Implementing Asynchronous Distributed Systems Using the IOA Toolkit

Chryssis Georgiou 1, Panayiotis P. Mavrommatis2, Joshua A. Tauber2

1 University of Cyprus, Department of Computer Science
2 MIT Computer Science and Artificial Intelligence Laboratory

September 27, 2004

Contents

1 Introduction 2

2 Experiments 2
  2.1 LCR Leader Election .................................................. 2
  2.2 Asynchronous Spanning Tree ........................................... 4
  2.3 Asynchronous Broadcast Convergecast .............................. 7
  2.4 Leader Election Using Broadcast Convergecast ...................... 10
  2.5 Unrooted Spanning Tree to Leader Election ......................... 12
  2.6 GHS Minimum Spanning Tree ........................................ 15

3 Results 23

4 Conclusions 23

5 Comments and Suggestions on the Toolkit 23

A Troubleshooting 29

B Bug Reports 29

C Automata Definitions and Input Files 31
  C.1 SendMediator and ReceiveMediator .................................. 31
  C.2 LCR ................................................................. 32
  C.3 Asynchronous Spanning Tree ......................................... 35
  C.4 Asynchronous Broadcast Convergecast ............................ 37
  C.5 Leader Election using Asynchronous Broadcast Convergecast .... 40
  C.6 Spanning Tree to Leader Election ................................. 45
  C.7 GHS ............................................................... 47

D Runtime Tables 56
1 Introduction

This document is a report about the capabilities and performance of the IOA Toolkit, and in particular the tools that provide support for implementing and running distributed systems (checker, composer, code generator). The Toolkit compiles distributed systems specified in IOA into Java classes, which run on a network of workstations and communicate using the Message Passing Interface (MPI). In order to test the toolkit, several distributed algorithms were implemented, ranging from simple algorithms such as LCR leader election in a ring network to more complex algorithms such as the GHS algorithm for computing the minimum spanning tree in an arbitrary graph. All of our experiments completed successfully, and several runtime measurements were made.

2 Experiments

Implementation Platform The machines used are located in the Theory of Computation Group of the MIT Computer Science and Artificial Intelligence Laboratory, forming a Local Area Network. They are all Red Hat Linux machines and with Intel Pentium III to IV with clock speed ranging from 1 GHz to 3.2 GHz. Even though MPI sets up a connection between every pair of nodes, the algorithms only use the communication channels they need. For example, a node (i) in LCR only sends to node i+1 and only receives from node i-1.

2.1 LCR Leader Election

The algorithm of Le Lann, Chang and Roberts for Leader Election in a ring network was the first experiment in running an IOA program on a network of computers. The automaton definition that appears in [1] (Section 15.1) was used, with some modifications. For all the algorithms that follow, the nodes are automatically numbered from 0 to (size - 1).

Automata Definitions The automata LCRProcess, LCRNode, SendMediator and ReceiveMediator were written. The mediator automata are given in Appendix C.1 (these automata implement the channel automata integrated with MPI functionality). The composed automaton (LCR) appears in Appendix C.2 (this is the one that is translated into Java code by the IOA Toolkit).

LCR Leader Election process automaton

```plaintext
type Status = enumeration of idle, voting, elected, announced

automaton LCRProcess(rank: Int, size: Int)
signature
  input vote
  input RECEIVE(m: Int, const mod(rank - 1, size), const rank: Int)
  output SEND(m: Int, const rank: Int, const mod(rank+1, size))
```
output leader(const rank)

states
pending: Mset[Int] := {rank},
status: Status := idle

transitions
input vote
  eff status := voting
input RECEIVE(m, j, i) where m > i
  eff pending := insert(m, pending)
input RECEIVE(m, j, i) where m < i
input RECEIVED(i, j, i)
  eff status := elected
output SEND(m, i, j)
  pre status ≠ idle ∧ m ∈ pending
  eff pending := delete(m, pending)
output leader(rank)
  pre status = elected
  eff status := announced

LCR Leader Election composition automaton

automaton LCRNode(rank: Int, size: Int)
components
P: LCRProcess(rank, size);
RM[j: Int]: ReceiveMediator(Int, Int, j, rank)
  where j = mod(rank-1, size);
SM[j: Int]: SendMediator(Int, Int, rank, j)
  where j = mod(rank+1, size)

Results The trace of a run on 8 nodes (on 4 machines) can be found in Appendix E.1. A snapshot of the trace, representing the last five transitions, is shown below.

A snapshot of the trace of LCR leader election

transition: output SEND(7, 5, 6) in automaton LCR(5)
on condor.csail.mit.edu at 7:25:37:280
Modified state variables:
P → Tuple, modified fields: {pending -> ()}
SM → Map, modified entries: {[6 -> Tuple, modified fields: {[toSend ->
  Sequence, elements added: {7 } Elements removed: { }]}}

transition: output RECEIVE(7, 5, 6) in automaton LCR(6)
on parrot.csail.mit.edu at 7:25:37:755
Modified state variables:
P → Tuple, modified fields: {pending -> (7) }
SM → Map, modified entries: {[7 -> Tuple, modified fields: {[toSend ->

3
As the trace indicates, node 7’s message has made its way around the ring and eventually returned to node 7. At that point, node 7 announced itself as the leader. Figure 2.1 shows the messages sent by the nodes. All messages were sent from node \( i \) to node \( i + 1 \) (mod 8) and the message with value \( i \) was sent first. The message with value 7 followed and made the round of the ring, to elect node 7 as the leader.

![Figure 2.1: Running the LCR leader election algorithm on a ring of 8 nodes. The white squares represent the nodes, while the shaded parallelograms represent the messages sent.](image)

### 2.2 Asynchronous Spanning Tree

The Asynchronous Spanning Tree Algorithm, (see Section 15.3 of [1]) was the next test for the Toolkit. The algorithm is still very simple: Given a general graph it computes a spanning tree on the graph. This was the first test of the Toolkit on arbitrary graphs, where each node had more than one incoming and outgoing communication channels.

**Automata Definitions**  The AsynchSpanningTree automaton, as defined in [1](Section 15.3) was used. The process automaton and the composition one are listed below. The expanded automaton is shown in Appendix C.3.
Asynchronous Spanning Tree process automaton

type Message = enumeration of search, null

automaton sTreeProcess(i: Int)
signature
  input RECEIVE(m: Message, const i: Int, j: Int)
  output SEND(m: Message, const i: Int, j: Int)
  output PARENT(j: Int)

states
  nbrs: Set[Int] := {},
  parent: Int := -1, % -1 for null
  reported: Bool := false,
  send: Map[Int, Message]

transitions
  input RECEIVE(m, i, j)
    eff
    if i \neq 0 \land parent = -1 then
      parent := j;
      for k:Int in nbrs - {j} do
        send[k] := search
      od
    fi
  output SEND(m, i, j)
    pre send[j] = search
    eff send[j] := null
  output PARENT(j)
    pre parent = j \land reported = false
    eff reported := true

Asynchronous Spanning Tree composition automaton

type Message = enumeration of search, null

automaton sTreeNode(i: Int)
components
  P: sTreeProcess(i);
  RM[j:Int]: ReceiveMediator(Message, Int, i, j);
  SM[j:Int]: SendMediator(Message, Int, i, j)

Results  Figure 2.2 shows a graph of 16 nodes connected in a 4 \times 4 grid that was used on some of our tests. The source node was node 0. Some of the spanning trees computed are also shown in Figure 2.2. A snapshot of the trace follows, showing nodes 1 and 4 sending their message to node 5. The message of node 1 arrives first and thus node 5 announces node 1 as its parent. The complete trace of this run can be found in Appendix E.2.

Trace snapshot of the Asynchronous Spanning Tree algorithm

5
Figure 2.2: The $4 \times 4$ grid used to test the Asynchronous Spanning Tree Algorithm, along with 2 different spanning trees computed from 2 different runs of the algorithm.

transition: output SEND(search, 1, 5) in automaton sTreeNode(1) on blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (0 2 5), parent: 0, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 4, 5) in automaton sTreeNode(4) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (0 5 8), parent: 0, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 5, 1) in automaton sTreeNode(5) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (1 4 6 9), parent: 1, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output PARENT(1) in automaton sTreeNode(5) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (1 4 6 9), parent: 1, reported: true, send: ioa.runtime.adt.MapSort]
2.3 Asynchronous Broadcast Convergecast

This is essentially an extension of the previous algorithm, where along with the construction of a spanning tree, a broadcast and convergecast take place (using the computed spanning tree). The source node was node 0, and the message 99 (a dummy message) was broadcast on the network. Some complexity is added compared to the previous algorithm by the fact that different kinds of messages are exchanged.

Automata Definitions The AsynchBcastAck automaton, as defined in [1](section 15.3) was used. The process automaton and the composition automaton are shown below. The expanded automaton is given in Appendix C.4.

Asynchronous Broadcast Convergecast process automaton

type Kind = enumeration of bcast, ack
type BCastMsg = tuple of kind: Kind, w: Int
type Message = union of msg: BCastMsg, kind: Kind

automaton bcastProcess(rank: Int, nbrs: Set[Int])
signature
  input RECEIVE(m: Message, const rank, j: Int)
  output SEND(m: Message, const rank, j: Int)
  internal report(const rank)

states
  val: Int := -1, % -1 = special value denoting null
  parent: Int := -1,
  reported: Bool := false,
  acked: Set[Int] := {},
  send: Map[Int, Seq[Message]]

initially
  rank = 0 ⇒
    (val = 99 ∧ % 99 = the value to be broadcast
      (∀ j: Int
        ((j ∈ nbrs) ⇒ send[j] = {} ⊢ msg([bcast, val])))

transitions
  output SEND(m, rank, j)
    pre m = head(send[j])
    eff send[j] := tail(send[j])
  input RECEIVE(m, rank, j)
    eff
      if m = kind(ack) then
        acked := acked ∪ {j}
      else
        if val = -1 then
          val := m.msg.w;
          parent := j;
          for k:Int in nbrs - {j} do
            send[k] := send[k] ⊢ m
          od
        else
          send[j] := send[j] ⊢ kind(ack)

7
Asynchronous Broadcast Convergecast composition automaton

type Message = union of msg: BCastMsg, kind: Kind

automaton bcastNode(i: Int)
components
  P: bcastProcess(i);
  RM[j: Int]: ReceiveMediator(Message, Int, i, j);
  SM[j: Int]: SendMediator(Message, Int, i, j)

Results
The algorithm was tested on several graphs. One of them is shown in Figure 2.3 (a). Figure 2.3 (b) and (c) depicts some of the spanning trees that were computed and the communication sequence; the numbers next to the nodes represent the sequence at which the nodes reported done (through the internal report action). A snapshot of the trace is listed below, showing nodes 1 and 4 reporting that they are done. At that point, node 0 is enabled and after some communication between these nodes, node 0 also reports done. The complete trace of this run is shown in Appendix E.3.

Trace snapshot of the Asynchronous Broadcast Convergecast algorithm

```plaintext
transition: internal report(1) in automaton bcastNode
Modified state variables:
  P → [val: 99, acked: (2 5), nbrs: (0 2 5), parent: 0, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]

transition: output SEND(kind(ack), 1, 0) in automaton bcastNode
Modified state variables:
  P → [val: 99, acked: (2 5), nbrs: (0 2 5), parent: 0, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
  SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 4, 8) in automaton bcastNode
Modified state variables:
  P → [val: 99, acked: (5 8), nbrs: (0 5 8), parent: 0, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
  RM → ioa.runtime.adt.MapSort

transition: internal report(4) in automaton bcastNode
```
Figure 2.3: The grid shown in (a) was used to test the algorithm. (b) and (c) show the spanning trees computed in two (different) runs and the numbers next to the nodes indicate the sequence at which the nodes reported done.

Modified state variables:

\begin{align*}
P & \rightarrow [\text{val} : 99, \text{acked} : (5 8), \text{nbrs} : (0 5 8), \text{parent} : 0, \text{reported} : \text{true}, \\
& \text{send} : \text{ioa.runtime.adt.MapSort}, \text{temp} : [\text{kind} : \text{bcast}, w : 87]]
\end{align*}

transition: output SEND(kind(ack), 4, 0) in automaton bcastNode

Modified state variables:

\begin{align*}
P & \rightarrow [\text{val} : 99, \text{acked} : (5 8), \text{nbrs} : (0 5 8), \text{parent} : 0, \text{reported} : \text{true}, \\
& \text{send} : \text{ioa.runtime.adt.MapSort}, \text{temp} : [\text{kind} : \text{bcast}, w : 87]]
\end{align*}

RM $\rightarrow$ ioa.runtime.adt.MapSort

SM $\rightarrow$ ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 0, 1) in automaton bcastNode

Modified state variables:

\begin{align*}
P & \rightarrow [\text{val} : 99, \text{acked} : (1), \text{nbrs} : (1 4), \text{parent} : -1, \text{reported} : \text{false}, \text{send} : \\
& \text{ioa.runtime.adt.MapSort}, \text{temp} : [\text{kind} : \text{bcast}, w : 99]]
\end{align*}

RM $\rightarrow$ ioa.runtime.adt.MapSort

SM $\rightarrow$ ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 0, 4) in automaton bcastNode

Modified state variables:

\begin{align*}
P & \rightarrow [\text{val} : 99, \text{acked} : (1 4), \text{nbrs} : (1 4), \text{parent} : -1, \text{reported} : \text{false}, \text{send} : \\
& \text{ioa.runtime.adt.MapSort}, \text{temp} : [\text{kind} : \text{bcast}, w : 99]]
\end{align*}

RM $\rightarrow$ ioa.runtime.adt.MapSort

transition: internal report(0) in automaton bcastNode

Modified state variables:

\begin{align*}
P & \rightarrow [\text{val} : 99, \text{acked} : (1 4), \text{nbrs} : (1 4), \text{parent} : -1, \text{reported} : \text{true}, \\
& \text{send} : \text{ioa.runtime.adt.MapSort}, \text{temp} : [\text{kind} : \text{bcast}, w : 99]]
\end{align*}
2.4 Leader Election Using Broadcast Convergecast

In page 500 of [1], the author describes how the Asynchronous Broadcast Convergecast algorithm can be used to implement a leader election algorithm on a general graph using the Asynchronous Broadcast Convergecast algorithm. The main idea is to have every node act as a source node and create its own spanning tree, broadcast its UID using this spanning tree and hear from all the other nodes via a convergecast. During this convergecast, along with the acknowledge message, the children also send what they consider as the maximum UID in the network. The parents gather the maximum UIDs from the children, compare it to their own UID and send the maximum to their own parents. Thus, each source node learns the maximum UID in the network and the node whose UID equals the maximum one announces itself as a leader.

Automata Definitions  The process and composition automata are shown below. The expanded automaton is defined in Appendix C.5.

Leader Election Using Broadcast Convergecast process automaton

type Kind = enumeration of bcast, ack

type BCastMsg = tuple of kind: Kind, w: Int

type AckMsg = tuple of kind: Kind, mx: Int

type MSG = union of bmsg: BCastMsg, amsg: AckMsg, kind: Kind

type Message = tuple of msg: MSG, source: Int

automaton bcastLeaderProcess(rank: Int)

signature
input RECEIVE(m: Message, i: Int, j: Int)
output SEND(m: Message, i: Int, j: Int)
internal report(i: Int, source: Int)
internal finished
output LEADER

states
nbrs: Set[Int],
val: Map[Int, Int], % initially -1 (null) for all nodes
parent: Map[Int, Int], % initially -1 (null) for all nodes
reported: Map[Int, Bool], % initially false for all nodes
acked: Map[Int, Set[Int]], % initially {} for all nodes
send: Map[Int, Int, Seq[Message]], % First variable: source.
max: Map[Int, Int], % The max value found in the network , initially i
elected: Bool := false,
announced: Bool := false

initially
(∀ j: Int
((0 ≤ j ∧ j < 16) ⇒
(send[j,k] = {} ∧ (k ∈ nbrs ∧ rank = j) ⇒
send[j, k] = {bmsg([bcast, 99]), j}))))

transitions

10
output SEND(m, i, j)
  pre m = head(send[m.source, j])
  eff send[m.source, j] := tail(send[m.source, j])
input RECEIVE(m, i, j)
  eff if m.msg = kind(ack) then
    acked[m.source] := acked[m.source] ∪ {j}
  elseif tag(m.msg) = amsg then
    if max[m.source] < m.msg.amsg.mx then
      max[m.source] := m.msg.amsg.mx;
    fi;
    acked[m.source] := acked[m.source] ∪ {j}
  else
    %BcastMsg
    if val[m.source] = -1 then
      val[m.source] := bmsg.w;
      parent[m.source] := j;
      for k:Int in nbrs - {j} do
        send[m.source, k] := send[m.source, k] ⊢ m
      od
      else
        send[m.source, j] := send[m.source, j] ⊢ [kind(ack), m.source]
      fi
    fi
internal finished
  pre acked[rank] = nbrs ∧
    reported[rank] = false
  eff reported[rank] := true;
    if (max[rank] = rank) then
      elected := true
    fi;
output LEADER
  pre elected = true ∧ announced = false
  eff announced := true
internal report(i, source) where i ≠ source
  pre parent[source] ≠ -1 ∧
    acked[source] = nbrs - {parent[source]} ∧
    reported[source] = false
  eff send[source, parent[source]] :=
    send[source, parent[source]] ⊢ [amsg([ack, max[source]]), source];
    reported[source] := true;

Leader Election Using Broadcast Convergecast composition automaton

automaton bcastLeaderNode(i: Int)
  components
    P: bcastLeaderProcess(i);
    RM[j: Int]: ReceiveMediator(Message, Int, i, j);
    SM[j: Int]: SendMediator(Message, Int, i, j)

Results  This algorithm was also tested on the 4 × 4 grid that was used in the two previous algorithms (see Figure 2.3 (a)). The algorithm terminated correctly and announced node 15 as the leader. A snapshot of the trace is shown below. The complete trace of this run can be found on the web, at http://theory.csail.mit.edu/~pmavrom/ioaReport.html
Trace snapshot of the Leader Election Using Broadcast Convergecast algorithm

```plaintext
transition: output RECEIVE([msg: amsg([kind: ack, mx: 13]), source: 14], 14, 10) in automaton bcastLeader(14) on drake.csail.mit.edu at 9:07:20:455
Modified state variables:
P → Tuple, modified fields: {[acked → Map, modified entries: {[14 → (10 13 15)]}]}
c2 → 7

transition: internal finished() in automaton bcastLeader(14) on drake.csail.mit.edu at 9:07:20:464
Modified state variables:
P → Tuple, modified fields: {[reported → Map, modified entries: {[14 → true]}]}
c2 → 14
k → 15

Modified state variables:
P → Tuple, modified fields: {[acked → Map, modified entries: {[15 → (11 14)]}]}
c2 → 7

transition: internal finished() in automaton bcastLeader(15) on loon.csail.mit.edu at 9:07:20:687
Modified state variables:
P → Tuple, modified fields: {[reported → Map, modified entries: {[15 → true]}] [elected → true]}
c2 → 15

transition: output LEADER() in automaton bcastLeader(15) on loon.csail.mit.edu at 9:07:20:689
Modified state variables:
P → Tuple, modified fields: {[announced → true]}
```

2.5 Unrooted Spanning Tree to Leader Election

The algorithm STtoLeader of [1](page 501) was implemented as the next test for the Toolkit. The algorithm takes as input an unrooted spanning tree and returns a leader. The automaton listed below was written, according to the description of the algorithm.

Automata Definitions  The process and composition automata are listed below, while the expanded automaton can be found in Appendix C.6.
Unrooted Spanning Tree to Leader Election process automaton

\texttt{type Status = enumeration of idle, elected, announced} \\
\texttt{type Message = enumeration of elect} \\
\texttt{axioms ChoiceSet (Int for E)}

\texttt{automaton sTreeLeaderProcess(rank: Int, nbrs:Set[Int])} \\
\texttt{signature} \\
\texttt{input RECEIVE(m: Message, const rank: Int, j: Int)} \\
\texttt{output SEND(m: Message, const rank: Int, j: Int)} \\
\texttt{output leader} \\
\texttt{states} \\
\texttt{receivedElect: Set[Int],} \\
\texttt{sentElect: Set[Int],} \\
\texttt{status: Status,} \\
\texttt{send: Map[Int, Seq[Message]]} \\
\texttt{initially} \\
\texttt{true ⇒} \\
\texttt{receivedElect = {}} \\
\texttt{∧ sentElect = {}} \\
\texttt{∧ status = idle} \\
\texttt{∧ (size(nbrs) = 1 ⇒} \\
\texttt{ send[chooseRandom(nbrs)]} \\
\texttt{= send[chooseRandom(nbrs)] ⊨ elect)} \\
\texttt{transitions} \\
\texttt{input RECEIVE(m, i, j)} \\
\texttt{eff receivedElect := insert(j, receivedElect);} \\
\texttt{if size(receivedElect) = size(nbrs) - 1 then} \\
\texttt{send[chooseRandom(nbrs - receivedElect)] :=} \\
\texttt{send[chooseRandom(nbrs - receivedElect)] ⊨ elect;} \\
\texttt{sentElect :=} \\
\texttt{insert(chooseRandom(nbrs - receivedElect), sentElect)} \\
\texttt{elsif receivedElect = nbrs then} \\
\texttt{if j ∈ sentElect then if i > j then status := elected fi} \\
\texttt{else status := elected fi} \\
\texttt{fi} \\
\texttt{output SEND(m, i, j)} \\
\texttt{pre m = head(send[j])} \\
\texttt{eff send[j] := tail(send[j])} \\
\texttt{output leader} \\
\texttt{pre status = elected} \\
\texttt{eff status := announced}

Unrooted Spanning Tree to Leader Election composition automaton

\texttt{automaton sTreeLeaderNode(rank: Int, nbrs:Set[Int])} \\
\texttt{components} \\
\texttt{P: sTreeLeaderProcess(rank, nbrs);} \\
\texttt{RM[j:Int]: ReceiveMediator(Message, Int, rank, j);} \\
\texttt{SM[j:Int]: SendMediator(Message, Int, rank, j)}

\textbf{Results} The algorithm was tested on several graphs and different spanning trees as input. A spanning tree on a $4 \times 4$ grid is shown in Figure 2.4. The algorithm worked correctly, announcing
only one node as the leader. A snapshot of the trace is listed below. In that specific run, the edge between nodes 5 and 6 had *elect* messages sent in both directions, with node 6 being elected as the leader, having a larger UID. The complete trace of this run can be found in Appendix E.4.

![Diagram showing unrooted spanning tree and leader election process](image-url)

**Figure 2.4:** (a) shows the unrooted spanning tree used in one run. In (b), the arrows show the direction of the messages in a specific run, where the edge between nodes 5 and 9 had *elect* messages sent in both directions. Node 9, with a larger UID was elected as the leader.

### Trace Snapshot of the Unrooted Spanning Tree to Leader Election

```
transition: output SEND(elect, 6, 5) in automaton sTreeLeader(6) on drake.csail.mit.edu at 9:29:15:112
Modified state variables:
P → Tuple, modified fields: {
  send -> Map, modified entries: {
    5 -> Sequence, elements added: {} Elements removed: {elect}
  }
}
SM → Map, modified entries: {
  5 -> Tuple, modified fields: {
    toSend -> Sequence, elements added: {elect} Elements removed: {} 
  }
}
k → 5
tempNbrs2 → (10 2)

transition: output SEND(elect, 5, 6) in automaton sTreeLeader(5) on loon.csail.mit.edu at 9:29:16:182
Modified state variables:
P → Tuple, modified fields: {
  send -> Map, modified entries: {
    6 -> Sequence, elements added: {} Elements removed: {elect}
  }
}
SM → Map, modified entries: {
  6 -> Tuple, modified fields: {
    toSend -> Sequence, elements added: {elect} Elements removed: {} 
  }
}
k → 6
tempNbrs2 → (1 4 9)

transition: output RECEIVE(elect, 5, 6) in automaton sTreeLeader(5) on loon.csail.mit.edu at 9:29:16:494
Modified state variables:
P → Tuple, modified fields: {
  receivedElect -> (1 4 6 9) 
}
SM → Map, modified entries: {
  6 -> Tuple, modified fields: {
    toSend -> Sequence, elements added: {} Elements removed: {elect}
    sent -> Sequence, elements added: {elect} Elements removed: {} 
  }
}
```

---

14
2.6 GHS Minimum Spanning Tree

The last algorithm we ran using the Toolkit was the algorithm of Gallager, Humblet and Spira for finding the minimum-weight spanning tree in an arbitrary graph. Welch, Lamport and Lynch give an I/O automaton definition of the GHS algorithm in [2], which is used as a basis for our automata definitions. One technical detail in running this version of the algorithm is that the edge weights needed to be unique. This algorithm is significantly longer than the previous ones, with 7 different messages exchanged and 12 state variables for each automaton.

Automata Definitions  The automaton below defines the algorithm and the composition with the mediator automata is shown after that. The expanded automaton can be found in Appendix C.7.

GHS process automaton
% weight: Maps the Links ∈ links to their weight

%% automaton GHSProcess (rank : Int, size : Int, links : Set[Link], weight : Map[Link, Int])
signature
input startP
input RECEIVE(m: Message, const rank, i: Int)
output InTree(l:Link)
output NotInTree(l: Link)
output SEND(m: Message, const rank, j: Int)
internal ReceiveConnect(qp: Link, l:Int)
internal ReceiveInitiate(qp: Link, l:Int, c: Null[Edge], st: Status)
internal ReceiveTest(qp: Link, l:Int, c: Null[Edge])
internal ReceiveAccept(qp: Link)
internal ReceiveReject(qp: Link)
internal ReceiveReport(qp: Link, w: Int)
internal ReceiveChangeRoot(qp: Link)

states
nstatus : Nstatus,
nfrag : Null[Edge],
nlevel : Int,
bestlink : Null[Link],
bestwt : Int,
testlink : Null[Link],
inbranch : Link,
findcount : Int,
lstatus : Map[Link, Lstatus],
queueOut : Map[Link, Seq[Message]],
queueIn : Map[Link, Seq[Message]],
answered : Map[Link, Bool],
% Temporary variables
min : Int, minL : Null[Link], S : Set[Link]
initially
true ⇒ nstatus = sleeping
∧ nfrag = nil
∧ nlevel = 0
∧ bestlink = embed(chooseRandom(links))
∧ bestwt = weight[chooseRandom(links)]
∧ testlink = nil
∧ inbranch = chooseRandom(links)
∧ findcount = 0
∧ ∀ l: Link
(l ∈ links ⇒
 lstatus[l] = unknown
∧ answered[l] = false
∧ queueOut[l] = {}
∧ queueIn[l] = {})

transitions
% INPUT ACTIONS
input startP
eff if nstatus = sleeping then
%WakeUp
minL := embed(chooseRandom(links)); min := weight[minL.val];
for tempL: Link in links do
if weight[tempL] < min then minL := embed(tempL); min := weight[tempL] fi;
od;}
lstatus[\text{minL.val}] := \text{branch}; \ nstatus := \text{found};
queueOut[\text{minL.val}] := \text{queueOut[\text{minL.val}]} \vdash \text{connMsg([CONNECT, 0])};
\% \text{EndWakeUp}
\fi

\text{input} \ \text{RECEIVE}(m: \text{Message}, \ i: \text{Int}, \ j: \text{Int})
\quad \text{eff} \ \text{queueIn}[[i,j]] := \text{queueIn}[[i,j]] \vdash m
\% \text{OUTPUT ACTIONS}
\text{output} \ \text{InTree}(l: \text{Link})
\quad \text{pre} \ \text{answered}[l] = \text{false} \land \ lstatus[l] = \text{branch}
\quad \text{eff} \ \text{answered}[l] := \text{true}
\text{output} \ \text{NotInTree}(l: \text{Link})
\quad \text{pre} \ \text{answered}[l] = \text{false} \land \ lstatus[l] = \text{rejected}
\quad \text{eff} \ \text{answered}[l] := \text{true}
\text{output} \ \text{SEND}(m: \text{Message}, \ i: \text{Int}, \ j: \text{Int})
\quad \text{pre} \ m = \text{head(queueOut}[[i,j]])
\quad \text{eff} \ \text{queueOut}[[i,j]] := \text{tail(queueOut}[[i,j]])
\% \text{INTERNAL ACTIONS}
\text{internal} \ \text{ReceiveConnect}(qp: \text{Link}, \ l: \text{Int})
\quad \text{pre} \ \text{head(queueIn}[[qp]]) = \text{connMsg([CONNECT, l])}
\quad \text{eff} \ \text{queueIn}[[qp]] := \text{tail(queueIn}[[qp]])
\quad \text{if} \ \text{nstatus} = \text{sleeping} \text{then}
\quad \quad \text{\%WakeUp}
\quad \quad \text{minL} := \text{embed(chooseRandom(links))}; \ \text{min} := \text{weight[minL.val]};
\quad \quad \text{for tempL in links do}
\quad \quad \quad \text{if weight}[	ext{tempL}] < \text{min} \text{then minL} := \text{embed(tempL)}; \ \text{min} := \text{weight[tempL]}; \ \text{fi};
\quad \quad \text{od};
\quad \text{lstatus}[\text{minL.val}] := \text{branch}; \ \text{nstatus} := \text{found};
\quad \text{queueOut}[\text{minL.val}] := \text{queueOut}[\text{minL.val}] \vdash \text{connMsg([CONNECT, 0])};
\quad \text{\%EndWakeUp}
\fi;
\text{if} \ l < \text{nlevel} \text{then}
\quad \text{lstatus}[[qp.t,qp.s]] := \text{branch};
\quad \text{if} \ \text{testlink} \neq \text{nil} \text{then}
\quad \quad \text{queueOut}[[qp.t,qp.s]] := \text{queueOut}[[qp.t,qp.s]] \vdash \text{initMsg([INITIATE, nlevel, nfrag, find])};
\quad \quad \text{findcount} := \text{findcount + 1}
\quad \text{else}
\quad \quad \text{queueOut}[[qp.t,qp.s]] := \text{queueOut}[[qp.t,qp.s]] \vdash \text{initMsg([INITIATE, nlevel, nfrag, found])}
\quad \text{fi};
\text{else}
\quad \text{if} \ \text{lstatus}[[qp.t,qp.s]] = \text{unknown} \text{then}
\quad \quad \text{queueIn}[[qp]] := \text{queueIn}[[qp]] \vdash \text{connMsg([CONNECT, l])}
\quad \text{else}
\quad \quad \text{queueOut}[[qp.t,qp.s]] := \text{queueOut}[[qp.t,qp.s]] \vdash \text{initMsg([INITIATE, nlevel+1, embed([qp.t, qp.s]), find])}
\quad \text{fi}
\text{fi}
\text{internal} \ \text{ReceiveInitiate}(qp: \text{Link}, \ l: \text{Int}, \ c: \text{Null[Edge]}, \ \text{st: Status})
\quad \text{pre} \ \text{head(queueIn}[[qp]]) = \text{initMsg([INITIATE, l, c, st])}
\quad \text{eff} \ \text{queueIn}[[qp]] := \text{tail(queueIn}[[qp])
\quad \text{nlevel} := \text{l};
\quad \text{nfrag} := \text{c};
\quad \text{if} \ \text{st} = \text{find} \text{then nstatus} := \text{find} \text{else nstatus} := \text{found fi};
\% - \text{Let} \ S = \{ [\text{p,q}] : \text{lstatus}[[\text{p,r}]] = \text{branch}, \ r \neq q \} -
\text{S} := \{\};
for pr: Link in links do
    if pr.t ≠ qp.s ∧ lstatus[pr] = branch then
        S := S ∪ {pr}
    fi
od;
for k: Link in S do
    queueOut[k] := queueOut[k] ⊢ initMsg([INITIATE, l, c, st])
od;
if st = find then
    inbranch := [qp.t, qp.s];
    bestlink := nil;
    bestwt := 10000000; % Infinity
    %TestP
    minL := nil; min := 10000000; % Infinity
    for tempL: Link in links do
        if weight[tempL] < min ∧ lstatus[tempL] = unknown then
            minL := embed(tempL); min := weight[tempL]
        fi;
    od;
    if minL ≠ nil then
        testlink := minL;
        queueOut[minL.val] := queueOut[minL.val] ⊢ testMsg([TEST, nlevel, nfrag]);
    else
        testlink := nil;
        %Report
        if findcount = 0 ∧ testlink = nil then
            nstatus := found;
            queueOut[inbranch] := queueOut[inbranch] ⊢ reportMsg([REPORT, bestwt])
        fi
        %EndReport
        fi;
    %EndTestP
    findcount := size(S)
fi

internal ReceiveTest(qp: Link, l: Int, c: Null[Edge])
pre head(queueIn[qp]) = testMsg([TEST, l, c])
eff queueIn[qp] := tail(queueIn[qp]);
if nstatus = sleeping then
    %Wakeup
    minL := embed(chooseRandom(links)); min := weight[minL.val];
    for tempL: Link in links do
        if weight[tempL] < min then minL := embed(tempL); min := weight[tempL]
    fi;
    lstatus[minL.val] := branch; nstatus := found;
    queueOut[minL.val] := queueOut[minL.val] ⊢ connMsg([CONNECT, 0]);
    %EndWakeup
    fi;
if l > nlevel then
    queueIn[qp] := queueIn[qp] ⊢ testMsg([TEST, l, c]);
else
    if c ≠ nfrag then
        queueOut[[qp.t, qp.s]] := queueOut[[qp.t, qp.s]] ⊢ msg(ACCEPT)
    else
        if lstatus[[qp.t, qp.s]] = unknown then lstatus[[qp.t, qp.s]] := rejected
        if testlink ≠ embed([[qp.t, qp.s]]) then

queueOut[[qp.t, qp.s]] := queueOut[[qp.t, qp.s]] ⊢ msg(REJECT)
else
  %Test
  minL := nil; min := 10000000; % Infinity
  for tempL:Link in links do
    if weight[tempL] < min ∧ lstatus[tempL] = unknown then
      minL := embed(tempL); min := weight[tempL]
    fi;
  od;
  if minL ≠ nil then
    testlink := minL;
    queueOut[minL.val] := queueOut[minL.val] ⊢ testMsg([TEST, nlevel, nfrag]);
  else
    testlink := nil;
    %Report
    if findcount = 0 ∧ testlink = nil then
      nstatus := found;
      queueOut[inbranch] := queueOut[inbranch] ⊢ reportMsg([REPORT, bestwt])
    fi
    %EndReport
    fi;
  fi;
fi;

internal ReceiveAccept(qp: Link)
  pre head(queueIn[qp]) = msg(ACCEPT)
  eff queueIn[qp] := tail(queueIn[qp]);
  testlink := nil;
  if weight[[qp.t, qp.s]] < bestwt then
    bestlink := embed([qp.t, qp.s]);
    bestwt := weight[[qp.t, qp.s]];
  fi;
  %Report
  if findcount = 0 ∧ testlink = nil then
    nstatus := found;
    queueOut[inbranch] := queueOut[inbranch] ⊢ reportMsg([REPORT, bestwt])
  fi
  %EndReport

internal ReceiveReject(qp: Link)
  pre head(queueIn[qp]) = msg(REJECT)
  eff queueIn[qp] := tail(queueIn[qp]);
  if lstatus[[qp.t, qp.s]] = unknown then lstatus[[qp.t, qp.s]] := rejected fi;
  %Test
  minL := nil; min := 10000000; % Infinity
  for tempL:Link in links do
    if weight[tempL] < min ∧ lstatus[tempL] = unknown then
      minL := embed(tempL); min := weight[tempL]
    fi;
  od;
  if minL ≠ nil then
    testlink := minL;
    queueOut[minL.val] := queueOut[minL.val] ⊢ testMsg([TEST, nlevel, nfrag]);
  else

19
testlink := nil;
%Report
if findcount = 0 ^ testlink = nil then
  nstatus := found;
  queueOut[inbranch] := queueOut[inbranch] ⊢ reportMsg([REPORT, bestwt])
fi
%EndReport
fi
%EndTest

internal ReceiveReport(qp: Link, w: Int)
pre head(queueIn[qp]) = reportMsg([REPORT, w])
eff queueIn[qp] := tail(queueIn[qp]);
if [qp.t, qp.s] ≠ inbranch then
  findcount := findcount -1;
  if w < bestwt then
    bestwt := w;
    bestlink := embed([qp.t, qp.s])
  fi
%Report
if findcount = 0 ^ testlink = nil then
  nstatus := found;
  queueOut[inbranch] := queueOut[inbranch] ⊢ reportMsg([REPORT, bestwt])
fi
%EndReport
else
  if nstatus = find then
    queueIn[qp] := queueIn[qp] ⊢ reportMsg([REPORT, w])
  elseif w > bestwt then
    %ChangeRoot
    if lstatus[bestlink.val] = branch then
    else
      queueOut[bestlink.val] := queueOut[bestlink.val] ⊢ connMsg([CONNECT, nlevel]);
      lstatus[bestlink.val] := branch
    fi
  %EndChangeRoot
fi
fi

internal ReceiveChangeRoot(qp: Link)
pre head(queueIn[qp]) = msg(CHANGEROOT)
eff queueIn[qp] := tail(queueIn[qp]);
%ChangeRoot
if lstatus[bestlink.val] = branch then
else
  queueOut[bestlink.val] := queueOut[bestlink.val] ⊢ connMsg([CONNECT, nlevel]);
  lstatus[bestlink.val] := branch
fi
%EndChangeRoot

GHS composition automaton
Results  One of the graphs that were used to test the algorithm is shown in Figure 2.5 (a). The (unique) minimum spanning tree computed by the algorithm is shown in Figure 2.5 (b). A snapshot of the trace follows, showing the beginning of the algorithm with node 0 waking up (after a start action), sending a CONNECT message to its minimum weight outgoing edge (1) and reporting this edge as part of the minimum spanning tree. The complete trace of this run can be found at http://theory.csail.mit.edu/~pmavrom/ioaReport.html. The algorithm ran correctly, returning the unique minimum spanning tree in all cases.

Figure 2.5: (a) shows one of the graphs used to test the GHS algorithm. In (b), the (unique) minimum spanning tree computed by the algorithm

Trace snapshot of the GHS algorithm

Initialization starts (0) on loon.csail.mit.edu at 7:05:55:830
Modified state variables:
P → [nstatus: sleeping, nfrag: nil, nlevel: 87, bestlink: nil, bestwt:
87, testlink: nil, inbranch: [s: 87, t: 87], findcount: 87, lstatus: Map{}, queueOut: Map{}, queueIn: Map{}, answered: Map{}, min: 87, minL: nil, S: ()
RM → Map{}
SM → Map{}
lkns → ([s: 0, t: 1] [s: 0, t: 4])
lnk → [s: 87, t: 87]
lkns → ()
rank → null
size → null
templ → [s: 87, t: 87]
templns → ()
weight → Map{[[s: 0, t: 1] -> 8] [[s: 0, t: 4] -> 12]}
Initialization ends
transition: input startP() in automaton GHS(0)
on loon.csail.mit.edu at 7:05:55:852
Modified state variables:
SM → Map{[1 -> Tuple , modified fields : {[toSend -> Sequence , elements added : {connMsg([msg: CONNECT, l: 0])} Elements removed : {}] [sent -> Sequence , elements added : {connMsg([msg: CONNECT, l: 0])} Elements removed : {}]}}
lnk → [s: 87, t: 87]
lkns → ()
rank → 0
size → 16
templ → [s: 0, t: 4]
templns → ()
weight → Map{[[s: 0, t: 1] -> 8] [[s: 0, t: 4] -> 12]}
transiiton: output SEND(connMsg([msg: CONNECT, l: 0]), 0, 1) in automaton GHS(0) on loon.csail.mit.edu at 7:05:55:864
Modified state variables:
SM → Map, modified entries : {[1 -> Tuple , modified fields : {[toSend -> Sequence , elements added : {connMsg([msg: CONNECT, l: 0])} Elements removed : {}}]}}
lnk → Tuple, modified fields : {[s: 0] [t -> 1]}
transition: output InTree([s: 0, t: 1]) in automaton GHS(0)
on loon.csail.mit.edu at 7:05:55:915
Modified state variables:
P → Tuple, modified fields : {[answered -> Map, modified entries : {[[s: 0, t: 1] -> true]]]}
SM → Map, modified entries : {[1 -> Tuple , modified fields : {[toSend -> Sequence , elements added : {} Elements removed : {connMsg([msg: CONNECT, l: 0])}] [sent -> Sequence , elements added : {connMsg([msg: CONNECT, l: 0])} Elements removed : {}}]}}
3 Results

Figures 3.6, 3.7 and 3.8 depict the runtime results of the algorithms on 1 node per machine, a constant number of nodes and a constant number of machines respectively. The runtimes were measured in seconds, from the time when the first node started initialization until the time at which the algorithm terminated, and they are averages on 10 runs. The tables in Appendix D show the raw data collected from our experiments. In Figure 3.6(a) to (b) the relationship between time and number of nodes is not very clear due to the short time taken, number of nodes and messages exchanged. Figure 3.6(d) however, where the number of messages exchanged is large shows the clear (linear) relation.

4 Conclusions

During the months of June and July 2004 we had the opportunity to test the capabilities and the performance of the version of the toolkit that was then available. We started by mostly modifying the Java code the Toolkit generated to get LCR up and running. The toolkit was then gradually modified to automate the manual changes we had to make in Java. By the Spanning Tree to Leader algorithm the Toolkit was completely automated to provide Java code that compiled and ran successfully. After that, and after making sure that simple algorithms such as a simple broadcast and convergecast could be run, we implemented a more complicated algorithm, the GHS algorithm for computing the minimum spanning tree in an arbitrary graph. The successful implementation of this algorithm makes us very confident that the toolkit can be used to implement complex distributed systems successfully.

Performance We now comment on some performance issues:

1. Scalability: As Figure 3.6 suggests, the Toolkit is scalable. Letting the number of nodes double increases the runtime but no more than twice the previous runtime. Moreover, Figure 3.6(e) shows that the rate of increase of the running time is smaller as the number of messages increases.

2. Setup time: The time required for MPI to set up all the connections and enable the nodes to initialize was not measured in the runtime results. However, when the number of nodes was large, this time was also quite significant (around 5-10 minutes).

3. Resource usage: The generated programs used all the available processing power available to them during the whole run. This was mainly because there was no pause between successive tests for incoming or outgoing messages.

5 Comments and Suggestions on the Toolkit

- The NDR Language, used both in the scheduler and the initialization block could be extended to support the \texttt{for v:T in s:Set[T]} statement, which is already used in IOA. This would make the code much cleaner since using the current syntax, these loops are implemented using a while loop and an extra set of two or more variables.
Figure 3.6: (a) to (d) show the time taken to complete the algorithm in regard to the number of nodes. (e) shows the time of Broadcast Leader Election in regard to the number of messages exchanged.
Figure 3.7: (a) to (c) show the time taken to terminate, in regard to the number of physical machines used.
Figure 3.8: (a) to (c) show the time taken to terminate, in regard to the number of nodes.
• Initialization of Map (or Array) entries. The toolkit allows usage of possibly non-initialized map/array entries. As a result, when the compiled code is run, a NullPointerException is thrown from the Java environment when these variables are accessed. The following example illustrates the problem and demonstrates some possible solutions:

```java
type Tup = tuple of a: Int, b: Bool
automaton Test
signature ... states
M: Map[Tup]
initially (true ⇒ SM[1].a = 3)
det do
    SM[1].a := 3
...
```

The user here tries to initialize the tuple before constructing it, which is done using a command like `SM[1] := [3, true]`. One possible solution would be for the toolkit to check for an initialization statement of the type `SM[1] := ...` in the `initially` block, before the accessing statement (of the type `SM[1].a`). If none exists, it could print a warning. Another solution could be to arbitrarily initialize all Map/Array Entries. The toolkit already supports such kind of initialization, but again if it does not occur, a warning message could be printed.

• In GHS, some parts of the IOA code were used more than once. The code given in [2] makes use of a procedure, i.e. a block of statements that can be declared and then called as many times as necessary. Given the complexity of GHS, such a feature would be quite useful during the design, coding and debugging process. For GHS, there was no need for parameter passing to the procedure. Therefore, as a first step, a simple procedure support (without parameter passing) from the toolkit would be enough.

• The use of MPI for communication has both advantages and disadvantages. Implementing the Toolkit was probably easier using MPI. No low-level communication programming was necessary. Furthermore, it has been tested to work for a long time now, and indeed, it works. However, it introduces some issues that could have been avoided. Firstly, most of the error messages coming from MPI are far from descriptive. (see Troubleshooting # 4) Moreover, MPI sets up a connection between all pairs of nodes, even if these connections are not necessary. An n-node LCR needs only n connections, while MPI sets up \( \Theta(n^2) \) connections. This is probably the reason why on a larger number of nodes, MPI takes up a lot of time to setup. We suggest that the possibility of another communication interface, which gives more control over these issues (e.g. Java RMI) is examined at some time in the future.

• Finally, we have tested the possibility of pausing between consecutive tests for incoming and outgoing messages (MPI’s `test` and `Iprobe`). Right now a node might be in an infinite loop probing for messages for tens of seconds and using up all the processing power available to it. Forcing the nodes to sleep for some milliseconds before probing again showed to improve performance when the number of nodes per physical machine is large. We have experimented with some runtimes of LCR Leader election on 20 nodes on a single machine, using different sleep times. The results are shown below:
<table>
<thead>
<tr>
<th>Sleep time (milliseconds)</th>
<th>LCR on 20 nodes, 1 machine runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>40</td>
</tr>
<tr>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td>121</td>
</tr>
</tbody>
</table>

References


A Troubleshooting

1. Sometimes during the composition of the process and mediator automata, two SEND and/or two RECEIVE transitions were produced by the composer. This should be fixed before generating Java code because the code will not compile later. To prevent this one should make sure that the where clauses on the SEND and RECEIVE actions in the process automaton match the where clauses for the SendMediator and ReceiveMediator component automata in the node composition.

2. Send/Receive Convention. In LCR, we have used the convention of [1] for the Send and Receive transitions: Send(m, i, j) and Receive(m, j, i). This means that i is always the sender and j is always the receiver. This convention sometimes caused problems that were hard to debug. We found that using a different convention made it easier to understand and debug any problems: Send(m, i, j) and Receive(m, i, j), meaning that node i sends to node j and node i receives from node j respectively.

3. Currently the NDR Language, used in the schedule and initially blocks does not support the for v:T in S:Set[T]. The way to create such loops is to use a while loop of the form:

```java
uses ChooseSet
...

schedule
states
    tempNbrs: Set[Int],
    k: Int

    do
        tempNbrs := P.nbrs;
        while (~isEmpty(tempNbrs)) do
            k := chooseRandom(tempNbrs);
            tempNbrs := delete(k, tempNbrs);
            \% fire action on neighbor k
        od;
    od
```

4. MPI throws a SIGSEV error if the program tries to access a node using a negative number as the rank.

5. Several times when running an algorithm we encountered a `java.lang.NullPointerException` during the initialization or the first transitions of the automata. Almost always, this was due to a wrong initialization of the state variables in IOA.

B Bug Reports

1. In the LCRNode automaton some types such as Status and _States[LCRProcess] were not declared automatically. Manual declaration of these types was necessary. The latest version
of the toolkit generates these type definitions automatically. [Fixed]

2. The transitions that connect to the MPI have different signatures, but the implementation of the toolkit did not support that. (e.g. `Iprobe(i_v1, j_v2)` and `resp_Iprobe(i_v3, j_v4)` were not recognized as an MPI pair). We manually changed them to have exactly the same signature, for example `Iprobe(i_v1, j_v2)` and `resp_Iprobe(i_v1, j_v2)`. The latest version of the toolkit recognizes them as pairs even if they have different names. [Fixed]

3. In LCR a statement `(const 0)` was produced that was not recognized by the code generator. We manually changed the IL file produced to `(const v999)`, and declared the variable `v999` to something like `(v999 zero s8 (scope 1))` where `s8` was the sort corresponding to NatSort. [Pending]

4. Formal parameters were not translated in Java. We manually declared the field in the generated code, and initialized it to the correct value. The toolkit now translates all formal parameters and automatically initializes three special parameters: rank, size (of type Int or Nat) and hostName (of type String). [Fixed]

5. The initialization of the arguments from MPI (the statement `MPI.Init(args);`) had to happen in the `main()` method of the class LCRNode and not `ioa.runtime.Automaton`. Otherwise, the environment would not recognize the correct number of processes running. The toolkit has been modified so that this will happen automatically from now on. [Fixed]

6. The composer tool (incorrectly) removes all the preconditions of internal actions during composition. [Pending]

7. In the NDR Language translation, no `break;` statement was produced after a fire block. This caused unexpected termination in some algorithms. [Fixed]

8. IntSort’s modulo operator is not compatible with its specification. [Pending]

9. The syntax of declaring a map is `v: Map[D1, ..., Dn, R]` where `n ≥ 1`. If only one type is given, (e.g. `map1: Map[Int]`), the checker tool throws a `java.lang.InternalError` instead of a more descriptive Syntax Error message.
C Automata Definitions and Input Files

C.1 SendMediator and ReceiveMediator

These automata were used as the channel between two nodes and the connection of the automata to the MPI.

SendMediator

```
type sCall = enumeration of idle, Isend, test

automaton SendMediator(Msg, Node:Type, i:Node, j:Node)

  assumes Infinite(Handle)

  signature
    input SEND(m: Msg, const i, const j)
    output Isend(m: Msg, const i, const j)
    input resp_Isend(handle:Handle, const i, const j)
    output test(handle:Handle, const i, const j)
    input resp_test(flag:Bool, const i, const j)

  states
    status: sCall := idle,
    toSend: Seq[Msg] := {},
    sent: Seq[Msg] := {},
    handles: Seq[Handle] := {}

  transitions
    input SEND(m, i, j)
      eff toSend := toSend ⊢ m
    output Isend(m, i, j)
      pre head(toSend) = m;
      status := idle
      eff toSend := tail(toSend);
      sent := sent ⊢ m;
      status := Isend
    input resp_Isend(handle, i, j)
      eff handles := handles ⊢ handle;
      status := idle
    output test(handle, i, j)
      pre status = idle;
      handle = head(handles)
      eff status := test
    input resp_test(flag, i, j)
      eff if (flag = true) then
        handles := tail(handles);
        sent := tail(sent)
        fi;
      status := idle
```

ReceiveMediator

```
type rCall = enumeration of idle, receive, Iprobe
```
automaton ReceiveMediator(Msg, Node: Type, i: Node, j: Node)

assumes Infinite(Handle)

signature
  output RECEIVE(m: Msg, const i, const j)
  output Iprobe(const i, const j)
  input resp_Iprobe(flag: Bool, const i, const j)
  output receive(const i, const j)
  input resp_receive(m: Msg, const i, const j)

states
  status: rCall := idle,
  toRecv: Seq[Msg] := {},
  ready: Bool := false

transitions
  output RECEIVE(m, i, j)
    pre m = head(toRecv)
    eff toRecv := tail(toRecv)
  output Iprobe(i, j)
    pre status = idle;
    ready = false
    eff status := Iprobe
  input resp_Iprobe(flag, i, j)
    eff ready := flag;
    status := idle
  output receive(i, j)
    pre ready = true;
    status := idle
    eff status := receive
  input resp_receive(m, i, j)
    eff toRecv := toRecv ⊢ m;
    ready := false;
    status := idle

C.2 LCR

Expanded automaton with initialization and scheduling

uses ChoiceSet(Int)

automaton LCR(rank: Int, size: Int)

signature
  output receive(N6: Int, N7: Int)
    where N7 = rank ∧ N6 = mod((rank+size) -1, size)
  output SEND(m: Int, I2: Int, I3: Int)
    where I3 = mod(rank + 1, size) ∧ I2 = rank
  input resp_Iprobe(flag: Bool, N4: Int, N5: Int)
    where N5 = rank ∧ N4 = mod((rank+size) -1, size)
  input resp_test(flag: Bool, N18: Int, N19: Int)
    where N19 = mod(rank + 1, size) ∧ N18 = rank
  input resp_receive(m: Int, N8: Int, N9: Int)
    where N9 = rank ∧ N8 = mod((rank+size) -1, size)
  input vote
output test(handle: Handle, N16: Int, N17: Int)
  where N17 = mod(rank + 1, size) ∧ N16 = rank
output Iprobe(N2: Int, N3: Int)
  where N3 = rank ∧ N2 = mod((rank+size) -1, size)
output RECEIVE(m: Int, I0: Int, I1: Int)
  where I1 = rank ∧ I0 = mod((rank+size) -1, size)
output leader(I4: Int) where I4 = rank
input resp_Isend(handle: Handle, N14: Int, N15: Int)
  where N15 = mod(rank + 1, size) ∧ N14 = rank
output Isend(m: Int, N12: Int, N13: Int)
  where N13 = mod(rank + 1, size) ∧ N12 = rank

states
P: _States[LCRProcess],
RM: Map[Int, _States[ReceiveMediator, Int, Int]],
SM: Map[Int, _States[SendMediator, Int, Int]]
initially
(true ⇒ P.pend = {rank} ∧ P.status = idle) ∧
∀ j: Int
  ((j = mod((rank+size) -1, size)
  ⇒
    RM[j].status = idle
    ∧ RM[j].torecv = {}
    ∧ RM[j].read = false
    ∧ (j = mod((rank+size) -1, size) ⇔ defined(RM, j)))
  ∧
∀ j: Int
  ((j = mod(rank + 1, size)
  ⇒
    SM[j].status = idle
    ∧ SM[j].tosend = {}
    ∧ SM[j].sent = {}
    ∧ SM[j].handles = {})
  ∧ (j = mod(rank + 1, size) ⇔ defined(SM, j)))
det do
  P.pend := {rank};
P.status := idle;
RM[mod((rank+size) -1, size)] := [idle, {}, false];
SM[mod(rank+1, size)] := [idle, {}, {}, {}]
od

transitions
output receive(N6, N7)
  pre RM[mod((rank+size) -1, size)].read = true
  ∧ RM[mod((rank+size) -1, size)].status = idle
  eff RM[mod((rank+size) -1, size)].status := receive
output SEND(m, I2, I3)
  pre P.status ≠ idle ∧ m ∈ (P.pend)
  eff SM[mod(rank + 1, size)].tosend :=
    (SM[mod(rank + 1, size)].tosend) ⊔ m;
P.pend := delete(m, P.pend)
input resp_Iprobe(flag, N4, N5)
  eff RM[mod((rank+size) -1, size)].read := flag;
  RM[mod((rank+size) -1, size)].status := idle
input resp_test(flag, N18, N19)
  eff if flag = true then
    SM[mod(rank + 1, size)].handles :=
33
tail(SM[mod(rank + 1, size)].handles);
SM[mod(rank + 1, size)].sent :=
tail(SM[mod(rank + 1, size)].sent)
fi;
SM[mod(rank + 1, size)].status := idle

input

resp_receive(m, N8, N9)
eff RM[mod((rank+size) -1, size)].toRecv :=
(RM[mod((rank+size) -1, size)].toRecv) ⊢ m;
RM[mod((rank+size) -1, size)].ready := false;
RM[mod((rank+size) -1, size)].status := idle

input vote

eff P.status := voting

output

test(handle, N16, N17)

pre SM[mod(rank + 1, size)].status = idle
∧ handle = head(SM[mod(rank + 1, size)].handles)
eff SM[mod(rank + 1, size)].status := test

output

Iprobe(N2, N3)

pre RM[mod((rank+size) -1, size)].status = idle
∧ RM[mod((rank+size) -1, size)].ready = false
eff RM[mod((rank+size) -1, size)].status := Iprobe

output

RECEIVE(m, I0, I1) where m > I1 ∨ m < I1 ∨ m = I1

pren m = head(RM[mod((rank+size) -1, size)].toRecv)
eff if m > I1 then P.pending := insert(m, P.pending)
else if m = I1 then P.status := elected

fi;
RM[mod((rank+size) -1, size)].toRecv :=
tail(RM[mod((rank+size) -1, size)].toRecv)

output

leader(I4)

pre P.status = elected
eff P.status := announced

input

resp_Isend(handle, N14, N15)
eff SM[mod(rank + 1, size)].handles :=
(SM[mod(rank + 1, size)].handles) ⊢ handle;
SM[mod(rank + 1, size)].status := idle

output

Isend(m, N12, N13)

pre head(SM[mod(rank + 1, size)].toSend) = m
∧ SM[mod(rank + 1, size)].status = idle
eff SM[mod(rank + 1, size)].toSend :=
tail(SM[mod(rank + 1, size)].toSend);
SM[mod(rank + 1, size)].sent :=
(SM[mod(rank + 1, size)].sent) ⊢ m;
SM[mod(rank + 1, size)].status := Isend

schedule

do
fire input vote;
while (true) do
if P.pending ≠ {} then
fire output SEND(chooseRandom(P.pending), rank, mod(rank+1, size)) fi;
if SM[mod(rank+1, size)].status = idle
∧ SM[mod(rank+1, size)].toSend ≠ {} then
fire output
Isend(head(SM[mod(rank+1, size)].toSend), rank, mod(rank+1, size)) fi;
if SM[mod(rank+1, size)].status = idle
∧ SM[mod(rank+1, size)].handles ≠ {} then
fire output

test(head(SM[mod(rank+1, size)].handles), rank, mod(rank+1, size)) fi;

34
if \( \text{RM}[\text{mod}((\text{rank}+\text{size}) - 1, \text{size})].\text{status} = \text{idle} \)
\( \wedge \text{RM}[\text{mod}((\text{rank}+\text{size}) - 1, \text{size})].\text{ready} = \text{false} \) then
fire output \( \text{Iprobe} (\text{rank}, \text{mod}(\text{rank}+1, \text{size})) \)
fi;
if \( \text{RM}[\text{mod}((\text{rank}+\text{size}) - 1, \text{size})].\text{status} = \text{idle} \)
\( \wedge \text{RM}[\text{mod}((\text{rank}+\text{size}) - 1, \text{size})].\text{ready} = \text{true} \) then
fire output receive (rank, mod((rank+1), size)) fi;
fi
fire output
RECEIVE (head(RM[mod((rank+size) - 1, size)].toRecv),
mod((rank+size) - 1, size), rank) fi;
if \( \text{P}.\text{status} = \text{elected} \) then
fire output leader(rank) fi
od
od

\textbf{type Status} = \text{enumeration of idle, voting, elected, announced}
\textbf{type rCall} = \text{enumeration of idle, receive, Iprobe}
\textbf{type sCall} = \text{enumeration of idle, Isend, test}
\textbf{type }_\text{States}[\text{LCRProcess}] = \text{tuple of pending: Set[Int], status: Status}
\textbf{type }_\text{States}[\text{ReceiveMediator}, \text{Int}, \text{Int}] = \text{tuple of status: rCall, toRecv: Seq[Int], ready: Bool}
\textbf{type }_\text{States}[\text{SendMediator}, \text{Int}, \text{Int}] = \text{tuple of status: sCall, toSend: Seq[Message], sent: Seq[Message], handles: Seq[Handle]}

\textbf{C.3 Asynchronous Spanning Tree}
\textbf{Expanded automaton with initialization and scheduling}

\textbf{type Message} = \text{enumeration of search, null}
\textbf{type rCall} = \text{enumeration of idle, receive, Iprobe}
\textbf{type sCall} = \text{enumeration of idle, Isend, test}
\textbf{type }_\text{States}[\text{sTreeProcess}] = \text{tuple of parent: Int, reported: Bool, send: Map[Int, Message]}
\textbf{type }_\text{States}[\text{SendMediator}, \text{Message}, \text{Int}] = \text{tuple of status: sCall, toSend: Seq[Message], sent: Seq[Message], handles: Seq[Handle]}
\textbf{type }_\text{States}[\text{ReceiveMediator}, \text{Message}, \text{Int}] = \text{tuple of status: rCall, toRecv: Seq[Message], ready: Bool}

\textbf{uses ChoiceSet(Int)}

\textbf{automaton sTree}(rank: \text{Int}, \text{nbrs: Set[Int]})

\textbf{signature}
output SEND(m: Message, v0: Int, v1: Int) where v0 = rank
output RECEIVE(m: Message, v0: Int, v1: Int) where v0 = rank
output PARENT(j3: Int)
output test(handle: Handle, v16: Int, v17: Int) where v16 = rank
input resp_test(flag: Bool, v18: Int, v19: Int) where v18 = rank
output receive(v6: Int, v7: Int) where v6 = rank
input resp_receive(m: Message, v8: Int, v9: Int) where v8 = rank
output Iprobe(v2: Int, v3: Int) where v2 = rank
input resp_Iprobe(flag: Bool, v4: Int, v5: Int) where v4 = rank
output Isend(m: Message, v12: Int, v13: Int) where v12 = rank
input resp_Isend(handle: Handle, v14: Int, v15: Int) where v14 = rank

\textbf{states}
P: _\text{States}[sTreeProcess],
\[
\text{RM: Map[Int, States[ReceiveMediator, Message, Int]],}
\]
\[
\text{SM: Map[Int, States[SendMediator, Message, Int]],}
\]
\[
\% \text{ Temporary}
\]
\[
\text{tempNbrs: Set[Int], j: Int}
\]
\[
\text{initially}
\]
\[
\text{true } \Rightarrow \text{ P.parent \texttt{= -1}}
\]
\[
\text{det do}
\]
\[
\text{P.parent } \texttt{=} -1;
\]
\[
\text{P.reported } \texttt{=} \text{false;}
\]
\[
\text{tempNbrs } \texttt{=} \text{nbrs;}
\]
\[
\text{while } (\neg \text{isEmpty(tempNbrs)}) \text{ do}
\]
\[
\text{j } \texttt{=} \text{chooseRandom(tempNbrs);}
\]
\[
\text{tempNbrs } \texttt{=} \text{delete(j, tempNbrs);}
\]
\[
\text{if } \text{rank } = 0 \text{ then}
\]
\[
\text{P.send[j] } \texttt{=} \text{search}
\]
\[
\text{else}
\]
\[
\text{P.send[j] } \texttt{=} \text{null}
\]
\[
\text{fi;}
\]
\[
\text{RM[j]} \texttt{=} \text{[idle, {}, false];}
\]
\[
\text{SM[j]} \texttt{=} \text{[idle, {}, {}, {}]}
\]
\[
\text{od}
\]
\[
\text{od}
\]
\[
\text{transitions}
\]
\[
\text{output SEND(m, v10, v11) where v10 = rank}
\]
\[
\text{pre P.send[v11] } \texttt{=} \text{search}
\]
\[
\text{eff SM[v11].toSend } \texttt{=} (SM[v11].toSend) \vdash m;
\]
\[
\text{P.send[v11] } \texttt{=} \text{null}
\]
\[
\text{output RECEIVE(m, v0, v1)}
\]
\[
\text{pre m } \texttt{=} \text{head(RM[v1].toRecv)}
\]
\[
\text{eff if rank } \neq 0 \land \text{P.parent } = -1 \text{ then}
\]
\[
\text{P.parent } \texttt{=} v1;
\]
\[
\text{for k: Int in nbrs - \{v1\} do}
\]
\[
\text{P.send[k] } \texttt{=} \text{search}
\]
\[
\text{od}
\]
\[
\text{fi;}
\]
\[
\text{RM[v1].toRecv } \texttt{=} \text{tail(RM[v1].toRecv)}
\]
\[
\text{output PARENT(j3)}
\]
\[
\text{pre P.parent } = j3 \land \text{P.reported } = \text{false}
\]
\[
\text{eff P.reported } \texttt{=} \text{true}
\]
\[
\text{output receive(v6, v7)}
\]
\[
\text{pre RM[v7].ready } = \text{true } \land \text{RM[v7].status } = \text{idle}
\]
\[
\text{eff RM[v7].status } \texttt{=} \text{receive}
\]
\[
\text{input resp_receive(m, v6, v7)}
\]
\[
\text{eff RM[v7].toRecv } \texttt{=} (RM[v7].toRecv) \vdash m;
\]
\[
\text{RM[v7].ready } \texttt{=} \text{false;}
\]
\[
\text{RM[v7].status } \texttt{=} \text{idle}
\]
\[
\text{output test(handle, v16, v17)}
\]
\[
\text{pre SM[v17].status } = \text{idle } \land \text{handle } = \text{head(SM[v17].handles)}
\]
\[
\text{eff SM[v17].status } \texttt{=} \text{test}
\]
\[
\text{input resp_test(flag, v18, v19)}
\]
\[
\text{eff if flag } = \text{true then}
\]
\[
\text{SM[v19].handles } \texttt{=} \text{tail(SM[v19].handles)};
\]
\[
\text{SM[v19].sent } \texttt{=} \text{tail(SM[v19].sent)}
\]
\[
\text{fi;}
\]
\[
\text{SM[v19].status } \texttt{=} \text{idle}
\]
output Iprobe(v2, v3)
    pre RM[v3].status = idle ∧ RM[v3].ready = false
    eff RM[v3].status := Iprobe
input resp_Iprobe(flag, v2, v3)
    eff RM[v3].ready := flag;
    RM[v3].status := idle
output Isend(m, v12, v13)
    pre head(SM[v13].toSend) = m ∧ SM[v13].status = idle
    eff SM[v13].toSend := tail(SM[v13].toSend);
    SM[v13].sent := (SM[v13].sent) ⊢ m;
    SM[v13].status := Isend
input resp_Isend(handle, v12, v13)
    eff SM[v13].handles := (SM[v13].handles) ⊢ handle;
    SM[v13].status := idle

C.4 Asynchronous Broadcast Convergecast

Expanded automaton with initialization and scheduling

```plaintext
type Kind = enumeration of bcast, ack
type BCastMsg = tuple of kind: Kind, w: Int
type Message = union of msg: BCastMsg, kind: Kind
type rCall = enumeration of idle, receive, Iprobe
type sCall = enumeration of idle, Isend, test
type _States[bcastProcess] = tuple of val:Int, acked: Set[Int],
    parent: Int, reported:
    Bool, send: Map[Int, Seq[Message]], temp: BCastMsg
type _States[SendMediator, Message, Int] = tuple of status: sCall, toSend:
```

---

37
Seq[Message], sent : Seq[Message], handles : Seq[Handle]
type _States[ReceiveMediator, Message, Int] = tuple of status: rCall, toRecv: Seq[Message], ready: Bool

uses ChoiceSet(Int)

automaton bcast(rank: Int, nbrs: Set[Int])
    assumes bcastNode_Axioms
    assumes P_Axioms
    assumes RM_Axioms
    assumes SM_Axioms

signature
    output receive(v6: Int, v7: Int) where v6 = rank
    output SEND(m: Message, v10: Int, v11: Int) where v10 = rank
    input resp_Iprobe(flag: Bool, v4: Int, v5: Int) where v4 = rank
    internal report(v2: Int) where v2 = rank
    input resp_receive(m: Message, v8: Int, v9: Int) where v8 = rank
    output test(handle: Handle, v16: Int, v17: Int) where v16 = rank
    output Iprobe(v2: Int, v3: Int) where v2 = rank
    output RECEIVE(m: Message, v0: Int, v1: Int) where v0 = rank
    input resp_Isend(handle: Handle, v14: Int, v15: Int) where v14 = rank
    output Isend(m: Message, v12: Int, v13: Int) where v12 = rank

states
    P: _States[bcastProcess],
    RM: Map[Int, _States[ReceiveMediator, Message, Int]],
    SM: Map[Int, _States[SendMediator, Message, Int]],

%%Temporary variables
    tempNbrs : Set[Int] := {},
    j: Int

initially
    true ⇒ P.val = 99

det do
    if rank = 0 then
        P.val := 99;
        (P.temp).kind := bcast;
        (P.temp).w := P.val;
        tempNbrs := nbrs;
        while (~isEmpty(tempNbrs)) do
            j := chooseRandom(tempNbrs);
            tempNbrs := delete(j, tempNbrs);
            P.send[j] := {} ⊢ msg(P.temp)
        od
    else
        P.val := -1;
        tempNbrs := nbrs;
        while (~isEmpty(tempNbrs)) do
            j := chooseRandom(tempNbrs);
            tempNbrs := delete(j, tempNbrs);
            P.send[j] := {};
        od
    fi
    P.parent := -1;
    P.reported := false;
    P.acked := {};}
tempNbrs := nbrs;
while (~isEmpty(tempNbrs)) do
    j := chooseRandom(tempNbrs);
    tempNbrs := delete(j, tempNbrs);
    RM[j] := [idle, {}, false];
    SM[j] := [idle, {}, {}, {}]
od

transitions
output receive(v6, v7)
  pre RM[v7].ready = true \land RM[v7].status = idle
  eff RM[v7].status := receive
output SEND(m, v10, v11) where v10 = rank
  pre m = head(P.send[v11])
  eff SM[v11].toSend := (SM[v11].toSend) \cup \{ m \};
  P.send[v11] := tail(P.send[v11])
input resp_Iprobe(flag, v4, v5)
  eff RM[v5].ready := flag;
  RM[v5].status := idle
internal report(v2)
  % The preconditions were (incorrectly) removed
  eff if rank = 0 then P.reported := true
  elseif rank \neq 0 then
    P.send[P.parent] := P.send[P.parent] \cup kind(ack);
    P.reported := true
fi
input resp_test(flag, v18, v19)
  eff if flag = true then
    SM[v19].handles := tail(SM[v19].handles);
    SM[v19].sent := tail(SM[v19].sent)
fi;
SM[v19].status := idle
input resp_receive(m, v8, v9)
  eff RM[v9].toRecv := (RM[v9].toRecv) \cup \{ m \};
  RM[v9].ready := false;
  RM[v9].status := idle
output test(handle, v16, v17)
  pre SM[v17].status = idle \land handle = head(SM[v17].handles)
  eff SM[v17].status := test
output Iprobe(v2, v3)
  pre RM[v3].status = idle \land RM[v3].ready = false
  eff RM[v3].status := Iprobe
output RECEIVE(m, v0, v1)
  pre m = head(RM[v1].toRecv)
  eff if m = kind(ack) then P.acked := (P.acked) \cup \{ v1 \}
  else
    if P.val = -1 then
      P.val := (m.msg).w;
      P.parent := v1;
      for k : Int in (nbrs) - \{ v1 \} do
        P.send[k] := P.send[k] \cup m
      od
    else
      P.send[v1] := P.send[v1] \cup kind(ack)
    fi
  fi;
RM[v1].toRecv := tail(RM[v1].toRecv)
input resp_Isend(handle, v14, v15)
eff SM[v15].handles := (SM[v15].handles) \+ handle;
SM[v15].status := idle
output Isend(m, v12, v13)
pre head(SM[v13].toSend) = m \& SM[v13].status = idle
eff SM[v13].toSend := tail(SM[v13].toSend);
SM[v13].sent := (SM[v13].sent) \+ m;
SM[v13].status := Isend

schedule
states
tempNbrs2: Set[Int],
k: Int
do
while (true) do
  tempNbrs2 := nbrs;
  while (!isEmpty(tempNbrs2)) do
    k := chooseRandom(tempNbrs2);
    tempNbrs2 := delete(k, tempNbrs2);
    if P.send[k] \neq {} then
      fire output SEND(head(P.send[k]), rank, k) fi;
    if SM[k].status = idle \& SM[k].toSend \neq {} then
      fire output Isend(head(SM[k].toSend), rank, k) fi;
    if SM[k].status = idle \& SM[k].handles \neq {} then
      fire output test(head(SM[k].handles), rank, k) fi;
    if RM[k].status = idle \& RM[k].ready = false then
      fire output Iprobe(rank, k) fi;
    if RM[k].status = idle \& RM[k].ready = true then
      fire output receive(rank, k) fi;
    if RM[k].toRecv \neq {} then
      fire output RECEIVE(head(RM[k].toRecv), rank, k) fi
  od;
  if rank = 0 \& P.acked = nbrs \& P.reported = false then
    fire internal report(rank) fi;
  if rank \neq 0 \& P.parent \neq -1 \& P.acked = nbrs \& P.reported = false then
    fire internal report(rank) fi
od

C.5 Leader Election using Asynchronous Broadcast Convergecast

Expanded automaton with initialization and scheduling

type Kind = enumeration of bcast, ack
type BCastMsg = tuple of kind: Kind, w: Int
type AckMsg = tuple of kind: Kind, mx: Int
type MSG = union of bmsg: BCastMsg, amsg: AckMsg, kind: Kind
type Message = tuple of msg: MSG, source: Int
type rCall = enumeration of idle, receive, Iprobe
type sCall = enumeration of idle, Isend, test
type _States[bcastLeaderProcess] = tuple of val: Map[Int, Int],

40
parent: Map[Int, Int], reported: Map[Int, Bool], acked:
  Map[Int, Set[Int]], send: Map[Int, Int, Seq[Message]], max:
  Map[Int, Int], elected: Bool, announced: Bool

<table>
<thead>
<tr>
<th>parent</th>
<th>Map[Int, Int], reported: Map[Int, Bool], acked:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Map[Int, Set[Int]], send: Map[Int, Int, Seq[Message]], max:</td>
</tr>
<tr>
<td></td>
<td>Map[Int, Int], elected: Bool, announced: Bool</td>
</tr>
</tbody>
</table>

**_States[ReceiveMediator, Message, Int] = tuple of status: rCall, toRecv: Seq[Message], ready: Bool**

**_States[SendMediator, Message, Int] = tuple of status: sCall, toSend: Seq[Message], sent: Seq[Message], handles: Seq[Handle]**

**_States[bcastLeaderNode] = tuple of P: _States[bcastLeaderProcess], RM: Map[Int, _States[ReceiveMediator, Message, Int]], SM: Map[Int, _States[SendMediator, Message, Int]]**

**uses ChoiceSet (Int)**

**automaton bcastLeader(rank: Int, size:Int, nprs:Set[Int])**

**signature**

output receive(N6: Int, N7: Int) where N6 = rank

output SEND(m: Message, N10: Int, N11: Int) where N10 = rank

input resp_Iprobe(flag: Bool, N4: Int, N5: Int) where N4 = rank

internal finished

internal report(I2: Int, source: Int) where I2 = rank

output LEADER

input resp_test(flag: Bool, N18: Int, N19: Int) where N18 = rank

input resp_receive(m: Message, N8: Int, N9: Int) where N8 = rank

output test(handle: Handle, N16: Int, N17: Int) where N16 = rank

output Iprobe(N2: Int, N3: Int) where N2 = rank

output RECEIVE(m: Message, N0: Int, N1: Int) where N0 = rank

input resp_Isend(handle: Handle, N14: Int, N15: Int) where N14 = rank

output Isend(m: Message, N12: Int, N13: Int) where N12 = rank

**states**

P: _States[bcastLeaderProcess],

RM: Map[Int, _States[ReceiveMediator, Message, Int]],

SM: Map[Int, _States[SendMediator, Message, Int]],

tempNbrs: Set[Int], c: Int,

j: Int

**initially**

(true

⇒

∀ j: Int

(0 ≤ j ∧ j < 16

⇒

(rank = j

⇒

P.val[j] = rank

∧ (rank ≠ j ⇒ P.val[j] = -1)

∧ P.parent[j] = -1

∧ P.acked[j] = {}

∧ P.max[j] = rank

∧

∀ k: Int

(P.send[j, k] = {}

∧ k ∈ nprs

41
\[ \neg \text{rank} = j \Rightarrow \]
\[ \text{P. send}[j, k] = \{ \} \vdash [\text{bmsg}([\text{bcast}, 99]), j]\}
\[ \wedge \neg \text{P. elected} = \text{false} \]
\[ \forall j: \text{Int} \]
\[ (\text{RM}[j]. \text{status} = \text{idle} \wedge \text{RM}[j]. \text{toRecv} = \{\} \wedge \text{RM}[j]. \text{ready} = \text{false} \wedge \text{defined}(\text{RM}, j)) \]
\[ \wedge \forall j: \text{Int} \]
\[ (\text{SM}[j]. \text{status} = \text{idle} \wedge \text{SM}[j]. \text{toSend} = \{\} \wedge \text{SM}[j]. \text{sent} = \{\} \wedge \text{SM}[j]. \text{handles} = \{\} \wedge \text{defined}(\text{SM}, j)) \]

\text{det do}
\[ c := \text{size}; \]
\text{while} (c > 0) \text{ do}
\[ c := c - 1; \]
\[ \text{if} (\text{rank} = c) \text{ then} \]
\[ \text{P. val}[c] := \text{rank} \]
\[ \text{else} \]
\[ \text{P. val}[c] := -1 \]
\[ \text{fi}; \]
\[ \text{P. parent}[c] := -1; \]
\[ \text{P. acked}[c] := \{\}; \]
\[ \text{P. max}[c] := \text{rank}; \]
\[ \text{P. reported}[c] := \text{false}; \]
\[ \text{P. elected} := \text{false}; \]
\[ \text{P. announced} := \text{false}; \]
\[ \text{tempNbrs} := \text{nbrs}; \]
\text{while} (\neg \text{isEmpty}(\text{tempNbrs})) \text{ do}
\[ j := \text{chooseRandom}(\text{tempNbrs}); \]
\[ \text{tempNbrs} := \text{delete}(j, \text{tempNbrs}); \]
\[ \text{P. send}[c, j] := \{\}; \]
\[ \text{if} c = \text{rank} \text{ then} \]
\[ \text{P. send}[c, j] := \{\} \vdash [\text{bmsg}([\text{bcast}, 99]), c] \]
\[ \text{fi}; \]
\[ \text{od}; \]
\[ \text{tempNbrs} := \text{nbrs}; \]
\text{while} (\neg \text{isEmpty}(\text{tempNbrs})) \text{ do}
\[ j := \text{chooseRandom}(\text{tempNbrs}); \]
\[ \text{tempNbrs} := \text{delete}(j, \text{tempNbrs}); \]
\[ \text{RM}[j] := [\text{idle}, \{\}, \text{false}]; \]
\[ \text{SM}[j] := [\text{idle}, \{\}, \{\}, \{\}] \]
\[ \text{od}; \]
\[ \text{od}; \]
\text{transitions}
\text{output receive}(\text{N6}, \text{N7})
pre RM[N7].ready = true ∧ RM[N7].status = idle
  eff RM[N7].status := receive
output SEND(m, N10, N11) where N10 = rank
  pre m = head(P.send[m.source, N11])
  eff SM[N11].toSend := (SM[N11].toSend) ⊢ m;
  P.send[m.source, N11] := tail(P.send[m.source, N11])
input resp_Iprobe(flag, N4, N5)
  eff RM[N5].ready := flag;
  RM[N5].status := idle
output LEADER
  pre P.elected = true ∧ P.announced = false
  eff P.announced := true

internal finished
% preconditions didn’t pass through
pre P.acked[rank] = nbrs ∧
  P.reported[rank] = false
  eff P.reported[rank] := true;
  if (P.max[rank] = rank) then
    P.elected := true
  fi;
  P.reported := true;
internal report(I2, source)
  eff P.send[source, P.parent[source]] :=
    P.send[source, P.parent[source]]
    ⊢ [amsg([ack, P.max[source]]), source];
  P.reported[source] := true
input resp_test(flag, N18, N19)
  eff if flag = true then
    SM[N19].handles := tail(SM[N19].handles);
    SM[N19].sent := tail(SM[N19].sent)
  fi;
  SM[N19].status := idle
input resp_receive(m, N8, N9)
  eff RM[N9].toRecv := (RM[N9].toRecv) ⊢ m;
  RM[N9].ready := false;
  RM[N9].status := idle
output test(handle, N16, N17)
  pre SM[N17].status = idle ∧ handle = head(SM[N17].handles)
  eff SM[N17].status := test
output Iprobe(N2, N3)
  pre RM[N3].status = idle ∧ RM[N3].ready = false
  eff RM[N3].status := Iprobe
output RECEIVE(m, N0, N1)
  pre m = head(RM[N1].toRecv)
  eff if m.msg = kind(ack) then
    P.acked[m.source] := P.acked[m.source] ⊔ {N1}
  elseif tag(m.msg) = amsg then
    if P.max[m.source] < (((m.msg).amsg).mx) then
      P.max[m.source] := ((m.msg).amsg).mx
    fi;
    P.acked[m.source] := P.acked[m.source] ⊔ {N1}
  else
    if P.val[m.source] = -1 then
      P.val[m.source] := ((m.msg).bmsg).w;
    fi;
  fi;
  if P.acked[m.source] ⊔ {N1} ⊓ m then
    P.acked[m.source] := P.acked[m.source] ⊔ {N1}
  fi;
43
P.parent[m.source] := N1;
for k: Int in (nbrs) - {N1} do
    P.send[m.source, k] := P.send[m.source, k] ⊢ m
next
else
    P.send[m.source, N1] :=
    P.send[m.source, N1] ⊢ [kind(ack), m.source]
fi
fi;
RM[N1].toRecv := tail(RM[N1].toRecv)
input resp_Isend(handle, N14, N15)
eff SM[N15].handles := (SM[N15].handles) ⊢ handle;
SM[N15].status := idle
output Isend(m, N12, N13)
pre head(SM[N13].toSend) = m ∧ SM[N13].status = idle
eff SM[N13].toSend := tail(SM[N13].toSend);
SM[N13].sent := (SM[N13].sent) ⊢ m;
SM[N13].status := Isend

schedule
states
c2: Int, %% Source
tempNbrs2: Set[Int],
k: Int
do
while(true) do
    c2 := size;
    while (c2 > 0) do
        c2 := c2 - 1;
        tempNbrs2 := nbrs;
        while (¬isEmpty(tempNbrs2)) do
            k := chooseRandom(tempNbrs2);
            tempNbrs2 := delete(k, tempNbrs2);
            if P.send[c2, k] ≠ {} then
                fire output SEND(head(P.send[c2, k]), rank, k) fi;
            if SM[k].status = idle ∧ SM[k].toSend ≠ {} then
                fire output Isend(head(SM[k].toSend), rank, k) fi;
            if SM[k].status = idle ∧ SM[k].handles ≠ {} then
                fire output test(head(SM[k].handles), rank, k) fi;
            if RM[k].status = idle ∧ RM[k].ready = false then
                fire output Iprobe(rank, k) fi;
            if RM[k].status = idle ∧ RM[k].ready = true then
                fire output receive(rank, k) fi;
            if RM[k].toRecv ≠ {} then
                fire output RECEIVE(head(RM[k].toRecv), rank, k) fi
        od;
    if c2 ≠ rank ∧ P.parent[c2] ≠ -1 ∧
      P.acked[c2] = nbrs - {P.parent[c2]} ∧
      P.reported[c2] = false then
        fire internal report(rank, c2) fi;
    if c2 = rank ∧ P.acked[rank] = nbrs ∧
      P.reported[rank] = false then
        fire internal finished fi;
    if P.elected = true ∧ P.announced = false then
C.6 Spanning Tree to Leader Election

Expanded automaton with initialization and scheduling

uses ChoiceSet(Int)

automaton sTreeLeader(rank: Int, nbrs: Set[Int])

signature
output receive(N6: Int, N7: Int) where N6 = rank
output SEND(m: Message, N10: Int, N11: Int) where N10 = rank
input resp_Iprobe(flag: Bool, N4: Int, N5: Int) where N4 = rank
input resp_test(flag: Bool, N18: Int, N19: Int) where N18 = rank
input resp_receive(m: Message, N8: Int, N9: Int) where N8 = rank
output test(handle: Handle, N16: Int, N17: Int) where N16 = rank
output Iprobe(N2: Int, N3: Int) where N2 = rank
output RECEIVE(m: Message, N0: Int, N1: Int) where N0 = rank
output leader
input resp_Isend(handle: Handle, N14: Int, N15: Int) where N14 = rank
output Isend(m: Message, N12: Int, N13: Int) where N12 = rank

states
P: _States[sTreeLeaderProcess],
RM: Map[Int, _States[ReceiveMediator, Message, Int]],
SM: Map[Int, _States[SendMediator, Message, Int]],
% Temporary storage
j:

initially
(true ⇒

P.receivedElect = {}
∧ P.sentElect = {}
∧ P.status = idle
∧ (size(nbrs) = 1 ⇒

P.send[chooseRandom(nbrs)]
= P.send[chooseRandom(nbrs)] ⊢ elect)
∧ ∀ j: Int
{(j ∈ nbrs ⇒
(RM[j].status = idle
∧ RM[j].toRecv = {})
∧ RM[j].ready = false
∧ defined(RM, j)))
∧ ∀ j: Int
{(j ∈ nbrs ⇒
(SM[j].status = idle
∧ SM[j].toSend = {})
∧ SM[j].sent = {}
∧ SM[j].handles = {}
∧ defined(SM, j)))

det do
P.receivedElect := {};
P.sentElect := {};

45
P.status := idle;
tempNbrs := nbrs;
while (~isEmpty(tempNbrs)) do
j := chooseRandom(tempNbrs);
tempNbrs := delete(j, tempNbrs);
P.send[j] := {}; 
RM[j] := [idle, {}, false]; 
SM[j] := [idle, {}, {}, {}]
end;
if size(nbrs) = 1 then
P.send[chooseRandom(nbrs)] := {} ⊢ elect
fi
end

transitions

output receive(N6, N7)

pre RM[N7].ready = true ∧ RM[N7].status = idle
eff RM[N7].status := receive

output SEND(m, N10, N11) where N10 = rank

pre m = head(P.send[N11])
eff SM[N11].toSend := (SM[N11].toSend) ⊢ m;
P.send[N11] := tail(P.send[N11])

input resp_Iprobe(flag, N4, N5)
eff RM[N5].ready := flag;
RM[N5].status := idle

input resp_test(flag, N18, N19)
eff if flag = true then
SM[N19].handles := tail(SM[N19].handles);
SM[N19].sent := tail(SM[N19].sent)
fi;
SM[N19].status := idle

input resp_receive(m, N8, N9)
eff RM[N9].toRecv := (RM[N9].toRecv) ⊢ m;
RM[N9].ready := false;
RM[N9].status := idle

output test(handle, N16, N17)

pre SM[N17].status = idle ∧ handle = head(SM[N17].handles)
eff SM[N17].status := test

output Iprobe(N2, N3)

pre RM[N3].status = idle ∧ RM[N3].ready = false
eff RM[N3].status := Iprobe

output RECEIVE(m, N0, N1)

pre m = head(RM[N1].toRecv)
eff P.receivedElect := insert(N1, P.receivedElect);
if size(P.receivedElect) = size(nbrs) - 1 then
P.send[chooseRandom(nbrs - (P.receivedElect))] := 
P.send[chooseRandom(nbrs - (P.receivedElect))] ⊢ elect;
P.sentElect :=
insert
(chooseRandom(nbrs - (P.receivedElect)), P.sentElect)
elseif P.receivedElect = nbrs then
if N1 ∈ (P.sentElect) then
if N0 > N1 then P.status := elected
else P.status := elected
fi
fi
RM[N1].toRecv := tail(RM[N1].toRecv)
output leader
pre P. status = elected
eff P. status := announced
input resp_Isend(handle, N14, N15)
eff SM[N15]. handles := (SM[N15]. handles) \uplus handle;
SM[N15]. status := idle
output Isend(m, N12, N13)
pre head(SM[N13]. toSend) = m \land SM[N13]. status = idle
eff SM[N13]. toSend := tail(SM[N13]. toSend);
SM[N13]. sent := (SM[N13]. sent) \uplus m;
SM[N13]. status := Isend

schedule
states
tempNbrs2: Set[Int],
k: Int
do
while(true) do
tempNbrs2 := nbrs;
while (¬isEmpty(tempNbrs2)) do
  k := chooseRandom(tempNbrs2);
tempNbrs2 := delete(k, tempNbrs2);
if P.send[k] \neq \{\} then
  fire output SEND(head(P.send[k]), rank, k) fi;
if SM[k]. status = idle \land SM[k]. toSend \neq \{\} then
  fire output Isend(head(SM[k]. toSend), rank, k) fi;
if SM[k]. status = idle \land SM[k]. handles \neq \{\} then
  fire output test(head(SM[k]. handles), rank, k) fi;
if RM[k]. status = idle \land RM[k]. ready = false then
  fire output Iprobe(rank, k) fi;
if RM[k]. status = idle \land RM[k]. ready = true then
  fire output receive(rank, k) fi;
if RM[k]. toRecv \neq \{\} then
  fire output RECEIVE(head(RM[k]. toRecv), rank, k) fi
od;
if P. status = elected then fire output leader fi
od

C.7 GHS
Expanded automaton with initialization and scheduling
uses ChoiceSet(Link)

automaton
GHS(rank: Int, size: Int, links: Set[Link], weight: Map[Link, Int])
signature
  input resp_receive(m: Message, N8: Int, N9: Int) where N8 = rank
  internal ReceiveChangeRoot(qp: Link)
  input resp_Iprobe(flag: Bool, N4: Int, N5: Int) where N4 = rank
  output RECEIVE(m: Message, i: Int, j: Int) where i = rank
  input resp_Isend(handle: Handle, N14: Int, N15: Int) where N14 = rank
  output Isend(m: Message, N12: Int, N13: Int) where N12 = rank
  internal ReceiveConnect(qp: Link, l: Int)
  internal ReceiveReport(qp: Link, w: Int)
  internal ReceiveAccept(qp: Link)
  output NotInTree(l: Link)
  output InTree(l: Link)
  output Iprobe(N2: Int, N3: Int) where N2 = rank
  internal ReceiveTest(qp: Link, l: Int, c: Null[Edge])
  input resp_test(flag: Bool, N18: Int, N19: Int) where N18 = rank
  output SEND(m: Message, N10: Int, N11: Int) where N10 = rank
  output receive(N6: Int, N7: Int) where N6 = rank
  output test(handle: Handle, N16: Int, N17: Int) where N16 = rank
  internal ReceiveReject(qp: Link)
  input startP
states
  P: _States[GHSProcess],
  RM: Map[Int, _States[ReceiveMediator, Message, Int]],
  SM: Map[Int, _States[SendMediator, Message, Int]],
% Temporary
  templinks: Set[Link], tempL: Link
initially
  (true
  ⇒
    P.nstatus = sleeping
    ∧ P.nfrag = nil
    ∧ P.nlevel = 0
    ∧ P.bestlink = embed(chooseRandom(links))
    ∧ P.bestwt = weight[chooseRandom(links)]
    ∧ P.testlink = nil
    ∧ P.inbranch = chooseRandom(links)
    ∧ P.findcount = 0
    ∧
    ∀ l: Link
    (l ∈ links
    ⇒
      P.lstatus[l] = unknown
      ∧ P.answered[l] = false
      ∧ P.queueOut[l] = {}
      ∧ P.queueIn[l] = {}))
  ∧
  ∀ j: Int
  (RM[j].status = idle
  ∧ RM[j].toRecv = {}
  ∧ RM[j].ready = false
  ∧ defined(RM, j))
∀ j: Int
(SM[j].status = idle ∧ SM[j].toSend = {} ∧ SM[j].sent = {} ∧ SM[j].handles = {} ∧ defined(SM, j))

det do
  P.nstatus := sleeping;
P.nfrag := nil;
P.nlevel := 0;
P.bestlink := embed(chooseRandom(links));
P.bestwt := weight(chooseRandom(links));
P.testlink := nil;
P.inbranch := chooseRandom(links);
P.findcount := 0;
tempLinks := links;
while (~isEmpty(tempLinks)) do
  tempL := chooseRandom(tempLinks);
tempLinks := delete(tempL, tempLinks);
P.lstatus[tempL] := unknown;
P.answered[tempL] := false;
P.queueOut[tempL] := {};
P.queueIn[[tempL.t, tempL.s]] := {};
RM[tempL.t] := [idle, {}, false];
SM[tempL.t] := [idle, {}, {}, {}]
od
od

transitions
input resp_receive(m, N8, N9)
eff RM[N9].toRecv := (RM[N9].toRecv ⊢ m);
RM[N9].ready := false;
RM[N9].status := idle

internal ReceiveChangeRoot(qp)
eff P.queueIn[qp] := tail(P.queueIn[qp]);
if P.lstatus[(P.bestlink).val] = branch then
  P.queueOut[(P.bestlink).val] :=
  P.queueOut[(P.bestlink).val] ⊢ msg(CHANGERoot)
else
  P.queueOut[(P.bestlink).val] :=
  P.queueOut[(P.bestlink).val] ⊢ connMsg([CONNECT, P.nlevel]);
P.lstatus[(P.bestlink).val] := branch
fi

input resp_Iprobe(flag, N4, N5)
eff RM[N5].ready := flag;
RM[N5].status := idle
output RECEIVE(m, i, j)
pre m = head(RM[j].toRecv)
eff P.queueIn[[j, i]] := P.queueIn[[j, i]] ⊢ m;
RM[j].toRecv := tail(RM[j].toRecv)

input resp_Isend(handle, N14, N15)
eff SM[N15].handles := (SM[N15].handles) ⊢ handle;
SM[N15].status := idle
output Isend(m, N12, N13)
pre head(SM[N13].toSend) = m ∧ SM[N13].status = idle
eff SM[N13].toSend := tail(SM[N13].toSend);
SM[N13].sent := (SM[N13].sent) ⊢ m;
SM[N13].status := Isend

internal ReceiveConnect(qp, l)
pre head(P.queueIn[qp]) = connMsg([CONNECT, l])
eff P.queueIn[qp] := tail(P.queueIn[qp]);
if P.nstatus = sleeping then
    P.minL := embed(chooseRandom(links));
P.min := weight([P.minL].val);
for tempL: Link in links do
    if weight[tempL] < (P.min) then
        P.minL := embed(tempL);
P.min := weight[tempL]
fi;
P.lstatus[(P.minL).val] := branch;
P.nstatus := found;
P.queueOut[(P.minL).val] :=
P.queueOut[(P.minL).val] ⊢ connMsg([CONNECT, 0])
fi;
if l < (P.nlevel) then
    P.lstatus[(qp.t, qp.s)] := branch;
if P.testlink ≠ nil then
    P.queueOut[(qp.t, qp.s)] :=
P.queueOut[(qp.t, qp.s)] ⊢ initMsg([INITIATE, P.nlevel, P.nfrag, find]);
P.findcount := (P.findcount) + 1
else
    P.queueOut[(qp.t, qp.s)] :=
P.queueOut[(qp.t, qp.s)] ⊢ initMsg([INITIATE, P.nlevel, P.nfrag, found])
fi
else
    if P.lstatus[(qp.t, qp.s)] = unknown then
        P.queueIn[qp] := P.queueIn[qp] ⊢ connMsg([CONNECT, 1])
else
        P.queueOut[(qp.t, qp.s)] :=
P.queueOut[(qp.t, qp.s)] ⊢ initMsg
            ([INITIATE,
            (P.nlevel) + 1,
            embed([qp.t, qp.s]),
            find])
fi

internal ReceiveReport(qp, w)
pre head(P.queueIn[qp]) = reportMsg([REPORT, w])
eff P.queueIn[qp] := tail(P.queueIn[qp]);
if [qp.t, qp.s] ≠ P.inbranch then
    P.findcount := (P.findcount) - 1;
    if w < (P.bestwt) then
        P.bestwt := w;
P.bestlink := embed([qp.t, qp.s])
fi;
if \( P.\text{findcount} = 0 \land P.\text{testlink} = \text{nil} \) then
\[
P.\text{nstatus} := \text{found};
\]
\[
P.\text{queueOut}[P.\text{inbranch}] :=
\]
\[
  P.\text{queueOut}[P.\text{inbranch}] \vdash \text{reportMsg}([\text{REPORT}, P.\text{bestwt}])
\]
fi
else
if \( P.\text{nstatus} = \text{find} \) then
\[
P.\text{queueIn}[qp] := P.\text{queueIn}[qp] \vdash \text{reportMsg}([\text{REPORT}, w])
\]
elseif \( w > (P.\text{bestwt}) \) then
if \( P.\text{lstatus}[(P.\text{bestlink}).\text{val}] = \text{branch} \) then
\[
P.\text{queueOut}[(P.\text{bestlink}).\text{val}] :=
\]
\[
  P.\text{queueOut}[(P.\text{bestlink}).\text{val}] \vdash \text{msg}(\text{CHANGEROOT})
\]
else
\[
P.\text{queueOut}[(P.\text{bestlink}).\text{val}] :=
\]
\[
  P.\text{queueOut}[(P.\text{bestlink}).\text{val}]
\]
\[
  \vdash \text{connMsg}([\text{CONNECT}, P.\text{nlevel}])
\]
\[
P.\text{lstatus}[(P.\text{bestlink}).\text{val}] := \text{branch}
\]
fi
fi
fi

internal \( \text{ReceiveAccept}(qp) \)
\[
\text{pre head}(P.\text{queueIn}[qp]) = \text{msg}(\text{ACCEPT})
\]
\[
\text{eff P.\text{queueIn}[qp]} := \text{tail}(P.\text{queueIn}[qp]);
\]
\[
P.\text{testlink} := \text{nil};
\]
\[
\text{if weight}([qp.t, qp.s]) < (P.\text{bestwt}) \text{ then}
\]
\[
P.\text{bestlink} := \text{embed}([qp.t, qp.s]);
\]
\[
P.\text{bestwt} := \text{weight}([qp.t, qp.s])
\]
fi;
if \( P.\text{findcount} = 0 \land P.\text{testlink} = \text{nil} \) then
\[
P.\text{nstatus} := \text{found};
\]
\[
P.\text{queueOut}[P.\text{inbranch}] :=
\]
\[
  P.\text{queueOut}[P.\text{inbranch}] \vdash \text{reportMsg}([\text{REPORT}, P.\text{bestwt}])
\]
fi

internal \( \text{ReceiveInitiate}(qp, l, c, st) \)
\[
\text{pre head}(P.\text{queueIn}[qp]) = \text{initMsg}([\text{INITIATE}, l, c, st])
\]
\[
\text{eff P.\text{queueIn}[qp]} := \text{tail}(P.\text{queueIn}[qp]);
\]
\[
P.\text{nlevel} := l;
\]
\[
P.\text{nfrag} := c;
\]
\[
\text{if st = find then P.nstatus := find else P.nstatus := found fi;}
\]
\[
P.S := \{\}
\]
\[
\text{for pr : Link in links do}
\]
\[
\text{if pr.t} \neq qp.s \land P.lstatus[pr] = \text{branch then}
\]
\[
P.S := (P.S) \cup \{pr\}
\]
fi
\[
\text{od;}
\]
\[
\text{for k: Link in P.S do}
\]
\[
P.\text{queueOut}[k] := P.\text{queueOut}[k] \vdash \text{initMsg}([\text{INITIATE}, l, c, st])
\]
\[
\text{od;}
\]
\[
\text{if st = find then}
\]
\[
P.\text{inbranch} := [qp.t, qp.s];
\]
\[
P.\text{bestlink} := \text{nil};
\]
\[
P.\text{bestwt} := 1000;
\]
\[
P.\text{minL} := \text{nil};
\]
\[
P.\text{min} := 1000;
\]
\[
\text{for tempL: Link in links do}
\]
\[
\text{if weight[tempL] < (P.min) \land P.lstatus[tempL] = unknown}
\]
then
P. minL := embed(tempL);
P. min := weight[tempL]
fi
od;
if P. minL ≠ nil then
P. testlink := P. minL;
P. queueOut[(P. minL).val] :=
P. queueOut[(P. minL).val] ⊢ testMsg([TEST, P. nlevel, P. nfrag])
else
P. testlink := nil;
if P. findcount = 0 ∧ P. testlink = nil then
P. nstatus := found;
P. queueOut[P. inbranch] :=
P. queueOut[P. inbranch] ⊢ reportMsg([REPORT, P. bestwt])
fi
fi;
P. findcount := size(P. S)
fi
output NotInTree(l)
pre P. answered[l] = false ∧ P. lstatus[l] = rejected
eff P. answered[l] := true
output InTree(l)
pre P. answered[l] = false ∧ P. lstatus[l] = branch
eff P. answered[l] := true
output Iprobe(N2, N3)
pre RM[N3]. status = idle ∧ RM[N3]. ready = false
eff RM[N3]. status := Iprobe
internal ReceiveTest(qp, l, c)
pre head(P. queueIn[qp]) = testMsg([TEST, l, c])
eff P. queueIn[qp] := tail(P. queueIn[qp]);
if P. nstatus = sleeping then
P. minL := embed(chooseRandom(links));
P. min := weight[(P. minL).val];
for templ: Link in links do
if weight[templ] < (P. min) then
P. minL := embed(templ);
P. min := weight[templ]
fi
od;
P. lstatus[(P. minL).val] := branch;
P. nstatus := found;
P. queueOut[(P. minL).val] :=
P. queueOut[(P. minL).val] ⊢ connMsg([CONNECT, 0])
fi;
if l > (P. nlevel) then
P. queueIn[qp] := P. queueIn[qp] ⊢ testMsg([TEST, l, c])
else
if c ≠ P. nfrag then
P. queueOut[[qp.t, qp.s]] :=
P. queueOut[[qp.t, qp.s]] ⊢ msg(ACCEPT)
else
if P. lstatus[[qp.t, qp.s]] = unknown then
P. lstatus[[qp.t, qp.s]] := rejected
52
if $P.\text{testlink} \neq \text{embed}([qp.t, qp.s])$ then
  $P.\text{queueOut}([qp.t, qp.s]) := P.\text{queueOut}([qp.t, qp.s]) \vdash \text{msg(REJECT)}$
else
  $P.\text{minL} := \text{nil}$;
  $P.\text{min} := 1000$;
  for tempL : Link in links do
    if weight[tempL] < $(P.\text{min})$
      $P.\text{minL} := \text{embed}(\text{tempL})$;
      $P.\text{min} := \text{weight}(\text{tempL})$
    fi
  od;
  if $P.\text{minL} \neq \text{nil}$ then
    $P.\text{testlink} := P.\text{minL}$;
    $P.\text{queueOut}([P.\text{minL}.\text{val}]) := P.\text{queueOut}([P.\text{minL}.\text{val}]) \vdash \text{testMsg}([\text{TEST}, P.\text{nlevel}, P.\text{nfrag}])$
  else
    $P.\text{testlink} := \text{nil}$;
    if $P.\text{findcount} = 0 \land P.\text{testlink} = \text{nil}$ then
      $P.\text{status} := \text{found}$;
      $P.\text{queueOut}[P.\text{inbranch}] := P.\text{queueOut}[P.\text{inbranch}] \vdash \text{reportMsg}([\text{REPORT}, P.\text{bestwt}])$
    fi
  fi
fi

\textbf{input} \texttt{resp\_test(flag, N18, N19)}
\begin{itemize}
  \item \textbf{eff} if $flag = \text{true}$ then
    \begin{itemize}
      \item $\text{SM}[N19].\text{handles} := \text{tail}(\text{SM}[N19].\text{handles})$;
      \item $\text{SM}[N19].\text{sent} := \text{tail}(\text{SM}[N19].\text{sent})$
    \end{itemize}
  fi
\end{itemize}
\textbf{output} \texttt{SEND(m, N10, N11)} where $N10 = \text{rank}$
\begin{itemize}
  \item \textbf{pre} $m = \text{head}(\text{P.\text{queueOut}}([N10, N11]))$
  \item \textbf{eff} $\text{SM}[N11].\text{toSend} := (\text{SM}[N11].\text{toSend}) \vdash m$;
  \item $\text{P.\text{queueOut}}([N10, N11]) := \text{tail}(\text{P.\text{queueOut}}([N10, N11]))$
\end{itemize}
\textbf{output} \texttt{receive(N6, N7)}
\begin{itemize}
  \item \textbf{pre} $\text{RM}[N7].\text{ready} = \text{true} \land \text{RM}[N7].\text{status} = \text{idle}$
  \item \textbf{eff} $\text{RM}[N7].\text{status} := \text{receive}$
\end{itemize}
\textbf{output} \texttt{test(handle, N16, N17)}
\begin{itemize}
  \item \textbf{pre} $\text{SM}[N17].\text{status} = \text{idle} \land \text{handle} = \text{head}(\text{SM}[N17].\text{handles})$
  \item \textbf{eff} $\text{SM}[N17].\text{status} := \text{test}$
\end{itemize}
\textbf{internal} \texttt{ReceiveReject(qp)}
\begin{itemize}
  \item \textbf{pre} $\text{head}(\text{P.\text{queueIn}}[qp]) = \text{msg(REJECT)}$
  \item \textbf{eff} $\text{P.\text{queueIn}}[qp] := \text{tail}(\text{P.\text{queueIn}}[qp])$;
    \begin{itemize}
      \item \textbf{if} $\text{P.\text{lstatus}}([qp.t, qp.s]) = \text{unknown}$ then
        $\text{P.\text{lstatus}}([qp.t, qp.s]) := \text{rejected}$
      fi
    \end{itemize}
    $P.\text{minL} := \text{nil}$;
    $P.\text{min} := 1000$;
    for tempL : Link in links do 
if weight[tempL] < (P.min) ∧ P.lstatus[tempL] = unknown then
P.minL := embed(tempL);
P.min := weight[tempL]
fi
od;
if P.minL ≠ nil then
P.testlink := P.minL;
P.queueOut[(P.minL).val] :=
P.queueOut[(P.minL).val]
⊢ testMsg([TEST, P.nlevel, P.nfrag])
else
P.testlink := nil;
if P.findcount = 0 ∧ P.testlink = nil then
P.nstatus := found;
P.queueOut[P.inbranch] :=
P.queueOut[P.inbranch] ⊢ reportMsg([REPORT, P.bestwt])
fi
fi
input startP
eff if P.nstatus = sleeping then
P.minL := embed(chooseRandom(links));
P.min := weight[(P.minL).val];
for tempL: Link in links do
  if weight[tempL] < (P.min) then
    P.minL := embed(tempL);
P.min := weight[tempL]
  fi
od;
P.lstatus[(P.minL).val] := branch;
P.nstatus := found;
P.queueOut[(P.minL).val] :=
P.queueOut[(P.minL).val] ⊢ connMsg([CONNECT, 0])
fi

schedule
states
lnks: Set[Link],
lnk : Link
do
  fire input startP;
  while(true) do
    lnks := links;
    while (!isEmpty(lnks)) do
      lnk := chooseRandom(lnks);
      lnks := delete(lnk, lnks);
      if P.queueOut[lnk] ≠ {} then
        fire output SEND(head(P.queueOut[lnk]), rank, lnk.t) fi;
      if SM[lnk.t].status = idle ∧ SM[lnk.t].toSend ≠ {} then
        fire output Isend(head(SM[lnk.t].toSend), rank, lnk.t) fi;
      if SM[lnk.t].status = idle ∧ SM[lnk.t].handles ≠ {} then
        fire output test(head(SM[lnk.t].handles), rank, lnk.t) fi;
      if RM[lnk.t].status = idle ∧ RM[lnk.t].ready = false then
        fire output Iprobe(rank, lnk.t) fi;
      if RM[lnk.t].status = idle ∧ RM[lnk.t].ready = true then
        fire output receive(rank, lnk.t) fi;
      if RM[lnk.t].toRecv ≠ {} then

fire output RECEIVE(head(RM[lk.t].toRecv), rank, lk.t) fi;
if P. queueIn[[lk.t, lk.s]] \neq \{\} \^ tag(head(P. queueIn[[lk.t, lk.s]])) = connMsg then
  fire internal ReceiveConnect([[lk.t, lk.s],
  (head(P. queueIn[[lk.t, lk.s]])).connMsg.l]) fi;
if P. queueIn[[lk.t, lk.s]] \neq \{\} \^ tag(head(P. queueIn[[lk.t, lk.s]])) = initMsg then
  fire internal ReceiveInitiate([[lk.t, lk.s],
  (head(P. queueIn[[lk.t, lk.s]])).initMsg.l, (head(P. queueIn[[lk.t, lk.s]])).initMsg.c, (head(P. queueIn[[lk.t, lk.s]])).initMsg.st]) fi;
if P. queueIn[[lk.t, lk.s]] \neq \{\} \^ head(P. queueIn[[lk.t, lk.s]]) = msg(ACCEPT) then
  fire internal ReceiveAccept([[lk.t, lk.s]]) fi;
if P. queueIn[[lk.t, lk.s]] \neq \{\} \^ head(P. queueIn[[lk.t, lk.s]]) = msg(REJECT) then
  fire internal ReceiveReject([[lk.t, lk.s]]) fi;
if P. queueIn[[lk.t, lk.s]] \neq \{\} \^ tag(head(P. queueIn[[lk.t, lk.s]])) = reportMsg then
  fire internal ReceiveReport([[lk.t, lk.s],
  (head(P. queueIn[[lk.t, lk.s]])).reportMsg.w]) fi;
if P. answered[lk] = false \^ P.lstatus[lk] = branch then
  fire output InTree(lk) fi;
if P. answered[lk] = false \^ P.lstatus[lk] = rejected then
  fire output NotInTree(lk) fi
od
od

type Nstatus = enumeration of sleeping, find, found

type Edge = tuple of s: Int, t: Int

type Link = tuple of s: Int, t: Int

type Lstatus = enumeration of unknown, branch, rejected

type Msg = enumeration of CONNECT, INITIATE, TEST, REPORT, ACCEPT, REJECT,
  CHANGEROOT

type ConnMsg = tuple of msg: Msg, l: Int

type Status = enumeration of find, found

type InitMsg = tuple of msg: Msg, l: Int, c: Null[Edge], st: Status

type TestMsg = tuple of msg: Msg, l: Int, c: Null[Edge]

type ReportMsg = tuple of msg: Msg, w: Int

type Message = union of connMsg: ConnMsg, initMsg: InitMsg, testMsg:
TestMsg, reportMsg: ReportMsg, msg: Msg

\[\text{type } rCall = \text{enumeration of } \text{idle, receive, Iprobe}\]
\[\text{type } sCall = \text{enumeration of } \text{idle, Isend, test}\]

\[\text{type } \_\text{States[GSProcess]} = \text{tuple of } \nstatus: Nstatus, \ nfrag: \text{Null[Edge]}, \ nlevel: \text{Int}, \text{bestlink: Null[Link], bestwt: Int, testlink: Null[Link], inbranch: Link, findcount: Int, lstatus: Map[Link, Lstatus], queueOut: Map[Link, Seq[Message]], queueIn: Map[Link, Seq[Message]], answered: Map[Link, Bool], min: Int, minL: Null[Link], S: Set[Link]}\]

\[\text{type } \_\text{States[ReceiveMediator, Message, Int]} = \text{tuple of } \status: rCall, \ \text{toRecv: Seq[Message], ready: Bool}\]

\[\text{type } \_\text{States[SendMediator, Message, Int]} = \text{tuple of } \status: sCall, \ \text{toSend: Seq[Message], sent: Seq[Message], handles: Seq[Handle]}\]

### D RuntimeTables

<table>
<thead>
<tr>
<th>Physical machines</th>
<th>Nodes</th>
<th>Messages</th>
<th>Run time average (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>39</td>
<td>63</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>39</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>39</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>39</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>20</td>
<td>39</td>
<td>13</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>39</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>23</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>31</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>47</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>23</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>31</td>
<td>9</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>39</td>
<td>9</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>47</td>
<td>10</td>
</tr>
</tbody>
</table>

Table D.1: Runtime results of the LCR Leader Election Algorithm
<table>
<thead>
<tr>
<th>Physical machines</th>
<th>Nodes</th>
<th>Messages</th>
<th>Run time average (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

Table D.2: Runtime results of the Asynchronous Spanning Tree algorithm

<table>
<thead>
<tr>
<th>Physical machines</th>
<th>Nodes</th>
<th>Messages</th>
<th>Run time average (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16</td>
<td>66</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>66</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>66</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>66</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>66</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>66</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>66</td>
<td>11</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>66</td>
<td>9</td>
</tr>
<tr>
<td>16</td>
<td>24</td>
<td>106</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>16</td>
<td>48</td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td></td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>26</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>46</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>66</td>
<td>9</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>86</td>
<td>15</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>106</td>
<td>16</td>
</tr>
</tbody>
</table>

Table D.3: Runtime results of the Asynchronous Broadcast Convergecast algorithm
### Table D.4: Runtime results of the Leader Election using the Asynchronous Broadcast Convergecast Algorithm

<table>
<thead>
<tr>
<th>Physical machines</th>
<th>Nodes</th>
<th>Messages</th>
<th>Run time average (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>24</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>208</td>
<td>26</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>552</td>
<td>42</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>1048</td>
<td>64</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>1720</td>
<td>83</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>2530</td>
<td>96</td>
</tr>
</tbody>
</table>

### Table D.5: Runtime results of the Spanning Tree to Leader Election Algorithm

<table>
<thead>
<tr>
<th>Physical machines</th>
<th>Nodes</th>
<th>Messages</th>
<th>Run time average (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16</td>
<td>16</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>16</td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>11</td>
</tr>
<tr>
<td>16</td>
<td>24</td>
<td>24</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>48</td>
<td>48</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>24</td>
<td>9</td>
</tr>
</tbody>
</table>

### Table D.6: Runtime results of the GHS Minimum Spanning Tree algorithm

<table>
<thead>
<tr>
<th>Physical machines</th>
<th>Nodes</th>
<th>Messages</th>
<th>Run time average (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9</td>
<td>89</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>213</td>
<td>30</td>
</tr>
</tbody>
</table>
E Traces of runs

E.1 LCR on 8 nodes

Complete trace

Initialization starts (0) on loon.csail.mit.edu at 7:25:29:615
Modified state variables:
P → [pending: (), status: idle]
RM → Map{}
SM → Map{}
rank → null
size → null
Initialization ends

transition: input vote() in automaton LCR(0)
on loon.csail.mit.edu at 7:25:29:625
Modified state variables:
P → [pending: (0), status: voting]
RM → Map{[7 -> [status: idle, toRecv: {}, ready: false]} }
SM → Map{[1 -> [status: idle, toSend: {}, sent: {}, handles: {}]]}
rank → 0
size → 8

transition: output SEND(0, 0, 1) in automaton LCR(0)
on loon.csail.mit.edu at 7:25:29:629
Modified state variables:
P → Tuple, modified fields: {[pending -> ()] }
SM → Map, modified entries: {[1 -> Tuple, modified fields: {[toSend ->
Sequence, elements added: {0 } } Elements removed: {}}]}

Initialization starts (2) on parrot.csail.mit.edu at 7:25:29:820
Modified state variables:
P → [pending: (), status: idle]
RM → Map{}
SM → Map{}
rank → null
size → null
Initialization ends

Initialization starts (3) on tui.csail.mit.edu at 7:25:29:927
transition: input vote() in automaton LCR(2)
on parrot.csail.mit.edu at 7:25:29:960
Modified state variables:
P → [pending: (2), status: voting]
RM → Map{[1 -> [status: idle, toRecv: {}, ready: false]} }
SM → Map{[3 -> [status: idle, toSend: {}, sent: {}, handles: {}]]}
rank → 2
size → 8

Modified state variables:
P → [pending: (), status: idle]
RM → Map{}
SM → Map{}
rank → null
size → null
Initialization ends

Initialization starts (7) on tui.csail.mit.edu at 7:25:30:169
transition: output SEND(2, 2, 3) in automaton LCR(2)
on parrot.csail.mit.edu at 7:25:30:200
Modified state variables:
P → Tuple, modified fields: {[pending -> ()]}
SM → Map, modified entries: {[3 → Tuple, modified fields: {[toSend -> Sequence, elements added: {2 } Elements removed: {]}]}}

Modified state variables:
P → [pending: (), status: idle]
RM → Map{}
SM → Map{}
 rank → null
size → null
Initialization ends
Initialization starts (6) on parrot.csail.mit.edu at 7:25:30:247
Modified state variables:
P → [pending: (), status: idle]
RM → Map{}
SM → Map{}
 rank → null
size → null
Initialization ends
Initialization starts (5) on condor.csail.mit.edu at 7:25:30:286
transition: input vote() in automaton LCR(6)
on parrot.csail.mit.edu at 7:25:30:313
Modified state variables:
P → [pending: (6), status: voting]
RM → Map{{5 -> [status: idle, toRecv: {}, ready: false]}}
SM → Map{{7 -> [status: idle, toSend: {}, sent: {}, handles: {}]}}
 rank → 6
size → 8

Modified state variables:
P → [pending: (), status: idle]
RM → Map{}
SM → Map{}
 rank → null
size → null
Initialization ends
transition: input vote() in automaton LCR(5)
on condor.csail.mit.edu at 7:25:30:426
Modified state variables:
P → [pending: (5), status: voting]
RM → Map{{4 -> [status: idle, toRecv: {}, ready: false]}}
SM → Map{{6 -> [status: idle, toSend: {}, sent: {}, handles: {}]}}
 rank → 5
size → 8

transition: input vote() in automaton LCR(3)
on tui.csail.mit.edu at 7:25:30:460
Modified state variables:
P → [pending: (3), status: voting]
RM → Map{{2 -> [status: idle, toRecv: {}, ready: false]}}
SM → Map{{4 -> [status: idle, toSend: {}, sent: {}, handles: {}]}}
 rank → 3
size → 8
transition: output SEND(6, 6, 7) in automaton LCR(6)
on parrot.csail.mit.edu at 7:25:30:497
Modified state variables:
P → Tuple, modified fields: {[pending -> () ]}
SM → Map, modified entries: {[7 -> Tuple, modified fields: {[toSend -> Sequence, elements added: {6 }] Elements removed: {}}]}

transition: input vote() in automaton LCR(7)
on tui.csail.mit.edu at 7:25:30:710
Modified state variables:
P → [pending: (7), status: voting]
RM → Map{{6 -> [status: idle, toRecv: {}, ready: false]}}
SM → Map{{0 -> [status: idle, toSend: {}, sent: {}, handles: {}]}}
rank → 7
size → 8

transition: output SEND(5, 5, 6) in automaton LCR(5)
on condor.csail.mit.edu at 7:25:30:737
Modified state variables:
P → Tuple, modified fields: {[pending -> () ]}
SM → Map, modified entries: {[6 -> Tuple, modified fields: {[toSend -> Sequence, elements added: {5 }] Elements removed: {}}]}

transition: output SEND(3, 3, 4) in automaton LCR(3)
on tui.csail.mit.edu at 7:25:30:959
Modified state variables:
P → Tuple, modified fields: {[pending -> () ]}
SM → Map, modified entries: {[4 -> Tuple, modified fields: {[toSend -> Sequence, elements added: {3 }] Elements removed: {}}]}

transition: output SEND(7, 7, 0) in automaton LCR(7)
on tui.csail.mit.edu at 7:25:30:989
Modified state variables:
P → Tuple, modified fields: {[pending -> () ]}
SM → Map, modified entries: {[0 -> Tuple, modified fields: {[toSend -> Sequence, elements added: {7 }] Elements removed: {}}]}

Initialization starts (1) on condor.csail.mit.edu at 7:25:31:086
Modified state variables:
P → [pending: (), status: idle]
RM → Map{}
SM → Map{}
rank → null
size → null
Initialization ends

transition: input vote() in automaton LCR(1)
on condor.csail.mit.edu at 7:25:31:347
Modified state variables:
P → [pending: (1), status: voting]
RM → Map{{0 -> [status: idle, toRecv: {}, ready: false]}}
SM → Map{{2 -> [status: idle, toSend: {}, sent: {}, handles: {}]}}
rank → 1
size → 8

transition: output RECEIVE(7, 7, 0) in automaton LCR(0)
on loon.csail.mit.edu at 7:25:31:445
Modified state variables:
P → Tuple, modified fields: {
  [pending -> (7)]
}
SM → Map, modified entries: {
  [1 -> Tuple, modified fields: {
    [toSend -> Sequence, elements added: {} Elements removed: {0}]
    [sent -> Sequence, elements added: {0} Elements removed: {}]
  } ]
}

transition: output SEND(7, 0, 1) in automaton LCR(0)
on loon.csail.mit.edu at 7:25:31:448
Modified state variables:
P → Tuple, modified fields: {
  [pending -> ()]
}
SM → Map, modified entries: {
  [1 -> Tuple, modified fields: {
    [toSend -> Sequence, elements added: {7} Elements removed: {}]
    [sent -> Sequence, elements added: {0} Elements removed: {}]
  } ]
}

on condor.csail.mit.edu at 7:25:31:489
Modified state variables:
P → Tuple, modified fields: {
  [pending -> ()]
}
SM → Map, modified entries: {
  [1 -> Tuple, modified fields: {
    [toSend -> Sequence, elements added: {7} Elements removed: {}]
    [sent -> Sequence, elements added: {1} Elements removed: {}]
  } ]
}

transition: output SEND(1, 1, 2) in automaton LCR(1)
on condor.csail.mit.edu at 7:25:31:489

on parrot.csail.mit.edu at 7:25:32:243
Modified state variables:
SM → Map, modified entries: {
  [2 -> Tuple, modified fields: {
    [toSend -> Sequence, elements added: {7} Elements removed: {}]
    [sent -> Sequence, elements added: {1} Elements removed: {}]
  } ]
}

transition: output RECEIVE(0, 0, 1) in automaton LCR(1)
on condor.csail.mit.edu at 7:25:32:205
Modified state variables:
SM → Map, modified entries: {
  [2 -> Tuple, modified fields: {
    [toSend -> Sequence, elements added: {1} Elements removed: {}]
    [sent -> Sequence, elements added: {1} Elements removed: {}]
  } ]
}

transition: output RECEIVE(1, 1, 2) in automaton LCR(2)
on parrot.csail.mit.edu at 7:25:32:243

on tui.csail.mit.edu at 7:25:32:274
Modified state variables:
SM → Map, modified entries: {
  [3 -> Tuple, modified fields: {
    [toSend -> Sequence, elements added: {} Elements removed: {2}]
    [sent -> Sequence, elements added: {2} Elements removed: {}]
  } ]
}

transition: output RECEIVE(6, 6, 7) in automaton LCR(7)
on tui.csail.mit.edu at 7:25:32:274
Modified state variables:
SM → Map, modified entries: {
  [0 -> Tuple, modified fields: {
    [toSend -> Sequence, elements added: {} Elements removed: {7}]
    [sent -> Sequence, elements added: {7} Elements removed: {}]
  } ]
}

transition: output RECEIVE(7, 0, 1) in automaton LCR(1)
on condor.csail.mit.edu at 7:25:32:325
Modified state variables:
P → Tuple, modified fields: {
  [pending -> (7)]
}

transition: output RECEIVE(5, 5, 6) in automaton LCR(6)
on parrot.csail.mit.edu at 7:25:32:437
transition: output SEND(7, 1, 2) in automaton LCR(1)
on condor.csail.mit.edu at 7:25:32:466
Modified state variables:
P → Tuple, modified fields: {
  [pending -> (7)]
}
SM → Map, modified entries: {
  [2 -> Tuple, modified fields: {
    [toSend -> Sequence, elements added: {7} Elements removed: {}]
    [sent -> Sequence, elements added: {2} Elements removed: {}]
  } ]
}

62
Modified state variables:
SM → Map, modified entries: {
    [7 -> Tuple, modified fields: {
        [tosend -> Sequence, elements added: {6} Elements removed: {6}]
        [sent -> Sequence, elements added: {} Elements removed: {6}]
    }]
}

transition: output RECEIVE(2, 2, 3) in automaton LCR(3)
on tui.csail.mit.edu at 7:25:32:584

Modified state variables:
SM → Map, modified entries: {
    [4 -> Tuple, modified fields: {
        [tosend -> Sequence, elements added: {3} Elements removed: {3}]
        [sent -> Sequence, elements added: {} Elements removed: {3}]
    }]
}

transition: output RECEIVE(7, 1, 2) in automaton LCR(2)
on parrot.csail.mit.edu at 7:25:32:777

Modified state variables:
P → Tuple, modified fields: {{pending -> (7)}}

transition: output SEND(7, 2, 3) in automaton LCR(2)
on parrot.csail.mit.edu at 7:25:32:997

Modified state variables:
P → Tuple, modified fields: {{pending -> (7)}}
SM → Map, modified entries: {
    [3 -> Tuple, modified fields: {
        [tosend -> Sequence, elements added: {} Elements removed: {7}]
    }]
}

transition: output RECEIVE(7, 2, 3) in automaton LCR(3)
on tui.csail.mit.edu at 7:25:33:121

Modified state variables:
P → Tuple, modified fields: {{pending -> (7)}}

transition: output SEND(7, 3, 4) in automaton LCR(3)
on tui.csail.mit.edu at 7:25:33:135

Modified state variables:
P → Tuple, modified fields: {{pending -> (7)}}
SM → Map, modified entries: {
    [4 -> Tuple, modified fields: {
        [tosend -> Sequence, elements added: {7} Elements removed: {7}]
    }]
}

Initialization starts (4) on loon.csail.mit.edu at 7:25:36:355

Modified state variables:
P → [pending: (), status: idle]
RM → Map{}
SM → Map{}
rank → null
size → null
Initialization ends

transition: input vote() in automaton LCR(4)
on loon.csail.mit.edu at 7:25:36:365

Modified state variables:
P → [pending: (4), status: voting]
RM → Map{[3 -> [status: idle, toRecv: {}, ready: false]]}
SM → Map{[5 -> [status: idle, toSend: {}, sent: {}, handles: {}]]}
rank → 4
size → 8

transition: output SEND(4, 4, 5) in automaton LCR(4)
on loon.csail.mit.edu at 7:25:36:369

Modified state variables:
transition: output RECEIVE(3, 3, 4) in automaton LCR(4) on loon.csail.mit.edu at 7:25:36:553
Modified state variables:
SM → Map, modified entries: {5 → Tuple, modified fields: {[toSend → Sequence, elements added: {4} Elements removed: {}}}}

transition: output SEND(4, 4, 5) in automaton LCR(4) on condor.csail.mit.edu at 7:25:37:074
Modified state variables:
SM → Map, modified entries: {6 → Tuple, modified fields: {[toSend → Sequence, elements added: {5} Elements removed: {}} [sent → Sequence, elements added: {4} Elements removed: {}}]}

transition: output SEND(7, 5, 6) in automaton LCR(5) on parrot.csail.mit.edu at 7:25:37:280
Modified state variables:
SM → Map, modified entries: {7 → Tuple, modified fields: {[toSend → Sequence, elements added: {7} Elements removed: {}}]}

transition: output RECEIVE(7, 5, 6) in automaton LCR(6) on tui.csail.mit.edu at 7:25:37:872
Modified state variables:
P → Tuple, modified fields: { [status -> elected] }

transition: output leader(7) in automaton LCR(7)
on tui.csail.mit.edu at 7:25:37:874
Modified state variables:
P → Tuple, modified fields: { [status -> announced] }

E.2 Asynchronous Spanning Tree on 16 nodes

Complete trace

Begin initialization
Modified state variables:
P → [nbrs: (), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → null
End initialization

Begin initialization
Modified state variables:
P → [nbrs: (), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → null
End initialization

transition: input initialize() in automaton sTreeNode(6) on tui.csail.mit.edu
Modified state variables:
P → [nbrs: (10 2 5 7), parent: -1, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → 6

Begin initialization
Modified state variables:
P → [nbrs: (), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → null
End initialization

transition: input initialize() in automaton sTreeNode(0) on loon.csail.mit.edu
Modified state variables:
P → [nbrs: (1 4), parent: -1, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → 0

transition: output SEND(search, 0, 4) in automaton sTreeNode(0) on loon.csail.mit.edu
Modified state variables:
Begin initialization
Modified state variables:
P → [nbrs: (1 4), parent: -1, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → null
End initialization

transition: output SEND(search, 0, 1) in automaton sTreeNode(0) on loon.csail.mit.edu
Modified state variables:
P → [nbrs: (), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: input initialize() in automaton sTreeNode(11) on parrot.csail.mit.edu
Begin initialization
Modified state variables:
P → [nbrs: (10 15 7), parent: -1, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → 11

P → [nbrs: (), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → null
End initialization

transition: input initialize() in automaton sTreeNode(12) on parrot.csail.mit.edu
Begin initialization
Modified state variables:
P → [nbrs: (13 8), parent: -1, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → 12

transition: input initialize() in automaton sTreeNode(10) on nene.csail.mit.edu
transition: input initialize() in automaton sTreeNode(5) on parrot.csail.mit.edu

Begin initialization
Modified state variables:
P \rightarrow [nbrs: (), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM \rightarrow ioa.runtime.adt.MapSort
SM \rightarrow ioa.runtime.adt.MapSort
i \rightarrow null

End initialization

transition: input initialize() in automaton sTreeNode(1) on blackbird.csail.mit.edu

Begin initialization
Modified state variables:
P \rightarrow [nbrs: (0 2 5), parent: -1, reported: false, send: ioa.runtime.adt.MapSort]
RM \rightarrow ioa.runtime.adt.MapSort
SM \rightarrow ioa.runtime.adt.MapSort
i \rightarrow null
End initialization

Begin initialization
Modified state variables:
P \rightarrow [nbrs: (0 2 5), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM \rightarrow ioa.runtime.adt.MapSort
SM \rightarrow ioa.runtime.adt.MapSort
i \rightarrow null
End initialization

SM \rightarrow ioa.runtime.adt.MapSort
i \rightarrow 1

Modified state variables:
P \rightarrow [nbrs: (), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM \rightarrow ioa.runtime.adt.MapSort
SM \rightarrow ioa.runtime.adt.MapSort
i \rightarrow null
End initialization

Begin initialization

transition: input initialize() in automaton sTreeNode(4) on parrot.csail.mit.edu

transition: input initialize() in automaton sTreeNode(7) on blackbird.csail.mit.edu

Begin initialization
Modified state variables:
P \rightarrow [nbrs: (), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM \rightarrow ioa.runtime.adt.MapSort
SM \rightarrow ioa.runtime.adt.MapSort
i \rightarrow null
End initialization

Modified state variables:
P \rightarrow [nbrs: (11 3 6), parent: -1, reported: false, send: ioa.runtime.adt.MapSort]
RM \rightarrow ioa.runtime.adt.MapSort
SM -> ioa.runtime.adt.MapSort
i -> 7

transition: input initialize() in automaton sTreeNode(13) on
tui.csail.mit.edu

transition: input initialize() in automaton sTreeNode(9) on
condor.csail.mit.edu
Modified state variables:
P -> [nbrs : (12 14 9), parent : -1, reported : false, send:
ioa.runtime.adt.MapSort]
RM -> ioa.runtime.adt.MapSort
SM -> ioa.runtime.adt.MapSort
i -> 13

Begin initialization
Modified state variables:
Modified state variables:
P -> [nbrs : (10 13 5 8), parent : -1, reported : false, send:
ioa.runtime.adt.MapSort]
RM -> ioa.runtime.adt.MapSort
P -> [nbrs : (), parent : 87, reported : true, send : ioa.runtime.adt.MapSort]
SM -> ioa.runtime.adt.MapSort
RM -> ioa.runtime.adt.MapSort
i -> 9  SM -> ioa.runtime.adt.MapSort
i -> null

End initialization
transition: input initialize() in automaton sTreeNode(15) on
blackbird.csail.mit.edu
Modified state variables:
P -> [nbrs : (11 14), parent : -1, reported : false, send:
ioa.runtime.adt.MapSort]
RM -> ioa.runtime.adt.MapSort
SM -> ioa.runtime.adt.MapSort
i -> 15

Modified state variables:
P -> [nbrs : (1 4 6 9), parent : -1, reported : false, send:
ioa.runtime.adt.MapSort]
RM -> ioa.runtime.adt.MapSort
SM -> ioa.runtime.adt.MapSort
i -> 5

SM -> ioa.runtime.adt.MapSort
i -> null
End initialization
transition: input initialize() in automaton sTreeNode(14) on
blackbird.csail.mit.edu
Modified state variables:
P -> [nbrs : (10 13 15), parent : -1, reported : false, send:
ioa.runtime.adt.MapSort]
RM -> ioa.runtime.adt.MapSort
SM -> ioa.runtime.adt.MapSort
i → 14

Modified state variables:
P → [nbrs: (0 5 8), parent: -1, reported: false, send: 
  ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

Begin initialization
Modified state variables:
P → [nbrs: (), parent: 87, reported: true, send: 
  ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

End initialization

transition: input initialize() in automaton sTreeNode(8) on 
  blackbird.csail.mit.edu

transition: output RECEIVE(search, 4, 0) in automaton sTreeNode(4) on 
  parrot.csail.mit.edu

transition: output RECEIVE(search, 1, 0) in automaton sTreeNode(1) on 
  blackbird.csail.mit.edu

Modified state variables:
P → [nbrs: (0 2 5), parent: 0, reported: false, send: 
  ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort

Modified state variables:
P → [nbrs: (0 5 8), parent: 0, reported: false, send: 
  ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort

transition: output PARENT(0) in automaton sTreeNode(4) on 
  parrot.csail.mit.edu

Modified state variables:
P → [nbrs: (0 5 8), parent: 0, reported: true, send: 
  ioa.runtime.adt.MapSort]

transition: output PARENT(0) in automaton sTreeNode(1) on 
  blackbird.csail.mit.edu

Modified state variables:

transition: output SEND(search, 4, 8) in automaton sTreeNode(4) on 
  parrot.csail.mit.edu

Modified state variables:
P → [nbrs: (0 2 5), parent: 0, reported: true, send: 
  ioa.runtime.adt.MapSort]

transition: output SEND(search, 1, 2) in automaton sTreeNode(1) on 
  blackbird.csail.mit.edu

Modified state variables:
P → [nbrs: (0 2 5), parent: 0, reported: true, send: 
  ioa.runtime.adt.MapSort]
SM → ioa.runtime.adt.MapSort

P → [nbrs: (0 5 8), parent: 0, reported: true, send: 
  ioa.runtime.adt.MapSort]
transition: output SEND(search, 1, 5) in automaton sTreeNode(1) on blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (12 4 9), parent: -1, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → 8

transition: output SEND(search, 4, 5) in automaton sTreeNode(4) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (0 5 8), parent: 0, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 5, 1) in automaton sTreeNode(5) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (1 4 6 9), parent: 1, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output PARENT(1) in automaton sTreeNode(5) on parrot.csail.mit.edu

Modified state variables:
P → [nbrs: (1 4 6 9), parent: 1, reported: true, send: ioa.runtime.adt.MapSort]

transition: output SEND(search, 5, 4) in automaton sTreeNode(5) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (11 14 6 9), parent: -1, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → 10

transition: output RECEIVE(search, 5, 4) in automaton sTreeNode(5) on parrot.csail.mit.edu
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 5, 9) in automaton sTreeNode(5) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (1 4 6 9), parent: 1, reported: true, send: ioa.runtime.adt.MapSort]
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 5, 6) in automaton sTreeNode(5) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (1 4 6 9), parent: 1, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 9, 5) in automaton sTreeNode(9) on condor.csail.mit.edu
Modified state variables:
P → [nbrs: (10 13 5 8), parent: 5, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort

transition: output PARENT(5) in automaton sTreeNode(9) on condor.csail.mit.edu
Modified state variables:
P → [nbrs: (10 13 5 8), parent: 5, reported: true, send: ioa.runtime.adt.MapSort]

transition: output SEND(search, 9, 13) in automaton sTreeNode(9) on condor.csail.mit.edu
Modified state variables:
P → [nbrs: (10 13 5 8), parent: 5, reported: true, send: ioa.runtime.adt.MapSort]
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 9, 8) in automaton sTreeNode(9) on condor.csail.mit.edu
Modified state variables:
P → [nbrs: (10 13 5 8), parent: 5, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 9, 10) in automaton sTreeNode(9) on condor.csail.mit.edu
Modified state variables:
P → [nbrs: (10 13 5 8), parent: 5, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 8, 4) in automaton sTreeNode(8) on blackbird.csail.mit.edu

transition: output RECEIVE(search, 4, 5) in automaton sTreeNode(4) on
Modified state variables:
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

Modified state variables:
P \rightarrow [\text{nbrs: (12 4 9), parent: 4, reported: false, send: ioa.runtime.adt.MapSort}]
RM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output PARENT(4) in automaton sTreeNode(8) on blackbird.csail.mit.edu
Modified state variables:
P \rightarrow [\text{nbrs: (12 4 9), parent: 4, reported: true, send: ioa.runtime.adt.MapSort}]
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output SEND(search, 8, 9) in automaton sTreeNode(8) on blackbird.csail.mit.edu
Modified state variables:
P \rightarrow [\text{nbrs: (12 4 9), parent: 4, reported: true, send: ioa.runtime.adt.MapSort}]
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output RECEIVE(search, 8, 9) in automaton sTreeNode(8) on blackbird.csail.mit.edu
Modified state variables:
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output SEND(search, 8, 12) in automaton sTreeNode(8) on blackbird.csail.mit.edu
Modified state variables:
P \rightarrow [\text{nbrs: (12 4 9), parent: 4, reported: true, send: ioa.runtime.adt.MapSort}]
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output RECEIVE(search, 9, 8) in automaton sTreeNode(9) on condor.csail.mit.edu
Modified state variables:
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output RECEIVE(search, 10, 9) in automaton sTreeNode(10) on nene.csail.mit.edu
Modified state variables:
P \rightarrow [\text{nbrs: (11 14 6 9), parent: 9, reported: false, send: ioa.runtime.adt.MapSort}]
RM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output PARENT(9) in automaton sTreeNode(10) on nene.csail.mit.edu
Modified state variables:
P \rightarrow [\text{nbrs: (11 14 6 9), parent: 9, reported: true, send: ioa.runtime.adt.MapSort}]

transition: output SEND(search, 10, 6) in automaton sTreeNode(10) on
nene.csail.mit.edu
    Modified state variables:
P → [nbrs: (11 14 6 9), parent: 9, reported: true, send:
ioa.runtime.adt.MapSort]
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 10, 11) in automaton sTreeNode(10) on
nene.csail.mit.edu
    Modified state variables:
P → [nbrs: (11 14 6 9), parent: 9, reported: true, send:
ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 10, 14) in automaton sTreeNode(10) on
nene.csail.mit.edu
    Modified state variables:
P → [nbrs: (11 14 6 9), parent: 9, reported: true, send:
ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 6, 5) in automaton sTreeNode(6) on
tui.csail.mit.edu
    Modified state variables:
P → [nbrs: (10 2 5 7), parent: 5, reported: false, send:
ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort

transition: output PARENT(5) in automaton sTreeNode(6) on tui.csail.mit.edu
    Modified state variables:
P → [nbrs: (10 2 5 7), parent: 5, reported: true, send:
ioa.runtime.adt.MapSort]

transition: output SEND(search, 6, 2) in automaton sTreeNode(6) on
tui.csail.mit.edu
    Modified state variables:
P → [nbrs: (10 2 5 7), parent: 5, reported: true, send:
ioa.runtime.adt.MapSort]
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 6, 7) in automaton sTreeNode(6) on
tui.csail.mit.edu
    Modified state variables:
P → [nbrs: (10 2 5 7), parent: 5, reported: true, send:
ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 6, 10) in automaton sTreeNode(6) on
tui.csail.mit.edu
    Modified state variables:
P → [nbrs: (10 2 5 7), parent: 5, reported: true, send:
ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

73
transition: output RECEIVE(search, 6, 10) in automaton sTreeNode(6) on tui.csail.mit.edu
  Modified state variables:
  RM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 10, 6) in automaton sTreeNode(10) on nene.csail.mit.edu
  Modified state variables: RM → ioa.runtime.adt.MapSort
  SM → ioa.runtime.adt.MapSort

Modified state variables : RM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 13, 9) in automaton sTreeNode(13) on tui.csail.mit.edu
  Modified state variables:
  P → [nbrs: (12 14 9), parent: 9, reported: false, send:
    ioa.runtime.adt.MapSort]
  RM → ioa.runtime.adt.MapSort

transition: output PARENT(9) in automaton sTreeNode(13) on tui.csail.mit.edu
  Modified state variables:
  P → [nbrs: (12 14 9), parent: 9, reported: true, send:
    ioa.runtime.adt.MapSort]

transition: output SEND(search, 13, 14) in automaton sTreeNode(13) on tui.csail.mit.edu
  Modified state variables:
  P → [nbrs: (12 14 9), parent: 9, reported: true, send:
    ioa.runtime.adt.MapSort]
  SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 12, 8) in automaton sTreeNode(12) on parrot.csail.mit.edu
  Modified state variables:
  P → [nbrs: (13 8), parent: 8, reported: false, send:
    ioa.runtime.adt.MapSort]
  RM → ioa.runtime.adt.MapSort

transition: output PARENT(8) in automaton sTreeNode(12) on parrot.csail.mit.edu
  Modified state variables:
  P → [nbrs: (13 8), parent: 8, reported: true, send:
    ioa.runtime.adt.MapSort]

transition: output SEND(search, 12, 13) in automaton sTreeNode(12) on parrot.csail.mit.edu
  Modified state variables:
  P → [nbrs: (13 8), parent: 8, reported: true, send:
    ioa.runtime.adt.MapSort]
  SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 13, 12) in automaton sTreeNode(13) on tui.csail.mit.edu
  Modified state variables:
transition: output RECEIVE(search, 13, 12) in automaton sTreeNode(13) on tui.csail.mit.edu
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

modified state variables:
P → [nbrs: (10 15 7), parent: 10, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 12, 13) in automaton sTreeNode(12) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (10 15 7), parent: 10, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output PARENT(10) in automaton sTreeNode(11) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (10 15 7), parent: 10, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 11, 15) in automaton sTreeNode(11) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (10 15 7), parent: 10, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 7, 6) in automaton sTreeNode(7) on blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (11 3 6), parent: 6, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort

transition: output PARENT(6) in automaton sTreeNode(7) on blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (11 3 6), parent: 6, reported: true, send: ioa.runtime.adt.MapSort]

transition: output SEND(search, 7, 3) in automaton sTreeNode(7) on blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (11 3 6), parent: 6, reported: true, send: ioa.runtime.adt.MapSort]
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 7, 11) in automaton sTreeNode(7) on blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (11 3 6), parent: 6, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

Begin initialization
Modified state variables:
P → [nbrs: (), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → null
End initialization

transition: input initialize() in automaton sTreeNode(3) on nene.csail.mit.edu
transition: output SEND(search, 11, 7) in automaton sTreeNode(11) on parrot.csail.mit.edu
Modified state variables:
P → [nbrs: (10 15 7), parent: 10, reported: true, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 11, 7) in automaton sTreeNode(11) on parrot.csail.mit.edu
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 7, 11) in automaton sTreeNode(7) on blackbird.csail.mit.edu
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 14, 10) in automaton sTreeNode(14) on blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (10 13 15), parent: 10, reported: false, send: ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output PARENT(10) in automaton sTreeNode(14) on blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (10 13 15), parent: 10, reported: true, send: ioa.runtime.adt.MapSort]

transition: output SEND(search, 14, 15) in automaton sTreeNode(14) on blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (10 13 15), parent: 10, reported: true, send:
ioa.runtime.adt.MapSort
    SM → ioa.runtime.adt.MapSort

Modified state variables:
P → [nbrs: (2 7), parent: -1, reported: false, send:
    ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 3, 7) in automaton sTreeNode(3) on
nene.csail.mit.edu
Modified state variables:
P → [nbrs: (2 7), parent: 7, reported: false, send:
    ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort

transition: output PARENT(7) in automaton sTreeNode(3) on
nene.csail.mit.edu
Modified state variables:
P → [nbrs: (2 7), parent: 7, reported: true, send:
    ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort

transition: output SEND(search, 3, 2) in automaton sTreeNode(3) on
nene.csail.mit.edu
Modified state variables:
P → [nbrs: (2 7), parent: 7, reported: true, send:
    ioa.runtime.adt.MapSort]
SM → ioa.runtime.adt.MapSort

transition: output SEND(search, 14, 13) in automaton sTreeNode(14) on
blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (10 13 15), parent: 10, reported: true, send:
    ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 14, 13) in automaton sTreeNode(14) on
blackbird.csail.mit.edu
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 13, 14) in automaton sTreeNode(13) on
tui.csail.mit.edu
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 15, 11) in automaton sTreeNode(15) on
blackbird.csail.mit.edu
Modified state variables:
P → [nbrs: (11 14), parent: 11, reported: false, send:
    ioa.runtime.adt.MapSort]
RM → ioa.runtime.adt.MapSort

77
transition: output PARENT(11) in automaton sTreeNode(15) on blackbird.csail.mit.edu
Modified state variables:
P -> [nbrs: (11 14), parent: 11, reported: true, send: ioa.runtime.adt.MapSort]

transition: output SEND(search, 15, 14) in automaton sTreeNode(15) on blackbird.csail.mit.edu
Modified state variables:
P -> [nbrs: (11 14), parent: 11, reported: true, send: ioa.runtime.adt.MapSort]
SM -> ioa.runtime.adt.MapSort

transition: output RECEIVE(search, 15, 14) in automaton sTreeNode(15) on blackbird.csail.mit.edu
Modified state variables:
RM -> ioa.runtime.adt.MapSort
SM -> ioa.runtime.adt.MapSort

Begin initialization
Modified state variables:
P -> [nbrs: (), parent: 87, reported: true, send: ioa.runtime.adt.MapSort]
RM -> ioa.runtime.adt.MapSort
SM -> ioa.runtime.adt.MapSort
i -> null
End initialization

transition: output initialize() in automaton sTreeNode(2) on condor.csail.mit.edu
Modified state variables:
P -> [nbrs: (1 3 6), parent: -1, reported: false, send: ioa.runtime.adt.MapSort]
RM -> ioa.runtime.adt.MapSort
SM -> ioa.runtime.adt.MapSort
i -> 2

transition: output RECEIVE(search, 2, 1) in automaton sTreeNode(2) on condor.csail.mit.edu
Modified state variables:
P -> [nbrs: (1 3 6), parent: 1, reported: false, send: ioa.runtime.adt.MapSort]
RM -> ioa.runtime.adt.MapSort

transition: output PARENT(1) in automaton sTreeNode(2) on condor.csail.mit.edu
Modified state variables:
P -> [nbrs: (1 3 6), parent: 1, reported: true, send: ioa.runtime.adt.MapSort]

transition: output SEND(search, 2, 3) in automaton sTreeNode(2) on condor.csail.mit.edu
E.3 Asynchronous Broadcast Convergecast on 16 nodes

Complete trace

Begin initialization

Modified state variables:
P → [val: 87, acked: (), nbrs: (), parent: 87, reported: true, send:
ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

i → 0

End initialization

transition: input initialize() in automaton bcastNode

Modified state variables:

transition: output RECEIVE(msg([kind: bcast, w: 99]), 1, 0) in automaton bcastNode

P → [val: -1, acked: (), nbrs: (0 5 8), parent: -1, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → 4

Modified state variables:
P → [val: 99, acked: (), nbrs: (0 2 5), parent: 0, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 1, 2) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (0 2 5), parent: 0, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 1, 5) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (0 2 5), parent: 0, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

Begin initialization
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → 0

End initialization
transition: input initialize() in automaton bcastNode
Modified state variables:
P → [-1, acked: (), nbrs: (10 13 15), parent: -1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i → 14

transition: output RECEIVE(msg([kind: bcast, w: 99]), 4, 0) in automaton bcastNode
transition: output RECEIVE(msg([kind: bcast, w: 99]), 5, 1) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (1 4 6 9), parent: 1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 5, 4) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (1 4 6 9), parent: 1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

80
Modified state variables:
P \rightarrow \begin{array}{l}
\text{[val: 99, acked: (), nbrs: (0 5 8), parent: 0, reported: false, send:}
\text{ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]}\end{array}
RM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output SEND (msg([kind: bcast, w: 99]), 4, 8) in automaton bcastNode
Modified state variables:
P \rightarrow \begin{array}{l}
\text{[val: 99, acked: (), nbrs: (0 5 8), parent: 0, reported: false, send:}
\text{ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]}\end{array}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output SEND (msg([kind: bcast, w: 99]), 5, 9) in automaton bcastNode

transition: output SEND (msg([kind: bcast, w: 99]), 4, 5) in automaton bcastNode
Modified state variables:
P \rightarrow \begin{array}{l}
\text{[val: 99, acked: (), nbrs: (0 5 8), parent: 0, reported: false, send:}
\text{ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]}\end{array}
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output RECEIVE (msg([kind: bcast, w: 99]), 4, 5) in automaton bcastNode
Modified state variables:
P \rightarrow \begin{array}{l}
\text{[val: 99, acked: (), nbrs: (0 5 8), parent: 0, reported: false, send:}
\text{ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]}\end{array}
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output SEND (kind(ack), 4, 5) in automaton bcastNode
Modified state variables:
P \rightarrow \begin{array}{l}
\text{[val: 99, acked: (), nbrs: (0 5 8), parent: 0, reported: false, send:}
\text{ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]}\end{array}
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

Modified state variables:
P \rightarrow \begin{array}{l}
\text{[val: 99, acked: (), nbrs: (1 4 6 9), parent: 1, reported: false, send:}
\text{ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]}\end{array}
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output SEND (msg([kind: bcast, w: 99]), 5, 6) in automaton bcastNode
Modified state variables:
P \rightarrow \begin{array}{l}
\text{[val: 99, acked: (), nbrs: (1 4 6 9), parent: 1, reported: false, send:}
\text{ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]}\end{array}
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output RECEIVE (msg([kind: bcast, w: 99]), 5, 4) in automaton bcastNode
Modified state variables:
P \rightarrow \begin{array}{l}
\text{[val: 99, acked: (), nbrs: (1 4 6 9), parent: 1, reported: false,}
\text{send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]}\end{array}
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}
transition: output SEND(kind(ack), 5, 4) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (1 4 6 9), parent: 1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 5, 4) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (4), nbrs: (1 4 6 9), parent: 1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 8, 4) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (4), nbrs: (1 4 6 9), parent: 1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 8, 9) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (4), nbrs: (1 4 6 9), parent: 1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 4, 5) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (5), nbrs: (0 5 8), parent: 0, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 2, 1) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (5), nbrs: (1 3 6), parent: 1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 2, 3) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (5), nbrs: (1 3 6), parent: 1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 8, 12) in automaton bcastNode
Modified state variables:
Begin initialization
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

i → 0

End initialization

transition: input initialize() in automaton bcastNode

transition: output SEND(msg([kind: bcast, w: 99]), 2, 6) in automaton bcastNode

Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 6, 2) in automaton bcastNode

Modified state variables:
P → [val: -1, acked: (), nbrs: (10 2 5 7), parent: -1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

i → 6

transition: output SEND(msg([kind: bcast, w: 99]), 6, 10) in automaton bcastNode

Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 3, 2) in automaton bcastNode

Modified state variables:
RM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 3, 7) in automaton bcastNode

Modified state variables:
RM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 6, 7) in automaton bcastNode
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 6, 5) in automaton
bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 2 5 7), parent: 2, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 6, 5) in automaton
bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 2 5 7), parent: 2, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(kind(ack), 6, 5) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 2 5 7), parent: 2, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 12, 8) in automaton
bcastNode
Modified state variables:
P → [val: 99, acked: (4), nbrs: (13 8), parent: 8, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 12, 13) in automaton
bcastNode
Modified state variables:
P → [val: 99, acked: (4), nbrs: (13 8), parent: 8, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 5, 6) in automaton
bcastNode
Modified state variables:
P → [val: 99, acked: (4), nbrs: (1 4 6 9), parent: 1, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort

transition: output SEND(kind(ack), 5, 6) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (4), nbrs: (1 4 6 9), parent: 1, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
\[ \text{RM} \rightarrow \text{ioa.runtime.adt.MapSort} \]
\[ \text{SM} \rightarrow \text{ioa.runtime.adt.MapSort} \]

\textit{transition}: output \text{RECEIVE(kind(ack), 5, 6)} in automaton \text{bcastNode}

Modified state variables:
\[
P \rightarrow [\text{val}: 99, \text{acked}: (4 6), \text{nbrs}: (1 4 6 9), \text{parent}: 1, \text{reported}: \text{false},
\text{send}: \text{ioa.runtime.adt.MapSort}, \text{temp}: [\text{kind}: \text{bcast}, \text{w}: 87]]
\]
\[ \text{RM} \rightarrow \text{ioa.runtime.adt.MapSort} \]
\[ \text{SM} \rightarrow \text{ioa.runtime.adt.MapSort} \]

\textit{transition}: output \text{RECEIVE(kind(ack), 6, 5)} in automaton \text{bcastNode}

Modified state variables:
\[
P \rightarrow [\text{val}: 99, \text{acked}: (5), \text{nbrs}: (10 2 5 7), \text{parent}: 2, \text{reported}: \text{false},
\text{send}: \text{ioa.runtime.adt.MapSort}, \text{temp}: [\text{kind}: \text{bcast}, \text{w}: 87]]
\]
\[ \text{RM} \rightarrow \text{ioa.runtime.adt.MapSort} \]
\[ \text{SM} \rightarrow \text{ioa.runtime.adt.MapSort} \]

\textit{transition}: output \text{RECEIVE(msg([kind: bcast, w: 99]), 13, 12)} in automaton \text{bcastNode}

Modified state variables:
\[
P \rightarrow [\text{val}: 99, \text{acked}: (), \text{nbrs}: (12 14 9), \text{parent}: 12, \text{reported}: \text{false},
\text{send}: \text{ioa.runtime.adt.MapSort}, \text{temp}: [\text{kind}: \text{bcast}, \text{w}: 87]]
\]
\[ \text{RM} \rightarrow \text{ioa.runtime.adt.MapSort} \]
\[ \text{SM} \rightarrow \text{ioa.runtime.adt.MapSort} \]

\textit{transition}: output \text{SEND(msg([kind: bcast, w: 99]), 13, 9)} in automaton \text{bcastNode}

Modified state variables:
\[
P \rightarrow [\text{val}: 99, \text{acked}: (), \text{nbrs}: (12 14 9), \text{parent}: 12, \text{reported}: \text{false},
\text{send}: \text{ioa.runtime.adt.MapSort}, \text{temp}: [\text{kind}: \text{bcast}, \text{w}: 87]]
\]
\[ \text{SM} \rightarrow \text{ioa.runtime.adt.MapSort} \]

\textit{transition}: output \text{SEND(msg([kind: bcast, w: 99]), 13, 14)} in automaton \text{bcastNode}

Modified state variables:
\[
P \rightarrow [\text{val}: 99, \text{acked}: (), \text{nbrs}: (12 14 9), \text{parent}: 12, \text{reported}: \text{false},
\text{send}: \text{ioa.runtime.adt.MapSort}, \text{temp}: [\text{kind}: \text{bcast}, \text{w}: 87]]
\]
\[ \text{RM} \rightarrow \text{ioa.runtime.adt.MapSort} \]
\[ \text{SM} \rightarrow \text{ioa.runtime.adt.MapSort} \]

\textit{transition}: output \text{RECEIVE(msg([kind: bcast, w: 99]), 7, 3)} in automaton \text{bcastNode}

Modified state variables:
\[
P \rightarrow [\text{val}: 99, \text{acked}: (), \text{nbrs}: (11 3 6), \text{parent}: 3, \text{reported}: \text{false},
\text{send}: \text{ioa.runtime.adt.MapSort}, \text{temp}: [\text{kind}: \text{bcast}, \text{w}: 87]]
\]
\[ \text{RM} \rightarrow \text{ioa.runtime.adt.MapSort} \]
\[ \text{SM} \rightarrow \text{ioa.runtime.adt.MapSort} \]

\textit{transition}: output \text{SEND(msg([kind: bcast, w: 99]), 7, 11)} in automaton \text{bcastNode}

Modified state variables:
\[
P \rightarrow [\text{val}: 99, \text{acked}: (), \text{nbrs}: (11 3 6), \text{parent}: 3, \text{reported}: \text{false},
\text{send}: \text{ioa.runtime.adt.MapSort}, \text{temp}: [\text{kind}: \text{bcast}, \text{w}: 87]]
\]
\[ \text{SM} \rightarrow \text{ioa.runtime.adt.MapSort} \]

\textit{transition}: output \text{SEND(msg([kind: bcast, w: 99]), 7, 6)} in automaton \text{bcastNode}

\[85\]
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE (msg ([kind: bcast, w: 99]), 7, 6) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND (kind (ack), 7, 6) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE (msg ([kind: bcast, w: 99]), 6, 7) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort

transition: output SEND (kind (ack), 6, 7) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE (kind (ack), 6, 7) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (5 7), nbrs: (10 2 5 7), parent: 2, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE (kind (ack), 7, 6) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (6), nbrs: (11 3 6), parent: 3, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE (msg ([kind: bcast, w: 99]), 11, 7) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
transition: output SEND([kind: bcast, w: 99]), 11, 10 in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 15 7), parent: 7, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 14, 13) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 13 15), parent: 13, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 14, 10 in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 13 15), parent: 13, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 11, 15) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 15 7), parent: 7, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 14, 15) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 13 15), parent: 13, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 11, 10) in automaton bcastNode

transition: output RECEIVE(msg([kind: bcast, w: 99]), 10, 6) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (11 14 6 9), parent: 6, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 10, 11) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (11 14 6 9), parent: 6, reported: false,
send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 10, 11) in automaton bcastNode
Modified state variables:
\[ P \rightarrow [\text{val: 99, acked: ()}, \text{nbrs: (11 14 6 9)}, \text{parent: 6, reported: false}, \text{send: \(\text{ioa.runtime.adt.MapSort}\)}, \text{temp: [kind: \(\text{bcast}\), \text{w: 87}]})\]  
\[ \text{RM} \rightarrow \text{\(\text{ioa.runtime.adt.MapSort}\)}\]  
\[ \text{SM} \rightarrow \text{\(\text{ioa.runtime.adt.MapSort}\)}\]  

transition: output SEND(msg([kind: \(\text{bcast}\), \text{w: 99}]), 10, 14) in automaton bcastNode  
Modified state variables:  
\[ P \rightarrow [\text{val: 99, acked: ()}, \text{nbrs: (11 14 6 9)}, \text{parent: 6, reported: false}, \text{send: \(\text{ioa.runtime.adt.MapSort}\)}, \text{temp: [kind: \(\text{bcast}\), \text{w: 87}]})\]  
\[ \text{SM} \rightarrow \text{\(\text{ioa.runtime.adt.MapSort}\)}\]  

transition: output RECEIVE(msg([kind: \(\text{bcast}\), \text{w: 99}]), 10, 14) in automaton bcastNode  
Modified state variables:  
\[ P \rightarrow [\text{val: 99, acked: ()}, \text{nbrs: (11 14 6 9)}, \text{parent: 6, reported: false}, \text{send: \(\text{ioa.runtime.adt.MapSort}\)}, \text{temp: [kind: \(\text{bcast}\), \text{w: 87}]})\]  
\[ \text{SM} \rightarrow \text{\(\text{ioa.runtime.adt.MapSort}\)}\]  

transition: output SEND(msg([kind: \(\text{bcast}\), \text{w: 99}]), 10, 9) in automaton bcastNode  
Modified state variables:  
\[ P \rightarrow [\text{val: 99, acked: ()}, \text{nbrs: (10 15 7)}, \text{parent: 7, reported: false}, \text{send: \(\text{ioa.runtime.adt.MapSort}\)}, \text{temp: [kind: \(\text{bcast}\), \text{w: 87}]})\]  
\[ \text{SM} \rightarrow \text{\(\text{ioa.runtime.adt.MapSort}\)}\]  

transition: output SEND(kind(ack), 11, 10) in automaton bcastNode  
Modified state variables:  
\[ P \rightarrow [\text{val: 99, acked: ()}, \text{nbrs: (10 15 7)}, \text{parent: 7, reported: false}, \text{send: \(\text{ioa.runtime.adt.MapSort}\)}, \text{temp: [kind: \(\text{bcast}\), \text{w: 87}]})\]  
\[ \text{SM} \rightarrow \text{\(\text{ioa.runtime.adt.MapSort}\)}\]  

transition: output RECEIVE(msg([kind: \(\text{bcast}\), \text{w: 99}]), 14, 10) in automaton bcastNode  
Modified state variables:  
\[ P \rightarrow [\text{val: 99, acked: ()}, \text{nbrs: (10 13 15)}, \text{parent: 13, reported: false}, \text{send: \(\text{ioa.runtime.adt.MapSort}\)}, \text{temp: [kind: \(\text{bcast}\), \text{w: 87}]})\]  
\[ \text{SM} \rightarrow \text{\(\text{ioa.runtime.adt.MapSort}\)}\]  

transition: output SEND(kind(ack), 14, 10) in automaton bcastNode  
Modified state variables:  
\[ P \rightarrow [\text{val: 99, acked: ()}, \text{nbrs: (10 13 15)}, \text{parent: 13, reported: false}, \text{send: \(\text{ioa.runtime.adt.MapSort}\)}, \text{temp: [kind: \(\text{bcast}\), \text{w: 87}]})\]  
\[ \text{SM} \rightarrow \text{\(\text{ioa.runtime.adt.MapSort}\)}\]  

88
transition: output SEND(kind(ack), 10, 11) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 11, 10) in automaton bcastNode

transition: output RECEIVE(msg([kind: bcast, w: 99]), 15, 11) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 15, 14) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 15, 14) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(kind(ack), 15, 14) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 10, 11) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (11), nbrs: (11 14 6 9), parent: 6, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(kind(ack), 10, 14) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (11), nbrs: (11 14 6 9), parent: 6, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort
transition: output RECEIVE(kind(ack), 10, 14) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 14, 10) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 14, 15) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(kind(ack), 14, 15) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 14, 15) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

Begin initialization
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i = 0

End initialization
transition: input initialize() in automaton bcastNode
Modified state variables:
P → [val: -1, acked: (), nbrs: (10 13 5 8), parent: -1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort
i = 9

transition: internal report(14) in automaton bcastNode
Modified state variables:
transition: output SEND(kind(ack), 14, 13) in automaton bcastNode
Modified state variables:
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 13, 14) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (14), nbrs: (12 14 9), parent: 12, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 15, 14) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (14), nbrs: (11 14), parent: 11, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: internal report(15) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (14), nbrs: (11 14), parent: 11, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]

transition: output SEND(kind(ack), 15, 11) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (14), nbrs: (11 14), parent: 11, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 11, 15) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort

transition: internal report(11) in automaton bcastNode
Modified state variables:

transition: output SEND(kind(ack), 11, 7) in automaton bcastNode
Modified state variables:
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 7, 11) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (11 6), nbrs: (11 3 6), parent: 3, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort

transition: internal report(7) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (11 6), nbrs: (11 3 6), parent: 3, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]

transition: output SEND(kind(ack), 7, 3) in automaton bcastNode
transition: output RECEIVE(kind(ack), 3, 7) in automaton bcastNode

Modified state variables:
P → [val: 99, acked: (7), nbrs: (2 7), parent: 2, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: internal report(3) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (7), nbrs: (2 7), parent: 2, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]

transition: output SEND(kind(ack), 3, 2) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (7), nbrs: (2 7), parent: 2, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

Modified state variables:
P → [val: 99, acked: (11 6), nbrs: (11 3 6), parent: 3, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 2, 3) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (3), nbrs: (1 3 6), parent: 1, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 9, 5) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 13 5 8), parent: 5, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 9, 13) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 13 5 8), parent: 5, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output SEND(msg([kind: bcast, w: 99]), 9, 8) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 13 5 8), parent: 5, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
\[ SM \rightarrow \text{ioa.runtime.adt.MapSort} \]

**transition**: output \( \text{RECEIVE}([\text{kind : bcast, w : 99}]), 13, 9 \) in automaton \( \text{bcastNode} \)

Modified state variables:
\[
P \rightarrow [\text{val : 99, acked : (14), nbrs : (12 14 9), parent : 12, reported : false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87]}]\
RM \rightarrow \text{ioa.runtime.adt.MapSort}\
\]

**transition**: output \( \text{SEND(kind(ack)), 13, 9} \) in automaton \( \text{bcastNode} \)

Modified state variables:
\[
P \rightarrow [\text{val : 99, acked : (14), nbrs : (12 14 9), parent : 12, reported : false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87]}]\
RM \rightarrow \text{ioa.runtime.adt.MapSort}\
SM \rightarrow \text{ioa.runtime.adt.MapSort}\
\]

**transition**: output \( \text{RECEIVE}([\text{kind : bcast, w : 99}]), 9, 8 \) in automaton \( \text{bcastNode} \)

Modified state variables:
\[
P \rightarrow [\text{val : 99, acked : (), nbrs : (10 13 5 8), parent : 5, reported : false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87]}]\
RM \rightarrow \text{ioa.runtime.adt.MapSort}\
SM \rightarrow \text{ioa.runtime.adt.MapSort}\
\]

**transition**: output \( \text{SEND(msg([kind : bcast, w : 99]), 9, 10}) \) in automaton \( \text{bcastNode} \)

Modified state variables:
\[
P \rightarrow [\text{val : 99, acked : (), nbrs : (10 13 5 8), parent : 5, reported : false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87]}]\
SM \rightarrow \text{ioa.runtime.adt.MapSort}\
\]

**transition**: output \( \text{RECEIVE(msg([kind : bcast, w : 99]), 10, 9}) \) in automaton \( \text{bcastNode} \)

Modified state variables:
\[
P \rightarrow [\text{val : 99, acked : (11 14), nbrs : (11 14 6 9), parent : 6, reported : false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87]}]\
RM \rightarrow \text{ioa.runtime.adt.MapSort}\
SM \rightarrow \text{ioa.runtime.adt.MapSort}\
\]

**transition**: output \( \text{SEND(kind(ack), 10, 9}) \) in automaton \( \text{bcastNode} \)

Modified state variables:
\[
P \rightarrow [\text{val : 99, acked : (11 14), nbrs : (11 14 6 9), parent : 6, reported : false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87]}]\
RM \rightarrow \text{ioa.runtime.adt.MapSort}\
SM \rightarrow \text{ioa.runtime.adt.MapSort}\
\]

**transition**: output \( \text{RECEIVE(msg([kind : bcast, w : 99]), 9, 13}) \) in automaton \( \text{bcastNode} \)

Modified state variables:
\[
P \rightarrow [\text{val : 99, acked : (11 14), nbrs : (11 14 6 9), parent : 6, reported : false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87]}]\
RM \rightarrow \text{ioa.runtime.adt.MapSort}\
SM \rightarrow \text{ioa.runtime.adt.MapSort}\
\]

**transition**: output \( \text{RECEIVE(msg([kind : bcast, w : 99]), 9, 13}) \) in automaton \( \text{bcastNode} \)
bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 13 5 8), parent: 5, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort

transition: output SEND(kind(ack), 9, 8) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 13 5 8), parent: 5, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output SEND(kind(ack), 9, 10) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (), nbrs: (10 13 5 8), parent: 5, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 10, 9) in automaton bcastNode
Modified state variables:

transition: output RECEIVE(kind(ack), 9, 10) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (10), nbrs: (10 13 5 8), parent: 5, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort  RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort  SM → ioa.runtime.adt.MapSort

transition: internal report(10) in automaton bcastNode
Modified state variables:

transition: output SEND(kind(ack), 10, 6) in automaton bcastNode
Modified state variables:
SM → ioa.runtime.adt.MapSort
transition: output SEND(kind(ack), 9, 13) in automaton bcastNode

Modified state variables:
P → [val: 99, acked: (10), nbrs: (10 13 5 8), parent: 5, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort  RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort  SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 9, 13) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (10 13), nbrs: (10 13 5 8), parent: 5, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 13, 9) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (14 9), nbrs: (12 14 9), parent: 12, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: internal report(13) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (14 9), nbrs: (12 14 9), parent: 12, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]

transition: output SEND(kind(ack), 13, 12) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (14 9), nbrs: (12 14 9), parent: 12, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(kind(ack), 12, 13) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (13), nbrs: (13 8), parent: 8, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: internal report(12) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (13), nbrs: (13 8), parent: 8, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]

transition: output SEND(kind(ack), 12, 8) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (13), nbrs: (13 8), parent: 8, reported: true, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
SM → ioa.runtime.adt.MapSort

transition: output RECEIVE(msg([kind: bcast, w: 99]), 8, 9) in automaton bcastNode
Modified state variables:
RM → ioa.runtime.adt.MapSort
SM → ioa.runtime.adt.MapSort

transition: output SEND(kind(ack), 8, 9) in automaton bcastNode
Modified state variables:
transition: output RECEIVE(kind(ack), 6, 10) in automaton bcastNode
Modified state variables:
P → [val: 99, acked: (10 5 7), nbrs: (10 2 5 7), parent: 2, reported: false, send: ioa.runtime.adt.MapSort, temp: [kind: bcast, w: 87]]
RM → ioa.runtime.adt.MapSort
transition: internal report(6) in automaton bcastNode
Modified state variables:
P \rightarrow \{ val : 99, acked : (10 5 7), nbrs : (10 2 5 7), parent : 2, reported: true, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87] \}

transition: output SEND(kind(ack), 6, 2) in automaton bcastNode
Modified state variables:
P \rightarrow \{ val : 99, acked : (10 5 7), nbrs : (10 2 5 7), parent : 2, reported: true, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87] \}
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output RECEIVE(kind(ack), 2, 6) in automaton bcastNode
Modified state variables:
P \rightarrow \{ val : 99, acked : (3 6), nbrs : (1 3 6), parent : 1, reported: false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87] \}
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: internal report(2) in automaton bcastNode
Modified state variables:
P \rightarrow \{ val : 99, acked : (3 6), nbrs : (1 3 6), parent : 1, reported: true, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87] \}

transition: output SEND(kind(ack), 2, 1) in automaton bcastNode
Modified state variables:
P \rightarrow \{ val : 99, acked : (3 6), nbrs : (1 3 6), parent : 1, reported: true, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87] \}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output RECEIVE(kind(ack), 1, 2) in automaton bcastNode
Modified state variables:
P \rightarrow \{ val : 99, acked : (2), nbrs : (0 2 5), parent : 0, reported: false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87] \}
RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

RM \rightarrow \text{ioa.runtime.adt.MapSort}
SM \rightarrow \text{ioa.runtime.adt.MapSort}

transition: output RECEIVE(kind(ack), 9, 8) in automaton bcastNode
Modified state variables:
transition: output RECEIVE(kind(ack), 8, 9) in automaton bcastNode
P \rightarrow \{ val : 99, acked : (10 13 8), nbrs : (10 13 5 8), parent : 5, reported: false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87] \}
RM \rightarrow \text{ioa.runtime.adt.MapSort}

Modified state variables:
transition: internal report(9) in automaton bcastNode
Modified state variables:
P \rightarrow \{ val : 99, acked : (9), nbrs : (12 4 9), parent : 4, reported: false, send : ioa.runtime.adt.MapSort, temp : [kind : bcast, w : 87] \}
RM \rightarrow \text{ioa.runtime.adt.MapSort}

96
SM \rightarrow \text{ioa.runtime.adt.MapSort}

P \rightarrow [\text{val: 99, acked: (10 13 8), nbrs: (10 13 5 8), parent: 5, reported: true, send: \text{ioa.runtime.adt.MapSort}, temp: [\text{kind: bcast, w: 87}]}]

\text{transition: output SEND(kind(ack), 9, 5) in automaton bcastNode}

\text{Modified state variables:}

P \rightarrow [\text{val: 99,acked: (10 13 8), nbrs: (10 13 5 8), parent: 5, reported: true, send: \text{ioa.runtime.adt.MapSort}, temp: [\text{kind: bcast, w: 87}]}]

RM \rightarrow \text{ioa.runtime.adt.MapSort}

SM \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: output RECEIVE(kind(ack), 8, 12) in automaton bcastNode}

\text{Modified state variables:}

P \rightarrow [\text{val: 99, acked: (12 9), nbrs: (12 4 9), parent: 4, reported: false, send: \text{ioa.runtime.adt.MapSort}, temp: [\text{kind: bcast, w: 87}]}]

RM \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: internal report(8) in automaton bcastNode}

\text{Modified state variables:}

P \rightarrow [\text{val: 99, acked: (12 9), nbrs: (12 4 9), parent: 4, reported: true, send: \text{ioa.runtime.adt.MapSort}, temp: [\text{kind: bcast, w: 87}]}]

SM \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: output SEND(kind(ack), 8, 4) in automaton bcastNode}

\text{Modified state variables:}

P \rightarrow [\text{val: 99, acked: (12 9), nbrs: (12 4 9), parent: 4, reported: true, send: \text{ioa.runtime.adt.MapSort}, temp: [\text{kind: bcast, w: 87}]}]

RM \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: output RECEIVE(kind(ack), 5, 9) in automaton bcastNode}

\text{Modified state variables:}

P \rightarrow [\text{val: 99, acked: (4 6 9), nbrs: (1 4 6 9), parent: 1, reported: false, send: \text{ioa.runtime.adt.MapSort}, temp: [\text{kind: bcast, w: 87}]}]

RM \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: internal report(5) in automaton bcastNode}

\text{Modified state variables:}

P \rightarrow [\text{val: 99, acked: (4 6 9), nbrs: (1 4 6 9), parent: 1, reported: true, send: \text{ioa.runtime.adt.MapSort}, temp: [\text{kind: bcast, w: 87}]}]

SM \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: output SEND(kind(ack), 5, 1) in automaton bcastNode}

\text{Modified state variables:}

P \rightarrow [\text{val: 99, acked: (4 6 9), nbrs: (1 4 6 9), parent: 1, reported: true, send: \text{ioa.runtime.adt.MapSort}, temp: [\text{kind: bcast, w: 87}]}]

RM \rightarrow \text{ioa.runtime.adt.MapSort}

SM \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: output RECEIVE(kind(ack), 1, 5) in automaton bcastNode}

\text{Modified state variables:}

P \rightarrow [\text{val: 99, acked: (2 5), nbrs: (0 2 5), parent: 0, reported: false, send: \text{ioa.runtime.adt.MapSort}, temp: [\text{kind: bcast, w: 87}]}]

RM \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: internal report(1) in automaton bcastNode}

\text{Modified state variables:}
\[
P \rightarrow \{
 val : 99, 
 acked : (2 5), 
 nbrs : (0 2 5), 
 parent : 0, 
 reported : true, 
 send : \text{ioa.runtime.adt.MapSort}, 
 temp : [\text{kind} : \text{bcast}, w : 87]\}
\]

\text{transition: output SEND(kind(ack), 1, 0) in automaton bcastNode}

Modified state variables:
\[
P \rightarrow \{
 val : 99, 
 acked : (2 5), 
 nbrs : (0 2 5), 
 parent : 0, 
 reported : true, 
 send : \text{ioa.runtime.adt.MapSort}, 
 temp : [\text{kind} : \text{bcast}, w : 87]\}
\]

\text{SM} \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: output RECEIVE(kind(ack), 4, 8) in automaton bcastNode}

Modified state variables:
\[
P \rightarrow \{
 val : 99, 
 acked : (5 8), 
 nbrs : (0 5 8), 
 parent : 0, 
 reported : false, 
 send : \text{ioa.runtime.adt.MapSort}, 
 temp : [\text{kind} : \text{bcast}, w : 87]\}
\]

\text{RM} \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: internal report(4) in automaton bcastNode}

Modified state variables:
\[
P \rightarrow \{
 val : 99, 
 acked : (5 8), 
 nbrs : (0 5 8), 
 parent : 0, 
 reported : true, 
 send : \text{ioa.runtime.adt.MapSort}, 
 temp : [\text{kind} : \text{bcast}, w : 87]\}
\]

\text{transition: output SEND(kind(ack), 4, 0) in automaton bcastNode}

Modified state variables:
\[
P \rightarrow \{
 val : 99, 
 acked : (5 8), 
 nbrs : (0 5 8), 
 parent : 0, 
 reported : true, 
 send : \text{ioa.runtime.adt.MapSort}, 
 temp : [\text{kind} : \text{bcast}, w : 87]\}
\]

\text{RM} \rightarrow \text{ioa.runtime.adt.MapSort}

\text{SM} \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: output RECEIVE(kind(ack), 0, 1) in automaton bcastNode}

Modified state variables:
\[
P \rightarrow \{
 val : 99, 
 acked : (1), 
 nbrs : (1 4), 
 parent : -1, 
 reported : false, 
 send : \text{ioa.runtime.adt.MapSort}, 
 temp : [\text{kind} : \text{bcast}, w : 99]\}
\]

\text{RM} \rightarrow \text{ioa.runtime.adt.MapSort}

\text{SM} \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: output RECEIVE(kind(ack), 0, 4) in automaton bcastNode}

Modified state variables:
\[
P \rightarrow \{
 val : 99, 
 acked : (1 4), 
 nbrs : (1 4), 
 parent : -1, 
 reported : false, 
 send : \text{ioa.runtime.adt.MapSort}, 
 temp : [\text{kind} : \text{bcast}, w : 99]\}
\]

\text{RM} \rightarrow \text{ioa.runtime.adt.MapSort}

\text{SM} \rightarrow \text{ioa.runtime.adt.MapSort}

\text{transition: internal report(0) in automaton bcastNode}

Modified state variables:
\[
P \rightarrow \{
 val : 99, 
 acked : (1 4), 
 nbrs : (1 4), 
 parent : -1, 
 reported : true, 
 send : \text{ioa.runtime.adt.MapSort}, 
 temp : [\text{kind} : \text{bcast}, w : 99]\}
\]

\text{E.4 Spanning Tree to Leader Election on 16 nodes}

\text{Complete trace}

<table>
<thead>
<tr>
<th>Trace of sTreeLeader</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of machines: 5</td>
</tr>
<tr>
<td>Number of nodes: 16</td>
</tr>
<tr>
<td>Duration: 9 sec</td>
</tr>
</tbody>
</table>

98
Number of messages exchanged: 16
====================================
Initialization starts (15) on loon.csail.mit.edu at 9:29:05:372
Modified state variables:
P → [nbrs: (), receivedElect: (), sentElect: (), status: idle, send: Map{}}
RM → Map{}}
SM → Map{}}
j → 87
k → 87
rank → null
tempNbrs → ()
tempNbrs2 → ()
Initialization ends
transition: output SEND(elect, 15, 11) in automaton sTreeLeader(15)
on loon.csail.mit.edu at 9:29:05:388
Modified state variables:
P → [nbrs: (11), receivedElect: (), sentElect: (), status: idle, send: Map{[11 -> {}}] }}
RM → Map{[11 -> [status: idle, toRecv: {}, ready: false]] }}
SM → Map{[11 -> [status: idle, toSend: {elect}, sent: {}, handles: {}]} }
j → 11
k → 11
rank → 15
tempNbrs → ()
tempNbrs2 → ()

Initialization starts (0) on loon.csail.mit.edu at 9:29:05:847
Modified state variables:
P → [nbrs: (), receivedElect: (), sentElect: (), status: idle, send: Map{}}
RM → Map{}}
SM → Map{}}
j → 87
k → 87
rank → null
tempNbrs → ()
tempNbrs2 → ()
Initialization ends
transition: output SEND(elect, 0, 1) in automaton sTreeLeader(0)
on loon.csail.mit.edu at 9:29:05:860
Modified state variables:
P → [nbrs: (1), receivedElect: (), sentElect: (), status: idle, send: Map{[1 -> {}}] }
RM → Map{[1 -> [status: idle, toRecv: {}, ready: false]] }}
SM → Map{[1 -> [status: idle, toSend: {elect}, sent: {}, handles: {}]} }
j → 1
k → 1
rank → 0
tempNbrs → ()
tempNbrs2 → ()

Initialization starts (14) on tui.csail.mit.edu at 9:29:05:977
Initialization starts (3) on parrot.csail.mit.edu at 9:29:05:978
Modified state variables:
Modified state variables:
P → [nbrs: (), receivedElect: (), sentElect: (), status: idle, send: Map{}}
RM → Map{}}
SM → Map{}}
Initialization starts (13) on parrot.csail.mit.edu at 9:29:06:210
Modified state variables:
P → [nbrs: (), receivedElect: (), sentElect: (), status: idle, send: Map{}
RM → Map{}
SM → Map{}
j → 87
k → 87
rank → null
tempNbrs → ()
tempNbrs2 → ()
Initialization ends

Modified state variables:
P → [nbrs: (), receivedElect: (), sentElect: (), status: idle, send: Map{}
RM → Map{}
SM → Map{}
j → 87
k → 87
rank → null
tempNbrs → ()
tempNbrs2 → ()
Initialization ends

Modified state variables:
P → [nbrs: (), receivedElect: (), sentElect: (), status: idle, send: Map{}
RM → Map{}
SM → Map{}
j → 87
k → 87
rank → null
tempNbrs → ()
tempNbrs2 → ()
Initialization ends

Modified state variables:
P → [nbrs: (14), receivedElect: (), sentElect: (), status: idle, send: Map{}
RM → Map{}
SM → Map{}
j → 87
k → 87
rank → null
tempNbrs → ()
tempNbrs2 → ()
Initialization ends

transition: output SEND(elect, 13, 14) in automaton sTreeLeader(13)
on parrot.csail.mit.edu at 9:29:06:796
Modified state variables:
P → [nbrs: (14), receivedElect: (), sentElect: (), status: idle, send: Map[[14 -> {}]]}
RM → Map[[14 -> [status: idle, toRecv: {}, ready: false]]}
SM → Map[[14 -> [status: idle, toSend: {elect}, sent: {}, handles: {}]]}
j → 14
k → 14

100
rank → 13
tempNbrs → ()
tempNbrs2 → ()

Initialization starts (1) on drake.csail.mit.edu at 9:29:07:059
Initialization starts (2) on condor.csail.mit.edu at 9:29:07:062
Modified state variables:
Initialization starts (4) on tui.csail.mit.edu at 9:29:07:150
Modified state variables:
P → [nbrs: (), receivedElect: (), sentElect: (), status: idle, send: Map{]}
RM → Map{}
SM → Map{}
j → 87
k → 87
rank → null
tempNbrs → ()
tempNbrs2 → ()
Initialization ends

transition: output RECEIVE(elect, 11, 15) in automaton sTreeLeader(11)
on drake.csail.mit.edu at 9:29:07:180
Modified state variables:
P → [nbrs: (15 7), receivedElect: (15), sentElect: (7), status: idle, send: Map{[15 ->
RM → Map{[15 -> [status: idle, toRecv: {}, ready: false]} [7 -> [status: idle, toRecv:
SM → Map{[15 -> [status: idle, toSend: {}, sent: {}, handles: {]]} [7 -> [status: idle,
j → 7
k → 15
rank → 11
tempNbrs → ()
tempNbrs2 → ()

Initialization starts (6) on drake.csail.mit.edu at 9:29:07:304
Modified state variables:

transition: output SEND(elect, 4, 5) in automaton sTreeLeader(4)
on tui.csail.mit.edu at 9:29:07:401
Modified state variables:
P → [nbrs: (5), receivedElect: (), sentElect: (), status: idle, send: Map{[5 ->
RM → Map{[5 -> [status: idle, toRecv: {}] }]
SM → Map{[5 -> [status: idle, toSend: { elect }, sent: {}, handles: {]}]}
j → 5
k → 5
rank → 4
tempNbrs → ()
tempNbrs2 → ()

transition: output SEND(elect, 11, 7) in automaton sTreeLeader(11)
on drake.csail.mit.edu at 9:29:07:485
Modified state variables:
P → Tuple, modified fields: {[send -> Map, modified entries: {[7 -> Sequence, elements
SM → Map, modified entries: {[7 -> Tuple, modified fields: {[toSend -> Sequence, elements
k → 7

Initialization starts (7) on condor.csail.mit.edu at 9:29:07:260
Modified state variables:
P → [nbrs: (), receivedElect: (), sentElect: (), status: idle, send: Map{]}
RM → Map{}
SM → Map{]
j \rightarrow 87
k \rightarrow 87
rank \rightarrow \text{null}
temp\text{Nbrs} \rightarrow ()
temp\text{Nbrs2} \rightarrow ()
Initialization ends
Initialization starts (12) on condor.csail.mit.edu at 9:29:07:712
Modified state variables:
P \rightarrow [\text{nbrs}: (), \text{receivedElect}: (), \text{sentElect}: (), \text{status}: \text{idle}, \text{send}: \text{Map}]
RM \rightarrow \text{Map}
SM \rightarrow \text{Map}
j \rightarrow 87
k \rightarrow 87
rank \rightarrow \text{null}
temp\text{Nbrs} \rightarrow ()
temp\text{Nbrs2} \rightarrow ()
Initialization ends
Initialization starts (10) on loon.csail.mit.edu at 9:29:07:954
Modified state variables:
P \rightarrow [\text{nbrs}: (), \text{receivedElect}: (), \text{sentElect}: (), \text{status}: \text{idle}, \text{send}: \text{Map}]
RM \rightarrow \text{Map}
SM \rightarrow \text{Map}
j \rightarrow 87
k \rightarrow 87
rank \rightarrow \text{null}
temp\text{Nbrs} \rightarrow ()
temp\text{Nbrs2} \rightarrow ()
Initialization ends
Initialization starts (9) on tui.csail.mit.edu at 9:29:08:081
Modified state variables:
P \rightarrow [\text{nbrs}: (), \text{receivedElect}: (), \text{sentElect}: (), \text{status}: \text{idle}, \text{send}: \text{Map}]
RM \rightarrow \text{Map}
SM \rightarrow \text{Map}
j \rightarrow 87
k \rightarrow 87
rank \rightarrow \text{null}
temp\text{Nbrs} \rightarrow ()
temp\text{Nbrs2} \rightarrow ()
Modification state variables:
P \rightarrow \{ \text{nbrs} : (), \text{sentElect} : (), \text{status} : \text{idle}, \text{send} : \text{Map}{} \}
RM \rightarrow \text{Map}{}
SM \rightarrow \text{Map}{}
j \rightarrow 87
k \rightarrow 87
rank \rightarrow \text{null}
tempNbrs \rightarrow ()
tempNbrs2 \rightarrow ()

Transition: output \text{SEND} (\text{elect}, 12, 8) in automaton \text{sTreeLeader}(12)
on \text{condor.csail.mit.edu} at 9:29:08:395

Modified state variables:
P \rightarrow \{ \text{nbrs} : (8), \text{receivedElect} : (), \text{sentElect} : (), \text{status} : \text{idle}, \text{send} : \text{Map}{{8 \rightarrow \{\}}} \}
RM \rightarrow \text{Map}{{8 \rightarrow \{ \text{status} : \text{idle}, \text{toRecv} : \{\}, \text{ready} : \text{false} \}\}}
SM \rightarrow \text{Map}{{8 \rightarrow \{ \text{status} : \text{idle}, \text{toSend} : \{\text{elect}\}, \text{sent} : \{\}, \text{handles} : \{\}\}}}\}

j \rightarrow 8
k \rightarrow 8
rank \rightarrow 12
tempNbrs \rightarrow ()
tempNbrs2 \rightarrow ()

Transition: output \text{RECEIVE} (\text{elect}, 1, 0) in automaton \text{sTreeLeader}(1)
on \text{drake.csail.mit.edu} at 9:29:08:496

Transition: output \text{RECEIVE} (\text{elect}, 14, 13) in automaton \text{sTreeLeader}(14)
on \text{tui.csail.mit.edu} at 9:29:08:528

Modified state variables:
P \rightarrow \{ \text{nbrs} : (10 13), \text{receivedElect} : (13), \text{sentElect} : (10), \text{status} : \text{idle}, \text{send} : \text{Map}{{10 \rightarrow \{\}}} \}
RM \rightarrow \text{Map}{{10 \rightarrow \{ \text{status} : \text{idle}, \text{toRecv} : \{\}, \text{ready} : \text{false} \}\}[13 \rightarrow \{ \text{status} : \text{idle}, \text{toRecv} : \{\}, \text{ready} : \text{false} \}\]}
SM \rightarrow \text{Map}{{10 \rightarrow \{ \text{status} : \text{idle}, \text{toSend} : \{\}, \text{sent} : \{\}, \text{handles} : \{\}\}[13 \rightarrow \{ \text{status} : \text{idle}, \text{toSend} : \{\}, \text{sent} : \{\}, \text{handles} : \{\}\}]

j \rightarrow 10
k \rightarrow 13
rank \rightarrow 14
tempNbrs \rightarrow ()
tempNbrs2 \rightarrow ()

Transition: output \text{SEND} (\text{elect}, 14, 10) in automaton \text{sTreeLeader}(14)
on \text{tui.csail.mit.edu} at 9:29:08:536

Modified state variables:
P \rightarrow \text{Tuple}, modified fields: \{[send \rightarrow \text{Map}, modified entries: \{[10 \rightarrow \text{Sequence}, elements: \{\}\}\]}
SM \rightarrow \text{Map}, modified entries: \{[10 \rightarrow \text{Tuple}, modified fields: \{[toSend \rightarrow \text{Sequence}, elements: \{\}\}\}

k \rightarrow 10
tempNbrs2 \rightarrow (13)

Modified state variables:
P \rightarrow \{ \text{nbrs} : (0 5), \text{receivedElect} : (0), \text{sentElect} : (5), \text{status} : \text{idle}, \text{send} : \text{Map}{{0 \rightarrow \{\}}} \}
RM \rightarrow \text{Map}{{0 \rightarrow \{ \text{status} : \text{idle}, \text{toRecv} : \{\}, \text{ready} : \text{false} \}\}[5 \rightarrow \{ \text{status} : \text{idle}, \text{toRecv} : \{\}, \text{ready} : \text{false} \}\]}
SM \rightarrow \text{Map}{{0 \rightarrow \{ \text{status} : \text{idle}, \text{toSend} : \{\}, \text{sent} : \{\}, \text{handles} : \{\}\}[5 \rightarrow \{ \text{status} : \text{idle}, \text{toSend} : \{\}, \text{sent} : \{\}, \text{handles} : \{\}\}]

j \rightarrow 0
k \rightarrow 0
rank \rightarrow 1
tempNbrs \rightarrow ()
tempNbrs2 \rightarrow (5)

Transition: output \text{SEND} (\text{elect}, 1, 5) in automaton \text{sTreeLeader}(1)
Modified state variables:
P → Tuple, modified fields: {
  send -> Map, modified entries: {
    5 -> Sequence, elements: SM → Map, modified entries: {
      5 -> Tuple, modified fields: {
        toSend -> Sequence, elements: tempNbrs2 → (0)
      }
    }
  }
}

transition: output RECEIVE(elect, 7, 11) in automaton sTreeLeader(7)

Modified state variables:
  k → 7
  rank → 11
  tempNbrs2 → ()
}

transition: output SEND(elect, 7, 3) in automaton sTreeLeader(7)

Modified state variables:
P → Tuple, modified fields: {
  send -> Map, modified entries: {
    3 -> Sequence, elements: SM → Map, modified entries: {
      3 -> Tuple, modified fields: {
        toSend -> Sequence, elements: tempNbrs2 → ()
      }
    }
  }
}

transition: output RECEIVE(elect, 10, 14) in automaton sTreeLeader(10)

Modified state variables:
  k → 14
  rank → 10
  tempNbrs2 → ()
}

transition: output SEND(elect, 10, 6) in automaton sTreeLeader(10)

Modified state variables:
P → Tuple, modified fields: {
  send -> Map, modified entries: {
    6 -> Sequence, elements: SM → Map, modified entries: {
      6 -> Tuple, modified fields: {
        toSend -> Sequence, elements: tempNbrs2 → ()
      }
    }
  }
}

transition: output RECEIVE(elect, 6, 10) in automaton sTreeLeader(6)

Modified state variables:
  k → 10
  rank → 6
  tempNbrs2 → ()
}
tempNbrs2 → ()

transition: output RECEIVE(elect, 8, 12) in automaton sTreeLeader(8)
on parrot.csail.mit.edu at 9:29:11:670
Modified state variables:
P → [nbrs: (12 9), receivedElect: (12), sentElect: (9), status: idle, send: Map{[12 -> Map{[12 -> [status: idle, toRecv: {}, ready: false]} [9 -> [status: idle, toRecv: {}]}] [9 -> [status: idle], j -> 12, k -> 12, rank -> 8, tempNbrs2 → ()}], tempNbrs → ()]

transition: output SEND(elect, 8, 9) in automaton sTreeLeader(8)
on parrot.csail.mit.edu at 9:29:11:684
Modified state variables:

transition: output RECEIVE(elect, 3, 7) in automaton sTreeLeader(3)
on parrot.csail.mit.edu at 9:29:12:228
Modified state variables:
P → [nbrs: (2 7), receivedElect: (7), sentElect: (2), status: idle, send: Map{[2 -> Map{[2 -> [status: idle, toRecv: {}, ready: false]} [7 -> [status: idle, toRecv: {}]}] [7 -> [status: idle], j -> 7, k -> 7, rank -> 3, tempNbrs2 → (12)]}, tempNbrs → ()]

transition: output SEND(elect, 3, 2) in automaton sTreeLeader(3)
on parrot.csail.mit.edu at 9:29:12:262

transition: output RECEIVE(elect, 9, 8) in automaton sTreeLeader(9)
on tui.csail.mit.edu at 9:29:12:573
Modified state variables:
P → [nbrs: (5 8), receivedElect: (8), sentElect: (5), status: idle, send: Map{[5 -> Map{[5 -> [status: idle, toRecv: {}, ready: false]} [8 -> [status: idle, toRecv: {}]}] [8 -> [status: idle], j -> 5, k -> 8, rank -> 9, tempNbrs2 → ()}], tempNbrs → ()]

Modified state variables:

transition: output SEND(elect, 9, 5) in automaton sTreeLeader(9)
on tui.csail.mit.edu at 9:29:12:730
Modified state variables:
P → Tuple, modified fields: {[send -> Map, modified entries: {[5 -> Sequence, elements: 5]}], SM → Map, modified entries: {[5 -> Tuple, modified fields: {[toSend -> Sequence, elements: 5]}]}
k → 5
tempNbrs2 → (8)

Initialization starts (5) on loon.csail.mit.edu at 9:29:12:852

Modified state variables:
P → [nbrs: (), receivedElect: (), sentElect: (), status: idle, send: Map{}]
RM → Map{}
SM → Map{}
j → 87
k → 87
rank → null
tempNbrs → ()
tempNbrs2 → ()

Initialization ends

transition: output RECEIVE(elect, 2, 3) in automaton sTreeLeader(2) on condor.csail.mit.edu at 9:29:14:748

Modified state variables:
P → [nbrs: (3 6), receivedElect: (3), sentElect: (6), status: idle, send: Map{[3 -> {]}],
RM → Map{[3 -> [status: idle, toRecv: {}, ready: false]] [6 -> [status: idle, toRecv: {}]]}
SM → Map{[3 -> [status: idle, toSend: {}, sent: {}, handles: {}]] [6 -> [status: idle, ]]
jk → 3
k → 3
rank → 2
tempNbrs → ()
tempNbrs2 → ()

transition: output SEND(elect, 2, 6) in automaton sTreeLeader(2) on condor.csail.mit.edu at 9:29:14:759

Modified state variables:
P → Tuple, modified fields: {[receivedElect -> (10 2)] [sendElect -> (5)] [send -> Map{[1 -> {]}],
RM → Map{[1 -> [status: idle, toRecv: {}, ready: false]] [4 -> [status: idle, toRecv: {}]]}
SM → Map{[1 -> [status: idle, toSend: {}, sent: {}, handles: {}]] [4 -> [status: idle, ]]
jk → 6
k → 2
rank → 5
tempNbrs → ()
tempNbrs2 → (10)

transition: output RECEIVE(elect, 6, 2) in automaton sTreeLeader(6) on d...
transition: output RECEIVE(elect, 5, 4) in automaton sTreeLeader(5)
on loon.csail.mit.edu at 9:29:15:492
Modified state variables:
P → Tuple, modified fields: {[receivedElect → (1 4)]}
k → 4
tempNbrs2 → ()

Modified state variables:
P → Tuple, modified fields: {[send → Map, modified entries: {[5 → Sequence, elements: {1 4}]]]
SM → Map, modified entries: {[5 → Tuple, modified fields: {[toSend → Sequence, elements: {1 4 6 9]}]]
k → 5
tempNbrs2 → (10 2)

transition: output RECEIVE(elect, 5, 9) in automaton sTreeLeader(5)
on loon.csail.mit.edu at 9:29:15:836
Modified state variables:
P → Tuple, modified fields: {[receivedElect → (1 4 9)] [sentElect → (6)] [send → Map]} k → 9

transition: output SEND(elect, 5, 6) in automaton sTreeLeader(5)
on loon.csail.mit.edu at 9:29:16:182
Modified state variables:
P → Tuple, modified fields: {[send → Map, modified entries: {[6 → Sequence, elements: {1 4 6 9]}]]
SM → Map, modified entries: {[6 → Tuple, modified fields: {[toSend → Sequence, elements: {1 4 6 9]}]]
k → 6
tempNbrs2 → (1 4 9)

transition: output RECEIVE(elect, 5, 6) in automaton sTreeLeader(5)
on loon.csail.mit.edu at 9:29:16:494
Modified state variables:
P → Tuple, modified fields: {[receivedElect → (1 4 6 9)]}
SM → Map, modified entries: {[6 → Tuple, modified fields: {[toSend → Sequence, elements: {1 4 6 9]}]} k → 6

transition: output RECEIVE(elect, 6, 5) in automaton sTreeLeader(6)
on drake.csail.mit.edu at 9:29:16:554
Modified state variables:
P → Tuple, modified fields: {[receivedElect → (10 2 5)] [status → elected]}
SM → Map, modified entries: {[5 → Tuple, modified fields: {[toSend → Sequence, elements: {1 4 6 9]}]} k → 6

transition: output leader() in automaton sTreeLeader(6)
on drake.csail.mit.edu at 9:29:16:559
Modified state variables:
P → Tuple, modified fields: {[status → announced]}