Introduction to Distributed Algorithms: Solutions and Suggestions

Gerard Tel
Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
email: `g.tel@.uu.nl`

May 2002/January 2015

This booklet contains (partial) solutions to most of the exercises in the book *Introduction to Distributed Algorithms* [Tel00]. Needless to say, the book as well as this document contain numerous errors; please find them and mail them to me! I have not included answers or hints for the following exercises and projects: 2.3, 2.9, 2.10, 3.2, 3.7, 6.16, 10.2, 12.8, 12.10, 12.11, 14.6, 17.5, 17.9, 17.11, B.5.

Gerard Tel.

Contents

Chapter 2: The Model

Exercise 2.1. Let $\mathcal{M}_1$ be the set of messages that can be passed asynchronously, $\mathcal{M}_2$ the set of messages that can be passed synchronously, and $\mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2$. A process is defined as in Definition 2.4. A distributed system with hybrid message passing has four types of transitions: internal, send and receive of asynchronous messages, and communication of synchronous messages. Such a system is a transition system $S = (\mathcal{C}, \to, \mathcal{I})$ where $\mathcal{C}$ and $\mathcal{I}$ are as in Definition 2.6 and $\to = (\cup_{p\in\mathbb{P}} \to_p) \cup (\cup_{p,q\in\mathbb{P}} \to_{pq})$, where $\to_p$ are the transitions corresponding to the state changes of process p and $\to_{pq}$ are the transitions corresponding with a communication from p to q. So, $\to_{p_i}$ is the set of pairs

$$(c_{p_1}, \dots, c_{p_i}, \dots, c_{p_N}, M_1),\ (c_{p_1}, \dots, c'_{p_i}, \dots, c_{p_N}, M_2)$$

for which one of the following three conditions holds.

- $(c_{p_i}, c'_{p_i}) \in \vdash^{\mathrm{i}}_{p_i}$ and $M_1 = M_2$;
- for some $m \in \mathcal{M}_1$, $(c_{p_i}, m, c'_{p_i}) \in \vdash^{\mathrm{s}}_{p_i}$ and $M_2 = M_1 \cup \{m\}$;
- for some $m \in \mathcal{M}_1$, $(c_{p_i}, m, c'_{p_i}) \in \vdash^{\mathrm{r}}_{p_i}$ and $M_1 = M_2 \cup \{m\}$.

Similarly, $\to_{p_i p_j}$ is the set of pairs

$$(\dots, c_{p_i}, \dots, c_{p_j}, \dots),\ (\dots, c'_{p_i}, \dots, c'_{p_j}, \dots)$$

for which there is a message $m \in \mathcal{M}_2$ such that

$$(c_{p_i}, m, c'_{p_i}) \in \vdash^{\mathrm{s}}_{p_i} \quad \text{and} \quad (c_{p_j}, m, c'_{p_j}) \in \vdash^{\mathrm{r}}_{p_j} .$$

Exercise 2.2. Consider the system $S = (\{\gamma, \delta\}, \{(\gamma \to \delta)\}, \varnothing)$, and define P to be true in γ but false in δ. As there are no initial configurations, this system has no computations, hence trivially P is always true in S. On the other hand, P is falsified in the only transition of the system, so P is not invariant.

If an empty set of initial configurations is not allowed, the three state system $S' = (\{\beta, \gamma, \delta\}, \{(\gamma \to \delta)\}, \{\beta\}$ is the minimal solution.

The difference between invariant and always true is caused by the existence of transitions that falsify a predicate, but are applicable in an unreachable configuration, so that they do not "harm" any computation of the system.

Exercise 2.4. (1) If each pair (γ, δ) in $\to_1$ satisfies $P(\gamma) \Rightarrow P(\delta)$ and each such pair in $\to_2$ does so, then each pair in the union satisfies this relation.
(2) Similarly, if $(Q(\gamma) \wedge P(\gamma)) \Rightarrow (Q(\delta) \Rightarrow P(\delta))$ holds for each pair (γ, δ) in $\to_1$ and in $\to_2$, then it holds for each pair in the union.
(3) In the example, P is an invariant of S_1, and always true in S_2, but is falsified by a transition of S_2 which is unreachable in S_2 but reachable in S_1. This occurs when $S_1 = (\{\gamma, \delta, \epsilon\}, \{(\gamma \to \delta)\}, \{\gamma\})$ and $S_2 = (\{\gamma, \delta, \epsilon\}, \{(\delta \to \epsilon)\}, \{\gamma\})$ with the assertion P, which is false in ϵ.

Exercise 2.5. In $(\mathbb{N}^2, \leq_l)$, $(1, k) <_l (2, 0)$ for each k; hence there are infinitely many elements smaller than $(2, 0)$.
It is shown by induction on n that $(\mathbb{N}^n, \leq_l)$ is well-founded.

$n = 1$: $\mathbb{N}^1$ is identical to $\mathbb{N}$, which is well-founded.

$n + 1$: Assume $(\mathbb{N}^n, \leq_l)$ is well-founded and w is an ascending sequence in the partial order $(\mathbb{N}^{(n+1)}, \leq_l)$. Partition the sequence in subsequences, i.e., write $w = w_1.w_2\ldots$, where each w_i is a maximal subsequence of elements of w that have the same first component. The number of these subsequences is finite; because w is ascending, the first component of elements in w_{i+1} is smaller than the first component of elements in w_i. Each w_i is of finite length; because w is ascending and consecutive elements of w_i have the same first component, the tail of each element (which is in $\mathbb{N}^n$) is smaller than the tail of its predecessor in w_i. The well–foundedness of $\mathbb{N}^n$ implies that w_i is finite. As w is the concatenation of a finite number of finite sequences, w is finite.

Exercise 2.6. The event labels:

	a	b	c	d	e	f	g	h	i	j	k	l
Θ_L	1	2	5	3	4	2	3	4	8	5	6	7
Θ_v	$\left\vert\begin{array}{c}1\\0\\0\\0\end{array}\right\vert$	$\left\vert\begin{array}{c}2\\0\\0\\0\end{array}\right\vert$	$\left\vert\begin{array}{c}3\\2\\0\\0\end{array}\right\vert$	$\left\vert\begin{array}{c}2\\1\\0\\0\end{array}\right\vert$	$\left\vert\begin{array}{c}2\\2\\0\\0\end{array}\right\vert$	$\left\vert\begin{array}{c}1\\0\\1\\0\end{array}\right\vert$	$\left\vert\begin{array}{c}1\\0\\2\\0\end{array}\right\vert$	$\left\vert\begin{array}{c}1\\0\\3\\0\end{array}\right\vert$	$\left\vert\begin{array}{c}1\\0\\4\\3\end{array}\right\vert$	$\left\vert\begin{array}{c}1\\0\\3\\1\end{array}\right\vert$	$\left\vert\begin{array}{c}1\\0\\3\\2\end{array}\right\vert$	$\left\vert\begin{array}{c}1\\0\\3\\3\end{array}\right\vert$

Events d and g are concurrent; while $\Theta_L(d) = \Theta_L(g) = 3$, the vector time stamps $\Theta_v(d)$ and $\Theta_v(g)$, $\left|\begin{array}{c}2\\1\\0\\0\end{array}\right|$ and $\left|\begin{array}{c}1\\0\\2\\0\end{array}\right|$, are incomparable.

Exercise 2.7. The algorithm (see also [Mat89]) resembles Algorithm 2.3, but maintains a *vector* $\theta_p[1..N]$. An *internal* event in p increments p's own component of the vector, i.e., preceding an internal event, p executes

$$\theta_p[p] := \theta_p[p] + 1.$$

The same assignment precedes a *send* event, and as in Algorithm 2.3 the time stamp is included in the message.

Finally consider a *receive* event r following an event e in the same process, and with corresponding send event s. The history of r consists of the combined histories of e and s, and the event r itself; hence the clock is updated by

```
**forall** $q$ **do** $\theta_p[q] := \max(\theta_p[q], \theta[q])$ ;
$\theta_p[p] := \theta_p[p] + 1$
```

Exercise 2.8. For finite executions E and F this is possible, but for infinite executions there is the problem that no finite number of applications of Theorem 2.19 suffices. We can then obtain each finite prefix of F by applying 2.19, but as we don't have "convergence results", this would not prove Theorem 2.21.

Chapter 3: Communication Protocols

Exercise 3.1. Assume a configuration where packets s_p and $s_p + 1$ are sendable by q, but q starts to send packet $s_p + 1$ infinitely often (and the packet is received infinitely often). This violates **F1** because packet s_p remains sendable forever, but **F2** is satisfied, and no progress is made.

Exercise 3.3. (1) The steps in the execution are as follows:

1. The sender opens and sends $\langle\, \mathbf{data}, \mathit{true}, 0, x \,\rangle$.
2. The receiver receives the packet, opens, and delivers.
3. The receiver sends $\langle\, \mathbf{ack}, 1 \,\rangle$, but this message is lost.
4. Sender and receiver repeat their transmissions, but all messages are lost.
5. The receiver times out and closes.
6. The sender times out and reports data item 0 as lost.

(2) No, this is not possible! Assume a protocol in which response is guaranteed withing ρ time, and assume q will deliver the data upon receipt of a message M from p. Consider two executions: E_1 in which M is lost and no messages are received by p for the next ρ time units; E_2 in which q receives M and no messages are received by p for the next ρ time units. Process q delivers m in E_2 but not in E_1. To p, E_1 and E_2 are identical, so p undertakes the same action in the two executions. Reporting is inappropriate in E_2, not doing so is inappropriate in E_1; consequently, no protocol reports if and only if the message is lost.

Exercise 3.4. In the example, the sender reuses a sequence number (0) in a new connection, while the receiver regards it as a duplicate and does not deliver, but sends an ack.

1. The sender opens and sends $\langle\, \mathbf{data}, \mathit{true}, 0, x \,\rangle$.
2. The receiver receives the packet, opens, and delivers.
3. The receiver sends $\langle\, \mathbf{ack}, 1 \,\rangle$.
4. The sender receives the acknowledgement.
5. The sender times out and closes.
6. The sender opens and sends $\langle\, \mathbf{data}, \mathit{true}, 0, y \,\rangle$.
7. The receiver (still open!) receives the packet, observes $i = \mathit{Exp} - 1$
   and does nothing.
8. The receiver sends $\langle\, \mathbf{ack}, 1 \,\rangle$.
9. The sender receives the acknowledgement.
10. The sender times out and closes.
11. The receiver finally times out and closes.

Observe that, with correct timers, this scenario does not take place because the receiver always times out before the sender does so.

Exercise 3.5. Let the receiver time out and the sender send a packet with the start-of-sequence bit *true*:

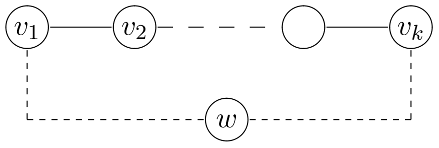
1. The sender opens and sends $\langle\, \mathbf{data}, \mathit{true}, 0, x \,\rangle$.
2. The receiver receives the packet, opens, and delivers.
3. The receiver sends $\langle\, \mathbf{ack}, 1 \,\rangle$.
4. The sender receives the acknowledgement (now $\mathit{Low} = 1$).
5. The receiver times out and closes.
6. The sender accepts new data and sends $\langle\, \mathbf{data}, \mathit{true}, 1, y \,\rangle$.
7. The receiver receives the packet, opens, and delivers.
8. The receiver sends $\langle\, \mathbf{ack}, 2 \,\rangle$.
9. The sender receives the acknowledgement (now $\mathit{Low} = 2$).
10. The receiver times out and closes.
11. The sender times out and closes.

Exercise 3.6. The action models the assumption of an upper limit on message delay, without concern of how this bound is implemented; thus, the remaining packet lifetime is an idealized "mathematical" timer rather than a really existing clock device.
Of course the situation can be considered where the network implements the bound using physical representations of a remaining packet lifetime for each process. A drift in these clocks result in a changed actual bound on the delay: if the timers are supposed to implement a maximum of μ but suffer ϵ-bounded drift, the actual maximum packet lifetime is $(1 + \epsilon) \cdot \mu$.

Exercise 3.8. The invariants of the protocol are insensitive to the threshold in action $\mathbf{E}_p$, as can be seen from the proofs of Lemmas 3.10, 3.11, and 3.13; therefore the protocol remains correct.
An advantage of the changed protocol is the shorter response time for the sender. A disadvantage is that more items may be inappropriately reported as lost.

Chapter 4: Routing Algorithms

Exercise 4.1. Consider a network that consists of a path of nodes v_1 through v_k, and a node w connected to both v_1 and v_k, but its links go up and down repeatedly.

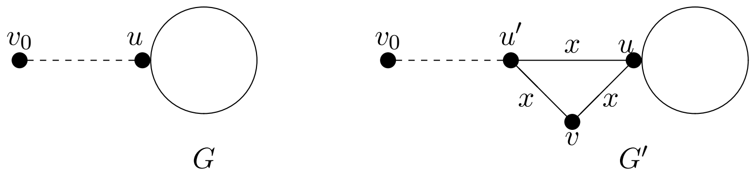


Consider a packet m with destination w, and assume the link v_1w is down; the packet must move in the direction of v_k, i.e., to the right. When the packet arrives in v_k, the link to w goes down but v_1w comes up; after adaptation of the routing tables the packet will move to v_1, i.e., to the left. When m arrives there, the link to w fails, but v_kw is restored again, and we are back in the situation we started with.

In this example the network is always connected, yet m never reaches its destination even if the routing algorithm is "perfect" and instantaneously adapts tables to the current topology.

Exercise 4.2. Yes, such a modification is possible; a message containing D_w is relayed to those neighbors from which a $\langle\, \mathbf{ys}, w\,\rangle$ message is received. Unfortunately, neighboring processes may now be several rounds apart in the execution of the algorithm, i.e., a process may receive this message while already processing several pivots further. This implies that the D_w table must be stored also during later rounds, in order to be able to reply to $\langle\, \mathbf{ys}, w\,\rangle$ messages appropriately. This causes the space complexity of the algorithm to become quadratic in N. The message complexity is reduced to quadratic, but the bit complexity remains the same because we unfortunately save only on small messages.

Exercise 4.3. To construct the example, let G be a part of the network, not containing v_0, connected to v_0 only through node u in G, and with the property that if $D_u[v_0]$ improves by x while no mydist-messages are in transit in G, then K messages may be exchanged in G:

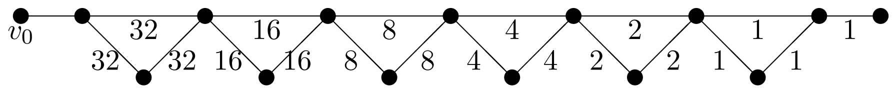


Construct G' by adding nodes u' and v, and three edges uu', $u'v$, and uv, each of weight x, and G' is connected to v_0 only via u'. When no messages are in transit in G', $D_u[v_0] = D_{u'}[v_0]+x$, and assume in this situation $D_{u'}[v_0]$ improves by $2x$ (due to the receipt of a mydist-message). Consider the following scenario, in which u is informed about the improvement in two steps by passing information via v.

1. u' sends an improved mydist-message to v and u.
2. v receives the message, improves its estimate, and sends an improved mydist-message to u.
3. u receives the message from v and improves its estimate by x, which causes K messages to be exchanged in G.

4. u receives the mydist-message from u' and improves its estimate again by x, causing another K messages to be exchanged in G.

It is seen that when $D_{u'}[v_0]$ improves by $2x$, more than $2K$ messages may be exchanged in G'. By iterating the construction the number of nodes and edges grows linearly, while the number of messages grows exponentially; the resulting graph is as follows:



A similar example can be given if the message delays are guaranteed to lie within a very small range. The assumption of the Shortest Path measure is, however, necessary; in the Minimum Hop measure the complexity of the algorithm is bounded by $O(N \cdot E)$ messages.

Exercise 4.4. The following table gives the values of $D_u[v]$ and, between parenthesis, the value or possible values of $Nb_u[v]$; *Recompute* is non-deterministic w.r.t. the selection of a preferred neighbor if the minimal estimate occurs multiply. For each neighbor w of u, $Ndis_u[v,\, w]$ equals $D_w[v]$.

	u					
v	A	B	C	D	E	F
A	0 (*loc*)	1 (A)	4 (F)	1 (A)	2 (B/D)	3 (E)
B	1 (B)	0 (*loc*)	3 (F)	2 (A/E)	1 (B)	2 (E)
C	4 (B/D)	3 (E)	0 (*loc*)	3 (E)	2 (F)	1 (C)
D	1 (D)	2 (A/E)	3 (F)	0 (*loc*)	1 (D)	2 (E)
E	2 (B/D)	1 (E)	2 (F)	1 (E)	0 (*loc*)	1 (E)
F	3 (B/D)	2 (E)	1 (F)	2 (E)	1 (F)	0 (*loc*)

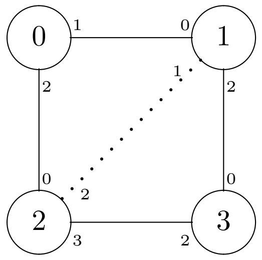
Following Algorithm 4.9, node F sends upon receiving the $\langle\, \textbf{repair}, A\,\rangle$ message all entries of its distance table, i.e., messages: $\langle\, \textbf{mydist}, A, 3\,\rangle$, $\langle\, \textbf{mydist}, B, 2\,\rangle$, $\langle\, \textbf{mydist}, C, 1\,\rangle$, $\langle\, \textbf{mydist}, D, 2\,\rangle$, $\langle\, \textbf{mydist}, E, 1\,\rangle$, and $\langle\, \textbf{mydist}, F, 0\,\rangle$.

Upon receipt of each of these six messages, *Recompute* is executed in A and leads to an improvement for destinations F and C, after which A sends messages $\langle\, \textbf{mydist}, F, 1\,\rangle$ and $\langle\, \textbf{mydist}, C, 2\,\rangle$.

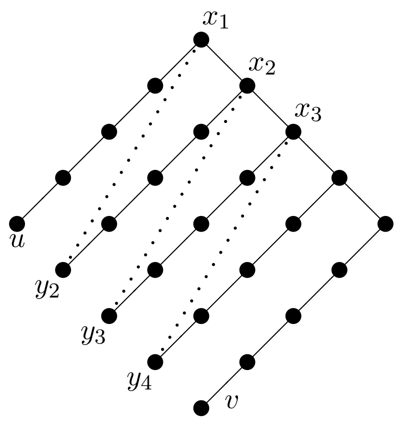
Exercise 4.5. Let G be a ring of N processes, where the cost of each edge in clockwise direction is 1 and in anticlockwise direction the cost is k; then D_G is approximately $\frac{k}{k+1}N$. A spanning tree of G is obtained by removing a single edge, and the distance between the two separated nodes is $N-1$ in one direction but $k \cdot (N-1)$ in the other direction; this is about $k+1$ times D_G.

Exercise 4.6. Let the label of link uw (in node u) be α_{uw}; if $\alpha_{uw} \neq l_u$, the link is used when a message with address α_{uw} is generated in u. If $\alpha_{uw} = l_u$ but the label $\alpha_{uw}+1$ does not occur in u, the link is used when a message with address $\alpha_{uw}+1$ is generated in u. But if $\alpha_{uw} = l_u$ and the label $\alpha_{uw}+1$ also occurs in u, the link is never used.

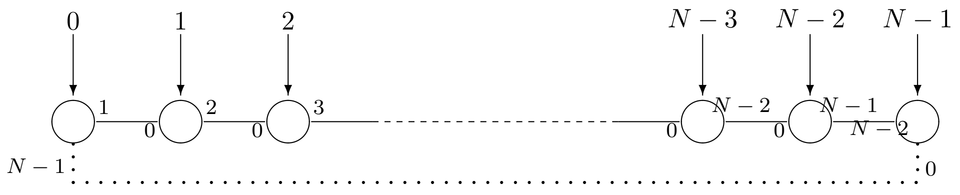
This picture shows an example of an ILS where this occurs for the edge 12 on both sides; hence the edge 12 is not used for traffic in neither direction. The scheme is valid.
A scheme not using edge uw is not optimal because traffic between u and w is sent over two or more hops while $d(u, w) = 1$.



Exercise 4.7. Exploit Figure 4.16 to design a detour running through all nodes of the network. In this network, such a detour is made by messages from u to v. The nodes marked x_i send the message down the tree via the dotted frond edges because y_{i+1} has a node label between x_{i+1} and v (both edges are labeled with the node label of the adjacent node).



Exercise 4.8. A DFS tree in a ring has depth $N-1$; the ILS is as indicated here, and a message from 0 to $N-2$ travels via $N-2$ hops (as does a message in the opposite direction).



Exercise 4.9. (1) Besides minimality, the only requirement on T is that it contains all c_i; consequently, every leaf of T is one of the c_i (otherwise, a leaf could be removed, contradicting minimality).
(2) Because a tree has $N-1$ edges, the node degrees sum up to $2(N-1)$, and leaves have degree 1. With m leaves, there are $N-m$ nodes with degree at least 2, and these degrees sum up to $2N-m-2$; so the number of nodes with degree larger than 2 is at most $2N-m-2-2(N-m) = m-2$.

Chapter 5: Deadlock–free Packet Switching

Exercise 5.1. Consider the scenario in which each process generates a packet (for another node); this generation in empty nodes is allowed by the controller by definition. Now all buffers are occupied but no packet is at its destination, hence the configuration is deadlocked.

Dropping the requirement that every node should be able to send packets to another node makes a solution feasible. If all packets ever sent have the same destination v_0 (so v_0 cannot send to another node), a single buffer per node, arranged as in the **dest** controller (Def. 5.8), suffices.

Exercise 5.2. Consider the scenario in which u_1, u_2, w_1, and w_2 concurrently generate a packet for v; this is allowed when the buffers for destination v are empty and results in deadlock.

Exercise 5.3. The buffers of node u are $b_u[0] \dots b_u[k]$, like for the hops-so-far scheme, but here an edge $b_u[i]b_w[j]$ exists in the buffer graph if $uw \in E$ and $i - 1 = j$. A packet p for v, generated in u, is placed in buffer $fb(p) = b_u[k]$, where k is the length of the used path from u to v. If packet p in buffer $b_u[i]$ must be forwarded to node w, it must be placed in buffer $nb(p, b_u[i]) = b_w[i-1]$ because from node w the packet has only $i - 1$ hops to go to v.

In this buffer graph, the class of buffer $b_u[i]$ is $k - i$. It will usually not be necessary to include the hop counter. To initialize it properly, the source of a packet with destination v must know the hop distance to v, knowledge which will usually be taken from routing tables at the source. The intermediate nodes also find the hop distance to v in their routing tables, so it is not necessary to include this information in the packet.

Exercise 5.4. Let $P = \langle u_0, u_1, \dots, u_l \rangle$ be the path for packet p and consider the sequence of buffers obtained by $b_0 = fb(p)$ and $b_{j+1} = nb(p, b_j)$. We must show that this sequence defines a path in the buffer graph, i.e., after reaching the highest buffer index, B, no further increment of the index is required.

This is not hard to show but some subtlety is involved because the embedding of P in the cover may skip some of the orientations (i.e., P_i is empty for some i), while the buffer index grows at most one in every hop. We first make explicit to what level of the cover each edge in P belongs: choose $1 = c_0 \leq c_1 \leq c_2 \dots \leq c_l \leq B$ such that $u_{j-1}u_j \in \vec{E_{c_j}}$. Finally, let i_j be the buffer index of b_j, i.e., $b_j = b_{u_j}[i_j]$; we show by induction that $i_j \leq c_j$.

Case $j = 0$**:** Because $fb(p) = b_{u_0}[1]$, $i_0 = 1$, and by definition $c_0 = 1$, hence $i_0 \leq c_0$.

Case $j + 1$**:** The induction hypothesis states $i_j \leq c_j$. By the definition of nb, $i_{j+1} \leq i_j + 1$ so if $i_j < c_j$ then $i_{j+1} \leq c_j \leq c_{j+1}$ and we are done. If $u_ju_{j+1} \in \vec{E_{i_j}}$ then $i_{j+1} = i_j$ and we are also done because $i_j \leq c_j \leq c_{j+1}$.

The remaining case is where $i_j = c_j$ and $u_ju_{j+1} \notin \vec{E_{i_j}}$; this implies that $i_{j+1} = i_j + 1 = c_j + 1$. However, $c_{j+1} > c_j$ can be concluded in this case: because $u_ju_{j+1} \notin \vec{E_{c_j}}$, we have $c_{j+1} \neq c_j$, hence $c_j \leq c_{j+1}$ implies $c_{j+1} > c_j$.

Thus, the buffer index in node u_j is *at most* the level of edge $u_{j-1}u_j$ in the embedding of P in the cover, hence remains bounded by B.

Project 5.5. We consider the Hypercube with the standard node labeling as in Definition B.12, i.e., the name of a node is a string of bits. The distance from u to v is the number of positions i with $u_i \neq v_i$ and an optimal path is obtained by reversing the differing bits in any order.

Now adopt the following routing algorithm. In the first phase, reverse the bits that are 1 in u and 0 in v (in any order), then reverse the bits that are 0 in u and 1 in v. Thus, the number of 1-bits decreases in the first phase and increases in the second phase. All obtained paths are covered by the acyclic orientation cover HC_0, HC_1, where in HC_0 all edges are directed towards the node with the smaller number of 1's and in HC_1 the other way. This shows that deadlock free packet switching is possible with only two buffers per node.

Shortest paths that are all emulated in the cover described here cannot be encoded in an interval labeling scheme. In fact, Flammini demonstrated that the minimal number of intervals needed globally to represent shortest paths covered by HC_0, HC_1 is lower bounded by $\Theta(N\sqrt{N})$, which means that at least $\sqrt{N}/\log N$ intervals per link (average) are needed.

On the other hand, the cover described above is not the only cover of size two that emulates a shortest path between every two nodes. Whether shortest paths emulated by another cover can be encoded in an interval labeling scheme is unknown.

Exercise 5.6. We start to prove the correctness of **BC**. First, if u is empty, $f_u = B$ hence $B > k$ implies $k - f_u < 0$, so the acceptance of every packet is allowed.

To show deadlock-freedom of **BC**, assume γ is a deadlocked configuration and obtain δ as in the proof of Theorem 5.17. Let p be a packet in δ that has traveled a maximal number of hops, i.e., the value of t_p is the highest of all packets in δ, and let p reside in u. Node u is not p's destination, so let w be the node to which p must be forwarded; as this is move is not allowed there is a packet in w (an empty node accepts every packet). Choose q as the most recently accepted one of the packets in w and let f'_w be the number of free buffers in w just before acceptance of q; we find $f'_w \leq f_w + 1$. The acceptance of q implies $t_q > k - f'_w$ but the refusal of p implies $t_p + 1 \leq k - f_w$. Consequently,

$$t_q > k - f'_w \geq k - (f_w + 1) \geq t_p,$$

contradicting the choice of p.

As expected, the correctness of **BS** is a little bit more intricate. Again, an empty node accepts every packet because $\sum_{t=0}^{j} i_t - B + k < 0$; this shows that generation in (and forwarding to) empty nodes is allowed and we use it in the proof below.

Again, assume a deadlocked configuration γ, and obtain δ as usual; choose p a packet in γ with maximal value of t_p, let u be the node where p resides and w be the node to which p must be forwarded. Choose q the packet in w that maximizes t_q; this choice implies $i_t = 0$ for $t > t_q$. Let r be the most recently accepted packet in w, and $\vec{i}$ and $\vec{i'}$ the state vector of w in δ, and just before accepting r, respectively.
(1) The acceptance of r implies

$$\forall j,\ t_r \leq j \leq k\ :\ j > \sum_{t=0}^{j} i'_t - B + k.$$

(2) The choice of r implies that $i_t \leq i'_t$ for all $t \neq t_s$ and $i_{t_r} \leq i'_{t_r} + 1$, so we find

$$\forall j,\ t_s \leq j \leq k\ :\ j > \sum_{t=0}^{j} i_t - B + k - 1.$$

(3) In particular, with $j = t_q$:

$$t_q > \sum_{t=0}^{t_q} i_t - B + k - 1.$$

(4) Now use that p is *not* accepted by w, i.e.,

$$\exists j_0,\ t_p + 1 \leq j_0 \leq k\ :\ j_0 \leq \sum_{t=0}^{j_0} i_t - B + k.$$

(5) This gives the inequality

$$\begin{array}{rcll} t_p & < & t_p + 1 & \\ & \leq & j_0 & \text{as chosen in (4)} \\ & \leq & \sum_{t=0}^{j_0} i_t - B + k & \text{from (4)} \\ & \leq & \sum_{t=0}^{k} i_t - B + k & \text{because } i_t \geq 0 \\ & = & \sum_{t=0}^{t_q} i_t - B + k & \text{because } i_t = 0 \text{ for } t > t_q \\ & \leq & t_q, & \text{see (3)} \end{array}$$

which contradicts the choice of p.

Exercise 5.7. Assume the acceptance of packet p by node u is allowed by **BC**; that is, $t_p > k - f_u$. As the path length is bounded by k, $s_p \leq k - t_p$, i.e., $s_p < k - (k - f_u) = f_u$; consequently the move is allowed by **FC**.

Chapter 6: Wave and Traversal Algorithms

Exercise 6.1. The crux of this exercise lies in the causal dependency between receive and send in systems with synchronous message passing; a dependency that is enforced without passing a message from the receiver to the sender. Consider a system with two processes p and q; p may broadcast a message with feedback by sending it to q and deciding; completion of the send operation implies that q has received the message when p decides. On the other hand, it will be clear that p cannot compute the infimum by only sending a message (i.e., without receiving from q).

The strength of the wave algorithm concept lies in the equivalence of causal chains and message chains under asynchronous communications. Hence, the causal dependency required for wave algorithms implies the existence of message chains leading to a decide event and allows infimum computations.

Wave algorithms for synchronous communications can be defined in two ways. First, one may require a *causal* relation between each process and a decide event; in this case, the definition is too weak to allow infimum computation with waves, as is seen from the above example. Second, one may require a *message chain* from each process to a decide event; in this case, the definition is too strong to conclude that each PIF algorithm satisfies it, as is also seen from the example. In either case, however, we lack the full strength of the wave concept for asynchronous communications.

Exercise 6.2. The proof assumes it is possible to choose j'_q such that $J \leq j'_q$ is not satisfied, i.e., that J is not a bottom.

Let X be an order with bottom $\perp$. If p has seen a collection of inputs whose infimum is $\perp$, a decision on $\perp$ can safely be made because it follows that $\perp$ is the correct output. Now consider the following algorithm. Process p receives the inputs of its neighbors and verifies whether their infimum is $\perp$; if so, p decides on $\perp$, otherwise a wave algorithm is started and p decides on its outcome. This algorithm correctly computes the infimum, but does not satisfy the dependency requirement of wave algorithms.

Exercise 6.3. The required orders on the natural numbers are:
(1) Define $a \leq b$ as a divides b.
(2) Define $a \leq b$ as b divides a.
The first order has bottom 1, the second has bottom 0.

The required orders on the subsets are:
(1) Define $a \leq b$ as a is a subset of b.
(2) Define $a \leq b$ as b is a subset of a.
The first order has bottom $\varnothing$, the second has bottom U.

The given solutions are unique; the proof of the Infimum Theorem (see Solution 6.4) reveals that the order $\leq_\star$ whose infimum function is $\star$ is defined by

$$a \leq_\star b \equiv (a \star b) = a.$$

Exercise 6.4. This exercise is solved by an extensive, but elementary manipulation with axioms and definitions. Let the commutative, associative, and idempotent operator $\star$ on X be given and define a binary relation $\leq_\star$ by $x \leq_\star y \iff (x \star y) = x$.

We shall first establish (using the assumed properties of $\star$) that the relation $\leq_\star$ is a partial order on X, i.e., that this relation is transitive, antisymmetric and reflexive.

1. **Transitivity:** Assume $x \leq_\star y$ and $y \leq_\star z$; by definition of $\leq_\star$, $(x\star y) = x$ and $(y\star z) = y$. Using these and associativity we find $(x \star z) = (x \star y) \star z = x \star (y \star z) = x \star y = x$, i.e., $x \leq_\star z$.

2. **Antisymmetry:** Assume $x \leq_\star y$ and $y \leq_\star x$; by definition of $\leq_\star$, $x \star y = x$ and $y \star x = y$. Using these and commutativity we find $x = y$.

3. **Reflexivity:** By idempotency, $x \star x = x$, i.e., $x \leq_\star x$.

In a partial order $(X, \leq)$, z is the infimum of x and y w.r.t. $\leq$ if z is a lower bound, and z is the largest lower bound, i.e., (1) $z \leq x$ and $z \leq y$; and (2) every t with $t \leq x$ and $t \leq y$ satisfies $t \leq z$ also. We continue to show that $x \star y$ is the infimum of x and y w.r.t. $\leq_\star$; let $z = x \star y$.

1. **Lower bound:** Expand z and use associativity, commutativity, associativity again, and idempotency to find $z \star x = (x \star y) \star x = x \star (y \star x) = x \star (x \star y) = (x \star x) \star y = x \star y = z$, i.e., $z \leq_\star x$.
   Expand z and use associativity and idempotency to find $z \star y = (x \star y) \star y \; x \star (y \star y) = x \star y = z$, i.e., $z \leq_\star y$.

2. **Smallest l.b.:** Assume $t \leq_\star x$ and $t \leq_\star y$; by definition, $t \star x = t$ and $t \star y = t$. Then, using these, $z = x \star y$, and associativity, we find $t \star z = t \star (x \star y) = (t \star x) \star y = t \star y = t$, i.e., $t \leq_\star z$.

In fact, the partial order is completely determined by its infimum operator, i.e., it is possible to demostrate that $\leq_\star$ as defined above is the *only* partial order of which $\star$ is the infimum operator. This proof is left as a further exercise!

Exercise 6.5. Consider the terminal configuration γ as in the proof of Theorem 6.16. It was shown there that at least one process has decided; by Lemma 6.4 all other processes have sent, and by the algorithm the process itself has sent, so $K = N$. Now F, the number of false *rec* bits, equals $N - 2$ and each process has either 0 or 1 of them; consequently, exactly two processes have 0 false *rec* bits.

Exercise 6.6. Each message exchanged in the echo algorithm contains a label (string) and a letter. The initiator is assigned label ϵ and every non-initiator computes its node label upon receipt of the first message (from its father), by concatenating the label and the letter in the message. When sending tokens, a process includes its node label, and the letter in the message is chosen different for each neighbor. In this way the label of each node extends the label of its father by one letter, and the labels of the sons of one node are different. To compute the link labels, each node assigns to the link to the father the label ϵ and to all other links the label found in the message received through that link. If a message from the initiator is received but the initiator is not the father, the node has a frond to the root and relabels its father link with the label of the father (cf. Definition 4.41).

The labeling can be computed in $O(D)$ time because it is not necessary that a node withhelds the message to its father until all other messages have been received (as in the Echo algorithm). In fact, as was pointed out by Erwin Kaspers, it is not necessary to send messages to the father at all. When forwarding the message $\langle\,\mathbf{tok}, l_u, a\,\rangle$ to neighbor w, u sets α_{uw} to $l_u \centerdot a$, which is w's label *in case it becomes u's son.* When a message $\langle\,\mathbf{tok}, l, b\,\rangle$ is later received from w, α_{uw} is set to l. The modified algorithm exchanges only $2E - (N-1)$ messages, but is more difficult to terminate. Processes must wait indefinitely for the possible receipt of a message from every neighbor that has become a son.

Exercise 6.7. The crucial step in the proof is that the first i messages received by q were sent in different send operations by p, and this is true even if some messages may get lost. If messages can be duplicated, q may receive the same message more than once, so the first i receive events may match less than i send events in p.

Exercise 6.8. According to Theorem 6.12, we must add a variable v_p for process p, initialized to p's input j_p; "send $\langle\,\mathbf{tok}\,\rangle$" and "receive $\langle\,\mathbf{tok}\,\rangle$" are replaced by "send $\langle\,\mathbf{tok}, v_p\,\rangle$" and "receive $\langle\,\mathbf{tok}, v\,\rangle$; $v_p := \max(v_p, v)$", respectively. Finally, upon deciding the output is written; here is the result:

```
cons $D$           : integer    = the network diameter ;
var  $Rec_p[q]$    : $0..D$     init $0$, for each $q \in In_p$ ;
     $Sent_p$      : $0..D$     init $0$ ;
     $v_p$         : integer    init $j_p$   (* the input *)
begin if $p$ is initiator then
          begin forall $r \in Out_p$ do send $\langle\,\mathbf{tok}, v_p\,\rangle$ to $r$ ;
                $Sent_p := Sent_p + 1$
          end ;
      while $\min_q Rec_p[q] < D$ do
            begin receive $\langle\,\mathbf{tok}, v\,\rangle$ from $r$; $v_p := \max(v_p, v)$ ;
                  $Rec_p[r] := Rec_p[r] + 1$ ;
                  if $\min_q Rec_p[q] \geq Sent_p$ and $Sent_p < D$ then
                     begin forall $r \in Out_p$ do send $\langle\,\mathbf{tok}, v_p\,\rangle$ to $r$ ;
                           $Sent_p := Sent_p + 1$
                     end
            end ;
      $out_p := v_p$
end
```

Exercise 6.9. (a) As the algorithm counts the number of messages received, a duplicated message may corrupt the count and cause a process to decide or send to its father too early. This can be remedied by administrating the received messages in a bit-array rather than in a count; when receiving form neighbor q, $rec_p[q]$ is set to true. Sending to the father or deciding occurs when all bits are true.

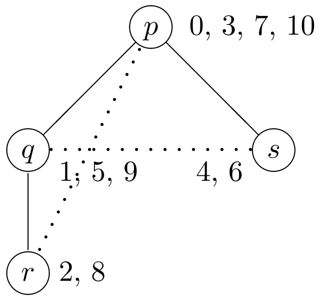
(b) If a message $\langle\,\mathbf{sets}, Inc, NInc\,\rangle$ is received for the second time (by p), $rec_p[q]$, $Inc \subseteq Inc_p$ and $NInc \subseteq NInc_p$ hold already, so the message does not change the state of p and no new message is sent. Consequently, the algorithm handles duplications correctly and no modification is necessary.

Exercise 6.10. In a complete bipartite network, if p is any process and q is any neighbor of p, then every process is a neighbor of p or a neighbor of q. The algorithm extends the sequential polling algorithm (Algorithm 6.10) in the sense that the initiator polls its neighbors, and one neighbor (q) polls all of its (q's) neighbors.

The initiator polls its neighbors, but uses a special token for exactly one of its neighbors (for example, the first). The polled processes respond as in Algorithm 6.10, except the process that receives the special token: it polls all other neighbors before sending the token back to the originator.

Project 6.11. My conjecture is that the statement is true; if so, it demonstrates that sense of direction "helps" in hypercubes.

Exercise 6.12. In this picture the dotted lines are fronds and p is the root of the tree. The numbers written at each node indicate the order in which the token visits the nodes.
Observe that after the third step, when p has received the token from r, p forwards the token to s, while rules R1 and R2 would allow to send it back to r. This step is a violation of rule R3; of course, R3 must be violated somewhere to obtain a tree that is not DFS.



Exercise 6.13. The node labels are of course computed by a DFS traversal, where the token carries a counter that is used as the label and then incremented every time the token visits a process for the first time. After such a traversal (with any of the DFS algorithms presented in Section 6.4) each process knows its father, and which of its neighbors are its sons. The value k_u in Definition 4.27 is found as the value of the token that u returns to its father. Now each node can compute the link labels as in Definition 4.27 if it can obtain the node label of each of its neighbors.

In the sequential algorithm (Algorithm 6.14) and the linear time variants (Algorithm 6.15 and Algorithm 6.16/6.17) a node sends a message to each of its neighbors after receiving the token for the first time. In this message the node label can be communicated to every neighbor, which allows to perform the computation as argued above.

In the linear message DFS algorithm (with neighbors knowledge, Algorithm 6.18) the labels of neighbors can also be obtained, but at the expense of further increasing the bit complexity. In addition to the list of all visited nodes, the token will contain the node label of every visited node. If node u has a frond to an ancestor w, then w and its label are contained in the token when u first receives it. If u has a frond to a descendant w, the token contains w and its label when the token returns to u after visiting the subtree containing w. In both cases u obtains w's label from the token before the end of the algorithm.

Exercise 6.14. Each process sorts the list of names; the token contains an array of N bits, where bit i is set true if the process whose name has rank i is in the list.

Exercise 6.15. Each message sent upwards in the tree contains the sum over the subtree; the sender can compute this sum because it knows its own input and has received messages (hence sums) from all its subtrees. In order to avoid an explicit distinction between messages from sons (reporting subtree sums) and messages from the father and via fronds (not carrying any relevant information) we put the value 0 in every message not sent upward. Assuming p's input is given as j_p, the program for the initiator becomes:

```
begin forall $q \in Neigh_p$ do send $\langle \mathbf{tok}, 0 \rangle$ to $q$ ;
        while $rec_p < \#Neigh_p$ do
                begin receive $\langle \mathbf{tok}, s \rangle$ ; $j_p := j_p + s$ ; $rec_p := rec_p + 1$ end ;
        Output: $j_p$
end
```

and for non-initiators:

```
begin receive $\langle \mathbf{tok}, s \rangle$ from neighbor $q$ ; $father_p := q$ ; $rec_p := 1$ ;
        forall $q \in Neigh_p$, $q \neq father_p$ do send $\langle \mathbf{tok}, 0 \rangle$ to $q$ ;
        while $rec_p < \#Neigh_p$ do
                begin receive $\langle \mathbf{tok}, s \rangle$ ; $j_p := j_p + s$ ; $rec_p := rec_p + 1$ end ;
        send $\langle \mathbf{tok}, j_p \rangle$ to $father_p$
end
```

Non-initiators ignore the value received in the first message; however, this value is *not* received from a son and it is always 0.

Exercise 6.17. In the *longest* message chain it is always the case that the receipt of m_i and the sending of m_{i+1} occur in the same process. Otherwise, the causal relation between these events is the result of a causal chain containing at least one additional message, and the message chain can be extended.

The chain time complexity of Algorithm 6.8 is exactly N.

First, consider a message chain in any computation. In the algorithm each process sends to every neighbor, and these sends are not separated by receives in the same process; consequently, each message in the chain is sent by a different process. This implies that the length of the chain is at most N.

Second, consider an execution on processes p_1 through p_N where only p_1 initiates the algorithm and the first message received by p_{i+1} (for $i > 1$) is the one sent by p_i. This computation contains a message chain of length N.

(The time complexity of the algorithm is 2; because p_1 sends a message to all processes upon initialization, all processes are guaranteed to send within one time unit from the start. Indeed, in the execution given above this single message is bypassed by a chain of $N - 1$ messages.)

Chapter 7: Election Algorithms

Exercise 7.1. Assume, to the contrary, that process l in network G becomes elected and its change to the leader state is not preceded by any event in process p. Construct network G', consisting of two copies of G, with one additional link between the two copies of process p; this is to ensure G' is connected. In both copies of G' the relative ordering of identities is similar to the ordering of the identities in G. Because a comparison algorithm is assumed, the computation on G leading to the election of l can be replayed in the two parts of G', which results in a computation in which two processes become leader.

Exercise 7.2. Because $\langle$ **wakeup** $\rangle$ messages are forwarded immediately by every process, all processes have started the tree algorithm withing D time units from the start of the whole algorithm. It remains to show that the time complexity of the tree algorithm, including the flooding of the answer, is D; assume the tree algorithm is started at time t.

Consider neighbors p and q in the tree; it can be shown by induction that process q sends a message to p at the latest at time $t + depth(T_{qp})$. (Indeed, the induction hypothesis allows to derive that by then q has received from all neighbors other then p. If no message was received from p the “first” token by q is sent to p, otherwise a “flooding” token is sent.) Consequently, the latest possible time for p to decide is $t + 1 + \max_{q \in Neigh_p} depth(T_{qp})$, which is at most $t + D$ (for p a leaf on the diameter path of the tree).

Exercise 7.3. For convenience, write $\mathbf{H}_0 = \sum_{j=1}^{0} \frac{1}{j} = 0$, so $\sum_{i=1}^{m} \mathbf{H}_i = \sum_{i=0}^{m} \mathbf{H}_i$. Further observe that $\mathbf{H}_i = \sum_{j=1}^{i} \frac{1}{j} = \mathbf{H}_m - \sum_{i<j\le m} \frac{1}{j}$. Now

$$\begin{array}{rcll}
\sum_{i=0}^{m} \mathbf{H}_i & = & \sum_{i=0}^{m} \left(\mathbf{H}_m - \sum_{i<j\le m} \frac{1}{j} \right) & \text{see above} \\
& = & (m+1)\mathbf{H}_m - \sum_{i=0}^{m} \sum_{i<j\le m} \frac{1}{j} & \\
& = & (m+1)\mathbf{H}_m - \sum_{j=1}^{m} \sum_{0\le i<j} \frac{1}{j} & \text{reverse summations} \\
& = & (m+1)\mathbf{H}_m - \sum_{j=1}^{m} 1 & \text{because } \sum_{0\le i<j} \frac{1}{j} = 1 \text{ !!} \\
& = & (m+1)\mathbf{H}_m - m &
\end{array}$$

Exercise 7.4. This is an example of solving a summation by integration. By drawing rectangles of sizes $\frac{1}{i}$ in a graph of the function $f(x) = \frac{1}{x}$ it is easily seen that $\mathbf{H}_N > \int_{x=1}^{N+1} \frac{dx}{x} = \ln(N+1)$. The other bound is shown similarly.

Exercise 7.5. As the token of the winner makes N hops, and each of the remaining $N-1$ processes initiates a token, which travels at least one hop, the number of messages is at least $2N-1$. This number is achieved when each process except the one with smallest identity is followed in the ring by a process with smaller identity, i.e., the identities are “ordered” decreasingly in the message direction (mirror Figure 7.4).

Exercise 7.6. Instead of repeating the argument in the proof of Theorem 7.6, we give the following informal argument.
A non-initiator only forwards messages, so the sequence of non-initiators between two successive initiators simulates an asynchronous link between the two initiators. It can thus be seen that the number of messages sent by initiators is $S \cdot \mathbf{H}_S$ on the average.
A message sent by an initiator may be forwarded for several hops by non-ititiators before be-

ing received by the next initiator; on the average, the number of links between two successive initiators is N/S. Consequently, the average complexity is $N \cdot \mathbf{H}_S$.

Exercise 7.7. We first construct the arrangements that yield long executions; if there are two processes ($N = 2^1$) with identities 1 and 2, the algorithm terminates in two $(1 + 1)$ rounds. Given an arrangement of 2^k processes with identities 1 through 2^k for which the algorithm takes $k + 1$ rounds, place one of the identities $2^k + 1$ through 2^{k+1} between each of the processes in the given arrangement. We obtain an arrangement of 2^{k+1} processes with identities 1 through 2^{k+1} for which the algorithm uses $k + 2$ rounds.

Assume the arrangement contains two local minima; if both minima and their neighbors initiate, both minima survive the first round, so no leader is elected in the second round. Hence, to guarantee termination within two rounds the arrangement can have only one local minimum, which means that the identities are arranged as in Figure 7.4. The algorithm terminates in one round if and only if there is a single initiator.

Exercise 7.8. $E_{\text{CR}} = \{(s_1, \ldots, s_k) : 1 < i \leq k \Rightarrow s_1 < s_i\}$.

Exercise 7.9. In the Chang-Roberts algorithm, process p compares a received identity q to its own, while in the extinction algorithm p compares q to the smallest identity p has seen so far.

A difference would occur if p receives an identity q smaller than its own, but larger than an identity m received previously. However, this does not occur on the ring if communication is fifo; p receiving m before q then implies that m is on the path from q to p in the ring, so m purges q's token before it reaches p. In case of non-fifo communication the extinction algorithm may safe some messages over Chang-Roberts.

Exercise 7.10. A connect message sent by a process at level 0 can be sent to a sleeping process. Connect messages at higher levels are only sent to processes from which an accept was previously received, implying that the addressee is awake. A test message can also be sent to a sleeping process.

The other messages are sent via edges of the tree (initiate, report, changeroot) or in reply to another message (accept, reject), both implying that the addressee is awake.

Exercise 7.11. Let the first node wake up at t_0, then the last one wakes up no later than $t_0 + N - 1$. By $t_0 + N$ all connect messages of level 0 have been received, and going to level 1 depends only on propagation of initiate messages through the longest chains that are formed, and completes by $t_1 = t_0 + 2N$.

Assume the last node reaches level l before time $t_l \leq (5l - 3)N$. Each node sends less than N test messages, hence has determined its locally least outgoing edge by $t_l + 2N$. The propagation of report, changeroot/connect, and initiate messages (for level $l + 1$) via MST edges takes less than $3N$ time, hence by time $t_{l+1} \leq t_l + 5N \leq (5(l + 1) - 3)N$, all nodes are at level $l + 1$.

Exercise 7.12. The crux of the exercise is to show that Tarry's traversal algorithm is an $O(x)$ traversal in this case. Indeed, after $6x - 3$ hops the token has visited a planar subgraph with at least $3x - 1$ edges, which implies that the visited subnetwork contains at least $x + 1$ nodes. The election algorithm is now implied by Theorem 7.24.

Exercise 7.13. *Solution 1:* Tarry's traversal algorithm is an $O(x)$ traversal algorithm for the

torus, because the degree of each node is bounded by 4. Therefore, after $4x$ hops the token has traversed at least $2x$ different edges, implying that $x+1$ nodes were visited. The election algorithm is now implied by Theorem 7.24.
Solution 2: By Theorem 7.21, election is possible in $O(N \log N + E)$ messages; for the torus, $E = 2 \cdot N$, which implies an $O(N \log N)$ bound.

Exercise 7.14. By Theorem 7.21, election is possible in $O(N \log N + E)$ messages; for the hypercube, $E = \frac{1}{2} N \log N$, which implies an $O(N \log N)$ bound.

Exercise 7.15. By Theorem 7.21, election is possible in $O(N \log N + E)$ messages; the k bound on the node degrees implies that $E \leq \frac{1}{2} kN$, showing that the GHS algorithm satisfies the requirements set in the exercise.
The class of networks considered in this exercise is kN-traversable (using Tarry's algorithm), but applying the KKM algorithm only gives an $\mathrm{O}(k.N \log N)$ algorithm.

Chapter 8: Termination Detection

Exercise 8.1. Process p is *active* if the guard on an internal or send step is enabled, which means, in Algorithm A.2, if

$$(\#\{q : \neg rec_p[q]\} \leq 1 \wedge \neg sent_p) \quad \vee \quad (\#\{q : \neg rec_p[q]\} = 0 \wedge \neg dec_p).$$

All other states are *passive*.

In Algorithm A.1 these *passive* states occur exactly when the process is waiting to execute a receive statement.

Exercise 8.2. The time complexity equals the maximal depth of the computation tree at the moment termination occurs; this can be seen by quantifying the liveness argument in Theorem 8.5. As only processes are internal nodes, this depth is bounded by N. To show that this is also a lower bound, observe that the time between termination and its detection is N when the algorithm is applied to the Ring algorithm (Algorithm 6.2).

Exercise 8.3. Construct the spanning tree in parallel with the basic computation and use the tree algorithm (Algorithm 6.3) for the detection algorithm. If the spanning tree is completed when the basic algorithm terminates, detection occurs within N time, but if the basic algorithm terminates very fast this modification gives no improvement.

Exercise 8.4. According to P_0, all p_i with $i > t$ are passive. So, if p_j with $j \leq t$ is activated, P_0 is not falsified because the state of all p_i with $i > t$ remains unchanged. If p_j is activated by p_i with $i > t$, P_0 is false already before the activation, hence it is not falsified.

Exercise 8.5. The basic computation circulates a token, starting from p_0, but in the opposite direction. A process becomes *passive* after sending the token to its predecessor. A process holding the basic token performs local computations until its predecessor has forwarded the detection token to it. In this way, the detection token is forwarded $(N-1)$ times between every two steps of the basic token.

Exercise 8.6. Action $\mathbf{R}_p$ becomes:

```
$\mathbf{R}_p$: { A message $\langle \mathbf{mes}, c \rangle$ has arrived at $p$ }
    begin receive $\langle \mathbf{mes}, c \rangle$ ;
          if $state_p = active$
             then send $\langle \mathbf{ret}, c \rangle$ to $p_0$
             else begin $state_p := active$ ; $cred_p := c$ end
    end
```

Exercise 8.7. Observe that the process names are exclusively used by a process to recognize whether it is the initiator of a token it receives; Knowledge of the ring size can be used alternatively. Tokens carry a hop counter, which is 1 for the first transmission, and incremented every time it is forwarded. When a token with hop count N is received (by a *quiet* process), termination is concluded.

Exercise 8.8. This was done by Michiel van Wezel; see [WT94]. Now try your hands on Mattern's algorithm in [Mat87, Section 6.1].

Chapter 9: Anonymous Networks

Exercise 9.1. Algorithm C repeats the subsequent execution of A and B until, according to B, ψ has been established. Explicit termination of A and B is necessary because the processes must be able to continue (with B, or A, respectively) after termination.

If A establishes ψ with probability P, the expected number of times A is executed by C is $1/P$; if the complexity of A and B is known, this allows to compute the expected complexity of C.

Exercise 9.2. If the expected message complexity is K, the probability that the LV algorithm exchanges more than K/ϵ messages is smaller than ϵ. We obtain an MC algorithm by simulating the LV algorithm for at least K/ϵ, but at most a finite number of messages.

Each process continues the execution until it has sent or received K/ϵ messages. Indeed, with probability $1-\epsilon$ or higher, termination with postcondition ψ has occurred before that time, and the number of overall communication actions is bounded (by $N \cdot K/\epsilon$), which shows the new algorithm is Monte Carlo. (If K depends on network parameters such as the size, these parameters must be known.)

Exercise 9.3. A solution based on Algorithm 9.1 does not meet the time bound mentioned in the exercise, because the time complexity of the echo algorithm is linear. However, the labeling procedure given in the solution to Exercise 6.6 assigns different names within the bounds.

Exercise 9.4. Each process tries to send to every neighbor and to receive from every neighbor. When a process receives from some neighbor, it it becomes *defeated* and will thereafter only receive messages. When a process succeeds to send to some neighbor, it keeps trying to send to and receive from the other neighbors. When a process has sent to every neighbor, it becomes *leader* and will thereafter not receive any messages.

At most one process becomes leader because if a process becomes leader, all its neighbors (i.e., all processes) are *defeated* and will not become leader. At most $N-1$ processes become *defeated* because if this many processes are *defeated* there is no process that can send to the last process. The algorithm terminates because only finitely many (*viz.*, $\frac{1}{2}N(N-1)$) communications can occur; consider a terminal configuration. At least one process is not *defeated*; if this process still wants to send, it can do so because all processes are ready to receive. Consequently, the process has sent to every neighbor and is *leader*.

Exercise 9.5. Let the input of p be given as a_p; the following algorithm terminates after exchanging at most NE messages, and in the terminal configuration, for every p: $b_p = \max_q a_q$.

```
$b_p := a_p$ ; shout ⟨ **val**, $b_p$ ⟩ ;
**while** *true* **do**
    **begin** receive ⟨ **val**, $b$ ⟩ ;
        **if** $b > b_p$ **then begin** $b_p := b$ ; shout ⟨ **val**, $b_p$ ⟩ **end** ;
    **end**
```

Exercise 9.6. (a) The processes execute the Tree algorithm (Algorithm 6.3), reducing the problem to election between two processes, namely the two neighbors that send each a message. These break symmetry by repeatedly exchanging a random bit: if the bits are equal, they continue; if they are different, the process that sent a 1 becomes leader.

(b) No; this can be shown with a symmetry preserving argument like the one used to prove Theorem 9.5.

Exercise 9.7. Both networks depicted in Figure 6.21 have diameter at most 2. Let any deterministic algorithm be given; by using symmetry arguments as in the proof of Theorem 9.5 it is shown that there is an execution in, say, the left network in which every process executes the same steps. The same sequence of steps, executed by every process in the right network, constitutes an execution of the algorithm in that network. Consequently, the same answer can be found by the processes in the left and in the right network; but as the sizes of these networks are different, the incorrect answer is found in at least one network.

This argument is valid for process and for message terminating algorithms. As the erroneous execution is finite, the argument shows that any probabilistic algorithm may err with positive probability of error.

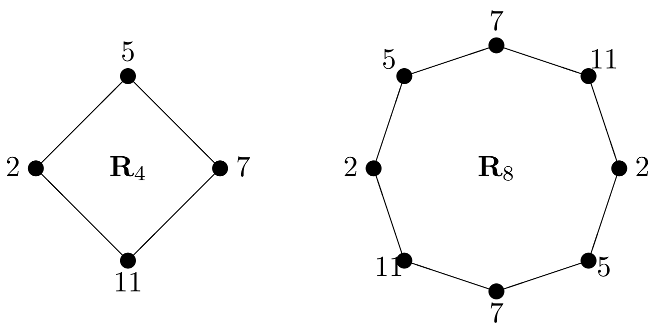
Exercise 9.8. Assume N is composite, i.e., $N = k \cdot l$ for k, $l > 1$; divide the processes in k groups of l each. Call a coniguration symmetric if the sequence of l states found in one group is the same for each group, i.e., $s_i = s_{i+l}$ for each i. If γ is a symmetric configuration in which a communication event between p_i and p_{i+1} is enabled, then the same event is enabled between p_{i+l} and p_{i_1+l}, between p_{i+2l} and p_{i+1+2l}, and so forth. These k events regard disjoint pairs of processes (because $l \geq 2$), hence the computation can continue with these k events in any order, after which a symmetric configuration is found again. In this way, either an infinite computation can be constructed, or a computation terminating in a symmetric configuration; the number of leaders in such a configuration is a multiple of k, hence not 1.

Exercise 9.9. The function f is non-constant (because there exist *true* as well as *false* valued inputs), and hence is not computable by a process terminating algorihm if N is not known (Theorem 9.8).

However, f can be computed by a process terminating algorithm. To this end, each process sends its input to its successor, who will determine if the two input equal. If and only if mutual equality holds for *all* processes, f is *true*, so the problem of computing f reduces to evaluating the logical AND (Theorem 9.9).

Exercise 9.10. The function g is cyclic, but non-constant, and hence is not computable by a process terminating algorihm (Theorem 9.8).

To show that evaluation of g is also impossible by a message terminating algorithm, consider the rings $\mathbf{R}_4$ and $\mathbf{R}_8$:



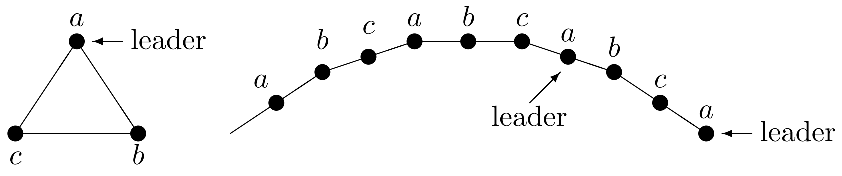
Assuming there exists a correct algorithm to compute g, there exists an execution, C_4 say, of this algorithm on $\mathbf{R}_4$ that (message) terminates with result *true*. Each process in $\mathbf{R}_8$ has neighbors with the same inputs as the neighbors of its counterpart (the unique process in $\mathbf{R}_4$ with the same input). Consequently, there exists a computation of the algorithm on $\mathbf{R}_8$ in which every process executes the same steps as its counterpart in $\mathbf{R}_4$. In particular, the algorithm (message) terminates with result *true*.

Exercise 9.11. Because failures are acceptable with small probability, it suffices to execute a single phase of the Itai–Rodeh algorithm: each process randomly selects a name from the range $[0..U)$ and verifies, by circulating a token, if its name is minimal. Smaller processes remover token, so if a process receives its own token (hopcount N) it becomes a leader. By increasing U the probability that the minimum is not unique (a situation that would lead to the selection of more than one leader) is made arbitrarily small. The message complexity is at most N^2 because each token makes at most N steps.

In its simplest form, this algorithm terminates only in the leader(s); Kees van Kemenade demonstrated how the leader(s) can process-terminate the other processes, still within N^2 messages (worst-case). Each token (as long as it survives) accumulates the number of hops from its initiator to the next process with the same name. When a process becomes leader and learns that the next process with the same name is h hops away, it sends a termination signal for $h-1$ hops. These hops do not push the message complexity over N^2, because the tokens of the next $h-1$ processes were not fully propagated.
(The simpler strategy of sending a termination signal to the next leader, i.e., over h hops, generates N^2+N messages in the case all processes draw the same name.)

Exercise 9.12. The processes execute input collection as in Theorem 9.6; process p obtains a sequence seq_p of names, consisting of an enumeration of the names on the ring, starting at p. Because the distribution of identities is non-periodic, the N sequences are different; the unique process whose string is lexicographically maximal becomes elected. (For p this is the case if and only if seq_p is maximal among the cyclic shifts of seq_p.)

Exercise 9.13. Assuming such an algorithm exists, it will generate a finite computation C on the small ring on the left at the end of which one process, a say, terminates as a leader. Let the maximal length of the trace of any message received in C be K, and the largest number of random bits used by any process.



The steps taken in C can occur in processes in a larger ring, as depicted on the right, if it contains a pattern of length $K+1+3$ labeled with the repeated pattern a, b, c, ... Moreover, each process in this segment must have the same first L random bits as the corresponding process on the small ring. The first K processes execute a strict prefix of the steps taken by their counterpart in the small ring, but the last $3+1$ processes simulate the entire sequence of steps, implying that two processes labeled a terminate as a leader. The probability that this happens is at least $2^{-(K+3+1).L}$.

A very large ring may contain sufficiently many segments as labeled above so as to make the probability of error arbitrarily large, and have the name d at one place in the ring to make it non-periodic. This shows that erroneous behavior of the algorithm is extremely likely, even in non-periodic rings.

Exercise 9.14. The solution performs input collection and can therefore be used to evaluate any cyclic function; it is based on Itai and Rodeh's algorithm to evaluate the ring size (Algorithm 9.5).

In a test message of Algorithm 9.5 a process includes its input; whenever a process receives a message with hop count h it learns the input of the process h hops back in the ring. When the algorithm terminates (with common value e of the *est* variables), each process knows the inputs of the e processes preceding it and evaluates g on this string.

With high probability (as specified in Theorem 9.15) e equals N, and in this case the result equals the desired value of g.

Exercise 9.15. It is not possible to use Safra's algorithm, because this algorithm pre-assumes that a leader (the process named p_0 in Algorithm 8.7) is available.

Neither can any other termination detection algorithm be used; it would turn Algorithm 9.5 into a process terminating algorithm, but such an algorithm does not exist by Theorem 9.12.

Chapter 10: Snapshots

Exercise 10.1. If S^* is *not* meaningful there exist p and q and corresponding send/receive events s and r such that $a_p \prec_p s$ and $r \prec_q a_q$, which implies $a_p \prec a_q$, so $a_p \not\parallel a_q$.

Next assume for some p and q, $a_p \prec a_q$; that is, there exists a causal chain from a_p to a_q. As a_p and a_q internal but in different processes, the chain contains at least one event e in p following a_p, and one event f in q preceding a_q. The cut defined by the snapshot includes f but not e, while $e \prec f$; hence the cut is not consistent, and S^* is not meaningful by Theorem 10.5.

Exercise 10.3. The meaningfulness of the snapshot is not guaranteed at all, but by careful use of colors meaningless snapshots are recognized and rejected. The snapshot shows a terminated configuration if (upon return of the token to p_0) $mc_{p_0} + q = 0$, but this allows calling *Announce* only if in addition $c = \mathit{white}$ and $\mathit{color}_{p_0} = \mathit{white}$.

Let m be a postshot message sent by p to q, which is received preshot. The receipt of the message colors q *black*, thus causing q to invalidate the snapshot by coloring the token *black* as well. Because the receipt of any message colors q *black*, meaningful snapshots may be rejected as well; but termination implies that a terminated snapshot is eventually computed and not rejected.

Chapter 12: Synchrony in Networks

Exercise 12.1. If m denotes the contents of the message, the sender computes $r = \lceil \sqrt{m} \rceil$ and $s = r^2 - m$; observe $r \leq \sqrt{m} + 1$ and $s \leq 2r$. The sender sends a $\langle\, \textbf{start}\, \rangle$ message in, say, pulse i and $\langle\, \textbf{defr}\, \rangle$ and $\langle\, \textbf{defs}\, \rangle$ messages in pulses $i + r$ and $i + s$, respectively.

The receiver counts the number of pulses between the receipt of the $\langle\, \textbf{start}\, \rangle$ and the $\langle\, \textbf{defr}\, \rangle$ and $\langle\, \textbf{defs}\, \rangle$ messages, thus finding r and s, and "decodes" the message by setting m to $r^2 - s$.

Needless to say, this idea can be generalized to a protocol that transmits the message in $O(m^{1/k})$ time by sending $k + 1$ small messages.

Exercise 12.2. Yes, and the proof is similar. The i^{th} process in the segment of the large ring goes through the same sequence of states as the corresponding process in the small ring, but up to the i^{th} configuration only.

Exercise 12.3. Assume p receives a j-message from q during pulse i; let σ and τ be the time of sending and receipt of the message. As the message is received during pulse i,

$$2i\mu \leq CLOCK_p^{(\tau)} \leq 2(i + 1)\mu.$$

As the message delay is strictly bounded by μ,

$$2i\mu - \mu < CLOCK_p^{(\sigma)} \leq 2(i + 1)\mu.$$

As the clocks of neighbors differ by less than μ,

$$2i\mu - 2\mu < CLOCK_q^{(\sigma)} < 2(i + 1)\mu + \mu.$$

By the algorithm, the message is sent when $CLOCK_q = 2j\mu$, so

$$2i\mu - 2\mu < 2j\mu < 2(i + 1)\mu + \mu,$$

which implies $i \leq j \leq i + 1$.

Exercise 12.4. The time between sending the $\langle\, \textbf{start}\, \rangle$ message and sending an i-message is exactly $2i\mu$. Each of the messages incurs a delay of at least 0 but less than μ, so the time between the receipt of these messages is between $2i\mu - \mu$ and $2i\mu + \mu$ (exclusive).

Exercise 12.6. In all three cases the message is propagated through a suitably chosen spanning tree.

The N-Clique: In the flooding algorithm for the clique, the initiator sends a message to its $N - 1$ neighbors in the first pulse, which completes the flooding. The message complexity is $N - 1$ because every process other than the initiator receives the message exactly once.

The $n \times n$-Torus: In the torus, the message is sent upwards with a hop counter for $n - 1$ steps, so every process in the initiator's column (except the initiator itself) receives the message exactly once. Each process in this column (that is, the initiator and every process receiving a message from *down*) forwards the message to the *right* through its row, again with a hop counter and for $n - 1$ steps. Here every process not in the initiator's column receives the messages exactly once and from the *left*. Again, every process except the initiator receives the message exactly once.

The last process to receive the message is in the row below and the column left of the initiator; in the $(n-1)^{\text{th}}$ pulse the message receives this row, and only in the $2(n-1)^{\text{th}}$ pulse the message is propagated to the end of the row.

The number of pulses can be reduced to $2\lfloor n/2 \rfloor$ by starting upwards and downwards from the initiator until the messages almost meet, and serve each row also by sending in two directions. *The n-Dimensional Hypercube:* The initiator sends a message through its n links in the first pulse. When receiving a message through link i, a process forwards the message via all links $< i$ in the next pulse.

It can be shown by induction on d that a node at distance d from th initiator receives the message exactly once, in pulse d, and via the highest dimension in which its node label differs from the initiator. This shows that the message complexity is $N-1$ and the number of pulses is n.

Exercise 12.7. When awaking, the processes initiate a wake-up procedure and then operate as in Section 12.2.1. As a result of the wake-up, the pulses in which processes start the election may differ by D (the network diameter) but this is compensated for by waiting until pulse $p \cdot (2D)$ before initiating a flood.

Exercise 12.9. The level of a node equals its distance to the initiator; if $d(l, p) = f$ and $q \in \mathit{Neigh}_p$, then $d(p, q) = 1$ and the triangle inequality implies $d(l, q) \geq f - 1$ and $d(l, q) \leq f + 1$.

Chapter 14: Fault Tolerance in Asynchronous Systems

Exercise 14.1. We leave the (quite trivial) correctness proofs of the algorithms as new exercises!

If only termination and agreement are required: Each correct process immediately decides on 0. (This algorithm is trivial because no 1-decision is possible.)

If only termination and non-triviality are required: Each process immediately decides on its input. (Agreement is not satisfied because different decisions can be taken.)

If only agreement and non-triviality are required: Each process shouts its input, then waits until $\lceil \frac{N+1}{2} \rceil$ messages with the same value have been received, and decides on this value. (This algorithm does not decide in every correct process if insufficiently many processes shout the same value.)

Exercise 14.2. We leave the (quite trivial) correctness proofs of the algorithms as new exercises!

1. The even parity implies that the entire input can be reconstructed if $N-1$ inputs are known. *Solution:* Every process shouts its input and waits for the receipt of $N-1$ values (this includes the process' own input). The N^{th} value is chosen so as to make the overall parity even. The process decides on the most frequent input (0 if there is a draw).

2. *Solution:* Processes r_1 and r_2 shout their input. A process decides on the first value received.

3. *Solution:* Each process shouts its input, awaits the receipt of $N-1$ values, and decides on the most frequently received value.

Can you generalize these algorithms to be t-crash robust for larger values of t?

Exercise 14.3. As in the proof of Theorem 14.11, two sets of processes S and T can be formed such that the processes in each group decide independently of the other group. If one group can reach a decision in which there is a leader, an execution can be constructed in which two leaders are elected. If one group can reach a decision in which no leader is elected, an execution can be constructed in which all processes are correct, but no leader is elected.

Exercise 14.4. Choose disjoint subsets S and T of $N-t$ processes and give all processes in S input 1 and the processes in T input $1+2\epsilon$. The processes in S can reach a decision on their own; because all inputs in this group are 1, so are the outputs. Indeed, the same sequence of steps is applicable if *all* processes have input 1, in which case 1 is the only allowed output. Similarly, the processes in T decide on output $1+2\epsilon$ on their own. The decisions of S and T can be taken in the same run, contradicting the agreements requirement.

Exercise 14.5. Write $f(s,r) = \frac{1}{2}(N-t+s-1)(s-(N-t)) + r$.

Exercise 14.7. Call the components G_1 through G_k. After determining that the decision vector belongs to G_i, decide on the parity of i.

Exercise 14.8. (1) Sometimes a leader is elected to coordinate a centralized algorithm. Execution of a small number of copies of this algorithm (e.g., in situations where election is not achievable) may still be considerably more efficient than execution by all processes.

(2) The decision vectors of $[k, k]$-election all have exactly k 1's, which implies that the task is disconnected (each node in G_T is isolated). The problem is non-trivial because a crashed process cannot decide 1; therefore, in each execution the algorithm must decide on a vector where the 1's are located in correct processes. Consequently, Theorem 14.15 applies.

(3) Every process shouts its identity, awaits the receipt of $N - t$ identities (including its own), and computes its rank r in the set of $N - t$ identities. If $r \leq k + t$ then it decides 1, otherwise it decides 0.

Exercise 14.9. (1) No, it doesn't. The probability distribution on K defined by $\mathbf{Pr}[\ K \geq k\ ] = 1/k$ satisfies $\lim_{k\to\infty}\ \mathbf{Pr}[\ K \geq k\ ] = 0$, but yet $\mathbf{E}[\,K\,]$ is unbounded.

(2) In all algorithms the probability distribution on the number of rounds K is geometric, i.e., there is a constant $c < 1$ such that $\mathbf{Pr}[\ (K > k)\ ] \leq c \cdot \mathbf{Pr}[\ K \geq k\ ]$, which implies that $\mathbf{E}[\,K\,] \leq (1 - c)^{-1}$.

Exercise 14.10. If all correct processes start round k, at least $N - t$ processes shout in that round, allowing every correct process to receive $N - t$ messages and finish the round.

Exercise 14.11. (1) In the first round, less than $\frac{N-t}{2}$ processes vote for a value other than v; this implies that *every* (correct) process receives a majority of votes for v. Hence in round 2 only v-votes are submitted, implying that in round 3 all processes witness for v, and decide on v.

(2) Assuming the processes with input v do not crash, more than $\frac{N-t}{2}$ processes vote for v in the first round. It is possible that every (correct) process receives all v-votes, and less than $\frac{N-t}{2}$ votes for the other value. Then in round 2 only v-votes are submitted, implying that in round 3 all (correct) processes witness for v, and decide on v.

(3) Assuming the processes with input v do not crash, exactly $\frac{N-t}{2}$ processes vote for v in the first round. In the most favorable (for v) case, all correct processes receive $\frac{N-t}{2}$ v-votes and $\frac{N-t}{2}$ votes for the other value. As can be seen from Algorithm 14.3, the value 1 is adopted for the next round in case of a draw. Hence, a decision for v is possible in this case if $v = 1$. (Of course the choice for 1 in case of draw is arbitrary.)

(4) An initial configurations is bivalent iff the number of 1's, (#1), satisfies

$$\frac{N-t}{2} \leq \#1 < \frac{N+t}{2}.$$

Exercise 14.12. (1) In the first round, more than $\frac{N+t}{2}$ correct processes vote for v; this implies that every correct process accepts a majority of v-votes in the first round, and hence chooses v. As in the proof of Lemma 14.31 it is shown that all correct processes choose v again in all later rounds, implying that a v-decision will be taken.

(2) As in point (1), in the first round all correct processes choose v in the first round, and vote for it in the second round. Hence, in the second round each correct process accepts at least $N - 2t$ v-votes; as $t < N/5$ implies $N - 2t > \frac{N+t}{2}$, is follows that in the second round every correct process decides on v.

Exercise 14.13. Assume such a protocol exists; partition the processes in three groups, S, T, and U, each of size $\leq t$, with the general in S. Let γ_i be the initial configuration where the general has input v.

Because all processes in U can be faulty, $\gamma_0 \rightarrow_{S \cup T} \delta_0$, where in δ_0 all processes in $S \cup T$ have decided on 0. Similarly, $\gamma_1 \rightarrow_{S \cup U} \delta_1$, where in δ_1 all processes in $S \cup U$ have decided on 1.

Now assume the processes in S are all faulty and γ_0 is the initial configuration. First the processes in S cooperate with the processes in T in such a way that all processes in $S \cup T$ decide on 0. Then the processes in S restore their state as in γ_1 and cooperate with the processes in U in such a way that all processes in $S \cup U$ decide on 1. Now the correct processes in S and U have decided differently, which contradicts the agreement.

Exercise 14.14. At most N initial messages are sent by a correct process, namely by the general (if it is correct). Each correct process shouts at most one echo, counting for N messages in each correct process. Each correct process shouts at most two ready messages, counting for $2N$ messages in each correct process. The number of correct processes can be as high as N, resulting in the $N \cdot (3N + 1)$ bound.

(Adapting the algorithm so as to suppress the second sending of ready messages by correct processes improves the complexity to $N \cdot (2N + 1)$.)

Chapter 15: Fault Tolerance in Synchronous Systems

Exercise 15.1. Denote this number by $M_t(N)$; it is easily seen that $M_0(N) = N - 1$. Further,

$$M_t(N) = (N-1) + (N-1) \cdot M_{t-1}(N-1). \qquad (\dagger)$$

The first term is the initial transmission by the general, the second term represents the $N-1$ recursive calls. The recursion is solved by $M_t = \sum_{i=0}^{t} \prod_{j=1}^{i+1} (N-j)$.

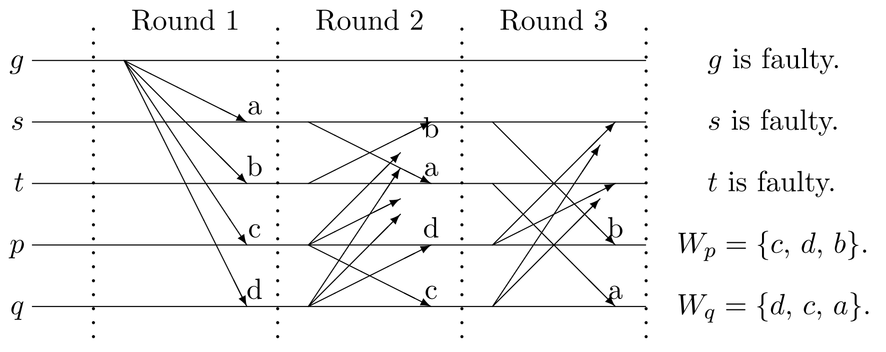
Exercise 15.2. If the decision value is 0 while the general is correct, the general's input is 0 and the discussion in Lemma 15.6 applies. No correct process p initiates or supports any correct process, hence the messages sent by p are for supporting a faulty process q; so p shouts at most t times, which means sending $(N-1).t$ messages.

If the general is faulty, at most $L - 1 = t$ correct processes can initiate without enforcing a 1-decision. Correct process p shouts at most 1 $\langle \mathbf{bm}, 1 \rangle$ message, at most t $\langle \mathbf{bm}, q \rangle$ messages for faulty q, and at most t $\langle \mathbf{bm}, r \rangle$ messages for correct r; this means sending at most $(N-1)(2t+1)$ messages.

Exercise 15.3. The faulty processes can send any number of messages, making the total number unbounded; this is why we usually count the number of messages sent by correct processes only. Faulty g, p_1, p_2 could conspire to send $\langle \mathbf{val}, v \rangle : g : p_1 : p_2$ for many values v to correct p to trap p into sending many messages. Therefore a message that repeats another message's sequence of signatures is declared invalid.

Then, a correct general sends at most $N-1$ messages in pulse 1, and for each sequence $g : p_2 : \ldots : p_i$ at most $N-i$ messages are sent in pulse i by a process (p_i). Thus the number of messages sent by correct processes is at most $\sum_{i=1}^{t+1} \prod_{j=1}^{i} (N-j)$.

Exercise 15.4. Let p and q be correct; the first and second value accepted by p or q are forwarded, and also accepted by the other. So we let them both receive a third value from faulty processes, which is not received by the other:



Observe that p and q forward the value c, and d, respectively, in round 2 and that they forward in round 3 their second received value, namely d, and c, respectively. The third received value (b and a, respectively) is not forwarded in the next round.

Exercise 15.5. The processes achieve interactive consistency on the vector of inputs; the name of p will be the rank of x_p in this vector.

Byzantine processes may refuse to participate, but then their default values are assumed to have the higher ranks in the list. They may also cheat by submitting other values than their input, for example, the name of a correct process. Then this name appears multiply in the list and the correct process decides on the smallest i for which (in the sorted vector) $V[i] = x_p$. Because all correct processes decide on the same vector and this decision vector contains the input of each correct process (by dependence), the decisions of correct processes are different and in the range $1 \dots N$.

Exercise 15.6. Process q receives signed messages M_1 and M_2 from p, i.e., the triples (M_1, r_1, s_1) and (M_2, r_2, s_2), where $r_i = g^{a_i} \bmod P$ and $s_i = (M - d\,r_i)a_i^{-1} \bmod (P-1)$. Reuse of a, i.e., $a_1 = a_2$ is detected by observing that $r_1 = r_2$ in this case. Subtracting s_2 from s_1 eliminates the unknown term $d\,r_i$, as $s_1 - s_2 = (M_1 - M_2)a^{-1}$, i.e., $a = (M_1 - M_2)\,(s_1 - s_2)^{-1}$ (mod ()$P-1$). Next we find $d = (M_1 - a\,s_1)\,r^{-1}$.

Exercise 15.7. In rings without zero-divisors, i.e., the axiom $a.b = 0 \Rightarrow a = 0 \vee b = 0$ applies, a square is the square of at most two elements. Indeed, $x^2 = z^2$ reads $(x-z)(x+z) = 0$ and hence, by the quoted axiom, $x - z = 0$ or $x + z = 0$, i.e., $z = x$ or $z = -x$. In particular, if p is a prime number, $\mathbb{Z}_p$ is free of zero-divisors, and $x^2 = y^2 \pmod p$ implies $y = \pm x \pmod p$.

Now let $n = p.q$ be a composit; the ring $\mathbb{Z}_n$ has zero-divisors because $a.b = 0$ holds if p divides a and q divides b, or vice versa. Each square can be the square of up to four numbers; if $y = x^2$, then also $y = z_i^2$ for

$$\begin{array}{llrrlcrrl}
z_1 & \text{s.t.} & z_1 = & x & \pmod p & \text{and} & z_1 = & y & \pmod q \\
z_2 & \text{s.t.} & z_2 = & x & \pmod p & \text{and} & z_2 = & -y & \pmod q \\
z_3 & \text{s.t.} & z_3 = & -x & \pmod p & \text{and} & z_3 = & y & \pmod q \\
z_4 & \text{s.t.} & z_4 = & -x & \pmod p & \text{and} & z_4 = & -y & \pmod q
\end{array}$$

The four numbers form two pairs of opposites: $z_1 = -z_4$ and $z_2 = -z_3$ and possessing z_i and z_j such that $z_i = \pm z_j$ is useless. However, if we possess z_i and z_j from different pairs, compute $a = z_i + z_j$ and $b = z_i - z_j$ and observe that neither a nor b equals 0, while $a.b = 0$. Hence each of a and b is divided by *one* of p and q, so $\gcd(a, n)$ is a non-trivial factor of n.

The square-root box can be used to produce such a pair. Select a random number x and use the box to produce a square root z of $y = x^2$. Because x was selected randomly, there is a probability of $\frac{1}{2}$ that x and y are from different pairs, revealing the factors of n.

(The main work in modern factoring methods is spent in finding a non-trivial pair of numbers with the same square [Len94].)

Exercise 15.8. The clock C_q has ρ-bounded drift, meaning (Eq. 15.1) that its advance in the real time span t_1–t_2 (where $t_2 \geq t_1$) differs at most a factor $1 + \rho$ from $t_2 - t_1$, i.e.,

$$(t_2 - t_1)(1+\rho)^{-1} \leq C_q(t_2) - C_q(t_1) \leq (t_2 - t_1)(1+\rho).$$

This implies that the amount of real time in which the clock advances from T_1 to T_2 (where $T_2 \geq T_1$) differs at most a factor $1 + \rho$ from $T_2 - T_1$, i.e.,

$$(T_2 - T_1)(1+\rho)^{-1} \leq c_q(T_2) - c_q(T_1) \leq (T_2 - T_1)(1+\rho).$$

Using only the second inequality we find, for any T, T':

$$|\, c_q(T) - c_q(T')\,| \leq (1+\rho).|\, T - T'\,|.$$

Consequently, for $T = C_p(t)$ we have

$$\begin{array}{rcll} |\, c_q(T) - c_p(T) \,| & = & |\, c_q(C_p(t)) - c_q(C_q(t)) \,| & \text{because } C_p(T) = t = c_q(C_q(t)) \\ & \leq & (1+\rho) \,.\, |\, C_p(t) - C_q(t) \,| & \text{by the bounded drift of } c_q \\ & \leq & (1+\rho) \,.\, \delta & \text{as } C_p,\, C_q \text{ are } \delta - \text{synchronized} \\ & & & \text{at real time } t. \end{array}$$

Exercise 15.9. Process p asks the server for the time, the server replies immediately with a $\langle\, \mathbf{time}, T\,\rangle$ message, and p measures the time I between sending the request and receiving the reply. As the delay of the $\langle\, \mathbf{time}, T\,\rangle$ message was at least $I - \delta_{\max}$ (this is because at most $\delta_{\max}$ of the I time interval was used by the request message) and at most $\delta_{\max}$, it differs at most $\delta_{\max} - I/2$ from $\frac{1}{2}I$. So, p sets its clock to $T + \frac{1}{2}I$, achieving a $\delta_{\max} - I/2$ precision; if ϵ is smaller than $\delta_{\max} - I/2$, the request is repeated.

The probability that the delay of the request (or response) message exceeds $\delta_{\max} - \epsilon$ is $1 - F(\delta_{\max} - \epsilon)$, so the probability that both delays exceed $\delta_{\max} - \epsilon$ is $[1 - F(\delta_{\max} - \epsilon)]^2$. This implies that the expected number of trials is at most $[1 - F(\delta_{\max} - \epsilon)]^{-2}$, hence the expected number of messages is at most $2.[1 - F(\delta_{\max} - \epsilon)]^{-2}$ and the expected time is at most $2\delta_{\max}.[1 - F(\delta_{\max} - \epsilon)]^{-2}$.

Exercise 15.10. No, certainly not. Assume $N - t$ processes submit the value x, and in addition p receives the values $x - \delta$ and $x + \delta$. Although at least one of these is erroneous (they differ by more than δ), neither of them is rejected because both have $N - t$ submissions that differ δ from it. This lower bounds $\mathit{width}(A_p)$ to 2δ.

Chapter 17: Stabilization

Exercise 17.1. The proof resembles that of Theorem 2.17. Consider an execution $E = (\gamma_0, \gamma_1, \ldots)$ of S. If E is finite, its last configuration is terminal and belongs to $\mathcal{L}$ by (1). Otherwise, consider the sequence $(f(\gamma_0), f(\gamma_1), \ldots)$; as there are no infinite decreasing sequences in W, there is an i such that $f(\gamma_i) > f(\gamma_{i+1})$ does not hold. By (2), $\gamma_{i+1} \in \mathcal{L}$.

Exercise 17.2. In the initial state F is at most $\frac{1}{2}N(N-1)$, and every step of process 0 increases F by at most $N-1$ (if it gives privilege to process 1). So the number of steps by processes other than 0 before the $(N+1)^{\text{th}}$ step of process 0 is bounded by $\frac{1}{2}N(N-1) + N(N-1)$, which brings the total number of steps to at most $\frac{3}{2}N(N-1) + N$.

A suitable norm function is defined as follows [DHS+94, Sec. 6.2]. Call configuration γ *single-segment* if there exists $j < N$ such that for all $i \leq j : \sigma_i = \sigma_0$ and for all $i > j : \sigma_i \neq \sigma_0$. Define

$$A(\gamma) = \begin{cases} 0 & \text{if } \gamma \text{ is single segment} \\ \text{the smallest } i \text{ s.t.} \sigma_0 + i & \\ \quad \text{does not occur in } \gamma & \text{otherwise} \end{cases}$$

and $W(\gamma)$ by $W(\gamma) = N.A(\gamma) + F(\gamma)$. For each step $\gamma \to \delta$, $[W(\gamma) > W(\delta) \ \vee \ \delta \in \mathcal{L}]$ holds.

Exercise 17.4. Modify action (1) so that the idle process p will receive the token only if p and q are oriented differently; if they are oriented the same way, the sending process becomes idle.

Action	q	p		p
(1a)	$\overrightarrow{S}$	$\overleftarrow{I}$	$\longrightarrow$	$\overrightarrow{R}$
(1b)	$\overleftarrow{I}$	$\overleftarrow{S}$	$\longrightarrow$	$\overleftarrow{I}$

Action (1a) is the same silent step as before, but action (1b) will kill the token where originally it would be forwarded to a process that already has the same orientation. This ensures that token circulation ceases, yet we are still guaranteed to have tokens as long as the ring is not legitimate (by Proposition 17.9).

Exercise 17.6. Replace the variable pref_p by a three-valued variable c_{pq} for each neighbor q, with value *you* if p has selected q, *other* if p has selected $r \neq q$, and *none* if p has not made a selection. These variables satisfy a local consistency:

$$\begin{array}{rcl} \mathit{lc}(p) & \equiv & (\forall q \in \mathit{Neigh}_p : c_{pq} = \mathit{none}) \\ & \vee & (\exists q \in \mathit{Neigh}_p : c_{pq} = \mathit{you} \ \wedge \ \forall r \neq q \in \mathit{Neigh}_p : c_{pr} = \mathit{other}). \end{array}$$

An additional "clearing" action $\mathbf{C}_p$, executed at most once for every process, establishes local consistency. Quantifications below run over Neigh_p.

```
**var** $c_{pq}$ : $(\mathit{you}, \mathit{other}, \mathit{none})$;
$\mathbf{C}_p$: $\{ \neg \mathit{lc}(p) \}$
      **forall** $q$ **do** $c_{pq} := \mathit{none}$
$\mathbf{M}_p$: $\{ \mathit{lc}(p) \wedge c_{pq} = \mathit{none} \wedge c_{qp} = \mathit{you} \}$
      **forall** $r$ **do** $c_{pr} := \mathit{other}$ ; $c_{qp} := \mathit{you}$
$\mathbf{S}_p$: $\{ \mathit{lc}(p) \wedge c_{pq} = c_{qp} = \mathit{none} \wedge \forall r \neq q : c_{rp} \neq \mathit{you}$
```

```
        **forall** $r$ **do** $c_{pr}$ := *other* ; $c_{qp}$ := *you*
**U**$_p$:  { $lc(p) \wedge c_{pq} = you \wedge c_{qp} = other$ }
        **forall** $r$ **do** $c_{pr}$ := *none*
```

Each process is locally consistent after at most one step, and the algorithm then behaves like the original one.

Can you prove a bound on length of an execution? Can you modify the algorithm so as to work for the link-register read-one model?

Exercise 17.7. In each matching step (action $\mathbf{M}_p$) the value of $c+f+w$ decreases by at least 2, and the other steps decrease the value of the second component of F. Hence the function $F = (\lfloor (c+f+w)/2 \rfloor, 2c+f)$ is also a norm, and proves stabilization within $N^2 + 2\frac{1}{2}N$ steps.

To show the lower bound we inductively construct an execution of $\frac{1}{2}k^2$ steps on a clique containing k (even) free processes; if $k = 2$ the two remaining processes are matched in two steps. If there are $k > 2$ free processes p_1 and p_2, first processes p_1 through p_{k-1} select p_k, then process p_k matches p_{k-1}, and finally processes p_1 through p_{k-2} unchain from p_k. After these $2k - 2$ steps the network contains $k - 2$ free processes, so by induction the execution can be continued for another $\frac{1}{2}(k-2)^2$ steps.

A more precise analysis is found in [Tel94]. A proof of stabilization within $O(|E|)$ steps is found in [DHS$^+$94, Section 7.2].

Exercise 17.8. Assume the state read-all model and have a boolean mis_p for each p, indicating if p is currently in the set M. There are two actions: (1) if both p and a neighbor are in M, $mis_p := false$; (2) if neither p nor any neighbor is in M, $mis_p := true$. Call p *black* if (1) applies, *gray* if (2) applies, and *white* if neither applies. Action (1) makes the black process white without making any other process black (a white neighbor could become gray) and action (2) makes a gray process white without darkening any other process. So the pair formed by the numbers of black and gray processes decreases lexicographically with every step and after less than N^2 steps a terminal configuration is reached. In a terminal configuration no two neighbors are in M, and every node not in M has a neighbor in M, hence M is a maximal independent set.

Project 17.10. An outerplanar graph has a node of degree two or less; now re-read pages 538 through 540, replacing "six" by "three" and "five" by "two".

Exercise 17.12. Let the cost of $\pi = \langle p_0, \ldots, p_k \rangle$ be the pair (p_0, k). Ordering is lexicographically, but to get length increase, paths of length N or longer are larger than paths of length below N.

Exercise 17.13. When a spanning tree is known, the size is computed by summation per subtree, i.e., each node assigns its count the sum of the counts of its children plus one. Construction of a spanning tree without a priory knowledge of the network size was solved by Dolev [Dol93].

Exercise 17.14. The update algorithm can compute the longest upward path for each node with the following choice of c, f, and the order. Let $c_p = 0$ for each process p, and let $f_{pq}(x) = x + 1$ if p is the father of q and $-\infty$ otherwise; the order is reversed, i.e., longer paths are smaller than short ones.

Appendix B: Graphs and Networks

Exercise B.1. To prove the first part, select a node v_0; because the graph is connected every node has a finite distance to v_0. Sort the remaining nodes as v_1 through v_{N-1} in increasing order of distance, i.e., $i > j \Rightarrow d(v_i,\, v_0) \leq d(v_j,\, v_0)$. For each of the v_i, $i > 0$, a shortest path from v_i to v_0 exists; the higher-numbered endpoint of the first edge is v_i; hence the number of edges is at least $N - 1$.

To show the second part we demonstrate that an acyclic graph contains at least one node with degree 1 or less; otherwise an infinite path without selfloops could be created, unavoidably hitting a node occuring earlier in the path and revealing a cycle. Then the proof is completed by induction; an acyclic graph on one node has degree 0, and removing a degree-0 or degree-1 node from an acyclic graph reduces the number of nodes, keeping the graph acyclic.

Exercise B.2. Let $G = (V,\, E)$ be an undirected graph. Assume G is bipartite; fix on a particular coloring and consider a cycle. As adjacent nodes in the cycle are also adjacent in G, they are colored differently, hence the cycle is of even length.

Now assume G has no odd-length cycles. Select an arbitrary node v_0 (or, if G is not connected: select a node in each connected component) and color node v red if $d(v_0,\, v)$ is odd and black if $d(v_0,\, v)$ is even. Let v and v' be adjacent nodes, π be any path from v_0 to v and π' any path from v' to v_0. The cycle formed by concatenating π, edge vv', and π' has even length, showing that the lengths of π and π' have different parity. In particular, $d(v_0,\, v)$ and $d(v_0,\, v')$ have different parity, and v and v' have different colors.

Exercise B.3. A ring of even length has no odd-length cycle and a tree has no cycle at all; use Exercise B.2. In the hypercube, consider a particular node labeling as in Definition B.12 and color node v red if the number of 1's in the label is even and black otherwise. As the labels of adjacent nodes differ in exactly one bit, their parity is different.

Exercise B.4. Theorem B.4 is Theorem 5.2 of Cormen *et al.* [CLR90]; I won't bother to copy the proof here.

Exercise B.6. *A clique on N nodes has $(N-1)!$ different labelings that are a sense of direction.* Select a node v_0 and observe that the $N-1$ different labels can bearranged in exactly $(N-1)!$ different ways on the edges incident to v_0.

1. Each arrangement can be extended to a (unique) sense of direction as follows. The arrangement defines $\mathcal{L}_{v_0}(w)$ for all $w \neq v_0$; set $\mathcal{L}(w,\, v_0) = N - \mathcal{L}_{v_0}(w)$ and $\mathcal{L}_u(w) = \mathcal{L}_{v_0}(w) - \mathcal{L}_{v_0}(u) \bmod N$ for $u,\, w \neq v_0$.

   To show that $\mathcal{L}$ is a sense of direction, label node v_0 with 0 and node $w \neq v_0$ with $\mathcal{L}_{v_0}(w)$; the obtained node labeling witnesses that $\mathcal{L}$ is an orientation.

   The $(N-1)!$ labelins are all different, because they already differ in their projection on v_0. Consequently, there are at least $(N-1)!$ different senses of direction.

2. Each sense of direction $\mathcal{L}$ has a different projection on v_0; indeed, $\mathcal{L}_w(v_0) = N - \mathcal{L}_{v_0}(w)$ and $\mathcal{L}_u(w) = \mathcal{L}_{v_0}(w) - \mathcal{L}_{v_0}(u) \bmod N$ can be shown by reasoning with a node labeling witnessing that $\mathcal{L}$ is a sense of direction. Consequently, there are at most $(N-1)!$ different senses of direction.

A hypercube of dimension n has n! different labelings that are a sense of direction. Select a node v_0 and observe that the n different labels can be arranged in exactly $n!$ different ways on the edges incident to v_0.

For edges $e = (u, v)$ and $f = (w, x)$, define e and f to be *parallel* iff $d(u, w) = d(v, x)$ and $d(u, x) = d(v, w)$. Reasoning with a witnessing node labeling it is shown that the labels of e and f in a sense of direction are equal if and only if e and f are parallel. Furthermore, for each edge f in the hypercube there is exactly one edge e incident to v_0 such that e and f are parallel. This shows that, like it is the case for cliques, the entire labeling is defined by the labeling of the edges of a single node.

Exercise B.7. Assume (1). By the definition of an orientation there exists a labeling $\mathcal{N}$ of the nodes such that $\mathcal{L}_u(v) = \mathcal{N}(v) - \mathcal{N}(u)$. Using induction on the path, it is easily shown that $Sum(P) = \mathcal{N}(v_k) - \mathcal{N}(v_0)$. As labels are from the set $\{1, .., N - 1\}$, $\mathcal{L}_u(v) \neq 0$, and all node labels are different. It now follows that $v_0 = v_k$ if and only if $Sum(P) = 0$.

Assume (2). Pick an arbitrary node v_0, set $\mathcal{N}(v_0) = 0$ and for all $u \neq v_0$ set $\mathcal{N}(u) = \mathcal{L}_{v_0}(u)$. It remains to show that this node labeling satisfies the constraints in the definition of an orientation. That is, for all nodes u_1 and u_2, $\mathcal{L}_{u_1}(u_2) = \mathcal{N}(u_2) - \mathcal{N}(u_1)$. For $u \neq v_0$, note $\mathcal{N}(v_0) = 0$ and $\mathcal{N}(u) = \mathcal{L}_{v_0}(u)$ so $\mathcal{L}_{v_0}(u) = \mathcal{N}(u) - \mathcal{N}(v_0)$. As $P = v_0, u, v_0$ is closed, $Sum(P) = 0$, hence $\mathcal{L}_u(v_0) = -\mathcal{L}_{v_0}(u) = \mathcal{N}(v_0) - \mathcal{N}(u)$. For $u_1, u_2 \neq v_0$, as $P = v_0, u_1, u_2, v_0$ is closed, $Sum(P) = 0$ and hence $\mathcal{L}_{u_1}(u_2) = -\mathcal{L}_{v_0}(u_1) - \mathcal{L}_{u_2}(v_0) = (\mathcal{N}(u_1) - \mathcal{N}(v_0)) - (\mathcal{N}(v_0) - \mathcal{N}(u_2)) = \mathcal{N}(u_2) - \mathcal{N}(u_1)$.

A discussion of the equivalence and similar results for the torus and hypercube are found in [Tel91].

References

[CLR90] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. *Introduction to Algorithms.* McGraw-Hill/MIT Press, 1990 (1028 pp.).

[DHS$^+$94] Dijkhuizen, G. H., Herman, T., Siemelink, W. G., Tel, G., and Verweij, A. Developments in distributed algorithms. Course notes, Dept Computer Science, Utrecht University, The Netherlands, 1994.

[Dol93] Dolev, S. Optimal time self-stabilization in dynamic systems. In proc. *Int. Workshop on Distributed Algorithms* (Lausanne, 1993), A. Schiper (ed.), vol. 725 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 160–173.

[Len94] Lenstra, A. Factoring. In proc. *8th Int. Workshop on Distributed Algorithms* (Terschelling (NL), 1994), G. Tel and P. Vitányi (eds.), vol. 857 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 28–38.

[Mat87] Mattern, F. Algorithms for distributed termination detection. *Distributed Computing* **2**, 3 (1987), 161–175.

[Mat89] Mattern, F. Virtual time and global states of distributed systems. In proc. *Parallel and Distributed Algorithms* (1989), M. Cosnard et al. (eds.), Elsevier Science Publishers, pp. 215–226.

[Tel91] Tel, G. Network orientation. Tech. rep. RUU–CS–91–8, Dept Computer Science, Utrecht University, The Netherlands, 1991. Anon. `ftp` from `ftp.cs.ruu.nl` file `/pub/RUU/CS/techreps/CS-1991/1991-08.ps.gz`.

[Tel94] Tel, G. Maximal matching stabilizes in quadratic time. *Inf. Proc. Lett.* **49**, 6 (1994), 271–272.

[Tel00] Tel, G. *Introduction to Distributed Algorithms*, 2nd edition. Cambridge University Press, 2000 (596 pp.).

[WT94] Wezel, M. C. van and Tel, G. An assertional proof of Rana's algorithm. *Inf. Proc. Lett.* **49**, 5 (1994), 227–233.