Processes and their Identifiers in Synchronous Network Systems

M.I.A. Stoelinga

Computing Science Institute/

CSI-R9807 March 1998

Computing Science Institute Nijmegen
Faculty of Mathematics and Informatics
Catholic University of Nijmegen
Toernooiveld 1
6525 ED Nijmegen
The Netherlands

Processes and their Identifiers in Synchronous Network Systems

Mariëlle Stoelinga

Computing Science Institute,
University of Nijmegen,
P.O. Box 9010, 6500 GL Nijmegen,
The Netherlands
marielle@cs.kun.nl

Abstract

This paper formalizes the two notions of "comparison based process" and of 'process isomorphism up-to process identifiers." The Key Lemma is proven which states that comparison based processes which are isomorphic up-to PIds behave very similarly, if the PIds in the processes in their enviromnents are what is called order equivalent.
Key words and phases: leader election, synchronous networks, comparison based processes, isomorphic processes, process identifiers, communication complexity.
AMS Subject Classification: 68Q10, 68Q22, 68Q25.
CR Subject Classification: F.1.1, F.1.2, F.2.m, F.3.3.

Nancy Lynch in Chapter Three of her book "Distributed algorithms" [Ly96] gives a proof of a lower bound result on a leader election problem: The amount of communication needed to solve this problem is $\Omega(n \log n)$. The result is stated in a synchronous ring network, consisting of processes that are isomprphic except for a unique process identifier – a unique PId for short. Furthermore, the algorithm solving the leader election problem should be comparison based, which means that the only operation on PIds is comparison.
The key to the lower bound result is a lemma expressing that a certain amount of symmetry can arise even in the presence of PIds. This symmetry can only be broken at the cost of $\Omega(n \log n)$ messages.
Several notions on the way to this result are introduced intuitively rather that formally. The proof of the Key Lemma takes some large and informal steps. This report provides a formal basis for all notions to complete a formal proof. In particular, formal definitions are proposed for the notions of "comparison based processes" and of "process isomorphism up-to PIds." This report provides syntax-free definitions of both notions for arbitrary network system. We prove and generalize the Key Lemma to any network system. We also sketch the lower bound result.

The crucial step in our formalization is the introduction of the operations pid and [ ], which retrieve and substitute respectively the PIds present in a message or state. These allow for access to the PIds in a state or message, without making further assumptions on their nature. Once a process contains a (single) PId it may copy it or send it to other processes though processing.
Using these operations we are able to express formally that two processes in a ring that are isomorphic up-to PIds, may only differ in the PIds of their start states. For ease of notation we firstly introduce this notion for ring networks and then for arbitrary networks.
Intuitively, a comparison based process is a process which, if implemented in code, performs no other operations on PIds than comparisons. Our definition uses the functions pid and [ ] again.
An important property of comparison based processes is that, comparison being the only serious operation on PIds, the relative ranks of PIds in states and messages are of interest rather than their particular values. In fact, the Key Lemma states that two processes in a network consisting of processes which are isomorphic except for a PId behave very similarly, if the processes in their neighbourhoods (which they receive messages from) contain PIds with the same relative ranks. We prove this lemma for ring networks first and then generalize it to other networks.

Relevance and related work

We believe that the main contribution of this report is the formalization of the concepts of "comparison based process" and of "isomorphic processes" in a synchronous network system.
The notion of comparison based process is introduced at a more syntactic level and more informally in [Ly96] and [FL87].
The notion of isomorphic processes (except for a PId), in our view, is more fundamental. Such processes often occur in real-world applications. It is therefore desirable to have an abstract and general definition of this notion. Moreover, the symmetry present in networks of isomorphic processes can be used to reduce the complexity of (verification) algorithms, c.f. [CFJ93], [ES95], [EA93], which introduce the notion of isomorphic processes in a more syntactic way.

Organization of the sections

This report is organized as follows. Section 1 introduces the synchronous network model. The operations pid and [ ] are specified axiomatically in Section 2. Then Section 3 defines the notion of process isomorphism up-to PIds in ring networks and Section 4 the notion of comparison based processes. We prove the Key Lemma to the lower bound result in Section 5 for rings and mention some obvious generalizations. Section 6 treats the generalization of the notion of process isomorphism up-to PIds and the Key Lemma to arbitrary networks. The reader is referred to Appendix A for notational conventions and a glossary of symbols.

1 The synchronous network model

This section recalls the general model of synchronous network systems. The ideas in this section are taken from Lynch [Ly96], Chapter 2. We deviate from Lynch's model in some minor details. Furthermore, we present our definitions at a higher level of formality and introduce some additional notations.
A *synchronous ring network system* consists of a number of processes, running on some pieces of hardware, processors, that communicate via channels. The adjective *synchronous* refers to the situation that the execution of the system, i. e. all processing and all communication, proceeds simultaneously, in infinitely many consecutive rounds. The systems begins with all processes in arbitrary start states. The processes, in lock step, repeatedly perform the following two steps: Firstly they simultaneously send messages through the channels, then they simultaneously do some processing, based on the current states and on inputs they received, in order to reach the next state. The message a process sends through a channel is indicated by the message generation function, the next state by the state transition function.
A *synchronous ring network system* consists of a number of processes, that communicate via channels. The systems begins with all processes in arbitrary start states. The processes simultaneously and repeatedly perform the following two steps: Firstly they send messages through the channels, then they move to the next state, using the current states and on inputs they received,

Convention 1.1 If M is some set of messages, then we assume that $\mathsf{null} \notin M$, as we write null as a placeholder for the absence of a message. We define $M_0 = M \cup \{\mathsf{null}\}$. Furthermore, we fix for the entire report that n, k are natural numbers.

Definition 1.2 A *synchronous network system*, or simply a *network* is a tuple $\langle G, P, M\rangle$, where G is a directed graph, P a V-tuple of processes and M a set of messages such that:

- $G = \langle V, E\rangle$, V is a finite set of vertices and $E \subseteq V^2$ a set of edges. If $(i, j) \in E$ then we write $i \to j$. For $i \in V$, $in(i)$ is the set of "incoming neighbours," that is $in(i) = \{j \in V \mid j \to i\}$ and $out(i) = \{j \in V \mid i \to j\}$ the set of "outgoing neighbours." The distance between i and j, denoted by $\mathrm{d}(i, j)$, is the length of a shortest path between i and j, if there is such a path, otherwise $\mathrm{d}(i, j) = \infty$.

- $P = (P_i)_{i\in V}$. The process P_i, which runs at node i, consists of the following components:

  - $states_i$, a (possibly infinite) set of states,
  - $start_i \subseteq states_i$, a non-empty set of start states,
  - $msgs_i : states_i \to M_0^{out(i)}$, the message generation function.
    $msgs_i(m)$ is the tuple of messages, indexed by $out(i)$, to be sent to the outgoing neighbours of P_i. We often write $msgs_{i\to j}(m)$ for $(msgs_i(m))_j$.

- $trans_i : states_i \times M_0^{in(i)} \to states_i$, the state transition function.
  $trans_i(s, \mu)$ yields the next state of process P_i, when being in state s and receiving the message μ_j from P_j, for all $j \in in(i)$.

We write $states = \bigcup_{i \in V} states_i$

The *network size* is the number of elements of V.

Convention 1.3 Since we only consider one system at the time, the following notations will do. Throughout this report S and M are sets of states and messages respectively. If G is a graph, then V denotes the set of vertices and E the edges. If P_i is some process, then we denote its component by $states_i, start_i, trans_i$ and $msgs_i$.

A special class of networks are ring networks. In order to facilitate discussion, the definition below fixes the names of the nodes and edges.

Definition 1.4
1. A *unidirectional ring* of size n is a graph $G = \langle V, E \rangle$ such that $V = \{1, \ldots, n\}$ and $E = \{1 \to 2, 2 \to 3, \ldots, n-1 \to n, n \to 1\}$.
2. A *bidirectional ring* of size n is a graph $G = \langle V, E' \rangle$, where V, E as above and $E' = E \cup \{2 \to 1, 3 \to 2, \ldots, n \to n-1, 1 \to n\}$.
3. A *unidirectional (bidirectional) ring network* is a network $\langle G, P, M \rangle$ such that G is a unidirectional (bidirectional) ring.

A ring network is either a uni- or bidirectional ring network. We often count modulo n in a ring network of size n; allowing 0 to be another name for n, $n+1$ for 1, etc. Furthermore, the *right* or *clockwise neighbour* of i is the node $i+1$; node $i-1$ the *left* or *counterclockwise* one.

Convention 1.5 The types of the state transition functions and the message generation functions in a ring network can be simplified, using the isomorphisms $M_0^{\{i-1\}} \cong M_0^{\{i+1\}} \cong M_0$ and $M_0^{\{i-1,i+1\}} \cong M_0^2$. For unidirectional rings we get:

$$trans_i : states_i \times M_0 \to states_i,$$
$$msgs_i : states_i \to M_0$$

and for bidirectional rings we get:

$$trans_i : states_i \times M_0^2 \to states_i,$$
$$msgs_i : states_i \to M_0^2.$$

By convention (m, m') denotes $(i-1 : m, i+1 : m') \in M_0^{\{i-1,i+1\}}$, so the message concerning the left neighbour is mentioned firstly and the message concerning the right one secondly.

As a running example, we consider the LCR algorithm – by Le Lann, Chang and Roberts [La77] and [CR79] – as described in [Ly96]. It solves the leader election problem in an unidirectional ring, which means that eventually exactly one of the processes becomes "leader." Later examples will show that the processes are isomorphic except for a PId and comparison based.

Example 1.6 (synchronous network system) Let R be a unidirectional ring of size n. Let $M = \{u_1, \ldots, u_n\} \subseteq \mathbb{N}$ be a set of n different objects; the value u_i will be associated to P_i. For each $i \in V$, define the process P_i as follows:

- $states_i = \{u_1, \ldots, u_n\} \times M_0 \times \{\mathsf{init}, \mathsf{unknown}, \mathsf{leader}\}$. A state should be thought of as a tuple $(u, send, status)$, where $u = u_i$ remains unchanged by the process, $send$ is the message to be send and $status$ indicates whether or not the process is in its initial state and whether or not it has declared itself leader.
- $start_i = \{(u_i, \mathsf{null}, \mathsf{init})\}$.
- $msgs_i : states_i \to M_0$, given by
  $msgs_i(u, send, status) =$ if $status = \mathsf{init}$ then u else $send$ fi.
- $trans_i : states_i \times M_0 \to states_i$, $trans_i((u, send, status), m)$ is described by the following pseudo code:

  $send :=$ if $m \neq \mathsf{null} \wedge m > u$ then m else null fi;
  $status :=$ if $m = u$ then leader else $\mathsf{unknown}$ fi.

The algorithm works in the following way: The process with the largest value u_i gets elected. Each process P_i sends the value u_i around the ring. When a process receives a non-null message, it compares it to u_i, the first field of its state. If the incoming message is greater than u_i, the process keeps passing this message; if it is less then it discards the message; if it is equal, then u_i has been passed through the entire ring, so it is the largest and the process declares itself leader.

1.1 The behaviour of a synchronous network system

Definition 1.7 1. A *state assignment function* for a network system $\langle G, P, M\rangle$ is a function which assigns a state to each process, i.e. a function $\nu : V \to states$ such that $\nu(i) \in states_i$ for all $i \in V$. We often write $\nu.i$ in stead of $\nu(i)$.

2. A *message assignment function* is a function which assigns a message to each channel, i.e. a function $\mu : E \to M_0$. We write $\mu.e$ in stead of $\mu(e)$.

Definition 1.8 An *execution* of a network system $\langle G, P, M\rangle$ is an infinite, alternating sequence

$$\nu_0, \mu_1, \nu_1, \mu_2, \nu_2, \ldots$$

of state and message assignment functions, starting with a state assignment function. This sequence should satisfy:

$$\nu_0.i \in start_i$$
$$\mu_{r+1}.i \to j = msgs_{i\to j}(\nu_r.i)$$
$$\nu_{r+1}.i = trans_i(\nu_r.i, (\mu_{r+1}.l \to i)_{l\in in(i)})$$

for all $r \in \mathbb{N}$, all $i, j \in V$ such that $i \to j$.

Notice that $(\mu_{r+1}.l \to i)_{l\in in(i)} \in M_0^{in(i)}$ is the tuple of messages that P_i receives in round $r+1$ from all its incoming neighbours. The state $\nu_r.i$ is called the state of process i in round r (or: after r rounds) of the execution; the message $\mu_r.i \to j$ is called the message sent by i to j in round r of the execution. If $\alpha = \nu_0, \mu_1, \nu_1, \ldots$ we also use $\dot{\alpha}_r \stackrel{\text{d}}{=} \nu_r$ and $\vec{\alpha}_r \stackrel{\text{d}}{=} \mu_r$.

Example 1.9 The diagram below shows the execution of the execution of the LCR algorithm (see Example 1.6), executing in ring of size 4, taking $u_1 = 4, u_2 = 3, u_3 = 2, u_4 = 1$.

	round 0	round 1	round 1	round 2	round 2
proc	states	msgs	state	msgs	states
P_1	(4,null,init)	4	(4,null,unknown)	null	(4,null,unknown)
P_2	(3,null,init)	3	(3,4,unknown)	4	(3,null,unknown)
P_3	(2,null,init)	2	(2,3,unknown)	3	(2,4,unknown)
P_4	(1,null,init)	1	(1,2,unknown)	2	(1,3,unknown)

	round 3	round 3	round 4	round 4	round 5
proc	msgs	states	msgs	states	
P_1	null	(4,null,unknown)	null	(4,null, leader)	...
P_2	null	(3,null,unknown)	null	(3,null,unknown)	...
P_3	4	(2,null,unknown)	null	(2,null,unknown)	...
P_4	3	(1,4,unknown)	4	(1,null,unknown)	...

Process 4 is elected as leader in round 4 of the execution. Then the processes neither send messages nor change to another state.

2 Dealing with PIds in processes

This section prepares for the definition of the notion of process isomorphism up-to PIds [1] as well as for the notion of "comparison based process." These notions are defined

[1] [Ly96] uses the word "uid" (unique identifier) and [FL87] "id" for what we call "PId." We deviate from this terminology because in general we do not assume that each process has a different PId.

independently in Sections 3 and 4 respectively. When combined, they form the setting of the Key Lemma.
An argument for introducing PIds is to break the symmetry present in networks consisting of entirely isomorphic processes. When starting in the same states, such processes remain in identical states throughout the entire execution. As a result, such processes can not solve the leader election problem.
However, PIds also make sense in other kind of networks containing very different processes. Once PIds are present in some process, it may manipulate them and send them around in various ways. A process that only uses comparison between PIds (not between a PId and a constant) is called comparison based. This notion is defined for any network in Section 4.
As we do not wish to make more assumptions on the nature of the PIds than necessary, all access on PIds in messages and states is performed via the operations $[\,]$ and pid introduced below.

Axioms 2.1 We assume the following functions and sets to be available on the sets M and S.

- A set U, the type of PIds. In examples we take $U = \mathbb{N}$.
- Two functions $\mathsf{pid_m} : M_0 \to U^*$ and $\mathsf{pid_s} : S \to U^*$, which retrieve a list of all PIds contained in a message and a state, respectively.
- Two partial functions $[\,]_\mathsf{m} : M_0 \times U^* \rightharpoonup M_0$ and $[\,]_\mathsf{s} : S \times U^* \rightharpoonup S$, substituting a sequence of PIds by some other sequence of the same length.

The subscripts $_\mathsf{m}$ and $_\mathsf{s}$ are omitted if it is clear whether the function is defined for messages or for states. This convention is also adopted for future objects defined on both states and messages.
The pairs of functions $\langle \mathsf{pid_m}, [\,]_\mathsf{m} \rangle$ and $\langle \mathsf{pid_s}, [\,]_\mathsf{s} \rangle$ should satisfy the following requirements, for all $v, w \in U^*, m \in M_0 \cup S$:

U-1 $m[v]{\downarrow} \iff |v| = |\mathsf{pid}(m)|$.

The substituent and the substituant should have the same length.

U-2 $m[v]{\downarrow} \implies \mathsf{pid}(m[v]) = v$

U-3 $m[\mathsf{pid}(m)] = m$.

Subsequent retrieval and (well-applied) substitution is the identity; information is neither created nor lost.

U-4 $m[v]{\downarrow} \wedge m[w]{\downarrow} \implies m[v][w] = m[w]$.

Substitution is destructive.

Moreover, the function $\mathsf{pid_m}$ should satisfy:

U-5 $\mathsf{pid_m}(\mathsf{null}) = \varepsilon$

For the sake of convenience we define

$$\mathsf{pidset_s}(s) = \mathsf{set}(\mathsf{pid_s}(s)),$$
$$\mathsf{pidset_m}(m) = \mathsf{set}(\mathsf{pid_m}(m)),$$

so pidset yields the set of all PIds in a state or message.

Remark 2.2 Although a more general treatment is possible, we have required that the number of PIds in a state or message is finite, mostly for sake of convenience. Moreover, this requirement is realistic in practice for physical processes only have a finite amount of memory.

Example 2.3 Reconsider the LCR algorithm from Example 1.6. The PIds in this algorithm are $U = \{u_1, \ldots, u_n\} \subseteq \mathbb{N}$. As $M = U$, we have

$$\mathsf{pid_m}(m) = \begin{cases} m & \text{if } m \in M, \\ \varepsilon & \text{if } m = \mathsf{null}. \end{cases}$$

Moreover, $S = \{u_1, \ldots, u_n\} \times M_0 \times \{\mathsf{unknown}, \mathsf{leader}, \mathsf{init}\}$. Then,

$$\mathsf{pid_s}\,(u, send, status) = [u] \star \mathsf{pid_m}(send) = \begin{cases} [u, send] & \text{if } send \in M, \\ [u] & \text{if } send = \mathsf{null}. \end{cases}$$

For example,

$$\mathsf{pid_s}\,(1, 1, \mathsf{leader}) = [1, 1],$$
$$\mathsf{pid_s}\,(1, \mathsf{null}, \mathsf{unknown}) = [1],$$
$$\mathsf{pid_m}(3) = [3],$$
$$(1, 1, \mathsf{leader})[3, 1] = (3, 1, \mathsf{leader}).$$

Notice that the LCR algorithm is independent of the implementation of the PIds; it works for any total order $(U, \leq)$.

3 Process isomorphism up-to PIds in ring networks

Now we as able to express what it means for two process in a ring that they are "isomorphic up-to PIds," also referred by as simply "isomorphic." Section 6 generalizes this notion to arbitrary networks.

Definition 3.1 Let $\langle R, P, M \rangle$ be a ring network and $i, j \in V$. The processes P_i and P_j. are *isomorpic up-to PIds*, notation $P_i \approx_{\text{ring}} P_j$, if the following requirements are met:

IP-1 $states_i = states_j$.

IP-2 $\#\, start_i = \#\, start_j = 1$.

The processes P_i and P_j have a single start state, denoted by s_i^0 and s_j^0 resp.

IP-3 $|\mathsf{pidset}(s_i^0)| = \left|\mathsf{pidset}(s_j^0)\right| = 1$.

Each process has a single PId present in its start state. Process i's PId is denoted by u_i.

IP-4 $s_i^0[\mathsf{pid}(s_j^0)] = s_j^0$.

Except for the PIds the start states, P_i and P_j are equal.

IP-5 The following functions are equal.

$$\begin{aligned} trans_i &= trans_j, \\ msgs_{i\to i+1} &= msgs_{j\to j+1}, \text{ for all rings,} \\ msgs_{i\to i-1} &= msgs_{j\to j-1}, \text{ for bidirectional rings only.} \end{aligned}$$

Notice that the start state may contain multiple occurrences of its PId. Axiom IP-4 requires that all start states have that same number of occurrences. Furthermore, axiom IP-4 implies $s_j^0[\mathsf{pid}(s_i^0)] = s_i^0$.

Example 3.2 It is easy to check that the processes in the LCR algorithm in 1.6 meet the requirements IP1 – IP5, so all processes in the LCR algorithm are isomorphic up-to PIds.

Remark 3.3 The notion of isomorphic processes up-to PIds can also be interpreted as "isomorphic processes performing on different input data." However, this requirement is rather weak. Take for instance the unidirectional ring of processes described by

$$\begin{aligned} &states_i = U \times \mathbb{N}, \\ &start_i = \{(u_i, 0)\}, \\ &trans_i((u,n),m) = (u, PrimRec(u,m)), \\ &msgs_{i\to i+1}(u,n) = u. \end{aligned}$$

These processes satisfy the axioms in 3.1, but behave very differently. $PrimRec(n,m)$ is the n^{th} primitive recursive function on input m. The only reason for using *primitive* recursive functions is because these are total.

4 Comparison based processes

As explained before, the syntactic idea behind the notion of comparison based processes is that comparison between PIds – not between a PId an a constant – the respect to

some order $\leq$ on U is the only serious operation allowed on PIds; we also admit syntactic operations like copying, appending PIds to lists, etc.
The intuition behind the semantic definition is the following. A comparison based process can not really distinguish between e.g. the state $(2, 3, \mathsf{unknown})$ and the state $(3, 8, \mathsf{unknown})$, that is between states or messages such that

1. The objects are equal except for their PIds.

2. All comparisons between the PIds in both objects yield the same results.

Property 1 is formally introduced as the notion of *σ-correspondence*. For technical reasons in the proof of the Key Lemma, the substitution of PIds is performed by a partial function $\sigma : U \rightharpoonup U$. The property 2 is formally expressed in the notion of *order equivalence*. However, we use an equivalent formulation in the definition of comparison based processes, expressing that the PIds in the objects can be obtained from each other by application of a strictly monotone function. Then a process is called comparison based if it the result of the *msgs* and *trans* applied to σ-corresponding objects yields σ-corresponding objects, for any strictly monotone σ.
From now on we assume that the set U of PIds is equipped with a total order $\leq$. We also assume that $\mathcal{I}$ is some index set and that X_i is either S or M_0, for all $i \in \mathcal{I}$. Note that the sets S, $S \times M_0^{in(i)}$ and $M_0^{out(i)}$ can be considered as dependent products $\prod_{i \in \mathcal{I}} X_i$. We use different brackets, () and $\langle\rangle$, only for sake of readability.

4.1 Order equivalence

Order equivalence expresses that two sequences of PIds have the same relative ranks.

Definition 4.1 For two sequences $v, w \in U^k$ we define

$$v \sim w \stackrel{\mathrm{d}}{=} \forall i, j \leq k[v_i \leq v_j \iff w_i \leq w_j].$$

The relation $\sim$ is called *order equivalence*. It is not difficult to see that it is an equivalence relation.

Example 4.2 $(2, 7, 5) \sim (2, 5, 3) \nsim (2, 3, 5)$.

Fact 4.3 *For all v,w,v',w' we have*

$$v \sim w \wedge v' \sim w' \stackrel{\Rightarrow}{\not\Leftarrow} v \star w \sim v' \star w'.$$

Lemma 4.4 *The following statements are equivalent for all $v, w \in U^k$:*

1. $v \sim w$,

2. $\forall i, j \leq k[v_i < v_j \iff w_i < w_j]$,

3. *There exists a partial function* $\sigma : U \rightharpoonup U$ *such that*

$$\sigma(v_i) = w_i$$

for all $i \in \mathcal{I}$ *and this function is strictly monotone. Notice that* $\mathsf{pidset}(v_i) = \{v_1 \dots v_k\} \subseteq \mathrm{Dom}(\sigma)$.

Example 4.5 As $(2,7,5) \sim (2,5,3)$, the partial function $\{2 \mapsto 2, 7 \mapsto 5, 5 \mapsto 3\}$ is strictly monotone.

4.2 Correspondence

We define the notion of *correspondence via a substitution* σ on S, on M and on dependent products of these. An object x corresponds with y via σ if we obtain y from x by applying the substitution σ to all PIds in x. Then x and y are the same except for their PIds. Firstly, we extend the definition of [ ] by allowing functions to be used for the substitution.

Definition 4.6 Let $\sigma : U \rightharpoonup U$ be a function. Let $m \in M$ and write $\mathsf{pid_m}(m) = v = v_1 \dots v_k$. Define

$$m[\sigma]_{\mathsf{m}} \stackrel{\mathrm{d}}{=} m[\sigma(v_1) \dots \sigma(v_k)]_{\mathsf{m}}.$$

By convention, a function is undefined whenever one of its arguments are so. Therefore, $m[\sigma]_{\mathsf{m}}\downarrow \iff \{v_1, \dots, v_k\} \subseteq \mathrm{Dom}(\sigma)$. Similarly, $s[\sigma]_{\mathsf{s}}$ is defined for $s \in S$ and subscripts are omitted like before.

Definition 4.7 1. Let $\sigma : U \rightharpoonup U$ and $m, m' \in M$. Define

$$m \xrightarrow{\sigma} m' \stackrel{\mathrm{d}}{=} m[\sigma]_{\mathsf{m}} = m'.$$

Similarly, define $s \xrightarrow{\sigma} s'$ for $s, s' \in S$.

2. For $x, y \in \prod_{i \in \mathcal{I}} X_i$, $\sigma : U \rightharpoonup U$, define

$$x \xrightarrow{\sigma} y \stackrel{\mathrm{d}}{=} \forall i \in \mathcal{I}[x_i \xrightarrow{\sigma} y_i].$$

If $x \xrightarrow{\sigma} y$ we say that x and y *correspond via* σ or simply that they are σ*-corresponding.*

Lemma 4.8 *For all* $x, y \in \prod_{i \in \mathcal{I}} X_i$ *and all* $\sigma, \tau : U \rightharpoonup U$ *we have*

$$x \xrightarrow{\sigma} y \wedge \sigma \subseteq \tau \implies x \xrightarrow{\tau} y.$$

Example 4.9 We investigate what correspondence boils down to for the states and the messages in the LCR algorithm from 1.6.

On states
When do we have $(u, send, status) \xrightarrow{\sigma} (u', send', status')$?

case 1 $send = \mathsf{null}$. Then $(u, send, status)[\sigma] = (\sigma(u), send, status)$. So we have

$$(u, send, status) \xrightarrow{\sigma} (u', send', status') \iff u' = \sigma(u) \wedge send' = send = \mathsf{null} \wedge status' = status.$$

case 2 $send \neq \mathsf{null}$. Then $(u, send, status)[\sigma] = (\sigma(u), \sigma(send), status)$. So we have

$$(u, send, status) \xrightarrow{\sigma} (u', send', status') \iff u' = \sigma(u) \wedge send' = \sigma(send) \wedge status' = status.$$

Lemma 4.4.3 implies that if σ is monotone then $(u, send) \sim (u', send')$.

Furthermore, it is easy to see that

$$\begin{aligned}(u, send, status) \xrightarrow{\sigma} (u', send', status') \implies\\ (u, u, status) \xrightarrow{\sigma} (u', u', status') \wedge\\ (u, send, \mathsf{unknown}) \xrightarrow{\sigma} (u', send', \mathsf{unknown}) \wedge\\ (u, \mathsf{null}, status) \xrightarrow{\sigma} (u', \mathsf{null}, status') \wedge\\ send \xrightarrow{\sigma} send'.\end{aligned}$$

On states and messages
Similarly, we get on $states \times M_0^{in(i)}$,

$$\begin{aligned}\langle (u, send, status), m\rangle \xrightarrow{\sigma} \langle (u', send', status'), m'\rangle \iff\\ (m = m' = \mathsf{null} \vee m' = \sigma(m)) \wedge\\ u' = \sigma(u) \wedge\\ (send = send' = \mathsf{null} \vee send' = \sigma(send)) \wedge\\ status' = status.\end{aligned}$$

4.3 Comparison based processes

If $x \xrightarrow{\sigma} y$ and σ is strictly monotone, then it is easy to see that $(\mathsf{pid}\, x_i)_{i\in\mathcal{I}} \sim (\mathsf{pid}\, y_i)_{i\in\mathcal{I}}$ — choose a fixed order on $\mathcal{I}$ such that $(\mathsf{pid}\, x_i)_{i\in\mathcal{I}}, (\mathsf{pid}\, y_i)_{i\in\mathcal{I}}$ can be considered as elements of U^*. So, x and y are equal except for their PIds and all comparisons between PIds in x and in y yield the same results. As we have argued in the introduction to this section, a comparison based process should yield similar results on these objects. We can make this more precisely now by saying that such a process yields σ-corresponding results on x and y. Thus, a process is *comparison based* if both its state transition and its message function preserve σ-correspondence, for any strictly monotone σ.

Definition 4.10 Let $\langle G, P, M\rangle$ be a synchronous network system. The process P_i is called *comparison based* if for all $s, s' \in states_i$, for all $\mu, \mu' \in M_0^{in(i)}$ and for all strictly monotone $\sigma : U \rightarrowtail U$:

CB-1 $\langle s, \mu\rangle \xrightarrow{\sigma} \langle s', \mu'\rangle \implies trans_i(s, \mu) \xrightarrow{\sigma} trans_i(s', \mu')$.

CB-2 $s \xrightarrow{\sigma} s' \implies (msgs_{i\to j}(s))_{j\in out(i)} \xrightarrow{\sigma} (msgs_{i\to j}(s'))_{j\in out(i)}$.

Notice that the relation $\xrightarrow{\sigma}$ above is used respectively on the sets $states_i \times M_0^{in(i)}$, $states_i$, $states_i$ and $M_0^{in(i)}$.

Example 4.11 All processes in the LCR algorithm from 1.6 are comparison based. We use some properties states in Example 4.9

PROOF: Consider the process P_i. Let $s, s' \in states_i$, $m, m' \in M_0$. We write $s = (u, send, status)$, $s' = (u', send', status')$. Let $\sigma : U \rightarrowtail U$ be a strictly monotone function.

CB-2 the message function: Assume $s \xrightarrow{\sigma} s'$. Then $status = status'$.

case 1 $status = status' = \mathsf{init}$. Then

$$msgs_i(s) = u \xrightarrow{\sigma} u' = msgs_i(s').$$

case 2 $status = status' \neq \mathsf{init}$. Then

$$msgs_i(s) = send \xrightarrow{\sigma} send' = msgs_i(s').$$

CB-1 the transition function: Assume $(s, m) \xrightarrow{\sigma} (s', m')$. Then $m = m' = \mathsf{null}$ or $(m, u) \sim (m', u')$.

case 1 $m = \mathsf{null}$. Then

$$\begin{aligned} trans_i(s, m) &= (u, \mathsf{null}, \mathsf{unknown}) \\ &\xrightarrow{\sigma} (u', \mathsf{null}, \mathsf{unknown}) \\ &= trans_i(s', m'). \end{aligned}$$

case 2 $m \neq \mathsf{null}$. Then $m' \neq \mathsf{null}$ and $(m, u) \sim (m', u')$ by monotony of σ.

case 2.1 $m > u$. Then $m' > u'$ and

$$\begin{aligned} trans_i(s, m) &= (u, u, \mathsf{unknown}) \\ &\xrightarrow{\sigma} (u', u', \mathsf{unknown}) \\ &= trans_i(s', m'). \end{aligned}$$

case 2.2 $m = u$. Then $m' = u'$ and

$$\begin{aligned} trans_i(s, m) &= (u, \mathsf{null}, \mathsf{leader}) \\ &\xrightarrow{\sigma} (u', \mathsf{null}, \mathsf{leader}) \\ &= trans_i(s', m'). \end{aligned}$$

case 2.3 $m < u$. Then $m' < u'$, so

$$\begin{aligned} trans_i(s, m) &= (u, \mathsf{null}, \mathsf{unknown}) \\ &\xrightarrow{\sigma} (u', \mathsf{null}, \mathsf{unknown}) \\ &= trans_i(s', m'). \end{aligned}$$

□

Remark 4.12 Some alternative formulations of CB-1 and CB-2 are possible. The proof of the Key Lemma uses the formulation given in the definition above.

1. CB-1 is equivalent to

   $$s[\sigma]{\downarrow} \wedge \mu[\sigma]{\downarrow} \implies trans_i(s[\sigma], \mu[\sigma]) = (trans_i(s, \mu))\,[\sigma].$$

   A similar reformulation can be given for CB-2.

2. The universal quantification over σ can be replaced by a specific σ. Let $s, s' \in states_i$ and $\mu, \mu' \in M_0^{in(i)}$. If there is a $\sigma : U \rightharpoonup U$ such that $(s, \mu) \xrightarrow{\sigma} (s', \mu')$ then there exists a smallest one (w.r.t. to function inclusion), let us say $\tau = \tau_{s,\mu,s',\mu'}$. Then the following statements are equivalent:

   1. $(s, \mu) \xrightarrow{\sigma} (s', \mu') \implies trans_i(s, \mu) \xrightarrow{\sigma} trans_i(s', \mu')$, for all σ,
   2. $(s, \mu) \xrightarrow{\tau} (s', \mu') \implies trans_i(s, \mu) \xrightarrow{\tau} trans_i(s', \mu')$.

   Again, a similar reformulation can be given for CB-2.

5 The Key Lemma and the lower bound result

This section proves the Key Lemma for rings networks. The point of this lemma is the following: suppose we have two processes P_i and P_j in a bidirectional ring of comparison based processes and isomorphic processes up-to PIds. Assume furthermore that the sequences of the PIds contained in the processes lying within distance k are order equivalent, i.e. $(u_{i-k}, u_{i-k+1} \ldots u_{i-1}, u_i, u_{i+1} \ldots u_{i+k}) \sim (u_{j-k}, u_{j-k+1} \ldots u_{j-1}, u_j, u_{j+1} \ldots u_{j+k})$. The start states of all processes are corresponding by axiom IP-4. Comparison based processes behave similarly when they are in corresponding states and get corresponding messages, so, P_i and P_j are in corresponding states within k rounds of the execution; until distinguishing information has had a chance to propagate to the processes P_i and P_j.
Throughout this section, $\langle R, P, M\rangle$ denotes a bidirectional ring consisting of comparison based processes that are isomorphic up-to PIds and n denotes its network size. Recall we count modulo n in such networks. Recall also that $\dot{\alpha}_{r,i}$ is the state of process P_i in round r and $\vec{\alpha}_{r,e}$ the message sent through channel e in round r. The PId of process i is denoted by u_i.
We need a few more definitions to formulate the lemma.

Definition 5.1 Define the k-neighbourhood of process P_i, notation $N_i(k)$, as the sequence in U^{2k+1} containing the PIds of process P_i's neighbours within distance k, i.e.

$$N_i(k) \stackrel{\mathrm{d}}{=} (u_{i-k}u_{i-k+1}\ldots u_{i-1}u_iu_{i+1}\ldots u_{i+k}).$$

Notice that, if $k \geq \lfloor \frac{n}{2} \rfloor$, then $N_i(k)$ contains the PIds of all processes.

Notation 5.2 If $N_i(k) \sim N_j(k)$ then Lemma 4.4 implies that the function in

$$\{u_{i-k}, u_{i-k+1}, \ldots, u_{i+k}\} \to U,$$
$$u_{i+l} \mapsto u_{j+l},$$

for every l, $i-k \leq l \leq i+k$, is well-defined and strictly monotone. This function is denoted by $\upsilon_{i,j,k}$.

Proposition 5.3 *For all $i, j \in V, k \in \mathbb{N}$ we have the following function inclusions*

$$\upsilon_{i-1,j-1,k-1} \subseteq \upsilon_{i,j,k}, \quad \upsilon_{i+1,j+1,k-1} \subseteq \upsilon_{i,j,k}, \quad \upsilon_{i,j,k-1} \subseteq \upsilon_{i,j,k}.$$

Definition 5.4 1. A message assignment function μ is called *active* if at least one non-null message is sent, i.e. $\exists e \in E[\mu.e \neq \mathsf{null}]$. A round > 0 is called active if its message assignment function is so.

2. For $r \in \mathbb{N}$ define $\mathsf{actives}(\alpha, r) = \#\{r' \mid r' \leq r \wedge \alpha_{r'} \text{ is active}\}$.

5.1 The Key Lemma

Now we can formulate and prove the Key Lemma for rings. The result is due to [FL87] and [Ly96]. Our result is slightly more general and the proof we present is more precise and uses fewer case distinctions. Section 6 treats its generalization to arbitrary networks.

Lemma 5.5 (Key Lemma) *Let $\langle R, P, M\rangle$ be a bidirectional ring network consisting of comparison based processes that are isomorphic up-to PIds. Then for all $k \in \mathbb{N}$, for all executions α of the ring we have*

> *If P_i and P_j are processes with order equivalent k-neighbourhoods, then at any point r in α such that $\dot{\alpha}_0$, $\vec{\alpha}_1$,$\dot{\alpha}_1$,$\ldots$ $\vec{\alpha}_r$,$\dot{\alpha}_r$ contains at most k active rounds, the states $\dot{\alpha}_r.i$ and $\dot{\alpha}_r.j$ are corresponding through $\upsilon_{i,j,k}$.*

Proof: Before presenting the formal proof, we give a sketch. The proof proceeds by induction on r. As P_i and P_j are processes with order equivalent k-neighbourhoods, the function $\upsilon_{i,j,k}$ is well-defined and strictly monotone. Furthermore, if P_i and P_j are corresponding via $\upsilon_{i-1,j-1,k-1}$ or via $\upsilon_{i+1,j+1,k-1}$ then also via $\upsilon_{i,j,k}$, by Lemma 5.3. The induction basis follows immediately from the fact that P_i and P_j are isomorphic up-to PIds. The induction step distinguishes between two cases: round $r+1$ is active and round $r+1$ is not active. In both cases we prove

1. The processes P_i and P_j are in $\upsilon_{i,j,k-1}$- corresponding states in round r by IH.
2. The processes receive $\upsilon_{i,j,k}$-corresponding messages in round $r+1$.
   - If $r+1$ is non-active this follows by the fact null-messages that are corresponding via any σ.
   - If $r+1$ is active it follows by application of the IH to the neighbours of P_i and P_j: these are in $\upsilon_{i,j,k}$-corresponding states in round r, so they send $\upsilon_{i,j,k}$-corresponding messages in round $r+1$.
3. Then P_i and P_j are in $\upsilon_{i,j,k}$-corresponding states in round $r+1$, for P_i and P_j are comparison based and isomorphic: applying the transition functions to $\upsilon_{i,j,k}$-corresponding states yield $\upsilon_{i,j,k}$-corresponding states, for $\upsilon_{i,j,k}$ is strictly monotone.

Now, for the formal proof, let α be an execution. By induction on r we prove for all r:

$$\forall i,j \in V \forall k \in \mathbb{N}[\mathsf{actives}(\alpha, r) \leq k \wedge N_i(k) \sim N_j(k) \quad \Longrightarrow \quad \dot{\alpha}_{r\cdot i} \xrightarrow{\upsilon_{i,j,k}} \dot{\alpha}_{r\cdot j}] \qquad (*)$$

induction basis: $r = 0$
Let $i,j \in V$. Let $k \in \mathbb{N}$. Assume $N_i(k) \sim N_j(k)$. Then $\upsilon_{i,j,k}$ is well-defined and strictly monotone. We prove $\dot{\alpha}_{0\cdot i} \xrightarrow{\upsilon_{i,j,k}} \dot{\alpha}_{0\cdot j}$:
Axiom IP-4 requires that $s_i^0[\mathsf{pid}(s_i^0)] = s_j^0$. The sequence $\mathsf{pid}(s_i^0)$ only contains occurrences of u_i, $\mathsf{pid}(s_j^0)$ only of u_j and $\upsilon_{i,j,k}$ maps u_i to u_j, so we have $s_i^0[\upsilon_{i,j,k}] = s_j^0$, i.e. $s_i^0 = \dot{\alpha}_{0\cdot i} \xrightarrow{\upsilon_{i,j,k}} \dot{\alpha}_{0\cdot j} = s_j^0$.

induction step: Fix $r \in \mathbb{N}$ such that formula (*) holds. Let $i,j \in V$, let $k \in \mathbb{N}$. Assume that $\mathsf{actives}(\alpha, r+1) \leq k$ and $N_i(k) \sim N_j(k)$. Then $\upsilon_{i,j,k}$ is well-defined and strictly monotone.

case 1: round $r+1$ is not active.
Then P_i and P_j receive only null-messages in round $r+1$. It follows from Axiom U-5 that null-messages are corresponding via every substitution. So $\mathsf{null} \xrightarrow{\upsilon_{i,j,k}} \mathsf{null}$. As $\mathsf{actives}(\alpha, r) \leq \mathsf{actives}(\alpha, r+1) \leq k$ the IH yields $\dot{\alpha}_{r\cdot i} \xrightarrow{\upsilon_{i,j,k}} \dot{\alpha}_{r\cdot j}$. By Definition 4.7.3 we have

$$\langle \dot{\alpha}_{r\cdot i}, (i-1 : \mathsf{null}, i+1 : \mathsf{null})\rangle \xrightarrow{\upsilon_{i,j,k}} \langle \dot{\alpha}_{r\cdot j}, (j-1 : \mathsf{null}, j+1 : \mathsf{null})\rangle.$$

So, we have isomorphic processes in $\upsilon_{i,j,k}$-corresponding states, receiving $\upsilon_{i,j,k}$-corresponding messages. Because $\upsilon_{i,j,k}$ is strictly monotone, they move to $\upsilon_{i,j,k}$-corresponding next states by IP-5 and CB-1:

$$\begin{aligned}
\dot{\alpha}_{r+1\cdot i} &= trans_i(\dot{\alpha}_{r\cdot i}, (i-1 : \mathsf{null}, i+1 : \mathsf{null})) \\
&= trans_j(\dot{\alpha}_{r\cdot i}, (j-1 : \mathsf{null}, j+1 : \mathsf{null})) \\
&\xrightarrow{\upsilon_{i,j,k}} trans_j(\dot{\alpha}_{r\cdot j}, (j-1 : \mathsf{null}, j+1 : \mathsf{null}))) \\
&= \dot{\alpha}_{r+1\cdot j}.
\end{aligned}$$

case 2: round $r+1$ is active.
We apply the IH to three cases: with the processes $i-1, j-1$ and k active rounds, with $i+1, j+1$ and k active rounds, and with i, j, k. We make the following observations:

1. As round $r+1$ is active, we have that $k > 0$ and the execution fragment $\dot{\alpha}_{0\cdot}, \vec{\alpha}_1, \ldots \vec{\alpha}_r, \dot{\alpha}_r$ contains at most $k-1$ active rounds.

2. Both the processes P_{i-1}, P_{j-1} and P_{i+1}, P_{j+1} have order equivalent $k-1$-neighbourhoods.

Applying the IH with $(i-1, j-1, k-1)$ and with $(i+1, j+1, k-1)$ yields that P_{i-1} and P_{j-1} are in corresponding states in round r, and so are P_{i+1} and P_{j+1}:

$$\dot{\alpha}_{r\cdot i-1} \xrightarrow{\upsilon_{i-1,j-1,k-1}} \dot{\alpha}_{r\cdot j-1} \quad \text{and} \quad \dot{\alpha}_{r\cdot i+1} \xrightarrow{\upsilon_{i+1,j+1,k-1}} \dot{\alpha}_{r\cdot j+1}.$$

Now, application of IP-5 and CB-2 leads to the conclusion that P_{i-1} and P_{j-1} send $\upsilon_{i-1,j-1,k-1}$-corresponding messages:

$$\begin{aligned}\vec{\alpha}_{r+1\cdot i-1} &= msgs_{i-1}(\dot{\alpha}_{r\cdot i-1}) \\ &= msgs_{j-1}(\dot{\alpha}_{r\cdot i-1}) \\ &\xrightarrow{\upsilon_{i-1,j-1,k-1}} msgs_{j-1}(\dot{\alpha}_{r\cdot j-1}) \\ &= \vec{\alpha}_{r+1\cdot j-1}\end{aligned}$$

We then obtain by Definition 4.7

$$\vec{\alpha}_{r+1\cdot i-1\to i} \xrightarrow{\upsilon_{i-1,j-1,k-1}} \vec{\alpha}_{r+1\cdot j-1\to j}$$

and by $\upsilon_{i-1,j-1,k-1} \subseteq \upsilon_{i,j,k}$ and by Lemma 4.8

$$\vec{\alpha}_{r+1\cdot i-1\to i} \xrightarrow{\upsilon_{i,j,k}} \vec{\alpha}_{r+1\cdot j-1\to j}.$$

Similarly, we obtain for the clockwise neighbours

$$\vec{\alpha}_{r+1\cdot i+1\to i} \xrightarrow{\upsilon_{i,j,k}} \vec{\alpha}_{r+1\cdot j+1\to j}.$$

Another application of the IH yields

$$\dot{\alpha}_{r\cdot i} \xrightarrow{\upsilon_{i,j,k}} \dot{\alpha}_{r\cdot j}.$$

Like in case 1, we have

$$\begin{aligned}&\langle \dot{\alpha}_{r\cdot i}, (j-1 : \vec{\alpha}_{r+1\cdot i-1\to i}, j+1 : \vec{\alpha}_{r+1\cdot i+1\to i})\rangle \\ &\xrightarrow{\upsilon_{i,j,k}} \langle \dot{\alpha}_{r\cdot j}, (j-1 : \vec{\alpha}_{r+1\cdot j-1\to j}, j+1 : \vec{\alpha}_{r+1\cdot j+1\to j})\rangle.\end{aligned}$$

by Definition 4.7.

Finally, Axioms CB-1 and IP-5 yield that

$$\begin{aligned}\dot{\alpha}_{r+1\cdot i} &= trans_i(\dot{\alpha}_{r\cdot i}, (i-1 : \vec{\alpha}_{r+1\cdot i-1\to i}, i+1 : \vec{\alpha}_{r+1\cdot i+1\to i})) \\ &= trans_j(\dot{\alpha}_{r\cdot i}, (j-1 : \vec{\alpha}_{r+1\cdot i-1\to i}, j+1 : \vec{\alpha}_{r+1\cdot i+1\to i})) \\ &\xrightarrow{\upsilon_{i,j,k}} trans_j(\dot{\alpha}_{r\cdot j}, (j-1 : \vec{\alpha}_{r+1\cdot j-1\to j}, j+1 : \vec{\alpha}_{r+1\cdot j+1\to j})) \\ &= \dot{\alpha}_{r+1\cdot j}.\end{aligned}$$

□ □

Example 5.6 Reconsider the execution of the LCR algorithm, given in Example 1.9. Although the algorithm works in a unidirectional ring, we can still apply the Key Lemma, for we may imagine each process sending all null-messages to the counterclockwise neighbours. This operation preserves the properties of being comparison based and consisting of isomorphic processes.

	round 0	round 1	round 1	round 2	round 2
proc	states	msgs	state	msgs	states
P_1	(4,null,init)	4	(4,null,unknown)	null	(4,null,unknown)
P_2	(3,null,init)	3	(3,4,unknown)	4	(3,null,unknown)
P_3	(2,null,init)	2	(2,3,unknown)	3	(2,4,unknown)
P_4	(1,null,init)	1	(1,2,unknown)	2	(1,3,unknown)

We apply the Key Lemma taking $r = 1$. All rounds before the election of the leader are active. We have $(4,3,2) = N_2(1) \sim N_3(1) = (3,2,1)$. Now the lemma concludes that P_2 and P_3 are in states corresponding via $\upsilon_{2,3,1} = \{4 \mapsto 3, 3 \mapsto 2, 2 \mapsto 1\}$ in round 1 of the execution. Indeed, $(3,4,\mathsf{unknown}) \xrightarrow{\upsilon_{2,3,1}} (2,3,\mathsf{unknown})$. We also see that $N_1(1) \nsim N_2(1)$ and that P_1 and P_2 are not in corresponding states.

5.2 From the Key Lemma to the lower bound result

This subsection formulates the lower bound result in our own terminology and sketches the steps in the proof by [Ly96]. The leader election problem assumes that all processes in the network have different PIds.

Theorem 5.7 *Solving the leader election problem in a bidirectional ring of size n by comparison based processes being isomorphic up-to PIds requires $\Omega(n \log n)$ messages.*

The steps taken in the proof by [Ly96] are:

LB-1 An assignment of PIds to the nodes of a ring of size n is called *c-symmetric* if there are at least $\lfloor \frac{c \cdot n}{2k+1} \rfloor$ order equivalent k-neighbourhoods, $\sqrt{n} \leq 2k+1 \leq n$.

LB-2 There exists a constant c such that, for all $n \in \mathbb{N}$, there is a c-symmetric ring of size n.

LB-3 A network system consisting of comparison based processes which are isomorphic up-to PIds and whose PIds in the processes form a c-symmetric ring, has more than $\lfloor \frac{c \cdot n - 2}{4} \rfloor$ active rounds before the leader is elected, if n is large enough.

LB-4 In order to prove the lower bound result, fix c as in LB-2 and take n sufficiently large. Let R be a c-symmetric ring of size n. Then every active round r with $r \geq \sqrt{n+1}$ such that the leader has not yet been elected sends more than $\frac{c \cdot n}{2r-1}$ messages through the network. This follows from: there is at least one process P_i that sends a non-null message. As R is c-symmetric, there are at least $\frac{c \cdot n}{2r-1}$ processes whose $r-1$-neighbourhoods are order equivalent to $N_i(r-1)$. Then it follows from the Key Lemma that all those $\frac{c \cdot n}{2r-1}$ processes are in corresponding states in round $r-1$ by the Key Lemma, so all sent a non-null message in round r.

Because the network needs at least $\lfloor \frac{c \cdot n - 2}{4} \rfloor$ active rounds before the leader is elected, the total number of messages send before the election is larger than

$$\sum_{r=0}^{\lfloor \frac{c \cdot n - 2}{4} \rfloor} \text{number of messages sent in round } r \geq \sum_{r=\sqrt{n+1}}^{\lfloor \frac{c \cdot n - 2}{4} \rfloor} \frac{c \cdot n}{2r-1}$$

This last sum is shown to be $\Omega(n \log n)$ by integral approximation.

5.3 Generalizations of the result

This subsection briefly and informally discusses some generalizations of the Key Lemma and of the lower bound result.

Multiple start states

Definition 3.1, axiom IP-2, of "processes isomorphism up-to PIds" requires the processes to have a single start state. This is mainly done for the sake of simplicity.
There are several weaker alternatives for this axiom such that the lower bound result still holds. This is so because proving a lower bound only requires one "bad" execution.

Other relations than $\leq$

Given some relation R on U, there exists a sensible notion of *R-based process* such that the Key Lemma holds when the processes are R-based instead of comparison based: generalizing the notion of order equivalence to R-equivalence is easy. Then we can adapt the notion of congruence to R-congruence by replacing "order equivalence" by *R-equivalence*. The proof of the Key Lemma is generalized in the same way. The lower bound result need not hold for any relation, for the existence of large R-equivalent neighbourhoods like those in LB-2 is not guaranteed.

Other problems than leader election

It is not difficult to see that the proof given in [Ly96] also works for networks solving other problems than leader election. The only step in the proof of the lowerbound that uses the fact that the network solves leader election is LB-3. It uses that in the round in which the leader is elected there is a state that is not corresponding via any σ to the state of any other process. In the leader election problem, this is the state of the process which is elected.

Other networks than bidirectional rings

The following section provides a definition of the notion "process isomorphism up-to PIds," such that the Key Lemma can be generalized to other networks than rings. The lower bound result needs not hold for other networks, as the existence of large order equivalent neighbourhoods like in those LB-2 is not guaranteed.

Uniqueness of PIds in start states

The fact that each start state contains a *single* PId which is required by axiom IP-3, is used only in the leader election problem, not in the Key Lemma. However, uniqueness of PIds in the start states should not be confused with uniqueness of the PIds in different processes.

6 Process isomorphism and the Key Lemma more generally

This section generalizes the Key Lemma to other networks than rings. Therefore, the notions of process isomorphism up-to PIds and of neighbourhood need to be given a meaning in an arbitrary network. Since different processes have different channels, we need to relate these when comparing two processes.
This is done via a so-called local automorphism, which bijectively maps the incoming and outgoing channels of one process to those of another. Then we introduce the notion of "process isomorphism up-to PIds via a local automorphism." We also compare the k-neighbourhoods of two processes using a local automorphism.
We claim that the Key Lemma also holds for isomorphic processes in an arbitrary network. The generalized Key Lemma does not require all processes to be isomorphic up-to PIds, but only the pairs that are related by the local automorphism.
We assume that $\mathcal{I}, \mathcal{J}$ are index sets and that X_k is some set, for all $k \in \mathcal{I} \cup \mathcal{J}$.

Definition 6.1 Let $G = \langle V, E \rangle$ be a graph and $i \in V$. A *local automorphism* of G in i is a bijection $\sigma : V \hookrightarrow\!\!\!\!\rightarrow V$ such that for all $k \in V$

1. $k \to i \iff \sigma(k) \to \sigma(i)$,

2. $i \to k \iff \sigma(i) \to \sigma(k)$.

So, a local automorphism in i maps incoming neighbours of i to incoming neighbours of $\sigma(i)$ and the same for outgoing neighbours. The set of local automorphisms of G in i is denoted by $\mathrm{Loc}(G, i)$. Notice that an automorphism of G is local automorphism of G in i, for every i.

Definition 6.2 Let $\mathcal{I}, \mathcal{J}$ be index sets. Let $\sigma : \mathcal{I} \hookrightarrow\!\!\!\!\rightarrow \mathcal{J}$ be a bijection. We define for all $x \in \prod_{j \in \mathcal{J}} X_j$

$$x\langle\sigma\rangle \stackrel{\mathrm{d}}{=} \big(x_{\sigma(i)}\big)_{i \in \mathcal{I}}.$$

so, $x\langle\sigma\rangle \in \prod_{i \in \mathcal{I}} X_i$ and $(x\langle\sigma\rangle)_i = x_{\sigma(i)}$. If $\tau \supseteq \sigma$, then we also use $x\langle\tau\rangle \stackrel{\mathrm{d}}{=} x\langle\sigma\rangle$. If $x\langle\sigma\rangle = y$, then we say that y is obtained from x by substitution of indices.

Example 6.3 Let R be a bidirectional ring of size n. Let $i, j \in V$. Take $\sigma : V \to V$, $\sigma(i+l) = j+l$ for all $0 \leq l \leq n$, counting modulo n. This is a local automorphism if R in i; it is even an automorphism of R. For $m, m' \in M$, we have $x = (j-1 : m, j+1 : m') \in M_0^{in(j)}$ and

$$\begin{aligned} x\langle\sigma\rangle &= \big(x_{\sigma(i)}\big)_{i \in \mathcal{I}} \\ &= \big(i-1 : x_{\sigma(i-1)}, i+1 : x_{\sigma(i+1)}\big) \\ &= (i-1 : x_{j-1}, i+1 : x_{j+1}) \\ &= (i-1 : m, i+1 : m'). \end{aligned}$$

Remark, however,

$$\begin{aligned} x\langle\sigma\rangle &= \big(x_{(i)}\big)_{i \in \mathcal{I}}\langle\sigma\rangle \\ &\neq (\sigma(j-1) : m, \sigma(j+1) : m'), \end{aligned}$$

for, in general, $\sigma(j-1)$ needs not to be defined.

6.1 Process isomorphism up-to PIds more generally

It is not difficult to formulate a more general notion of process isomorphism up-to PIds by combining the axioms of the notions of process isomorphism for rings from Section 3 and by relating the incoming and outgoing channels via a local automorphism.

Definition 6.4 Let $\langle G, P, M\rangle$ be a network system. Let $i, j \in V$ and let $\sigma \in \mathrm{Loc}(G, i)$ such that $\sigma(i) = j$. The processes P_i and P_j are *isomorphic up-to PIds via* σ , notation $P_i \approx_\sigma P_j$, if the following properties are satisfied:

IP-**1** $states_i = states_j$

IP-**2** $\#\, start_i = \#\, start_j = 1$

IP-**3** $|\mathsf{pidset}(s_i^0)| = \left|\mathsf{pidset}(s_j^0)\right| = 1$

IP-**4** $s_i^0[\mathsf{pid}(s_j^0)] = s_j^0$

GIP-**5** For all $s \in states_i$

$$msgs_i(s) = (msgs_j(s))\,\langle\sigma\rangle.$$

GIP-**6** For all $s \in states_i$ and all $\mu \in M_0^{in(j)}$

$$trans_i(s, \mu\langle\sigma\rangle) = trans_j(s, \mu)$$

Remark 6.5 1. Notice that $\approx_\sigma$ is not an equivalence relation. We have $P_i \approx_{id} P_i$, $P_i \approx_\sigma P_j \implies P_j \approx_{\sigma^{-1}} P_i$ and $P_i \approx_\sigma P_j \wedge P_j \approx_\tau P_k \implies P_i \approx_{\tau \circ \sigma} P_k$.

2. Compare the definition of $\approx_\sigma$ above, applied to a bidirectional ring network, and Definition 3.1, of $\approx_{\text{ring}}$, i.e. of "process isomorphism up-to PIds" of rings A correspondence between the relations $\approx_{\text{ring}}$ and $\approx_\sigma$ in a bidirectional ring network is given by

$$P_i \approx_{\text{ring}} P_j \iff \exists \sigma \in C_n[P_i \approx_\sigma P_j]$$

Here C_n is the cyclic group of order n, $C_n = \{\sigma_k \mid \sigma_k(i) = i + k \text{ modulo } n\}$. So, definition 3.1 uses very specific (local) isomorphisms to compare the processes, whereas the definition above allows any local automorphism.

6.2 The Key Lemma generalized

In order to generalize the Key Lemma, we redefine the notion of k-neighbourhood and generalize the notion of order equivalence to $\mathcal{I}$-tuples over U. Recall that $\mathrm{d}(i,l)$ is the distance between i and l, i.e. the length of a shortest path.

Definition 6.6 Let $\langle G, P, M\rangle$ be a network system. Let $i \in V$. The k-neighbourhood of a process P_i is defined by

$$\mathrm{N}_i(k) \stackrel{\mathrm{d}}{=} (u_l)_{l \in V, \mathrm{d}(i,l) \le k}.$$

Definition 6.7 Let $v, w \in U^{\mathcal{I}}$. We define

$$v \sim w \stackrel{\mathrm{d}}{=} \forall i \in \mathcal{I}[v_i \le v_j \iff w_i \le w_j].$$

Now order equivalence between neighbourhoods of different processes is expressed via a substitution of indices, e.g. $\mathrm{N}_i(k)\langle\sigma\rangle \sim \mathrm{N}_j(k)$.

Notation 6.8 If $N_i(k)\langle\sigma\rangle \sim N_j(k)$ then Lemma 4.4 implies that the function in $\{u_{i-k}, u_{i-k+1}, \ldots, u_{i+k}\} \to U$, mapping u_{i+l} to u_{j+l}, for every l, is well-defined and strictly monotone. This function is denoted by $v_{ik\sigma}$.

The proposition below is an analogue of proposition 5.3.

Proposition 6.9 *For all $i, k \in \mathbb{N}$ and $\sigma \in \mathrm{Loc}(G, i)$ we have*

1. $v_{j,k-1,\sigma} \subseteq v_{ik\sigma}$, *for all $j \in in(i) \cup \{i\}$,*
2. $\mathrm{N}_i(k)\langle\sigma\rangle \sim \mathrm{N}_{\sigma(i)}(k) \implies \forall l \in in(i) \cup \{i\}[\mathrm{N}_l(k-1)\langle\sigma\rangle \sim \mathrm{N}_{\sigma(l)}(k-1)]$.

Now we can generalize the Key Lemma. Suppose we have two processes P_i, P_j which are isomorphic up-to PIds via σ and whose k-neighbourhoods are order equivalent via σ also. Then the processes remain in corresponding states via $v_{ik\sigma}$, during the first k rounds of the execution. Notice that, contrary to the Key Lemma for rings, we do not require all processes to be isomorphic, but only the pairs that are related by σ, i.e. only P_l and $P_{\sigma(l)}$, for all l within distance k from i.

Lemma 6.10 (Key Lemma generalized) *Let $\langle G, P, M\rangle$ be a network system, consisting of comparison based processes. Let $i \in V, k \in \mathbb{N}$ and $\sigma \in \mathrm{Loc}(G, i)$. Write $\sigma(i) = j$. If*

$$\forall l \in V, d(i,l) \leq k[P_l \approx_\sigma P_{\sigma(l)} \wedge \mathrm{N}_i(k)\langle\sigma\rangle \sim \mathrm{N}_j(k)]$$

then for all r

$$\mathsf{actives}(r) \leq k \implies \dot{\alpha}_{r\cdot i} \xrightarrow{v_{ik\sigma}} \dot{\alpha}_{r\cdot j}.$$

PROOF: Similar to the proof of lemma 5.5. □ □

Example 6.11 The Key Lemma concludes that the processes 3 and 4 in the execution of the LCR agorithm of Example 1.9 are in corresponding states during the *first round* of the execution, the Generalized Key Lemma that these are so during the *first two rounds* of the execution.

	round 0	round 1	round 1	round 2	round 2
proc	states	msgs	state	msgs	states
P_1	(4,null,init)	4	(4,null,unknown)	null	(4,null,unknown)
P_2	(3,null,init)	3	(3,4,unknown)	4	(3,null,unknown)
P_3	(2,null,init)	2	(2,3,unknown)	3	(2,4,unknown)
P_4	(1,null,init)	1	(1,2,unknown)	2	(1,3,unknown)

7 Further research

It would be interesting to investigate whether or not comparison based processes can be verified automatically. We conjecture that this is indeed possible. We suggest an adaption of a result by [PJ96], stating that bisimulation equivalence is decidable for programs that can only read, write and store their data, i.e. programs that do not depend on the actual data values. The idea behind this suggestion is that given a fixed assignment of PIds to all the processes, the network can be considered as only moving around data. As a comparison based process behaves similarly on order equivalent input, there is only a finite number of really different assignments. If n is the number of processes, we have n^n, assignments leading to different behaviour which we need to check.

As a second topic for further research, it would be more elegant to formulate the notion of "process isomorphism up-to PIds" more generally using bisimulation equivalence. Our formalization requires the state spaces of all processes to be equal. It would be more natural to allow each process to have its own state space, expressing equivalence of behaviour by means of bisimulation.

Moreover, it would be elegant to have an absolute notion of isomorphic processes, independent of an local automorphism, like we have for ring networks. We could try to find an analogon of C_n as used in Remark 6.5, using a specific class of (local) isomorphisms to compare processes. It is not immediately clear which class to take.
Another direction for further investigation is to adapt the lower bound result on the leader election problem to other networks than rings. The proof of the lower bound given by [Ly96] does not apply to other networks than rings because the existence of as many and as large order equivalent neighbourhoods as those in LB-2 is guaranteed in rings only.
In fact, we conjecture that the lower bound result $\Omega(n \log n)$ does not hold for arbitrary networks. networks. It is likely that the higher the connectivity of the network, i.e. the number of channels, the less messages are needed to elect a leader. It would be an interesting combinatorial problem to find large order equivalent neighbourhoods in other networks than rings, in order to prove an adapted lower bound.

A Notational conventions

Sets

$\mathcal{P}(A)$	powerset of A
$A \dot{\cup} B$	the disjoint union of A and B
	$\{(a,0) \mid a \in A\} \cup \{(b,1) \mid b \in B\}$
$\# A$	the number of elements in A

Functions

$A \to B$	the set of (total) functions from A to B
$A \hookrightarrow B$	the set of injective functions from A to B
$A \twoheadrightarrow B$	the set of surjective functions from A to B
$A \hookrightarrow\!\!\!\!\!\rightarrow B$	the set of bijective functions from A to B
$A \rightharpoonup B$	the set of partial functions from A to B
$\mathrm{Dom}(f)$	the domain of f
$\mathrm{Ran}(f)$	the range of f
$f \subseteq g$	$\mathrm{Dom}(f) \subseteq \mathrm{Dom}(g)$ & $\forall x \in \mathrm{Dom}(f)[f(x) = g(x)]$
$f(x)\!\downarrow$	$x \in \mathrm{Dom}(f)$
$f(x) = f(y)$	$x, y \in \mathrm{Dom}(f)$ and they are equal as elements in $\mathrm{Ran}(f)$

Sequences

A^*	set of finite sequences over A
ε	empty sequence
$\lvert x \rvert$	length of the sequence x
x_i	$1 \leq i \leq \lvert x \rvert$, the i^{th} element of x
$\star$	concatenation of sequences
set	$\mathsf{set} : U^* \to \mathcal{P}(U), \mathsf{set}(x_1 \cdots x_k) = \{x_1, \ldots x_k\}$

Dependent products

$\mathcal{I}$-tuples are tuples over another index set $\mathcal{I}$ than $\{0, 1, \ldots k\}$ or $\mathbb{N}$.

$\prod_{i \in \mathcal{I}} A_i$	the set of over tuples an index set $\mathcal{I}$, such that the element at place i is an element of A_i
$A^{\mathcal{I}}$	$\prod_{i \in \mathcal{I}} A$
$(x_i)_{i \in \mathcal{I}}$	$\mathcal{I}$-tuple consisting of elements x_i at index i
$(i_1 : x_1, i_2 : x_2, i_3 : x_3, \ldots)$	$\mathcal{I}$-tuples consisting of elements x_{i_j} at index i_j, $\mathcal{I} = \{i_1, i_2, i_3, \ldots\}$
$x_i, x.i$	$i \in \mathcal{I}$, the i^{th} of x

Typically, we use the dependent product as a "common super type" of the sets S, $S \times M_0^{in(i)}$ and M_0^{out}. Note that $S \times M_0^{in(i)} \simeq \prod_{i \in \{1\} \cup in(i)} X_i$, where $X_{(1,0)} = S$, $X_{(i,1)} = M^0, i \in in(i)$. We mix the notations of the product and the dependent product. Some examples: if $out(i) = \{3, 5\}$, then $(3 : m, 5 : m') \in M_0^{out(i)}$. Furthermore, if $in(i) = \{i - 1, i + 1\}$ and $status \in S$, then $\langle status, (i - 1 : m, i + 1 : m') \rangle \in S \times M_0^{in(i)}$. If $m_{i-1}, m_{i+1} \in M_0$ then $(m_i)_{i \in in(i)} \in M_0^{in(i)}$.

Glossary of symbols

G	graph
R	uni- or bidirectional ring
V	set of vertices
E	set of edges
$\mathcal{I}$	index set
i, j, l	elements of V, $\mathcal{I}$ or $\mathbb{N}$
P	(sequence of) processes
S	set of states
s	state
s_0^i	start state of process i
M	set of messages
null	absence of a message
M_0	$M \cup \{\mathsf{null}\}$
m	message
x, y	indexed sequences (of messages and states)
α	execution
n, k	natural numbers
U	set of PIds
u	PId
u_i	the PId of process i
v, w	sequence of PIds

References

[CR79] E. Chang, R. Roberts *An improved algorithm for finding extrema in circular configurations of processes.* In: Communications of the ACM, Vol. 22-5, pp. 144 – 156, May 1979.

[CFJ93] E.M. Clarke, T. Filkorn, S.Jha *Exploiting symmetry in temporal logic model checking.* In: Proceedings of the international conference on computer-aided verification, Lecture notes in computer science, vol. 697, pp. 451 – 462, 1993. Springer-Verlag, Berlin.

[EA93] E. A. Emerson, A.P. Sistla *Symmetry and model checking.* In: Proceedings International conference on computer-aided verification, Lecture notes in computer science, vol. 697, 463 – 477, 1993. Springer-Verlag, Berlin.

[ES95] E.A. Emerson, A.P. Sistla *Utilizing symmetry when model checking under fairness assumptions.* In: Proceedings International conference on computer-aided verification, Lecture notes in computer science, vol. 939, 309 – 318, 1995. Springer-Verlag, Berlin.

[FL87] G.N.Frederickson, N.A. Lynch *Electing a leader in a synchronous ring.* In: Journal of the ACM, Vol. 34-I, pp. 98 – 115, 1987.

[PJ96] B.Johnson, J. Parrow *Deciding Bisimulation equivalences for a class of non-finite-state programs.* In: Information and Computation, 107-II:pp. 272 – 302, 1993.

[La77] G. Le Lann *Distributed algorithms – towards a formal approach.* In: B. Gilchrist, editor, Information Proceeding 77, Toronto, August 1977, vol. 7, Proceedings of IFIP Congress, pp 155 – 160, North Holland, Amsterdam, 1977

[Ly96] N.A. Lynch *Distributed algorithms,* pp. 17 – 49. Morgan Kaufmann Publishers, Inc., 1996.