \( p \) 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).
- The current state \( s \) 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 [58]. 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 \( s(i, j, rq) \) is the sending of the parent request \( rq \) by node \( i \) to node \( j \), and an action \( r(i, j, rq) \) 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 synchronize with each other to produce communication actions: an action \( c(i, j, rq) \) 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 \( Mssg \) of messages has only one element: the parent request message \( rq \).

We define the set of actions

\[
\text{Act} = \{ s, r, c : \text{Nat} \times \text{Nat} \times Mssg, \text{leader} \}
\]

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 by

\[
\text{NodeA}(i : \text{Nat}, p : \text{NatSet}, s : \text{Nat}) = \\
\text{leader} \cdot \text{NodeA}(i, p, 1) \triangleleft s = 0 \land \text{empty}(p) \triangleright \delta \\
+ \sum_{j : \text{Nat}} r(j, i, rq) : \text{NodeA}(i, p \setminus \{j\}, s) \triangleleft s = 0 \land j \in p \triangleright \delta \\
+ \sum_{j : \text{Nat}} s(i, j, rq) : \text{NodeA}(i, p, 1) \triangleleft s = 0 \land p = \{j\} \triangleright \delta.
\]

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

\[
\text{ImpA}(n : \text{Nat}, P_0 : \text{NatSetList}) = \delta_H (\text{NodesA}(n, P_0)),
\]

where \( H = \{ s, r \} \) and

\[
\text{NodesA}(n, P_0) = \text{NodeA}(0, P_0[0], 0) \triangleleft n = 0 \triangleright \\
(\text{NodeA}(n, P_0[n], 0) || \text{NodesA}(n - 1, P_0)).
\]

Here, \( P_0 \) is a list of sets of connections for all nodes, indexed by node number; these are the initial values for the sets of potential parents. Initially all nodes are in state 0.

2.3. Implementation B. Implementation A assumed handshaking communication between nodes; in reality messages are sent by variations in voltage along wires of various lengths and are therefore not received instantaneously, that is, 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

\[
\text{ImpA}(n : \text{Nat}, P_0 : \text{NatSetList}) = \delta_H (\text{NodesA}(n, P_0)),
\]

where \( H = \{ s, r \} \) and
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 actions also become more complex: in addition to the parent requests, nodes must also send acknowledgments (since a node cannot assume its parent request is successful until an acknowledgment is received). Therefore we introduce the acknowledgment message \textit{ok}. Let \textit{Msg} be the sort of messages with the two elements \textit{rq} and \textit{ok}. A buffer transmitting from node \textit{i} to node \textit{j} is defined by

\[ \text{Buffer}(i, j : \text{Nat}) = \sum_{m : \text{Msg}} s(i, j, m) \cdot \tilde{r}(i, j, m) \cdot \text{Buffer}(i, j). \]

The names of actions in this definition may be confusing; for a buffer an \textit{s} action is a \textit{read} action and a \textit{\tilde{r}} action is a \textit{send} action. This is a consequence of the names used in the specification of nodes (defined below).

The process \textit{Buffers}(n) is the parallel composition of all buffers between nodes \textit{i}, \( j \leq n \):

\[ \text{Buffers}(n : \text{Nat}) = X(n, n) \parallel (\text{Buffers}(n - 1) < n > 0 \bowtie \delta), \]

\[ X(k, l : \text{Nat}) = \text{Buffer}(k, l) \parallel \text{Buffer}(l, k) \parallel (X(k, l - 1) < l > 0 \bowtie \delta). \]

Again individual nodes of the network are specified by separate processes. The parameters are similar to those for implementation A, except that there are now three more state values, and there is an extra parameter: a set \textit{c} of naturals 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, then it is the leader and can do a leader action. If not, a node waits for an acknowledgment from its parent. In state 2, a node may receive a parent request instead of an acknowledgment 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. Since the timing of actions cannot be specified in \(\mu\text{CRL}\), we just have 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
The Tree Identify Protocol of IEEE 1394

### Table 1. Definition of NodeB.

<table>
<thead>
<tr>
<th>NodeB(i : Nat, p : NatSet, c : NatSet, s : Nat)</th>
</tr>
</thead>
<tbody>
<tr>
<td>leader · NodeB(i, p, c, 4) &lt; s = 0 \lor s = 2 \land empty(p) \triangleright \delta</td>
</tr>
<tr>
<td>+ \sum_{j : Nat} r(j, i, rq) · NodeB(i, p {j}, c \cup {j}, if{singleton(p), 1, 0})</td>
</tr>
<tr>
<td>\langle s = 0 \land j \in p \triangleright \delta</td>
</tr>
<tr>
<td>+ \sum_{j : Nat} \tilde{s}(i, j, ok) · NodeB(i, p, c {j}, 1)</td>
</tr>
<tr>
<td>\langle s = 0 \land singleton(p) \land j \in c \triangleright \delta</td>
</tr>
<tr>
<td>+ \sum_{j : Nat} \tilde{s}(i, j, rq) · NodeB(i, p, c, 2) &lt; s = 0 \land p = {j} \land empty(c) \triangleright \delta</td>
</tr>
<tr>
<td>+ \sum_{j : Nat} \tilde{s}(i, j, ok) · NodeB(i, p, c {j}, if{empty(p) \land singleton(c), 2, 1})</td>
</tr>
<tr>
<td>\langle s = 1 \land j \in c \triangleright \delta</td>
</tr>
<tr>
<td>+ \sum_{j : Nat} \tilde{s}(i, j, rq) · NodeB(i, p, c, 2) &lt; s = 1 \land p = {j} \land empty(c) \triangleright \delta</td>
</tr>
<tr>
<td>+ \sum_{j : Nat} r(j, i, ok) · NodeB(i, p, c, 4) &lt; s = 2 \land p = {j} \triangleright \delta</td>
</tr>
<tr>
<td>+ \sum_{j : Nat} r(j, i, rq) · NodeB(i, p, c, 3) &lt; s = 2 \land p = {j} \triangleright \delta</td>
</tr>
<tr>
<td>+ \sum_{j : Nat} r(j, i, rq) · NodeB(i, p {j}, c \cup {j}, 1) &lt; s = 3 \land p = {j} \triangleright \delta</td>
</tr>
<tr>
<td>+ \sum_{j : Nat} \tilde{s}(j, i, rq) · NodeB(i, p, c, 2) &lt; s = 3 \land p = {j} \triangleright \delta</td>
</tr>
</tbody>
</table>

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 \( n = 0 \), that is, when there is only one node in the network. In this case this one node can do the leader action immediately.

An action \( \tilde{s}(i, j, rq) \) is the sending of a parent request from node \( i \) to node \( j \). Through the buffer, the \( \tilde{s} \) action is transformed into a \( r \) action, synchronizing with \( r \) actions in other nodes. An action \( r(j, i, rq) \) is therefore the receiving of a parent request from \( j \) by \( i \). Acknowledgments \( \tilde{s}(i, j, ok) \) from \( i \) to \( j \) acknowledge that \( i \) will be \( j \)'s parent.

We define the set of actions

\[
Act = \{ r, \tilde{r}, r, s, \tilde{s}, \tilde{s} : \text{Nat} \times \text{Nat} \times \text{Msg}, \text{leader} \}
\]

and the communications \( r | \tilde{r} = \tilde{r} \) and \( s | \tilde{s} = \tilde{s} \). There are no other communications defined.

Individual nodes NodeB are specified in Table 1. 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 synchronization between NodesB and Buffers. We define

\[
ImpB(n : \text{Nat}, P_0 : \text{NatSetList}) = \partial_H (\text{NodesB}(n, P_0) || \text{Buffers}(n)),
\]
where $H = \{r, \bar{r}, s, \bar{s}\}$ and

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

$$(NodeB(n, P_0[n], \emptyset, 0) \ | \ | NodesB(n - 1, P_0)).$$

3. 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 [53] 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 characterized 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 can be phrased as follows.

(i) The implementation must be convergent.

(ii) Internal actions in the implementation preserve the mapping.

(iii) If the implementation can do a visible action then so can the specification.

(iv) If the specification can do a visible action and the focus condition holds, then so can the implementation.

(v) The implementation and the specification have the same data on visible actions.
(vi) 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 so-called general equality theorem of [53] (repeated in the appendix here as Theorem 7.1). The general forms of the matching criteria are given in Definition 7.3. Given the particular actions, conditions and state mapping for a system, the matching criteria can be mechanically derived. Of course, the choice of the state 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, [53] 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 7.2). Fairness allows that we define convergence with respect to progressing internal actions only, that is, to those which are somehow moving towards a focus point. A measure of progression is defined which allows us to formalize 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 7.4 and 7.6).

A requirement of the cones and foci proof method is that the process must be defined by a linear equation (Definition 7.1). The linearization of process terms is a common transformation in process algebra. Informally, all operators other than \( \cdot, + \) and the conditional are eliminated. The linearization technique of [46] provides rules for the transformation in the special case that the process is the parallel composition of similar processes (as in NodesA and NodesB).

**Preliminary Definitions.** We introduce some preliminary notions for the correctness proofs.

- **Good Topology.** As mentioned earlier, the protocol operates correctly only on tree networks, that is, assuming the network has a good topology. Networks with loops will cause a time-out in the real protocol, and unconnected nodes will simply be regarded as another network. We shall describe a network by the number of nodes \((n + 1)\) and by a list \(P_0 : \text{NatSetList}\) describing the configuration of the network. For example, there is an edge between nodes \(i\) and \(j\) if \(j \in P_0[i]\). We shall assume that \(P_0\) describes a network with a good topology, that is, that edges are symmetric \((i \in P_0[j]\) implies \(j \in P_0[i]\), for all \(i, j \leq n\), and the network is a connected graph with no loops. We shall refer to this assumption by GoodTopology\((n, P_0)\).
4. Correctness of Implementation A

- **Linearization of the Specification.** As a preliminary step to applying the cones and foci proof method, the specification process (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 the implementations. We define

$$LSpec(b : \text{Bool}) = \text{leader} \cdot LSpec(\perp) < b \triangleright \delta.$$  

Clearly $$LSpec(\top) = \text{Spec}.$$

- **Notation.** We may abbreviate a term of the form  

$$\sum_{i : \text{Nat}} x < b \land i \leq n \triangleright \delta$$  

as  

$$\sum_{i \leq n} x < b \triangleright \delta.$$ 

4. Correctness of Implementation A

In this section we prove that implementation A is correct using the methodology outlined in the previous section. We start with the linearization of the implementation that we defined in Section 2.2; the linearization is given here as the process $$\text{LImpA}$$:

$$\text{LImpA}(n : \text{Nat}, P : \text{NatSetList}, S : \text{NatList}) =$$  

$$\sum_{i \leq n} \text{leader} \cdot \text{LImpA}(1/S[i]) < S[i] = 0 \land \text{empty}(P[i]) \triangleright \delta$$  

$$+ \sum_{i,j \leq n} \varepsilon(j, i, rq) \cdot \text{LImpA}((P[i] \setminus \{j\})/P[i], 1/S[j])$$  

$$< S[j] = 0 \land P[j] = \{i\} \land S[i] = 0 \land j \in P[i] \land i \neq j \triangleright \delta.$$ 

For recursive calls of the process only those parameters which are updated are given, for example, $$\text{LImpA}(1/S[i])$$ means replace the $$i$$th element of $$S$$ by 1, leaving all other elements as they are. This linearization can be derived straightforwardly from the definition of individual nodes using the linearization technique of [46]. We assert

$$\text{ImpA}(n, P_0) = \text{LImpA}(n, P_0, S_0),$$

where $$S_0$$ is the list of initial state values for the nodes (so $$S_0[i] = 0$$ for all $$i \leq n$$).

4.1. Invariants.** The proof of correctness also requires an invariant on the data states of the implementation. The invariant $$I$$ is defined as

$$I(n, P, S) = \forall i, j \leq n (I_1 \land \cdots \land I_6),$$

where the conjuncts on the right are listed below.

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

1The value of an invariant should be a boolean term. Since the booleans do not have universal quantification in \(\mu\text{CRL}\), this definition is not strictly legal. However, in this case the use of quantification can be avoided at the cost of a much more complicated definition using recursively defined auxiliary functions.
The invariants are satisfied in every state that can be reached from the initial state \((n, P_0, S_0)\) (we assume that \(\text{GoodTopology}(n, P_0)\)). The proofs of the first five invariants are straightforward, and omitted here. We give a proof for the last invariant:

We assume distinct nodes \(i, j \leq n\) such that

\[
empty(P[i]) \land S[j] = 0,
\]

and derive a contradiction. By \(\text{GoodTopology}\) there is a path \(i_0 \ldots i_m\) of distinct nodes with \(i = i_0\) and \(i_m = j\), such that \(i_{k+1} \in P[i_k]\) for all \(k < m\). By \(I_2\) and \(empty(P[i_0])\) we see that \(i_0 \in P[i_1]\). Then \(S[i_1] = 1\) by \(I_3\), and \(\text{singleton}(P[i_1])\) by \(I_4\). 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 \(S[j] = 1\), which yields the required contradiction.

The invariant \(I_6\) says that if a node can do the leader action (if it has an empty set of potential parents), 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.

4.2. An Auxiliary Function. 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.

We introduce a function

\[
pr : \text{Nat} \times \text{NatSetList} \times \text{NatList} \rightarrow \text{Nat}
\]

(for "possible root") on data states of the implementation. This function will allow us to omit the summation over de node identifiers from the summand for the leader action in the definition of \(\text{LimpA}\). We define the function \(pr\) so, that if a node \(i\) can perform the leader action, that is, if it satisfies

\[
S[i] = 0 \land empty(P[i]),
\]

then it will be the value of \(pr\) applied on the current data state. Observe that it follows from \(empty(P[i])\) and invariant \(I_6\) that all other nodes are in state 1. So, we let the value of \(pr\) be one (the largest) of the identifiers of nodes in state 0:

\[
pr(n, P, S) = \text{if}(n = 0, 0, \text{if}(S[n] = 0, n, pr(n - 1, P, S))).
\]
Note that if all nodes are in state 1, then the value of \textit{pr} is 0. Now it is safe to eliminate the summation over \textit{i} in the first summand of the linearization, by instantiating it with \textit{pr}(n, P, S). This elimination yields the following redefinition of the linearized process:

\[
\text{LimpA}(n : \text{Nat}, P : \text{NatSetList}, S : \text{NatList}) =
\]
\[
\begin{align*}
\text{leader} \cdot \text{LimpA}(1/S[pr]) & \triangleright S[pr] = 0 \land \text{empty}(P[pr]) \triangleright \delta \\
+ \sum_{i,j \leq n} c(j, i, rq) \cdot \text{LimpA}((P[i] \setminus \{j\})/P[i], 1/S[j]) & \triangleright S[j] = 0 \land P[j] = \{i\} \land S[i] = 0 \land j \in P[i] \land i \neq j \triangleright \delta.
\end{align*}
\]

We often simply write \textit{pr} to denote the value of \textit{pr} in the current state.

4.3. Verification. The theorem to be demonstrated can now be stated as follows.

\textbf{Theorem 4.1.} If \textit{GoodTopology}(n, P_0) and \textit{I}(n, P_0, S_0), then

\[
\tau \cdot \text{LSpec}(\top) = \tau \cdot \tau_{\textit{c}} \text{LimpA}(n, P_0, S_0).
\]

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

\[
\text{LSpec}(\top) = \tau_{\textit{c}} \text{LimpA}(n, P_0, S_0).
\]

This is a direct instantiation of Theorem 7.1 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 \(\tau_{\textit{c}} \text{LimpA}\). The focus condition is the condition under which no more \(\tau\)-steps can be made. So, it is defined as the negation of the condition for making a \(\tau\)-step:

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

The state mapping \(h\) is a function mapping data states of the implementation into data states of the simple specification. In this case the state mapping is defined so that it is \(T\) before the visible leader action occurs and \(\bot\) afterwards:

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

Intuitively \(h\) says that as long as the possible root, \textit{pr}, introduced in the last section, has not moved to state 1, then the leader action has not yet occurred.

We shall now show that the state mapping indeed satisfies the matching criteria (see Definition 7.3). The instantiated matching criteria are stated below, together with the proofs that they are satisfied.
(i) The implementation is convergent.

Using the number of nodes that are in state 0 as a measure, each \( \tau \)-step decreases that measure by one.

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

Suppose an internal action is possible, that is, that there are nodes \( i, j \leq n \) such that

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

We see that \( S[pr] = 0 \). We have to show that if a state \((n', P', S')\) is reached by the communication between nodes \( i \) and \( j \), then \( S'[pr'] = 0 \), where \( pr' \) is the value of \( pr \) in the new state. Observe 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.

(iii) If the implementation can do the leader action, then so can the specification:

\[
S[pr] = 0 \land empty(P[pr]) \rightarrow S[pr] < 1.
\]

Trivial.

(iv) 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:

\[
S[pr] < 1 \land FC \rightarrow S[pr] = 0 \land empty(P[pr]).
\]

Assume \( S[pr] < 1 \land FC \). Then trivially \( S[pr] = 0 \). We prove \( empty(P[pr]) \) by assuming \( \neg empty(P[pr]) \) and deriving a contradiction. Let \( i_1 \in P[pr] \). By \( I_5 \) we have \( S[i_1] = 0 \) and \( pr \in P[i_1] \). By FC we see that \( \neg singleton(P[i_1]) \), so there is an \( i_2 \neq pr \) 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 \( i_0 i_1 i_2 \ldots \), where \( pr = i_0 \), such that \( S[i_k] = 0 \), \( i_k \in P[i_{k+1}] \) and \( i_k \neq i_{k+2} \), for all \( k \). By \( I_2 \) we see that this infinite path is also a path in \( P_0 \). This contradicts \( GoodTopology \).

(v) The implementation and the specification perform external actions with the same parameter.

Trivial; the action leader involves no data.

(vi) After the implementation and the specification perform the leader action, the mapping \( h \) still holds: if the implementation can reach a data state by the execution of the leader action, then \( h \) maps this state to \( \bot \).

Assume \( S[pr] = 0 \land empty(P[pr]) \) (the leader action can be executed). By \( I_6 \) 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, and then the value of \( h \) will be \( \bot \).
5. Correctness of Implementation B

In Table 2 we give a new definition for individual nodes of the second implementation. The definition in Table 1 is easier to read, but we will use the new definition because it is more compact and therefore easier to reason about. Using $s = 3 \to empty(c)$ and $s > 0 \to empty(p) \lor singleton(p)$, that are satisfied in every state reachable from the initial state, it is easy to check that these definitions are equivalent (cf. the invariants $I_4$ and $I_8$ of Section 5.2).

<table>
<thead>
<tr>
<th>Table 2. New definition of process NodeB.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NodeB(i : Nat, p : NatSet, c : NatSet, s : Nat) =$</td>
</tr>
<tr>
<td>$leader \cdot NodeB(i, p, c, 4) \triangleleft (s = 0 \lor s = 2) \land empty(p) \triangleright \delta$</td>
</tr>
<tr>
<td>$+ \sum_{j : Nat} t(j, i, rq) \cdot NodeB(i, if(s = 2, p, p \setminus {j}),$</td>
</tr>
<tr>
<td>$\quad if(s = 2, c, c \cup {j}),$</td>
</tr>
<tr>
<td>$\quad if(s = 2, 3, if(singleton(p), 1, 0)))$</td>
</tr>
<tr>
<td>$\triangleleft (s = 0 \lor s = 2 \lor s = 3) \land j \in p \triangleright \delta$</td>
</tr>
<tr>
<td>$+ \sum_{j : Nat} t(j, i, ok) \cdot NodeB(i, p, c, 4) \triangleleft s = 2 \land p = {j} \triangleright \delta$</td>
</tr>
<tr>
<td>$+ \sum_{j : Nat} s(i, j, rq) \cdot NodeB(i, p, c, 2)$</td>
</tr>
<tr>
<td>$\triangleleft (s = 0 \lor s = 1 \lor s = 3) \land p = {j} \land empty(c) \triangleright \delta$</td>
</tr>
<tr>
<td>$+ \sum_{j : Nat} s(i, j, ok) \cdot NodeB(i, p, c \setminus {j}, if(\text{empty}(p) \land singleton(c), 1, 2),)$</td>
</tr>
<tr>
<td>$\triangleleft ((s = 0 \land singleton(p)) \lor s = 1) \land j \in c \triangleright \delta$</td>
</tr>
</tbody>
</table>

5.1. Linearization. The linearization of the process Buffers is defined in Table 3 as the process $LBuffers$. We left out the linearization 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 $Mssg$. The parameters $BS$ and $BM$ in the definition of $LBuffers$ are tables containing entries for pairs of naturals: for all naturals $i$ and $j$, the natural $BS[i, j]$ is the state value of the buffer from node $i$ to node $j$ and the message $BM[i, j]$ is the message value of the buffer from node $i$ to node $j$.

The linearization of the process $NodesB$ is defined in Table 3 as process $LNodesB$.

Let the initial values of the parameters be such that $GoodTopology(n, P_0)$ and

$$C_0[i] = \emptyset \land S_0[i] = 0 \land BS_0[i, j] = 0 \land BM_0[i, j] = ok,$$
The Tree Identify Protocol of IEEE 1394

TABLE 3. The linearizations of Buffers and NodesB.

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sum_{i,j \leq n} \sum_{m:MSSG} s(i, j, m) \cdot \text{LBuffers}(m/\text{BM}[i, j], 1/\text{BS}[i, j]) )</td>
<td>( \text{LBuffers}(n : \text{Nat}, BS : \text{NatTable}, BM : \text{MssgTable}) = )</td>
</tr>
<tr>
<td>( \sum_{i,j \leq n} \sum_{m:MSSG} s(i, j, m) \cdot \text{LBuffers}(m/\text{BM}[i, j], 1/\text{BS}[i, j]) )</td>
<td>( \triangleleft BS[i, j] = 0 \triangleright \delta )</td>
</tr>
<tr>
<td>( + \sum_{i,j \leq n} \tau(i, j, BM[i, j]) \cdot \text{LBuffers}(0/BS[i, j]) \triangleleft BS[i, j] = 1 \triangleright \delta )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sum_{i \leq n} \text{leader} \cdot \text{LNodesB}(4/S[i]) \triangleleft (S[i] = 0 \lor S[i] = 2) \land \text{empty}(P[i]) \triangleright \delta )</td>
<td>( \text{LNodesB}(n : \text{Nat}, P : \text{NatSetList}, C : \text{NatSetList}, S : \text{NatList}) = )</td>
</tr>
<tr>
<td>( + \sum_{i,j \leq n} \tau(j, i, rq) \cdot \text{LNodesB}(if(S[i] = 2, P[i], P[i] \setminus {j})/P[i], )</td>
<td></td>
</tr>
<tr>
<td>( \text{if}(S[i] = 2, C[i], C[i] \cup {j})/C[i], )</td>
<td></td>
</tr>
<tr>
<td>( \text{if}(S[i] = 2, 3, (\text{singleton}(P[i]), 1, 0))/S[i]) )</td>
<td>( \triangleleft (S[i] = 0 \lor S[i] = 2 \lor S[i] = 3) \land j \in P[i] \land i \neq j \triangleright \delta )</td>
</tr>
<tr>
<td>( + \sum_{i,j \leq n} \tau(j, i, ok) \cdot \text{LNodesB}(4/S[i]) )</td>
<td></td>
</tr>
<tr>
<td>( \triangleleft S[i] = 2 \land P[i] = {j} \land i \neq j \triangleright \delta )</td>
<td></td>
</tr>
<tr>
<td>( + \sum_{i,j \leq n} \tilde{s}(i, j, rq) \cdot \text{LNodesB}(2/S[i]) )</td>
<td></td>
</tr>
<tr>
<td>( \triangleleft (S[i] = 0 \lor S[i] = 1 \lor S[i] = 3) \land P[i] = {j} \land )</td>
<td></td>
</tr>
<tr>
<td>( \text{empty}(C[i]) \land i \neq j \triangleright \delta )</td>
<td></td>
</tr>
<tr>
<td>( + \sum_{i,j \leq n} \tilde{s}(i, j, ok) \cdot \text{LNodesB}((C[i] \setminus {j})/C[i], )</td>
<td></td>
</tr>
<tr>
<td>( \text{if}(\text{empty}(P[i]) \land \text{singleton}(C[i]), 2, 1)/S[i]) )</td>
<td>( \triangleleft (S[i] = 0 \lor \text{singleton}(P[i])) \lor S[i] = 1 \land j \in C[i] \land i \neq j \triangleright \delta )</td>
</tr>
</tbody>
</table>

for all \( i, j \leq n \). We took the initial message values to be acknowledgments for convenience; this is not essential. The implementation \( \text{ImpB} \) is given by

\[
\text{ImpB}(n : \text{Nat}, P_0 : \text{NatSetList}) =
\partial_H(\text{LNodesB}(n, P_0, C_0, S_0) \parallel \text{LBuffers}(n, BS_0, BM_0)).
\]

The linearization of \( \text{ImpB} \) is the process \( L\text{ImpB} \) defined in Table 4.

5.2. Invariants. The invariant \( I \) on data states is given by\(^2\)

\[
I(n, P, C, S, BS, BM) = \forall i, j \leq n (I_1 \land \cdots \land I_{20}),
\]

where the first 16 conjuncts on the right are listed below. The correctness of this part of the invariant is easy to check. The last four conjuncts are presented in Lemmas 5.1, 5.2, 5.3, and 5.5.

\[
I_1 : S[i] \leq 4
\]

\[
I_2 : j \in P_0[i] \iff j \in P[i] \lor i \in P[j]
\]

\[
I_3 : S[i] = 0 \land \text{empty}(P[i]) \rightarrow \text{empty}(P_0[i])
\]

\[
I_4 : S[i] > 0 \rightarrow \text{empty}(P[i]) \lor \text{singleton}(P[i])
\]

\(^2\text{Cf. footnote 1.}\)
5. Correctness of Implementation B

TABLE 4. The linearization of ImpB.

\[
\begin{align*}
L_{\text{ImpB}}(n : \text{Nat}, P : \text{NatSetList}, C : \text{NatSetList}, S : \text{NatList}, BS : \text{NatTable}, BM : \text{MsgTable}) &= \\
&= \sum_{i \leq n} \text{leader} \cdot L_{\text{ImpB}}(4/S[i]) \land (S[i] = 0 \lor S[i] = 2) \land \text{empty}(P[i]) \lor \delta \\
&+ \sum_{i, j \leq n} \tilde{r}(j, i, rq) \cdot L_{\text{ImpB}}(\text{if}(S[i] = 2, P[i], P[i] \setminus \{j\}) / P[i], \\
&\quad \text{if}(S[i] = 2, C[i], C[i] \cup \{j\}) / C[i], \\
&\quad \text{if}(S[i] = 2, 3, \text{if}(\text{singleton}(P[i]), 1, 0)) / S[i], \\
&\quad 0 / BS[j, i]) \\
&\land (S[i] = 0 \lor S[i] = 2 \lor S[i] = 3) \land j \in P[i] \land i \neq j \land BS[j, i] = 1 \land BM[j, i] = rq \lor \delta \\
&+ \sum_{i, j \leq n} \tilde{s}(i, j, ok) \cdot L_{\text{ImpB}}(4/S[i], 0 / BS[j, i]) \\
&\land (S[i] = 2 \land P[i] = \{j\} \land i \neq j \land BS[j, i] = 1 \land BM[j, i] = ok \lor \delta \\
&+ \sum_{i, j \leq n} \tilde{s}(i, j, ok) \cdot L_{\text{ImpB}}(2/S[i], 1 / BS[i, j], rq / BM[i, j]) \\
&\land (S[i] = 0 \lor S[i] = 1 \lor S[i] = 3) \land P[i] = \{j\} \land \text{empty}(C[i]) \land i \neq j \land BS[i, j] = 0 \lor \delta \\
&+ \sum_{i, j \leq n} \tilde{s}(i, j, ok) \cdot L_{\text{ImpB}}((C[i] \setminus \{j\}) / C[i], \\
&\quad \text{if}(\text{empty}(P[i]) \land \text{singleton}(C[i]), 2, 1) / S[i], \\
&\quad 1 / BS[i, j], ok / BM[i, j]) \\
&\land ((S[i] = 0 \land \text{singleton}(P[i])) \lor S[i] = 1) \land j \in C[i] \land i \neq j \land BS[i, j] = 0 \lor \delta
\end{align*}
\]

\(I_5: S[i] = 0 \land j \in P[i] \rightarrow (BS[j, i] = 0 \leftrightarrow BM[j, i] = ok)\)
\(I_6: S[i] \leq 1 \lor (j \in P[i] \lor j \in C[i]) \rightarrow BS[i, j] = 0 \land BM[i, j] = ok\)
\(I_7: S[i] = 1 \rightarrow \neg(\text{empty}(P[i]) \land \text{empty}(C[i]))\)
\(I_8: S[i] = 3 \rightarrow \text{empty}(C[i]) \land \text{singleton}(P[i])\)
\(I_9: S[i] = 3 \land P[i] = \{j\} \rightarrow BM[j, i] = rq\)
\(I_{10}: S[i] = 3 \land P[i] = \{j\} \rightarrow P[j] = \{i\} \land (S[j] = 2 \lor S[j] = 3)\)
\(I_{11}: S[i] > 0 \land j \in P_0[i] \rightarrow P[i] = \{j\} \land (S[j] > 0 \land P[j] = \{i\})\)
\(I_{12}: S[i] = 4 \land P[i] = \{j\} \rightarrow i \notin P[j]\)
\(I_{13}: S[i] = 0 \land j \in P[i] \rightarrow i \in P[j] \land (S[j] = 0 \lor S[j] = 1 \lor (S[j] = 2 \land BS[j, i] = 1))\)
\(I_{14}: S[i] = 3 \land P[i] = \{j\} \land S[j] = 3 \rightarrow BS[i, j] = 0\)
\(I_{15}: S[i] = 2 \land S[j] = 2 \land P[i] = \{j\} \land P[j] = \{i\} \rightarrow BS[i, j] = 1\)
\(I_{16}: S[i] = 2 \land S[j] = 3 \land P[i] = \{j\} \land P[j] = \{i\} \rightarrow (BS[j, i] = 0 \rightarrow BS[i, j] = 1)\)

The last three conjuncts relate to contention in the system; they are illustrated by the picture in Figure 1. The picture shows nodes \(i\) and \(j\), and the buffers between them.
5.3. An Auxiliary Function and Contention. Linearization of implementation B 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 using an auxiliary function on data states of the implementation, the value of which will be the identifier of the leader at the moment it performs the leader action: it is defined by\(^3\)

\[
pr(n, P, C, S, BS, BM) = \begin{cases} 
\inf\{i \leq n \mid \text{empty}(P[i])\} & \text{if } \exists i \leq n(\text{empty}(P[i])), \\
\inf\{i \leq n \mid \neg\exists j \leq n (S[j] < S[i])\} & \text{otherwise.}
\end{cases}
\]

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 see that this \(i\) will then be the value of the function \(pr\). Given the function \(pr\), the new linearization of implementation B is as presented in Table 5.

**Lemma 5.1.** The formula

\[
\forall i, j \leq n \ (\text{empty}(P[i]) \land i \neq j \rightarrow \neg\text{empty}(P[j])) \quad (I_{17})
\]

is an invariant.

\(^3\)The use of quantifiers in this definition can be avoided at the cost of introducing recursively defined auxiliary functions. The case distinction can easily be written using the function \(\text{if} : \text{Bool} \times \text{Nat} \times \text{Nat} \rightarrow \text{Nat} \).
5. Correctness of Implementation B

TABLE 5. The process $L\text{Imp}B$ (redefined).

\[
L\text{Imp}B(n : \text{Nat}, P : \text{NatSetList}, C : \text{NatSetList}, S : \text{NatList}, \\
BS : \text{NatTable}, BM : \text{MssgTable}) = \\
\text{leader} \cdot L\text{Imp}B(4/S[pr]) \triangleleft (S[pr] = 0 \lor S[pr] = 2) \land \text{empty}(P[pr]) \triangleright \delta \\
+ \sum_{i,j \leq n} \bar{r}(j, i, rq) \cdot L\text{Imp}B(\text{if}(S[i] = 2, P[i], P[i] \setminus \{j\})/P[i], \\
\text{if}(S[i] = 2, C[i], C[i] \cup \{j\})/C[i], \\
\text{if}(S[i] = 2, 3, \text{if}(\text{singleton}(P[i]), 1, 0))/S[i], \\
0/BS[j, i]) \\
\triangleleft (S[i] = 0 \lor S[i] = 2 \lor S[i] = 3) \land j \in P[i] \land \\
i \neq j \land BS[j, i] = 1 \land BM[j, i] = rq \triangleright \delta \\
+ \sum_{i,j \leq n} \tilde{r}(i, j, ok) \cdot L\text{Imp}B(4/S[i], 0/BS(j, i)) \\
\triangleleft S[i] = 2 \land P[i] = \{j\} \land i \neq j \land \\
BS[j, i] = 1 \land BM[j, i] = \text{ok} \triangleright \delta \\
+ \sum_{i,j \leq n} \bar{g}(i, j, rq) \cdot L\text{Imp}B(2/S[i], 1/BS[i, j], rq/BM[i, j]) \\
\triangleleft (S[i] = 0 \lor S[i] = 1 \lor S[i] = 3) \land P[i] = \{j\} \land \\
\text{empty}(C[i]) \land i \neq j \land BS[i, j] = 0 \triangleright \delta \\
+ \sum_{i,j \leq n} \tilde{g}(i, j, ok) \cdot L\text{Imp}B((C[i] \setminus \{j\})/C[i], \\
\text{if}(\text{empty}(P[i]) \land \text{singleton}(C[i]), 2, 1))/S[i], \\
1/BS[i, j], \text{ok}/BM[i, j]) \\
\triangleleft (S[i] = 0 \land \text{singleton}(P[i])) \lor S[i] = 1 \land j \in C[i] \land \\
i \neq j \land BS[i, j] = 0 \triangleright \delta
\]

Proof. Take distinct $i, j \leq n$ with $\text{empty}(P[i])$. By $I_1$ we find that $S[i] \leq 4$. If $S[i] = 0$, then $\text{empty}(P_0[i])$ by $I_3$, and by GoodTopology there is only one node in the network, so the lemma trivially holds.

If $S[i] > 0$, then by GoodTopology there is a path of distinct nodes $i_0 i_1 \ldots i_m$ with $i = i_0$, $j = i_m$ and $i_k+1 \in P_0[i_k]$ for all $k < m$. By $I_{11}$ we see that it follows from $P[i_0] \neq \{i_1\}$ that $S[i_1] > 0$ and $P[i_1] = \{i_0\}$. Also by $I_{11}$, $P[i_k] = \{i_{k-1}\}$ for all $0 < k \leq m$. So $\neg \text{empty}(P[j])$. \hfill \Box

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

\[
(S[i] = 2 \lor S[i] = 3) \land (S[j] = 2 \lor S[j] = 3) \land P[i] = \{j\} \land P[j] = \{i\}.
\]

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. The formula

\[
S[pr] > 1 \land \neg \text{empty}(P[pr]) \rightarrow \text{contention}
\]

is an invariant.

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 \(I_4\) we find that \(\text{singleton}(P[i])\) for all \(i \leq n.\) Now supposing there is no pair of nodes that have each other as potential parent leads to a contradiction: Take any node \(i_0.\) Construct a path \(i_0i_1 \ldots\) such that \(P[i_k] = \{i_{k+1}\}\) for all \(k.\) By assumption there is no \(k\) such that \(P[i_{k+1}] = \{i_k\}.\) Now \(\text{GoodTopology}\) and \(I_2\) tell us \(i_k \notin \{i_0, \ldots, i_{k-1}\}\) for all \(k.\) So this path must visit infinitely many nodes. Contradiction.

So there are nodes \(i, j\) such that \(S[i] > 1\) and \(S[j] > 1\) and \(P[i] = \{j\}\) and \(P[j] = \{i\}.\) By \(I_{12}\) we know that \(S[i] \neq 4\) and \(S[j] \neq 4.\) The lemma follows by invariant \(I_1\).

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

is an invariant.

**Proof.** Suppose that \(S[pr] = 4\) and \(\neg \text{empty}(P[pr]).\) By definition of the function \(\text{pr},\) all nodes are in state 4. Hence, we see that \(\neg \text{contention},\) and this contradicts Lemma 5.2. \(\square\)

### 5.4. Verification

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

**Theorem 5.4.** \(\text{GoodTopology}(n, P_0)\) and \(I(n, P_0, C_0, S_0, BS_0, BM_0)\) imply
\[
\tau \cdot LSpec(\tau) = \tau \cdot \tau_{[\tilde{r}, \tilde{s}]}LImpB(n, P_0, C_0, S_0, BS_0, BM_0).
\]

We prove this theorem by application of Theorem 7.2 (taking \(\text{Int} = \{\tilde{r}, \tilde{s}\}\) and \(\text{Ext} = \{\text{leader}\}); first, we present a pre-abstraction function, the focus condition and a state mapping. Then, we prove that the matching criteria hold for these.

As we have seen, implementation B is not convergent due to the possibility of contention. Application of Theorem 7.2 requires that we distinguish between progressing and non-progressing internal actions. We define a pre-abstraction function on actions and their data, that yields \(\tau\) on 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

\[
\xi(a) = \begin{cases} 
\neg(S[i] = 3 \land S[j] = 2) & \text{if } a = \tilde{s}(i, j, rq), \\
\tau & \text{otherwise.}
\end{cases}
\]
The focus condition of $\text{LLmpB}$ relative to $\xi$ is the conjunction of the negations of the conditions for performing a progressing internal action (see Definition 7.4). Using invariants 4–9 we can simplify this formula to

$$
\begin{align*}
FC_\xi(n, P, C, S, BS, BM) = \forall i, j \leq n. i \neq j \rightarrow \\
& \land S[i] = 0 \rightarrow (j \in P[i] \rightarrow BS[j, i] = 0) \land \neg \text{singleton}(P[i]) \\
& \land S[i] \neq 1 \\
& \land S[i] = 2 \land P[i] = \{j\} \rightarrow BS[j, i] = 0 \\
& \land S[i] = 3 \land P[i] = \{j\} \rightarrow BS[j, i] = 0 \land (BS[i, j] = 1 \lor S[j] = 2).
\end{align*}
$$

(Recall that we let $\land$ bind more strongly than $\rightarrow$.)

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 state values:

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

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

**Lemma 5.5.** The formula

$$
\text{contention} \rightarrow \neg FC_\xi
$$

is an invariant.

**Proof.** Suppose that in a state both $\text{contention}$ and $FC_\xi$ are true. So there are nodes $i, j \leq n$ such that

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

Assume that one of these nodes, say $i$, is in state 2. The value of $BS[j, i]$ can be 0 or 1. If it is 1 then we find a contradiction with the third conjunct of $FC_\xi$. If it is 0, then by $I_{15}$ it must be the case that $S[j] = 3$. By $I_{16}$ we find that also $BS[i, j] = 1$, which contradicts the last conjunct of $FC_\xi$. We conclude that both nodes are in state 3. Then $BS[i, j] = BS[j, i] = 0$ by $I_{14}$. This contradicts the last conjunct of $FC_\xi$. $\square$

We shall now prove that the matching criteria (see Definition 7.6) hold for the processes $\text{LLmpB}$ and $\text{LSpec}$, the state mapping $h$ and the pre-abstraction function $\xi$:

(i) The process $\text{LLmpB}$ is convergent with respect to $\xi$.

For any data state

$$(n, P, C, S, BS, BM),$$

let $Pr$ be $\sum_{i \leq n} |P[i]|$; let $Ac$ be $\sum_{i \leq n} |C[i]|$; let $S_k$ be the number of nodes in state $k$; and let $B$ be the number of requests sent to nodes in state 2, but not received yet. In other words: the number of buffers in state 1 with the receiving node in state 2.
We define the following measure on data states:

$$(Pr, Ac, S_0, S_1, B, S_3, S_2).$$

The lexicographical ordering on 7-tuples of naturals is a well-founded ordering on the data states of $LimpB$ such that the measure decreases at every execution of a progressing internal step.

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

First, suppose that $S[pr] < 4$. The only internal action that can change the state of a node $i$ to 4, is the receiving of an acknowledgment 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 $S'[pr'] = 4 \land \text{singleton}(P'[pr'])$. This contradicts $I_{19}$. So in every state $d'$ reachable by an internal action $S'[pr'] < 4$.

Second, suppose that $S[pr] \neq 4$. Then $\text{empty}(P[pr])$ by $I_1$ and $I_{19}$. We see by $I_{17}$ that $pr$ will keep the same value.

(iii) If the implementation can do the leader action, then so can the specification:

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

Trivial.

(iv) 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:

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

Suppose $FC_\xi$ and $S[pr] < 4$. $S[pr] \neq 1$ by $FC_\xi$. If $S[pr] = 3$, then we have by $I_8$ and $I_{10}$ that $\text{contention}$, contradicting the assumption $FC_\xi$ by $I_{20}$. So $S[pr] = 0 \lor S[pr] = 2$. 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 $I_{13}$ we can make the following case distinction, where $S[i_1] \neq 1$ by $FC_\xi$:

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

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 an $i_2 \neq i_0$ in $P[i_1]$. We can repeat the argument above for $i_1$ and $i_2$. But we cannot construct an infinite path $i_0i_1i_2\ldots$ where $S[i_k] = 0$ and $i_{k+1} \in P[i_k]$ and $i_k \neq i_{k+2}$ for all $k$, as this would violate $\text{GoodTopology}$ by $I_2$. 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.
6. Conclusions

We have described the tree identify protocol of the 1394 multimedia serial bus. This was an exercise in specification using μCRL 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 [53] used here. In particular, the initial linearization can be generated automatically, and some development in this area is underway. In fact, computer checked proofs using this technique are described in [64]. 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 linearization of implementation A. 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. Automation of this and linearization would leave the verifier free to consider the more tricky questions of the definition of the state mapping 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 [45] 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

- \( S[pr] = 2 \). Suppose \( \neg \text{empty}(P[pr]) \). Then we find \( \neg FC_\xi \) by \( I_{18} \) and \( I_{20} \). Contradiction.

(v) The implementation and the specification perform external actions with the same parameter. Trivial; the leader action involves no data.

(vi) If from a data state \( d \), the implementation reaches state \( d' \) by the execution of the leader action, then \( h(d') = \bot \).

We see by \( I_{17} \) that the value of \( pr \) will be the same for \( d \) and \( d' \). Note that \( S = S' \) except that \( S'[pr] = 4 \). So \( h(d') = (4 < 4) = \bot \).
to extract some general principles which could be coded in some specialized 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 [53] 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 [54] 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 μCRL.

This proof technique compares favourably with earlier proofs in μCRL, e.g., [48, 37], which relied on much lower level proof rules (the usual rules for manipulating process algebra expressions), although we note that the proof given in [37] contains some similar features to the specifications here and in [54] (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 [38] in which automated proofs of branching bisimulation are carried out using the CADP toolbox, and [79] which again uses the CADP toolbox, but this time to check the validity of modal formulas 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.

Acknowledgments. 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 [Shankland] 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.

7. Appendix: Theorems and Definitions

We repeat here the most important definitions and theorems from [53].
Definition 7.1. Let $A \subseteq \Act \cup \{\tau\}$ be a finite set of action names; assume that the action names $a \in A$ are parametrized with data of type $D_a$.\footnote{In fact, silent actions $\tau$ do not have a parameter, but it is convenient to have a uniform treatment of the elements of $A$. Of course this can be made precise easily.} Also, assume for every $a \in A$ an additional data type $E_a$. A linear process equation (LPE) over $A$ and a data type $D$ is an equation of the form

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

for some functions

$$f_a : D \to E_a \to D_a,$$
$$g_a : D \to E_a \to D,$$
$$b_a : D \to E_a \to \text{Bool}.$$

The data type $D$ represents the state space of the process defined by the equation for $X$ in the definition above: if, for some $d$ of type $D$, some $a \in A$, and some $e$ of type $E_a$, the condition $b_a(d, e)$ is satisfied, then from state $d$ there is a transition to state $g_a(d, e)$, and this transition is labelled with action $a(f_a(d, e))$.

An important feature of linear process equations is that for each element of $A$ there is at most one summand in the alternative composition. Note that therefore the definition of process $\text{LimpB}$ in Table 5 does not directly fit into this format. We made sure that theorems were applied correctly.

Definition 7.2. An LPE $X$ written as in Definition 7.1 is called convergent if it does not admit infinite $\tau$-paths, that is, if 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 7.1 is a function $I : D \to \text{Bool}$ such that for all $a \in A$, $e : E_a$, and $d : D$ we have

$$b_a(d, e) \land I(d) \rightarrow I(g_a(d, e)).$$

Definition 7.3. Let $X$ and $X'$ be LPEs given as follows:

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

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

Let $FC$ be a formula over $d : D$ describing exactly the states of $X$ from which no $\tau$-action is enabled (i.e., equivalent to $\neg \exists e : E_\tau (b_\tau(d, e))$).

A state mapping

$$h : D \to D'.$$
is said to satisfy the matching criteria for state $d : D$, if for all $e_\tau : E_\tau$, $a \in A \setminus \{\tau\}$, and $e_a : E_a$ the following conditions hold:

(i) $X$ is convergent;
(ii) $b_\tau(d, e_\tau) \to h(d) = h(g_\tau(d, e_\tau));$
(iii) $b_a(d, e_a) \to b_a'(h(d), e_a);$
(iv) $FC_X(d) \land b_a'(h(d), e_a) \to b_a(d, e_a);$
(v) $b_a(d, e_a) \to f_a(d, e_a) = f_a'(h(d), e_a);$
(vi) $b_a(d, e_a) \to h(g_a(d, e_a)) = g_a'(h(d), e_a).$

**Theorem 7.1 (General Equality Theorem).** Let $X$ and $X'$, the focus condition, and the state mapping $h$ be written as above. Let $I$ be an invariant of $X$ such that $h$ satisfies the matching criteria for all $d : D$ with $I(d)$. Assume that $p$ and $p'$ are solutions of $X$ and $X'$, respectively, then it holds for all $d : D$ with $I(d)$ that

$$\tau \cdot p(d) = \tau \cdot p'(h(d)).$$

If $FC(d)$ then we also have the stronger result $p(d) = p'(h(d))$.

**7.1. Abstraction and Idle Loops.** Let $Ext$ and $Int$ be disjoint finite sets of action names; let $Int_\tau = Int \cup \{\tau\}$. Let $X$ and $X'$ be LPEs given as follows:

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

$$X'(d : D') = \sum_{a \in Ext} \sum_{e : E_a} a(f_a'(d, e)) \cdot X'(g_a'(d, e)) < b_a'(d, e) \triangleright \delta.$$

**Definition 7.4.** Let $\xi$ be a pre-abstraction function. The *focus condition* of $X$ relative to $\xi$ is defined by:

$$FC_\xi(d) = \bigwedge_{a \in Int_\tau} \forall e : E_a (\neg(b_a(d, e) \land \xi(a)(d, e))).$$

**Definition 7.5.** The LPE $X$ is *convergent* with respect to $\xi$ if there is a well-founded ordering $<$ on $D$ such that for all $a \in Int_\tau$, $d : D$ and all $e : E_a$ we have that $b_a(d, e)$ and $\xi(a)(d, e)$ imply $g_a(d, e) < d$.

**Definition 7.6.** Let $X, X'$ be as above. Let $h : D \to D'$ be a state mapping and let $\xi$ be a pre-abstraction function. The state mapping satisfies the *matching criteria for idle loops* with respect to state $d : D$, if for all $i \in Int_\tau$, $e_i : E_i$, $a \in Ext$, and $e : E_a$

(i) $X$ is convergent with respect to $\xi$;
(ii) $b_i(d, e_i) \to h(d) = h(g_i(d, e_i));$
(iii) $b_a(d, e) \to b_a'(h(d), e);$
(iv) $FC_\xi(d) \land b_a'(h(d), e) \to b_a(d, e);$
(v) $b_a(d, e) \to f_a(d, e) = f_a'(h(d), e);$. 

(vi) \( b_a(d, e) \rightarrow h(g_a(d, e)) = g'_a(h(d), e) \).

**Theorem 7.2.** Let \( X \) and \( X' \), the pre-abstraction function and the state mapping \( h \) be written be as above. Let \( p \) and \( p' \) be solutions of \( X \) and \( X' \), respectively. If \( I \) is an invariant of \( X \) such that \( h \) satisfies the matching criteria for idle loops for all \( d : D \) with \( I(d) \), then for all \( d : D \) with \( I(d) \) it holds that

\[
\tau \cdot \tau_{int}(p(d)) = \tau \cdot p'(h(d)).
\]