Theoretical Computer Science 412 (2011) 5580–5601

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

The Asynchronous Bounded-Cycle model☆

Peter Robinson*, Ulrich Schmid

Technische Universität Wien, Embedded Computing Systems Group (E182/2), Treitlstrasse 1-3, A-1040 Vienna, Austria

ARTICLE INFO

Keywords:
Fault-tolerant distributed algorithms
Partially synchronous models
Clock synchronization
VLSI

ABSTRACT

This paper shows how synchrony conditions can be added to the purely asynchronous model in a way that avoids any reference to message delays and computing step times, as well as system-wide constraints on execution patterns and network topology. Our Asynchronous Bounded-Cycle (ABC) model just bounds the ratio of the number of forward- and backward-oriented messages in certain ("relevant") cycles in the space–time diagram of an asynchronous execution. We show that clock synchronization and lock-step rounds can be implemented and proved correct in the ABC model, even in the presence of Byzantine failures. Furthermore, we prove that any algorithm working correctly in the partially synchronous Θ-Model also works correctly in the ABC model. In our proof, we first apply a novel method for assigning certain message delays to asynchronous executions, which is based on a variant of Farkas' theorem of linear inequalities and a non-standard cycle space of graphs. Using methods from point-set topology, we then prove that the existence of this delay assignment implies model indistinguishability for time-free safety and liveness properties. We also introduce several weaker variants of the ABC model, and relate our model to the existing partially synchronous system models, in particular, the classic models of Dwork, Lynch and Stockmayer and the query–response model by Mostefaoui, Mourgaya, and Raynal. Finally, we discuss some aspects of the ABC model's applicability in real systems, in particular, in the context of VLSI Systems-on-Chip.

1. Introduction

Adding synchrony conditions, relating the occurrence times of certain events in a distributed system to each other, is the "classic" approach for circumventing impossibility results as (Fischer et al. [23]) in fault-tolerant distributed computing. The following models in between synchrony and asynchrony, which are all sufficiently strong for solving the pivotal consensus problem, have been proposed in the literature:

(1) The Archimedean model by Vitányi [38] bounds the ratio between maximum end-to-end delays and minimal computing step times.
(2) The classic partially synchronous models by Dwork et al. [17], Chandra and Toueg [11] and the semi-synchronous models of Ponzio and Strong [36], Attiya et al. [7] bound message delays as well as the ratio of minimal and maximal computing step times.
(3) The MCM model by Fetzer [20] assumes that all the received messages are correctly classified as "slow" or "fast", depending on the message delays.

☆ This research is supported by the Austrian Science Foundation (FWF) projects P17757 and P20529. A preliminary version of this paper (Robinson and Schmid (2008)) received the best paper award at SSS'08.

* Corresponding author. Tel.: +43 158801 18253; fax: +43 158801 18297.
E-mail addresses: robinson@ecs.tuwien.ac.at (P. Robinson), s@ecs.tuwien.ac.at (U. Schmid).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.08.001

(4) The Θ-Model by Le Lann and Schmid [29], Widder et al. [41] and Widder and Schmid [40] bounds the ratio between the maximal and minimal end-to-end delay of messages simultaneously in transit.
(5) The FAR Model by Fetzer et al. [22] assumes lower bounded computing step times and message delays with finite average.
(6) The Weak Timely Link (WTL) models of Aguilera et al. [4], Malkhi et al. [30] and Hutle et al. [26] assume that only messages sent via certain links have bounded end-to-end delay.
(7) The MMR model by Mostefaoui et al. [35] suggested for implementing failure detectors in systems with process crashes, which assumes certain order properties for round-trip responses.

All these models, except (3) and (7), refer to individual message delays and/or computing step times, and most of them involve explicit time bounds and constraints that must hold within the entire system.

This paper shows how to add synchrony assumptions – sufficiently strong for implementing lock-step rounds, and hence for solving many important distributed computing problems – to the asynchronous model in a way that

(1) entirely avoids any reference to message delays and computing step times, and
(2) does not require system-wide constraints on potential computing step and communication patterns and network topology.

More specifically, our *Asynchronous Bounded-Cycle* (ABC) model bounds the ratio of the *number* of forward and backward messages in certain "relevant" cycles in the space–time diagram of an asynchronous execution only. Intuitively speaking, there is only one scenario that is admissible in the purely asynchronous model but not in the ABC model: A chain C_1 of k_1 consecutive messages, starting at process q and ending at p, that properly "spans" (i.e., covers w.r.t. real time; see Fig. 1) another causal chain C_2 from q to p involving $k_2 \geqslant k_1\Xi$ messages, for some model parameter $\Xi > 1$.

Consequently, individual message delays can be arbitrary, ranging from 0 to any finite value; they may even continuously increase. There is no relation at all between computing step times and/or message delays at processes that do not exchange messages; this also includes purely one-way communication ("isolated chains"). For processes that do exchange messages, message delays and step times in non-relevant cycles and isolated chains can also be arbitrary. Only *cumulative* delays of chains C_1 and C_2 in *relevant* cycles must yield the event order as shown in Fig. 1. That is, the *sum* of the message delays along C_2 must not become so small that C_1 could span $k_1\Xi$ or more messages in C_2. Note carefully that this is not just a static system-wide condition that is imposed by the system model, but also depends on the message pattern created by an algorithm. Nevertheless, ABC algorithms can exploit it for "timing out" relevant message chains, and hence for failure detection.

Besides introducing the ABC model in Section 2, our paper provides the following contributions:

- In Section 3, we provide and prove correct a simple Byzantine fault-tolerant clock synchronization algorithm and a lock-step round simulation for the ABC model.
- In Section 4, we prove that all message-driven algorithms designed and proved correct for the Θ-Model also work correctly in the ABC model, despite the fact that most ABC executions are not admissible in the Θ-Model. In the quite involved proofs, we use methods from linear algebra and point-set topology, some of which may also be applicable in other contexts as well.
- In Section 5, we relate the ABC model to the existing partially synchronous models, and show that it is strictly weaker in terms of synchrony. We also provide a short discussion of some practical aspects, in the context of VLSI systems-on-chip.
- In Section 6 we introduce some weaker variants of the ABC model, including unknown and/or eventual model parameters.

Some conclusions and directions of further research in Section 7 eventually complete our paper.

2. The ABC model

We consider a system of n distributed processes, connected by a (not necessarily fully connected) point-to-point network with finite but unbounded message delays. We neither assume FIFO communication channels nor an authentication service, but we do assume that processes know the sender of a received message.

Every process executes an instance of a distributed algorithm and is modeled as a state machine. Its local execution consists of a sequence of atomic, zero-time computing steps, each involving the reception of exactly one[1] message, a state transition, and the sending of zero or more messages to a subset of the processes in the system. Since the ABC model is entirely time-free, i.e., does not introduce any time-related bounds, we restrict our attention to message-driven algorithms, cp. [8,9,31]: Computing steps at process p are exclusively triggered by a single incoming message at p, with an external "wake-up message" initiating p's very first computing step; we assume that this very first step occurs before any message from another process is received.

Among the n processes, at most f may be Byzantine faulty. A faulty process may deviate arbitrarily from the behavior of correct processes as described above[2]; it may of course just crash as well, in which case it possibly fails to complete some

[1] An algorithm cannot learn anything from receiving multiple asynchronous messages at the same time, cp. [16].
[2] More specifically, we do not assume that Byzantine processes or messages sent by such processes adhere to any synchrony requirements.

computing step and does not take further steps later on. In order to properly capture the interaction of correct and faulty processes in the ABC model, we conceptually distinguish the receive event that triggers a computing step and the computing step itself. In case of a correct process, both occur at the same time. In case of a faulty receiver process, however, we separate the reception of a message, which is not under the receiver's control but initiated by the network, from the processing of this message, which is under the receiver's control and hence arbitrary in case of a faulty receiver. Consequently, even faulty processes eventually receive messages sent by correct processes, and since processes can only receive one message per step, there is a total order on the receive events at every process. We say that a message m sent by process p has been *processed* (or *executed*) by the correct process q, if a computing step triggered by m has been executed by q. Similarly, we say that m has been received by the (correct or faulty) process r, if a receive event for m has occurred at r.

We can now specify admissible executions for our asynchronous message-driven system, cp. [9]:

1. If an infinite number of messages are sent to a correct process, it executes infinitely many computing steps.
2. Every message sent by a correct process is received by every (correct or faulty) recipient within finite time.

Note that we do not say anything about messages sent by faulty processes here, which are usually unconstrained anyway.

The ABC model just puts one additional constraint on admissible executions. It is based on the space–time diagram representing the happens-before relation introduced in Lamport [28], which captures the causal flow of information in an admissible execution α. In order to properly include faulty processes, we just drop every message sent by a faulty process (along with both its send step and its receive event + step) in the space–time diagram. Note that a similar message dropping can be used for exempting certain messages, say, of some specific type or sent/received by some specific processes, from the ABC synchrony condition. Note that it is the algorithm, not the ABC model, that determines whether the order of certain receive events matters.

Definition 1 (*Execution Graph*)**.** The *execution graph* G_α is the digraph corresponding to the space–time diagram of an admissible execution α, with nodes $V(G_\alpha) = \Phi$ corresponding to the receive events[3] in α, and edges reflecting the happens-before relation without its transitive closure: (ϕ_i, ϕ_j) is in the edge relation $\rightarrow_\alpha : \Phi \times \Phi$ if and only if one of the following two conditions holds:

1. The receive event ϕ_i triggers a computing step where a message m is sent from correct process p to process q; event ϕ_j is the receive event of m at q. We call the edge $\phi_i \rightarrow_\alpha \phi_j$ *non-local edge* or simply *message* in G_α.
2. The events ϕ_i and ϕ_j both take place at the same processor p and there exists no event ϕ_k in α occurring at p with $i < k < j$. The edge $\phi_i \rightarrow_\alpha \phi_j$ is said to be a *local edge*.

We will simply write G and $\rightarrow$ instead of G_α and $\rightarrow_\alpha$ when α is clear from the context. Note that we will also consider execution graphs of finite prefixes of executions in Section 4.

Definition 2 (*Causal Chain and Cycle*)**.** A *causal chain* $\phi_1 \rightarrow \cdots \rightarrow \phi_l$ is a directed path in the execution graph, which consists of messages and local edges. The *length of a causal chain* D is the number of non-local edges (i.e., messages) in D, denoted by $|D|$. A *cycle* Z in G is a subgraph of G that corresponds to a cycle in the undirected shadow graph $\hat{G}$ of G.

Since messages cannot be sent backwards in time, every cycle can be decomposed into at least 2 causal chains having opposite directions. We now take a closer look at such cycles, which capture all causal information relevant for ABC algorithms.

Definition 3 (*Relevant Cycles*)**.** Let Z be a cycle in the execution graph, and partition the edges of Z into the *backward* edges $\hat{Z}^-$ and the *forward* edges $\hat{Z}^+$ as follows: Identically directed edges are in the same class, and

$$|Z^+| \leqslant |Z^-|, \tag{1}$$

where $Z^- \subseteq \hat{Z}^-$ and $Z^+ \subseteq \hat{Z}^+$ are the restrictions of $\hat{Z}^-$ resp. $\hat{Z}^+$ to non-local edges (messages). The *orientation* of the cycle Z is the direction of the forward edges Z^+, and Z is said to be *relevant* if all local edges are backward edges, i.e., if $\hat{Z}^+ = Z^+$; otherwise it is called *non-relevant*.

Fig. 1 shows an example of a relevant cycle: Its orientation is opposite to the direction of all local edges, and the backward messages are traversed oppositely w.r.t. their direction when traversing the cycle according to its orientation. In Fig. 4 on the other hand, the order of events ϕ and ψ make the cycle N non-relevant. Bear in mind, however, that labeling the edges in a cycle as forward and backward is only of local significance. For example, in Fig. 2, the forward message e in cycle X is actually a backward one in cycle Y (i.e., $e \in X^+$ and $e \in Y^-$).

Definition 4 (*ABC Synchrony Condition*)**.** Let Ξ be a given rational number $\Xi > 1$, and let G be the execution graph of an execution α. Then α is *admissible in the ABC model* if, for every relevant cycle Z in G, we have

$$\frac{|Z^-|}{|Z^+|} < \Xi. \tag{2}$$

[3] Recall that we consider message-driven algorithms with zero-time atomic receive + compute + send steps only; every send event hence corresponds to some receive event.

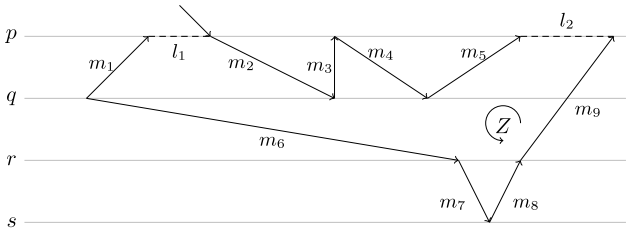

Fig. 1. A relevant cycle Z, where a causal chain $C_2 = m_1 l_1 m_2 \ldots m_5 l_2$ is spanned by the "slow" chain $C_1 = m_6 m_7 m_8 m_9$. Message m_3 has zero delay.

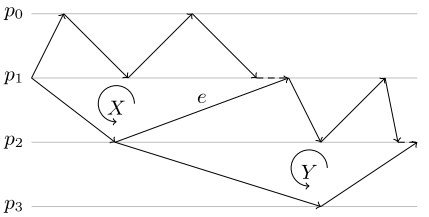

Fig. 2. An execution graph containing relevant cycles X, Y, and the combined cycle $X \oplus Y$, consisting of all edges except the oppositely oriented edge e.

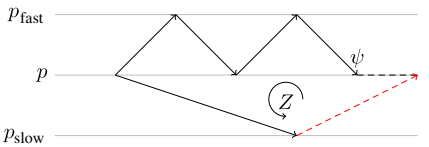

Fig. 3. If a reply message arrived from p_{slow} after event ψ, there would be a relevant cycle Z where $\frac{|Z^-|}{|Z^+|} = \frac{4}{2}$.

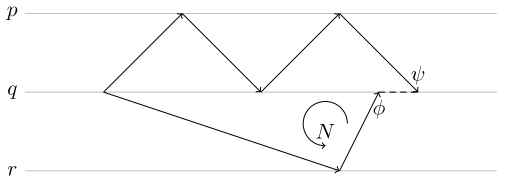

Fig. 4. Example of a non-relevant cycle N.

Note carefully that, compared to the purely asynchronous model, there is no other constraint in the ABC model: Only the ratio of the *number* of backward vs. forward messages in relevant cycles is constrained. There is no system-wide assumption that restricts the behavior of processes that do not communicate with each other, no delay constraints whatsoever are put on individual messages, and messages in non-relevant cycles and isolated chains are totally unconstrained. Nevertheless, in Section 3, we will prove that the ABC synchrony condition is sufficient for simulating lock-step rounds, and hence for solving e.g. consensus by means of any synchronous consensus algorithm.

Informally, this is true because (2) facilitates "timing out" message chains and hence failure detection[4]: For example, a correct process p could use its knowledge of Ξ to timeout a crashed process p_{slow}, by communicating in a ping-pong-like manner[5] with a correct process p_{fast}. Suppose that, as depicted in Fig. 3, p has initially broadcast a message to p_{slow} and p_{fast}. Assume that, after $\Xi = 2$ ping-pong sequences (i.e., a causal chain of length 2Ξ) between p and p_{fast}, no reply message from p_{slow} has yet arrived at p. If this reply message arrived at p at some point later on, then the receive event of this message would close a relevant cycle and thereby violate the synchrony assumption (2). Hence, in the computing step ψ, process p can safely conclude that p_{slow} has crashed. Note that the ABC synchrony condition is used indirectly here: The absence of a reply message allows p to timeout p_{slow}, because its later arrival would violate the ABC synchrony condition.

It is instructive to consider what happens if the message from p_{slow} arrives before the event ψ occurs, as shown in Fig. 4. In that case, ψ closes a non-relevant cycle—the local edge (ϕ, ψ) between the receive events ϕ and ψ has the same direction as the orientation of the cycle N. In sharp contrast to the situation depicted in Fig. 3, however, p does not gain any new information in the computing step ψ closing the resulting non-relevant cycle: Process p has already inferred that p_{slow} is still alive in the previous computing step ϕ (which actually closes a smaller relevant cycle). Therefore, non-relevant cycles are indeed non-relevant for an ABC algorithm.

At a first glance, the above examples suggest that we could simplify the ABC synchrony condition by stating certain order properties on fast resp. slow message chains, i.e., by considering relevant cycles that consist of exactly one fast and one

[4] Note that we consider only crash-faulty processes rather than Byzantine behavior in this example.

[5] Process p sends a message to p_{fast} which immediately sends back a reply. This in turn causes p to send another message to p_{fast} and so forth.

slow message chain only. However, this would unnecessarily restrict the way algorithms could exploit the ABC synchrony condition: It may well be the case that a clever algorithm could infer synchrony information from more complex message patterns. Moreover, in case of Byzantine failures, there can be cycles involving multiple overlapping message chains. For example, the correctness proof of Algorithm 1 shown in Fig. 5 would no longer work with this simplified assumption, as the cycle consists of 3 separate message chains: $\phi_0 \to \psi_2$, $\psi_1 \to \phi''$, and $\phi_0 \to \phi''$. This makes it necessary to state Definition 4 in terms of general cycles, as we did, rather than via causal chains.

Weaker variants of the ABC model, which comprise the ?ABC model (unknown Ξ), the $\diamond$ABC model (eventually holding Ξ), and the $\diamond$?ABC model (eventually holding unknown Ξ) defined analogously to Dwork et al. [17] and Widder and Schmid [40], as well as ABC models defined on even more restricted execution graphs (where (2) needs to hold for certain relevant cycles only, analogously to the WTL models of Aguilera et al. [4,30,26]) will be introduced in Section 6.

3. Clock synchronization in the ABC model

In this section, we show that the simple fault-tolerant tick generation algorithm introduced in [40] can be used for clock synchronization in the ABC model. It tolerates up to f Byzantine process failures in a system consisting of a fully connected network of $n \geqslant 3f + 1$ processes adhering to the ABC model.

Algorithm 1 Byzantine Clock Synchronization

```
1: VAR k: integer ← 0;
2: send (tick 0) to all [once];

   /* catch-up rule */
3: if received (tick l) from f + 1 distinct processes and l > k then
4:    send (tick k + 1), ...,(tick l) to all [once];
5:    k ← l;
   /* advance rule */
6: if received (tick k) from n − f distinct processes then
7:    send (tick k + 1) to all [once];
8:    k ← k + 1;
```

In Algorithm 1, every process p maintains a local variable k that constitutes p's local clock as follows: Every process initially sets $k \leftarrow 0$ and broadcasts the message (*tick* 0); for simplicity, we assume that a process sends messages also to itself. If a correct process p receives $f + 1$ (*tick* ℓ) messages (catch-up rule, line 3), it can be sure that at least one of them was sent by a correct process that has already reached clock value l. Therefore, p can safely catch-up to l and broadcast (*tick* $k + 1$), ..., (*tick* l). If some process p receives $n - f \geqslant 2f + 1$ (*tick* k) messages (advance rule, line 6) and thus advances its clock to $k + 1$, it follows that at least $f + 1$ of those messages will also be received by every other correct process, which then executes line 3. Hence, all correct processes will eventually receive $n - f$ (*tick* k) messages and advance their clock to $k + 1$.

We will now prove that the algorithm guarantees progress of clocks and a certain synchrony condition, which can be stated in terms of consistent cuts in the execution graph. Note that using causality as a reference – rather than a common point in time, as in traditional clock synchronization – is natural in the time-free ABC model. Since the classic definition of consistent cuts does not take faulty processes into account, we will use the following correct-restricted version tailored to our execution graphs:

Definition 5 (*Consistent Cut*)**.** Let G be an execution graph and denote by $\xrightarrow{*}$ the reflexive and transitive closure of the edge relation $\to$. A subset $\mathcal{S}$ of events in G is called *consistent cut*, if (1) for every correct process p, there is an event $\phi \in \mathcal{S}$ taking place at p, and (2) the set $\mathcal{S}$ is left closed for $\xrightarrow{*}$; i.e., $\mathcal{S}$ contains the whole causal past of all events in $\mathcal{S}$.

Given an event ϕ_p at process p, we denote by $C_p(\phi_p)$ the clock value *after* executing the computing step corresponding to ϕ_p. Recall that the latter need not be correctly executed if p is faulty. The clock value of a (correct) process p in the frontier of a consistent cut $\mathcal{S}$ is denoted by $C_p(\mathcal{S})$; it is the last clock value of p w.r.t. $\xrightarrow{*}$ in $\mathcal{S}$. Since it follows immediately from the code of Algorithm 1 that local clock values of correct processes are monotonically increasing, $C_p(\mathcal{S})$ is the maximum clock value at p over all events $\phi_p \in \mathcal{S}$.

We will first show that correct clocks make progress perpetually.

Lemma 1 (*One Step Progress*)**.** *Let $\mathcal{S}$ be a consistent cut such that all correct processes p satisfy $C_p(\mathcal{S}) \geqslant k$, for a fixed $k \geqslant 0$. Then there is a consistent cut $\mathcal{S}'$ where every correct process has set its clock to at least $k + 1$.*

Proof. If all correct processes p_i have a (possibly distinct) clock value $k_i \geqslant k$ in the frontier of $\mathcal{S}$, the code of Algorithm 1 ensures that they have already sent (*tick* k). Since all messages in transit are eventually delivered, there must be a (not necessarily consistent) cut $\mathcal{S}''$, in the frontier of which every correct process has received $n - f$ tick k messages and thus set its clock to $k + 1$. The left closure of $\mathcal{S}''$ yields the sought consistent cut $\mathcal{S}'$. □

Theorem 1 (*Progress*). *In every admissible execution of Algorithm 1 in a system with $n \geqslant 3f+1$ processes, the clock of every correct process progresses without bound.*

Proof. The theorem follows from a trivial induction argument using Lemma 1, in conjunction with the fact that the cut $\mathcal{S}^0$ comprising the initial event ϕ_p^0 of every process p is trivially consistent and satisfies $C_p(\mathcal{S}^0) \geqslant 0$. □

Lemma 2 (*First Advance*). *If a correct process q sets its clock to $k \geqslant 1$ in event ψ_q, then there is a correct process p that sets its clock to k using the advance rule in some event ψ_p with $\psi_p \xrightarrow{*} \psi_q$.*

Proof. If q uses the advance rule for setting its clock to k in ψ_q, the lemma is trivially true. If q uses the catch-up rule instead, it must have received $f+1$ (*tick* k) messages, at least one of which was sent by a correct process q' in an event $\psi_{q'} \xrightarrow{*} \psi_q$. If q' also sent its (*tick* k) via the catch-up rule (line 3), we apply the same reasoning to q'. Since every process sends (*tick* k) only once and there are only finitely many processes, we must eventually reach a correct process p that sends (*tick* k) in event $\psi_p \xrightarrow{*} \psi_q$ via the advance rule. □

Lemma 3 (*Causal Chain Length*). *Assume that a correct process sets its clock to $k+m$, for some $k \geqslant 0$, $m \geqslant 0$, at some event ϕ', or has already done so. Then, there is a causal chain D of length $|D| \geqslant m$ involving correct processes only, which ends at ϕ' and starts at some event ϕ where a correct process sets its clock to k using the advance rule ($k \geqslant 1$) or the initialization rule ($k = 0$).*

Proof. Let p be the process where ϕ' occurs. If p has set its clock in some earlier computing step $\phi''' \xrightarrow{*} \phi'$, we just replace ϕ' by ϕ''' and continue with the case where p sets its clock to $k+m$ in ϕ'. If p sets its clock in ϕ' using the catch-up rule, applying Lemma 2 yields a correct process that sets its clock to $k+m$ in an event $\psi \xrightarrow{*} \phi'$ using the advance rule. To prove Lemma 3, it hence suffices to assume that p sets its clock to $k+m$ in ϕ' via the advance rule ($k+m \geqslant 1$) or the initialization rule ($k+m=0$), as we can append the chains cut before to finally get the sought causal chain D.

The proof is by induction on m. For $m=0$, the lemma is trivially true. For $m > 0$, at least $n-2f \geqslant f+1$ correct processes must have sent (*tick* $k+m-1$). Let q be any such process, and ϕ'' be the event in which (*tick* $k+m-1$) is sent. Since q also sets its clock to $k+m-1$ at ϕ'', we can invoke Lemma 2 in case $k+m-1 \geqslant 1$ to assure that the advance rule is used in ϕ''; for $k+m-1=0$, the initialization rule is used in ϕ''. We can hence apply the induction hypothesis and conclude that there is a causal chain D' of length at least $m-1$ leading to ϕ''. Hence, appending q's (*tick* $k+m-1$) message [and the initially cut off chains] to D' provides D with $|D| \geqslant m$. □

The following Lemma 4 will be instrumental in our proof that Algorithm 1 maintains synchronized clocks. It reveals that when a correct process p updates its clock value in some event ϕ', then all messages of correct processes of a certain lower tick value must have already been received by p, i.e., must originate from the causal cone of ϕ'.

Lemma 4 (*Causal Cone*). *For some $k \geqslant 0$, suppose that $C_p(\phi') = k + 2\Xi$ at the event ϕ' of a correct process p. Then, for every $0 \leqslant \ell \leqslant k$, process p has already received (tick ℓ) from every correct process.*

Proof. The general proof idea is to show that the arrival of (*tick* ℓ) in some event ϕ'' after ϕ' would close a relevant cycle in which the synchrony assumption (2) is violated. See Fig. 5 for a graphical representation of the scenario described below.

Let $C_p(\phi') = k + 2\Xi$ and assume, for the sake of contradiction, that (*tick* ℓ) from some correct process q was not yet received by p before or at ϕ', for some $\ell \leqslant k$. Consider the last message that p received from q before (or at) ϕ'. If such a message exists, we denote its send event at q as ψ'; otherwise, we simply define ψ' to be the (externally triggered) initial computing step at q.

From Lemma 3, we know that there is a causal chain $D = \phi_1' \to \cdots \to \phi'$ of length $|D| \geqslant k + 2\Xi - (\ell+1)$, where a (*tick* $\ell+1$) message is sent in ϕ_1' by some correct process p_1 via the advance rule and, by assumption, $C_p(\phi') = k + 2\Xi$. Since ϕ_1' executes the advance rule, p_1 must have received $n-f$ (*tick* ℓ) messages to do so. Denoting by $0 \leqslant f' \leqslant f$ the actual number of faulty processes among the $n \geqslant 3f+1$ ones, it follows that $n-f-f' \geqslant f+1$ of these messages were sent by correct processes; we denote this set by P_1.

Since Theorem 1 ensures progress of all correct processes, there must be an event ψ_1, coinciding with or occurring after ψ', in which q broadcasts (*tick* ℓ). Eventually, this message is received by p in some event ϕ'', which must be after ϕ' since by assumption (*tick* ℓ) was not received before (or at) ϕ'. Furthermore, we claim that q receives at least $n-f'-f$ (*tick* ℓ) messages from correct processes after (or at) event ψ_1; let P_2 be that set. Otherwise, q would have received at least

$$n - f' - (n - f' - f) + 1 = f + 1$$

(*tick* ℓ) messages from correct processes by some event $\psi_1' \xrightarrow{*} \psi_1$, and therefore would have broadcast (*tick* ℓ) already in ψ_1' according to the catch-up rule.

Since $P_1 \cup P_2$ is of size at most $2n - 2f' - 2f$ and we have only $n - f'$ correct processes, it follows by the pigeonhole principle that

$$2n - 2f' - 2f - (n - f') = n - 2f - f' \geqslant n - 3f > 0$$

correct processes are in $P_1 \cap P_2$. Choose any process $p_0 \in P_1 \cap P_2$, which broadcasts its (*tick* ℓ) in some event ϕ_0. This message is received at q in some event ψ_2, and at p_1 in event ϕ_1.

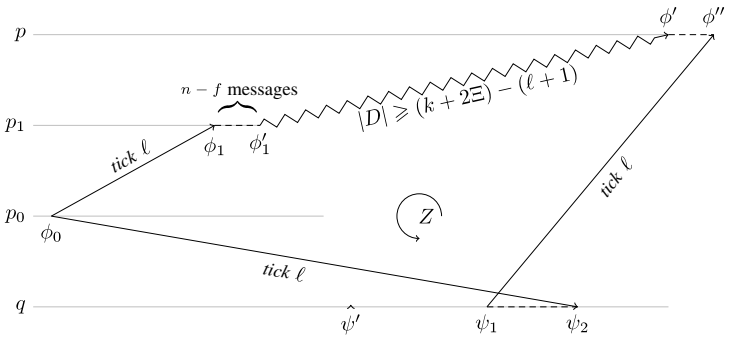

Fig. 5. Proof of Lemma 4.

It is immediately apparent from Fig. 5 that the causal chains $\phi_0 \to \phi_1 \overset{*}{\to} D \overset{*}{\to} \phi''$, $\phi_0 \to \psi_2$, $\psi_1 \overset{*}{\to} \psi_2$, and $\psi_1 \to \phi''$ form a relevant cycle Z: The number of backward messages is

$$|Z^-| = |D| + 1 \geqslant k - \ell + 2\Xi \geqslant 2\Xi,$$

since $\ell \leqslant k$; the number of forward messages $|Z^+|$ is 2. But this yields $\frac{|Z^-|}{|Z^+|} \geqslant \frac{2\Xi}{2} = \Xi$, contradicting the ABC synchrony assumption (2). □

We can now easily prove that Algorithm 1 maintains the following synchrony condition:

Theorem 2 (*Synchrony*)**.** *For any consistent cut $\mathcal{S}$ in an admissible execution of Algorithm 1 in a system with $n \geqslant 3f+1$ processes, we have $|C_p(\mathcal{S}) - C_q(\mathcal{S})| \leqslant 2\Xi$ for all correct processes p, q.*

Proof. Assume that the maximum clock value in the frontier of $\mathcal{S}$ is $k + 2\Xi$, and let p be a correct process with $C_p(\mathcal{S}) = k + 2\Xi$. From Lemma 4, we know that p must have seen (*tick* ℓ) from every correct process q for any $\ell \leqslant k$. Since $\mathcal{S}$ is consistent, all the corresponding send events at q must be within $\mathcal{S}$, such that $C_q(\mathcal{S}) \geqslant k$. □

Even though the ABC model is entirely time-free, we can immediately transfer the above synchrony property to real-time cuts according to Mattern [32], to derive the following theorem:

Theorem 3 (*Clock Precision*)**.** *Let $C_p(t)$ denote the clock value of process p at real-time t. For any time t of an admissible execution of Algorithm 1 in a system with $n \geqslant 3f+1$ processes, we have $|C_p(t) - C_q(t)| \leqslant 2\Xi$ for all correct processes p, q.*

We have already seen that in Algorithm 1 clocks make progress perpetually and remain synchronized while doing so. However, *precision* and *progress* define a fairly weak version of the clock synchronization problem only. In our setting, for example, one could easily simulate optimal precision clock synchronization (i.e., a precision of 1) by introducing an artificial "macro"-clock, which ticks once every 2Ξ ticks of our "micro"-clock C_p.[6] Clearly, 2Ξ is just a lower bound on the number of "micro"-clock ticks X per "macro"-clock tick here; optimal precision of the "macro"-clocks would also be maintained if we increased X with every "macro"-clock tick throughout the execution.[7] Clearly, X would grow without bound in such a simulation, which makes it apparent that neither precision nor progress constrains the actual rate of progress of a clock synchronization algorithm.

In classic clock synchronization, this is usually enforced by means of some *linear envelope condition*, which asserts a linear relation between clock time and real time and thereby also rules out "degenerated" solutions [15]. In our asynchronous setting, we obviously cannot refer to real time, but what we can do is to relate the rate of progress of the fastest and the slowest correct clock to each other. We will show that our algorithm also satisfies a suitably defined *bounded progress condition* (see Definition 7) based on consistent cuts.

Definition 6 (*Consistent Cut Interval*)**.** Let ϕ be an event at a correct process and denote the left closure w.r.t. the causality relation $\to$ as $\langle\phi\rangle$. If ψ is an event such that $\phi \to \psi$, we define the *consistent cut interval* as $[\langle\phi\rangle, \langle\psi\rangle] := \langle\psi\rangle \setminus \langle\phi\rangle$.

Note that, when considering the real-time transformation of Mattern [32], a consistent cut interval can essentially be seen as a real-time interval.

Since the ABC model is a message-driven model, we only care about the rate of progress of certain *distinguished events* that affect the message complexity, i.e., we do not want to include events where messages are only received but not sent.

Definition 7 (*Bounded Progress Condition*)**.** An algorithm A satisfies a *bounded progress* ϱ for some (set of) distinguished events *iff* the following holds true in all admissible executions of A: Whenever a correct process p performs at least $\varrho > 0$ distinguished events in a consistent cut interval $[\langle\phi_p\rangle, \langle\phi'_p\rangle]$, every correct process performs at least one distinguished event in $[\langle\phi_p\rangle, \langle\phi'_p\rangle]$.

[6] In fact, this is the main idea of the lock-step round simulation Algorithm 2 below.

[7] This can be used to simulate eventual lock-step rounds in the weaker variants of the ABC model introduced in Section 6, cp. [40].

Theorem 4. *Algorithm 1 satisfies the bounded progress $\varrho = 4\Xi+1$ for the distinguished event that represents clock incrementing and message broadcasting (=send to all).*

Proof. From the code of Algorithm 1, it is apparent that incrementing the clock value and broadcasting messages is done in the same step. Let the distinguished events considered here be exactly those steps. Suppose that a correct process p has performed at least $4\Xi+1$ distinguished events in between events ϕ_p and ϕ'_p, i.e., in the cut interval $[\langle\phi_p\rangle, \langle\phi'_p\rangle]$. Furthermore, assume in contradiction that there is a correct process q that does not perform any distinguished event in $[\langle\phi_p\rangle, \langle\phi'_p\rangle]$. Assuming that $C_p(\langle\phi_p\rangle) = k$, for some $k \geqslant 0$, it follows that $C_p(\langle\phi'_p\rangle) \geqslant k + 4\Xi + 1$. By assumption, q does not perform a distinguished event in $[\langle\phi_p\rangle, \langle\phi'_p\rangle]$, hence $C_q(\langle\phi_p\rangle) = C_q(\langle\phi'_p\rangle)$.

We distinguish two cases for the number of distinguished events, i.e., the clock values, in event ϕ_p:

1. $C_p(\langle\phi_p\rangle) > C_q(\langle\phi_p\rangle)$: Since $C_q(\langle\phi_p\rangle) = C_q(\langle\phi'_p\rangle)$, we immediately arrive at a contradiction to Theorem 2.
2. $C_p(\langle\phi_p\rangle) \leqslant C_q(\langle\phi_p\rangle)$: We have

$$\begin{aligned} C_p(\langle\phi'_p\rangle) - C_q(\langle\phi'_p\rangle) &\geqslant k + 4\Xi + 1 - C_q(\langle\phi'_p\rangle) = C_p(\langle\phi_p\rangle) - C_q(\langle\phi'_p\rangle) + 4\Xi + 1 \\ &\geqslant C_p(\langle\phi_p\rangle) - C_q(\langle\phi_p\rangle) + 4\Xi + 1. \end{aligned}$$

Applying Theorem 2 to $C_p(\langle\phi_p\rangle) - C_q(\langle\phi_p\rangle$ yields

$$C_p(\langle\phi'_p\rangle) - C_q(\langle\phi'_p\rangle) \geqslant -2\Xi + 4\Xi + 1 = 2\Xi + 1,$$

contradicting Theorem 2 for $\langle\phi'_p\rangle$. □

Finally, we will show how to build a lock-step round simulation in the ABC model atop of Algorithm 1. A lock-step round execution proceeds in a sequence of rounds $r = 1, 2, \ldots$, where all correct processes take their round r computing steps (consisting of receiving the round $r-1$ messages,[8] executing a state transition, and broadcasting the round r messages for the next round) exactly at the same time.

Algorithm 2 Lock-Step Round Simulation

```
1:  VAR r: integer ← 0;
2:  call start(0);

3:  Whenever k is updated do
4:  if k/(2Ξ) = r + 1 then
5:      r ← r + 1
6:      call start(r)

7:  procedure start(r:integer)
8:      if r > 0 then
9:          read round r − 1 messages
10:         execute round r computation
11:     send round r messages
```

We use the same simulation as in Widder and Schmid [40], which just considers clocks as phase counters and introduces rounds consisting of 2Ξ phases. Algorithm 2 shows the code that must be merged with Algorithm 1; the round r messages are piggybacked on (*tick* k) messages every 2Ξ phases, namely, when $k/(2\Xi) = r$. The round r computing step[9] is encapsulated in the function start(r) in line 7; start(0) just sends the round 0 messages that will be processed in the round 1 computing step.

To prove that this algorithm achieves lock-step rounds, we need to show that all round r messages from correct processes have arrived at every correct process p before p enters round $r+1$, i.e., executes start($r+1$).

Theorem 5 (*Lock-Step Rounds*)**.** *In a system with $n \geqslant 3f+1$ processes, Algorithm 2 merged with Algorithm 1 correctly simulates lock-step rounds in the ABC model.*

Proof. Suppose that a correct process p starts round $r+1$ in event ϕ. By the code, $C_p(\phi) = k$ with $k/(2\Xi) = r+1$, i.e., $k = 2\Xi r + 2\Xi$. By way of contradiction, assume that the round r message, sent by some correct process q in the event ψ, arrives at p only after ϕ. By the code, $C_q(\psi) = k'$ with $k'/(2\Xi) = r$, i.e., $k' = 2\Xi r$. However, Lemma 4 reveals that p should have already seen (*tick* $2\Xi r$) from q before event ϕ, a contradiction. □

[8] For notational convenience, we enumerate the messages with the index of the previous round.

[9] Note that we assume that computing steps happen atomically in zero time.

We note that the above proof(s) actually establish *uniform* (cp., [25]) lock-step rounds, i.e., lock-step rounds that are also obeyed by faulty processes until they behave erroneously for the first time: If the messages sent by faulty processes also obey the ABC synchrony condition (2), then the proof of the key Lemma 4 actually establishes a uniform causal cone property: Assuming that (i) process q performs correctly up to and including at least one more step after event ψ', and (ii) p works correctly up to and including event ϕ', then p would receive (though not necessarily process) the message from q in ϕ'', thereby closing a relevant cycle that violates Ξ. Hence, p must have received all messages from its causal cone by ϕ' already, which carries over to a uniform version of Theorem 5.

4. Model indistinguishability

In this section, we will develop a non-trivial model indistinguishability argument[10] in order to show that any algorithm designed for the Θ-Model by Le Lann and Schmid [29], Widder et al. [41] and Widder and Schmid [40] also works correctly in the ABC model. It is non-trivial, since there are (many) admissible ABC executions which are *not* admissible in the Θ-Model. Nevertheless, no simulation will be involved in our argument; the original algorithms can just be used "as is" in the ABC model, without sacrificing performance. This "timing invariance" of algorithms and their properties confirms again that timing constraints are not really essential for solving certain distributed computing problems.

More specifically, provided that $\Xi < \Theta$, we will show that every algorithm designed and proved correct for the Θ-Model[11] will also work in the corresponding ABC model.

Like the ABC model, the Θ-Model is a message-driven model, without real-time clocks, and hence also relies on end-to-end delays. If $\tau^+(t)$ resp. $\tau^-(t)$ denotes the (unknown) maximum resp. minimum delay of all messages in transit system-wide between correct processes at time t in an execution, it just assumes that there is some $\Theta > 1$ such that

$$\frac{\tau^+(t)}{\tau^-(t)} \leqslant \Theta \tag{3}$$

at all times t, in any admissible execution. In the simple static Θ-Model (which is sufficient for our model indistinguishability argument, since it has been shown in [40] to be equivalent to the general Θ-Model from the point of view of algorithms), it is assumed that there are (unknown) upper resp. lower bounds $\infty > \tau^+ \geqslant \tau^+(t)$ resp. $0 < \tau^- \leqslant \tau^-(t)$ on the end-to-end delays of all correct messages in all executions, the ratio of which matches the (known) $\Theta = \frac{\tau^+}{\tau^-}$.

Formally, fix some algorithm A and let *ASYNC* be the set of executions of A running in a purely asynchronous message-driven system; note that we consider *timed executions* here, i.e., executions along with the occurrence times of their events, as measured on a discrete real-time clock (that is of course not available to the algorithms). A *property* P is a subset of the admissible executions in *ASYNC*, i.e., a property is defined via the executions of A that satisfy it. Let $\mathcal{M}$ be the set of admissible executions of A in some model M that augments the asynchronous model, by adding additional constraints like the ABC synchrony condition. Clearly, $\mathcal{M}$ is the intersection of some model-specific safety and liveness properties in *ASYNC*. We say that an execution α is in model M if $\alpha \in \mathcal{M}$, i.e., if α is admissible in M. If $\mathcal{M} \subseteq P$, we say that A *satisfies* property P in the model $\mathcal{M}$.

Definition 8 (*Timing-Independent Property*)**.** A property P is called *timing independent*, if $\alpha \in P \Rightarrow \alpha' \in P$ for every pair of causally equivalent executions α, α', i.e., executions where $G_\alpha = G_{\alpha'}$.

First, using a trivial model indistinguishability argument, it is easy to show that properties of an algorithm proved to hold in the ABC model M_{ABC} also hold in the Θ-Model M_Θ, for any $\Theta < \Xi$: The following Theorem 6 exploits the fact that the relevant cycles in the execution graph G_α, corresponding to an admissible execution α in the Θ-Model, also satisfy the ABC synchrony condition (2), i.e. that α is an admissible execution in the ABC model as well.

Theorem 6. *For any $\Theta < \Xi$, it holds that $\mathcal{M}_\Theta \subseteq \mathcal{M}_{ABC}$. Hence, if an algorithm satisfies a property P in the ABC model, it also satisfies P in the Θ-Model.*

Proof. If Z is any relevant cycle in G_α, then no more than $|Z^+|\Theta$ backward messages can be in Z; otherwise, at least one forward-backward message pair would violate (3). It follows that $|Z^-|/|Z^+| \leqslant \Theta < \Xi$ as required. Hence, $\mathcal{M}_\Theta \subseteq \mathcal{M}_{ABC} \subseteq P$, since the algorithm satisfies P in the ABC model. □

The converse of Theorem 6 is not true, however: The time-free synchrony assumption (2) of the ABC model allows arbitrary small end-to-end delays for individual messages, recall Fig. 1, violating (3) for every Θ. From a timing perspective, the ABC model is indeed strictly weaker than the Θ-Model, hence $\mathcal{M}_{ABC} \not\subseteq \mathcal{M}_\Theta$. Nevertheless, Theorem 7 below shows that, given an arbitrary finite execution graph G in $\mathcal{M}_{ABC}$, it is always possible to assign end-to-end delays $\in (1, \Xi)$ to the individual messages *without changing the event order* at any process.

[10] A trivial model indistinguishability argument is often used in conjunction with asynchronous algorithms, which obviously work correctly also in synchronous systems: Since synchronous admissible executions are just a subset of asynchronous admissible executions, every property guaranteed by an algorithm in the asynchronous model also holds in the synchronous model.

[11] Since the algorithms analyzed in Section 3 are essentially the same as the algorithms for clock synchronization and lock-step rounds in the Θ-Model introduced in [40], one may wonder whether it would not have been possible to just transfer the results of the Θ-based analysis to the ABC model using this general result. This is not possible, however, since some of the properties studied in [40] refer to real time and are hence not timing independent.

Theorem 7. *For every finite ABC execution graph G, there is an end-to-end delay assignment function τ, such that the weighted execution graph G^τ is causally equivalent to G and all messages in G^τ satisfy* (3)*.*

The very involved proof of Theorem 7 can be found in Section 4.1.

Let τ be such a delay assignment function, and G^τ be the weighted execution graph obtained from G by adding the assigned delays to the messages. Since Θ-algorithms are message-driven, without real-time clocks, G and G^τ are indistinguishable for every process. Consequently, an algorithm that provides certain timing-independent properties when being run in the Θ-Model also maintains these properties in the ABC model, as stated in Theorem 9 below.

In order to formally prove the claimed "model indistinguishability" of the ABC model and the Θ-Model, we proceed with the following Lemma 5. It says that processes cannot notice any difference in finite prefixes in the ABC model and in the Θ-Model, and therefore make the same state transitions.

Lemma 5 (*Safety Equivalence*)**.** *If an algorithm satisfies a timing-independent safety property S in the Θ-Model, then S also holds in the ABC model, for any $\Xi < \Theta$.*

Proof. Suppose, by way of contradiction, that there is a finite prefix β of an ABC model execution $\alpha \in \mathcal{M}_{ABC}$, where S does not hold. Furthermore, let β' be a finite extension of β such that all messages sent by correct processes in β arrive in β', and denote the execution graph of β' by $G_{\beta'}$. From Theorem 7, we know that there is a delay assignment τ such that the synchrony assumption (3) of the Θ-Model is satisfied for all messages in the timed execution graph $G^\tau_{\beta'}$, while the causality relation in $G_{\beta'}$ and $G^\tau_{\beta'}$ (and, since $G^\tau_{\beta'} \supseteq G^\tau_\beta$, also in G_β and G^τ_β) is the same.

We will now construct an admissible execution γ in the Θ-Model, which has the same prefix $G^\tau_{\beta'}$: If t is the greatest occurrence time of all events in $G^\tau_{\beta'}$, we simply assign an end-to-end delay of τ^+ to all messages still in transit at time t and to all messages sent at a later point in time. Note that γ may be totally different from the ABC execution α w.r.t. the event ordering *after* the common prefix β'. Anyway, γ is admissible in the Θ-Model since (3) holds for all messages, but violates S, which provides the required contradiction. □

Unfortunately, we cannot use the same reasoning for transferring liveness properties from the Θ-Model to the ABC model, since finite prefixes of an execution are not sufficient to show that "something good" must eventually happen or happens infinitely often. Nevertheless, Theorem 8 below reveals that all properties satisfiable by an algorithm in the Θ-Model can be reduced to safety properties, in the following sense: For every property P (which could be a liveness property like termination) satisfied by A in M_Θ, there is actually a (typically stronger) safety property $P' \subseteq P$ (like termination within time X) that is also satisfied by A in M_Θ and immediately implies P. Hence, there is no need to deal with liveness properties here at all.

For our proof, we utilize the convenient topological framework introduced in [5], where safety properties correspond to closed sets of executions in *ASYNC*, and liveness properties correspond to dense sets.

Definition 9 (*Closed Model*)**.** If a model M is determined solely by safety properties $S_1, \ldots, S_k$, then the set $\mathcal{M} = \bigcap_{i=1}^k \mathcal{S}_i$ corresponding to the executions admissible in M is closed, since finite intersections of closed sets are closed. We say that M is a *closed model*.

Theorem 8 (*Safety only in Closed Models*)**.** *Let M be a closed model augmenting the asynchronous model, and let $\mathcal{M} \subseteq ASYNC$ be the set of all admissible executions of an algorithm A in M. To show that A satisfies some arbitrary property P in M, it suffices to show that A satisfies the safety property $P' = P \cap \mathcal{M}$.*

Proof. Suppose that A satisfies some property $P \subseteq ASYNC$ in M, i.e., $\mathcal{M} \subseteq P$. Then, $\mathcal{M} = \mathcal{M} \cap P$ and since $\mathcal{M}$ is closed, it follows that $P' = \mathcal{M} \cap P$ is closed (in *ASYNC*) as well. But this is exactly the definition of a safety property $P' \subseteq ASYNC$ and, since $\mathcal{M} \subseteq P' \subseteq P$, it indeed suffices to show that A satisfies P' in M. □

Note that Theorem 8 is not limited to the context of the Θ-Model, but rather applies to *any* system model that is specified by safety properties.

Lemma 6 (*Closedness of Θ-Model*)**.** *The Θ-Model is closed.*

Proof. We just need to show that the set $\mathcal{M}_\Theta$ of executions in the Θ-Model is closed. If some execution α violated the end-to-end timing assumption (3) of the Θ-Model, there would be a finite prefix of α within which this violation has happened. This characterizes a safety property in *ASYNC*, which by definition coincides with some closed set. □

Theorem 8 in conjunction with Lemma 6 reveals that every property satisfiable in the Θ-Model is a safety property. Hence, Lemma 5 finally implies Theorem 9.

Theorem 9. *All timing-independent properties satisfied by an algorithm in the Θ-Model also hold in the ABC model, for any $\Xi < \Theta$.*

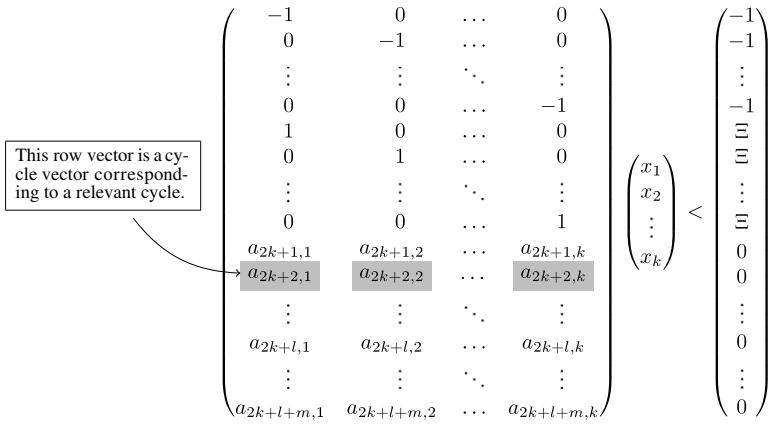

Fig. 6. Matrix form of the linear system $\mathbf{Ax} < \mathbf{b}$.

4.1. Proof of Theorem 7

We start our detailed treatment with the definition of a non-negative weight function τ on the edges of a given execution graph G, where τ will be such that (3) is satisfied for all messages in G. A necessary and sufficient condition on τ for preserving the causality relation $\rightarrow$ is to require that the sum over all edges in a cycle, taking into account their direction, is zero. Let Z be a (relevant or non-relevant) cycle and let the map $\mathrm{sgn}_Z : \Phi \times \Phi \rightarrow \{-1, 0, 1\}$ be such that $\mathrm{sgn}_Z(e^-) = +1$ for $e^- \in \bar{Z}^-$, $\mathrm{sgn}_Z(e^+) = -1$ for $e^+ \in \bar{Z}^+$, and $\mathrm{sgn}_Z(e) = 0$ for every e not in Z. The mapping $\tau : \Phi \times \Phi \rightarrow \mathbb{Q}^+$ is said to be an *assignment for the cycle* Z if $\sum_{e \in Z} \mathrm{sgn}_Z(e)\tau(e) = 0$. Note carefully that this "zero-sum" condition must hold for all cycles. A message e that is not contained in any cycle can safely be disregarded, since any value $\tau(e)$ will do for preserving $\rightarrow$. We call τ a *normalized assignment for* G if

$$1 < \tau(e) < \Xi, \tag{4}$$

$$0 < \tau(\bar{e}) < \infty, \tag{5}$$

for all non-local edges (=messages) e and all local edges $\bar{e}$ in G. Furthermore, we call G together with a normalized assignment τ a *timed execution graph*, denoted as G^τ.

Due to (4), a normalized assignment τ satisfies (3) for all messages in G since $\Xi < \Theta$. In addition, condition (5) ensures that all receive events at the same process are strictly ordered; since $\tau(\bar{e})$ may be chosen arbitrarily small, this is not a restriction in practice. In fact, allowing $\tau(\bar{e}) = 0$ would allow a local edge $\bar{e} = (\phi_1, \phi_2)$ to "disappear", which could alter the causality relation: The event order of simultaneous receive events ϕ_1, ϕ_2 could be determined by some tie-breaking rule, which might end up with $\phi_2 \rightarrow \phi_1$.

To show the existence of a normalized assignment, for a given finite execution graph (corresponding to a finite prefix of an ABC execution; this will be sufficient for our purposes, see Lemma 5), we combine the above conditions on τ in a system of linear inequalities of the form $\mathbf{Ax} < \mathbf{b}$ as follows: First, we split (4) into the conjunction of the lower bound condition $-\tau(e) < -1$ and the upper bound condition $\tau(e) < \Xi$, for all messages e. Moreover, assigning weights to all messages in a relevant cycle Z in a way such that

$$\sum_{e^- \in Z^-} \tau(e^-) - \sum_{e^+ \in Z^+} \tau(e^+) < 0 \tag{6}$$

holds, leaves "space" for assigning a positive weight to the local edges of Z, cp. Fig. 1, and hence to satisfy (5). Since every cycle has at least one local edge, the local edge weights can in fact be chosen such that the required cycle condition $\sum_{e \in Z} \mathrm{sgn}_Z(e)\tau(e) = 0$ also holds. The same reasoning applies if Z is a non-relevant cycle, except that the sums in (6) must have the opposite sign, cp. Fig. 4. As a consequence, it suffices to deal with assignments τ for messages only.

Listing these conditions for, say, k messages, l relevant cycles, and m non-relevant cycles in G yields the system of linear inequalities $\mathbf{Ax} < \mathbf{b}$ illustrated in Fig. 6. The matrix $\mathbf{A}$ is of size $n \times k$, for $n = 2k + l + m$; the coefficient x_j of the solution vector $\mathbf{x}$ is just the assignment $\tau(e_j)$ for message $e_j \in G$. The matrix–vector multiplication of the first k rows of $\mathbf{A}$ by vector $\mathbf{x}$ corresponds to the lower bound condition on τ for each message, while the upper bound conditions are represented by the multiplication of rows $k + 1$ to $2k$ by $\mathbf{x}$. The next l rows of $\mathbf{A}$ reflect condition (6) for all relevant cycles Z_i; a forward message will have a coefficient -1 here, whereas a backward message has $+1$, i.e., $a_{2k+i,j} = \mathrm{sgn}_{Z_i}(e_j)$. The remaining m rows represent the sign-flipped version of (6) for the non-relevant cycles Z_i, i.e., $a_{2k+i,j} = -\mathrm{sgn}_{Z_i}(e_j)$. Note that $a_{2k+i,j} = 0$ for every message e_j that is not in Z_i. Fig. 7 shows an example of this correspondence of cycles and cycle vectors.

We will use the following variant of Farkas' lemma for linear inequalities to prove that $\mathbf{Ax} < \mathbf{b}$ always has a solution, i.e., that a normalized assignment for G always exists.

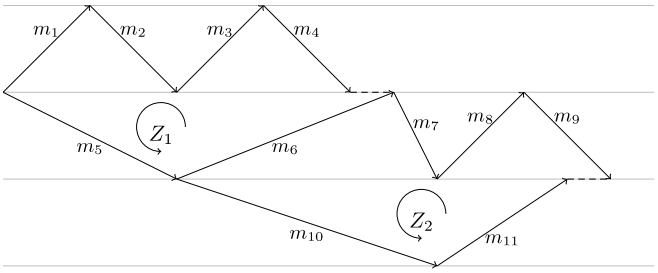

Fig. 7. For the relevant cycle Z_1 and the non-relevant cycle Z_2 we have cycle vectors $\mathbf{z_1} = (1, 1, 1, 1, -1, -1, 0, 0, 0, 0, 0)$ and $\mathbf{z_2} = (0, 0, 0, 0, 0, -1, -1, -1, -1, 1, 1)$.

Theorem 10 (*Farkas Lemma Variant, See [10]*)**.** *The system* $\mathbf{Ax} < \mathbf{b}$ *has a solution* $\mathbf{x}$ *if and only if all vectors* $\mathbf{y} > 0$ *where* $\mathbf{y}^T\mathbf{A} = 0$ *satisfy* $\mathbf{y}^T\mathbf{b} > 0$.

If a vector $\mathbf{y} > 0$ satisfies $\mathbf{y}^T\mathbf{A} = 0$, we have

$$\sum_{i=1}^{k} a_{i,j}y_i + \sum_{i=k+1}^{2k} a_{i,j}y_i + \sum_{i=1}^{l+m} a_{2k+i,j}y_{2k+i} = 0,$$

for all columns j in the matrix $\mathbf{A}$. Observing that the first $2k$ rows of $\mathbf{A}$ correspond to the identity matrices $-\mathbf{I}_k$ and $\mathbf{I}_k$, we can rewrite this as

$$y_{k+j} - y_j + \sum_{i=1}^{l+m} a_{2k+i,j}y_{2k+i} = 0. \tag{7}$$

Note that we call the first $2k$ rows of $\mathbf{A}$ the *upper part of* $\mathbf{A}$, and the first $2k$ coefficients of $\mathbf{y}$ *upper coefficients of* $\mathbf{y}$; the rest of the rows of $\mathbf{A}$ resp. the coefficients of $\mathbf{y}$ is called *cycle part of* $\mathbf{A}$ resp. *cycle coefficients of* $\mathbf{y}$. Moreover, we call a row vector in the cycle part of $\mathbf{A}$ *cycle vector*. We split the indices of the cycle part of $\mathbf{A}$ into the set R, containing all indices $1 \leqslant i \leqslant l$ of relevant cycle vectors, and the set N, containing all indices $l + 1 \leqslant i \leqslant l + m$ of non-relevant cycle vectors. Since equation (7) sums up to zero for every column j, the sum over all columns is also zero, that is,

$$\sum_{j=1}^{k} y_{k+j} - \sum_{j=1}^{k} y_j + \sum_{i\in R\cup N}\sum_{j=1}^{k} a_{2k+i,j}y_{2k+i} = 0.$$

For a relevant cycle Z_i, the sum of the positive resp. the sum of the negative entries in a row i in the cycle part of $\mathbf{A}$ is just the number of backward messages $|Z_i^-|$ resp. forward messages $|Z_i^+|$; for non-relevant cycles, we have the opposite correspondence. We can therefore rewrite the sum over all columns as

$$\sum_{j=1}^{k} y_j - \sum_{j=1}^{k} y_{k+j} = \sum_{i\in R} \left(|Z_i^-| - |Z_i^+|\right) y_{2k+i} + \sum_{i\in N} \left(|Z_i^+| - |Z_i^-|\right) y_{2k+i}. \tag{8}$$

We will first prove that $\bar{\mathbf{y}}^T\mathbf{b} > 0$ for a special type of solutions, called *canonical solutions* $\bar{\mathbf{y}}$, from which we will derive $\mathbf{y}^T\mathbf{b} > 0$ for arbitrary solutions $\mathbf{y}$ later on. A canonical solution $\bar{\mathbf{y}}$ must satisfy $\bar{\mathbf{y}} > 0$, with *integer* coefficients, $\bar{\mathbf{y}}^T\mathbf{A} = 0$, and either $\bar{y}_j = 0$ or $\bar{y}_{k+j} = 0$, or both, for all upper coefficients j of $\bar{\mathbf{y}}$.

Comparing (8) with $\bar{\mathbf{y}}^T\mathbf{b} = \Xi \sum_{j=1}^{k} \bar{y}_{k+j} - \sum_{j=1}^{k} \bar{y}_j$ suggests to consider the linear combination of all rows in the cycle part of $\mathbf{A}$, multiplied with the cycle coefficients $\bar{\mathbf{y}}^T$, which sum up to the "combined" row vector $\mathbf{s}$: Recalling $\bar{\mathbf{y}}^T\mathbf{A} = 0$, we need to distinguish 2 cases for the upper coefficients of $\bar{\mathbf{y}}$:

1. If $\bar{y}_{k+j} > 0$ and $\bar{y}_j = 0$, for some upper coefficient j, then Eq. (7) implies $\bar{y}_{k+j} = -s_j > 0$.
2. If $\bar{y}_j \geqslant 0$ and $\bar{y}_{k+j} = 0$, then $\bar{y}_j = s_j \geqslant 0$.

Hence, we can rewrite condition

$$\bar{\mathbf{y}}^T\mathbf{b} = \Xi \sum_{j=1}^{k} \bar{y}_{k+j} - \sum_{j=1}^{k} \bar{y}_j > 0$$

as

$$\Xi\mathbf{s}^+ + \mathbf{s}^- < 0, \tag{9}$$

where $\mathbf{s}^+$ is the sum over all negative coefficients in the sum vector $\mathbf{s}$, while $\mathbf{s}^-$ is the sum over all non-negative coefficients in $\mathbf{s}$. Observe that we have chosen the same notation for the entries of $\mathbf{s}$ as we have used for relevant cycles. This is by no means a coincidence: In fact, if $\mathbf{s}$ represented a relevant cycle, condition (9) would immediately follow from the ABC

synchrony condition (2).[12] Even though $\mathbf{s}$ is not a cycle vector in general, since its coefficients are usually not in $\{0, \pm 1\}$, we will exploit the fact that $\mathbf{s}$ is always a non-negative integer (since $\bar{\mathbf{y}} > 0$) linear combination of relevant and non-relevant cycles. Since we will prove (9) separately for non-negative linear combinations of relevant and non-relevant cycles, we split $\Xi\mathbf{s}^+ + \mathbf{s}^- < 0$ into two parts:

$$\Xi\mathbf{s}^+ + \mathbf{s}^- = \Xi\mathbf{s}_R^+ + \mathbf{s}_R^- + \Xi\mathbf{s}_N^- + \mathbf{s}_N^+,$$

where $\mathbf{s}_R^+, \mathbf{s}_R^-, \mathbf{s}_N^+, \mathbf{s}_N^-$, are the appropriate restrictions to the index sets R and N. Bear in mind that the sign of the coefficients of non-relevant cycle vectors are opposite to the relevant case.

Lemma 7 proves (9) for the non-relevant part; Lemma 11 will show the same result for the relevant part.

Lemma 7 (*Non-Relevant Sum Property*)**.** *Let $\mathbf{z}_1, \ldots, \mathbf{z}_\ell$, $\ell \geqslant 1$, be cycle vectors representing non-relevant cycles and let $\mathbf{s}_N$ be the vector corresponding to a non-negative linear combination of $\mathbf{z}_1, \ldots, \mathbf{z}_\ell$. Then, it holds that $\Xi\mathbf{s}_N^- + \mathbf{s}_N^+ < 0$.*

Proof. Since $|Z_i^-| - |Z_i^+| \geqslant 0$ for every i, it follows immediately, by summing up, that every non-negative linear combination $\mathbf{s}_N$ also satisfies $|\mathbf{s}_N^-| - |\mathbf{s}_N^+| \geqslant 0$. Since $\Xi > 1$, this implies $\Xi\mathbf{s}_N^- + \mathbf{s}_N^+ < 0$. □

Unfortunately, proving (9) for the relevant part is much more involved. The reason for this is that coefficients with opposite sign in a row cancel; this situation occurs in case of edge e in Fig. 2, for example. As a consequence, we cannot carry over the ABC synchrony condition (2) that holds for every constituent cycle vector to their sum (9). In order to solve this problem, we will show that there is a way to get rid of such cancellations, by constructing an equivalent set of cycle vectors that do not have coefficients with opposite sign in any row.

For proving Lemma 11, we will make use of some non-standard[13] *cycle space* of the underlying execution graph G. Formally, the cycle-space $\mathcal{C}$ of G is the sub-space of the vector space of the edge sets in G over $\mathbb{Q}$ spanned by G's (oriented) cycles. Since every cycle Z_i corresponds to a unique set of messages in G, which can be uniquely identified by a k-tuple ordered according to the columns in the matrix $\mathbf{A}$, there is a one-to-one correspondence between cycles Z_i in G and the cycle vectors $\mathbf{z}_i$ in $\mathbf{A}$, cp. Fig. 7. To avoid ambiguities w.r.t. indices, we will usually denote the coefficient for message e in $\mathbf{z}_i$ by $(\mathbf{z}_i)_e$. Every *cycle-space element*

$$Z = \lambda_1 Z_1 \oplus \lambda_2 Z_2 \oplus \cdots \oplus \lambda_\ell Z_\ell$$

is a linear combination of some relevant cycles $Z_1, \ldots, Z_\ell$, with all coefficients $\lambda_i \in \mathbb{Q}$, and the corresponding cycle vector reads

$$\mathbf{z} = \lambda_1 \mathbf{z}_1 + \lambda_2 \mathbf{z}_2 + \cdots + \lambda_\ell \mathbf{z}_\ell.$$

Note that we will use both representations interchangeably in the sequel.

The cycle addition operation $\oplus$ is defined as follows: If the cycles Z_1, Z_2 corresponding to the cycle vectors $\mathbf{z}_1, \mathbf{z}_2$ are disjoint, i.e., $Z_1 \cap Z_2 = \emptyset$, then the cycle-space element

$$Z = Z_1 \oplus Z_2 = Z_1 \cup Z_2$$

is the union of the two cycles Z_1, Z_2; it corresponds to the sum of the cycle vectors $\mathbf{z} = \mathbf{z}_1 + \mathbf{z}_2$. Note that disjoint cycles may have common vertices (and even partially overlapping local edges), but no common messages. If the cycles have a common message e, the outcome of adding $\mathbf{z}_1$ and $\mathbf{z}_2$ depends on the cycle vector orientation of Z_1 and Z_2: If e is *oppositely oriented* in Z_1 and Z_2, formally $(\mathbf{z}_1)_e \cdot (\mathbf{z}_2)_e < 0$ [we also say that e is a *mixed edge*, i.e., $e \in Z_1^- \cap Z_2^+$ or $e \in Z_1^+ \cap Z_2^-$ in relevant cycles, cp. message e in Fig. 2], then the coefficients cancel and hence $(\mathbf{z})_e = 0$. Consequently, e is no longer present in $Z = Z_1 \oplus Z_2$. Otherwise, if e is *identically oriented* in Z_1 and Z_2, formally $(\mathbf{z}_1)_e \cdot (\mathbf{z}_2)_e > 0$ [we say that message e is either a forward or a backward edge in both cycles], then the coefficients do not cancel and $(\mathbf{z})_e \neq 0$. In this case, e becomes a double edge in $Z = Z_1 \oplus Z_2$.

Hence, in general, the subgraph $Z = Z_1 \oplus Z_2$ corresponding to $\mathbf{z} = \mathbf{z}_1 + \mathbf{z}_2$ is not a cycle, and $Z_1 \oplus Z_2 \neq Z_1 \cup Z_2$. In fact, the general cycle-space element

$$Z = \lambda_1 Z_1 \oplus \cdots \oplus \lambda_n Z_n$$

is made up of multi-edges e with arbitrary multiplicity that is,

$$(\mathbf{z})_e = \lambda_1 (\mathbf{z}_1)_e + \cdots + \lambda_n (\mathbf{z}_n)_e \in \mathbb{Q}.$$

We will show, however, that every *non-negative integer* linear combination of cycle vectors representing relevant cycles yields a "relevant cycle like" vector $\mathbf{z}$, in the sense that its coefficients satisfy the ABC synchrony assumption (2). This immediately implies $\Xi\mathbf{s}_R^+ + \mathbf{s}_R^- < 0$ and thus proves (9) for the relevant part, see Lemma 11.

We start with the following Definition 10 of consistent cycles, which are such that all common edges consistently have either the same or the opposite orientation.

[12] If $\mathbf{s}$ corresponds to a relevant cycle S, the definition of the cycle vector coefficients yields $|S^-| = \mathbf{s}^-$ and $|S^+| = -\mathbf{s}^+$ and hence $-\Xi\mathbf{s}^+ - \mathbf{s}^- = \Xi|S^+| - |S^-| > 0$ by (2).

[13] Our "cycle space" is quite different from the well known cycle space in graph theory, cp. [14], since our notion of "cycles" correspond to cycles in the undirected shadow graph while still taking edge orientation into account.

Definition 10 (*Consistent Cycles*)**.** The cycles Z_1 and Z_2 are *consistent*, if there is some $d \in \{-1, +1\}$ such that the entries in the corresponding cycle vectors satisfy $(\mathbf{z}_1)_e \cdot (\mathbf{z}_2)_e = d$ for all messages $e \in Z_1 \cap Z_2$. If $d = +1$ resp. $d = -1$, then Z_1 and Z_2 are called *i-consistent* (identically consistent) resp. *o-consistent* (oppositely consistent). If Z_1 and Z_2 are disjoint, then they are i-consistent by definition. A set of cycles $M_1, \ldots, M_n$ is consistent [i-consistent/o-consistent] w.r.t. a cycle Z, if M_i and Z are consistent [i-consistent/o-consistent], for every $1 \leqslant i \leqslant n$.

For convenience, we say that $Z_1 \cap Z_2$ *contains* resp. *consists of* oppositely oriented messages, if for some resp. every message $e \in Z_1 \cap Z_2$ it holds that $(\mathbf{z}_1)_e \cdot (\mathbf{z}_2)_e = -1$. We proceed with several technical lemmas devoted to the removal of mixed edges in sums of cycle vectors.

Lemma 8 (*Mixed Edge Removal (Two Cycles)*)**.** *Let Z_1 and Z_2 be o-consistent cycles, such that all common message chains $m_1, \ldots, m_n$ in $Z_1 \cap Z_2$ consist of oppositely oriented messages only. Then, there are disjoint cycles $M_1, \ldots, M_n$ that are i-consistent w.r.t. both Z_1 and Z_2, such that $Z_1 \oplus Z_2 = M_1 \oplus \cdots \oplus M_n$. Moreover,*

$$|M_1 \oplus \cdots \oplus M_n| = |Z_1 \oplus Z_2| - 2 \sum_{i=1}^{n} |m_i|.$$

Proof. Let $Z_1 = v_1 m_1 v_1' \ldots v_2 m_2 v_2' \ldots v_{n-1} m_{n-1} v_{n-1}' \ldots v_n m_n v_n' \ldots v_1$, where v_i and v_i' denote the vertices incident to the message chain m_i, be the sequence of vertices and edges of Z_1 listed according to its cycle vector orientation. Since Z_2 traverses all common edges in the opposite direction, its analogous representation reads

$$Z_2 = v_n' m_n v_n \ldots v_{n-1}' m_{n-1} v_{n-1} \ldots v_2' m_2 v_2 \ldots v_1' m_1 v_1 \ldots v_n'.$$

Hence, the chains $m_1, \ldots, m_n$ are cancelled in $Z_1 \oplus Z_2$, only leaving the disjoint cycles

$$\begin{aligned} M_1 &= v_1 \ldots v_n' \ldots v_1, \\ M_2 &= v_2 \ldots v_1' \ldots v_2, \\ &\vdots \\ M_n &= v_n \ldots v_{n-1}' \ldots v_n. \end{aligned}$$

Every $M_i = v_i \ldots v_{i-1}' \ldots v_i$ (with $v_0' = v_n'$) consists of exactly one chain of messages $v_{i-1}' \ldots v_i$ in Z_1, and one chain of messages $v_i \ldots v_{i-1}'$ in Z_2, and is hence trivially i-consistent w.r.t. both Z_1 and Z_2. □

Lemma 9 (*Mixed Edge Removal (Single Set)*)**.** *Let Z be a cycle. If $M_1, \ldots, M_n$ are disjoint cycles such that every M_i and Z are either o-consistent or disjoint, then there is a set of disjoint cycles $M_1', \ldots, M_{l'}'$, all of which are i-consistent w.r.t. Z, such that $M_n \oplus \cdots \oplus M_1 \oplus Z = M_1' \oplus \cdots \oplus M_{l'}'$.*

Proof. We will construct $M_1', \ldots, M_{l'}'$ recursively. For $n = 1$, if Z and M_1 are disjoint (and hence i-consistent by definition), we just set $M_1' = Z$, $M_2' = M_1$ and trivially get $M_1 \oplus Z = M_1' \oplus M_2'$. Otherwise, Z and M_1 must be o-consistent and the statement of our lemma follows immediately from Lemma 8, applied to Z and M_1.

Now suppose that we have already constructed a set of disjoint cycles $M_1', \ldots, M_{l'}'$ that are all i-consistent w.r.t. Z with

$$M_{n-1} \oplus \cdots \oplus M_1 \oplus Z = M_1' \oplus \cdots \oplus M_{l'}'.$$

Since $\oplus$ is commutative and associative, it holds that

$$M_n \oplus M_{n-1} \oplus \cdots \oplus M_1 \oplus Z = M_n \oplus M_1' \oplus \cdots \oplus M_{l'}' = M_1' \oplus \ldots\ldots M_{l'}' \oplus M_n.$$

By our hypothesis, every M_ℓ', $1 \leqslant \ell \leqslant l'$, and Z are i-consistent, whereas M_n and Z are either disjoint or o-consistent. It follows immediately that every M_ℓ' and M_n is either o-consistent or disjoint. Since these are exactly the preconditions of our lemma, our recursive construction can be applied again. The termination of this recursive construction is guaranteed, since every application of Lemma 8 reduces the number of edges in the result. □

Lemma 10 (*Mixed Edge Removal (General Set)*)**.** *Let $Z_1, \ldots, Z_n$ be a set of cycles such that, for $1 \leqslant i < j \leqslant n$, cycles Z_i and Z_j are either disjoint or o-consistent. Then, there exist disjoint cycles $M_1, \ldots, M_l$ that are all i-consistent w.r.t. every Z_i, $1 \leqslant i \leqslant n$, such that $Z_1 \oplus \cdots \oplus Z_n = M_1 \oplus \cdots \oplus M_l$.*

Proof. The proof is by induction. For $n = 1$, Z_1 and $M_1 = Z_1$ are trivially i-consistent, hence establishing the induction base. For the induction step, suppose that there are disjoint cycles $M_1, \ldots, M_l$ that are i-consistent w.r.t. every Z_i, $1 \leqslant i \leqslant n-1$, such that

$$Z_1 \oplus \cdots \oplus Z_{n-1} = M_1 \oplus \cdots \oplus M_l.$$

Now, since every M_ℓ, $1 \leqslant \ell \leqslant l$, and every Z_i, $1 \leqslant i \leqslant n-1$, are i-consistent, whereas every Z_i and Z_n are o-consistent, it follows immediately that every M_ℓ and Z_n is either o-consistent or disjoint. Hence, we can apply Lemma 9 with $Z = Z_n$, which provides the required set $M_1', \ldots, M_{l'}'$ of disjoint cycles that are i-consistent w.r.t. every Z_i, $1 \leqslant i \leqslant n$ and satisfy $M_1' \oplus \cdots \oplus M_{l'}' = Z_1 \oplus \cdots \oplus Z_n$ as required. □

Theorem 11 (*Mixed-Free Decomposition*). *Let $C \in \mathcal{C}$ be a cycle-space element such that $C = Z_1 \oplus \cdots \oplus Z_n$. Then, there are cycles $M_1, \ldots, M_l$, which are all i-consistent w.r.t. every Z_i, $1 \leqslant i \leqslant n$ and no $M_j \cap M_{j'}$, $1 \leqslant j < j' \leqslant l$, contains oppositely oriented messages, such that*

$$Z_1 \oplus \cdots \oplus Z_n = M_1 \oplus \cdots \oplus M_l.$$

Proof. Let Γ be any non-empty subset of the multi-edges in C, i.e., of messages e that have some integer coefficient $(\mathbf{c})_e \notin \{0, \pm 1\}$ in the cycle vector corresponding to C. We can define an extended cycle-space $\mathcal{C}[\Gamma]$ as follows: Given some multi-edge $e \in \Gamma$, there must be at least $k = |(\mathbf{c})_e|$ cycles $Z_{\pi_1}, \ldots, Z_{\pi_k}$ where e has the same orientation $d = \mathrm{sgn}((\mathbf{c})_e) = \mathrm{sgn}((\mathbf{z}_{\pi_i})_e)$, $1 \leqslant i \leqslant k$. For every such Z_{π_i}, we introduce a new edge labeled $e^{Z_{\pi_i}}$ and replace Z_{π_i} by Z'_{π_i}, where $(\mathbf{z}'_{\pi_i})_e = 0$ but $(\mathbf{z}'_{\pi_i})_{e^{Z_{\pi_i}}} = d$. Doing this for all $e \in \Gamma$ provides a new set of cycles $Z_1[\Gamma], \ldots, Z_{n^\Gamma}[\Gamma] \in \mathcal{C}[\Gamma]$, which sum up to

$$C[\Gamma] = Z_1[\Gamma] \oplus \cdots \oplus Z_{n^\Gamma}[\Gamma].$$

Note that the only difference between C and $C[\Gamma]$ is that we have split all multi-edges $\in \Gamma$ occurring in C into separate new edges (which all have coefficients $\in \{0, \pm 1\}$) in $C[\Gamma]$. Let Γ^* denote the set of *all* multi-edges in C; note that $\Gamma \subset \Gamma^*$ implies that $C[\Gamma]$ still contains multi-edges in $\Gamma^* \backslash \Gamma$.

We will now prove by means of backwards induction on $|\Gamma|$ that the statement of our theorem actually holds for every cycle-space element $C[\Gamma]$. Since $C[\emptyset] = C$, this will also prove Theorem 11.

For the induction base, let $\Gamma = \Gamma^*$. Since all multi-edges have been split in $C[\Gamma^*]$, every pair $Z_i[\Gamma^*], Z_j[\Gamma^*]$, $1 \leqslant i < j \leqslant n^{\Gamma^*}$, is either disjoint or o-consistent. Lemma 10 thus reveals that there are disjoint cycles $M_1[\Gamma^*], \ldots, M_k[\Gamma^*] \in \mathcal{C}[\Gamma^*]$ that are all i-consistent w.r.t. every $Z_i[\Gamma^*]$, where

$$M_1[\Gamma^*] \oplus \cdots \oplus M_k[\Gamma^*] = Z_1[\Gamma^*] \oplus \cdots \oplus Z_{n^{\Gamma^*}}[\Gamma^*]$$

as required. Note that no $M_i[\Gamma^*] \cap M_j[\Gamma^*]$ can contain oppositely oriented messages because they are disjoint.

For the induction step, we assume that there are cycles $M'_1[\Gamma], \ldots, M'_k[\Gamma] \in \mathcal{C}[\Gamma]$, which are all i-consistent w.r.t. every $Z_i[\Gamma]$, $1 \leqslant i \leqslant n^\Gamma$ and no $M'_j[\Gamma] \cap M'_{j'}[\Gamma]$, $1 \leqslant j < j' \leqslant k$, contains oppositely oriented messages, such that

$$M'_1[\Gamma] \oplus \cdots \oplus M'_k[\Gamma] = C[\Gamma].$$

Let $M'_j[\Gamma]$, $1 \leqslant j \leqslant k$, be such a cycle. Suppose that $M'_j[\Gamma]$ contains $\alpha \geqslant 1$ "instances" of a multi-chain $c \subseteq \Gamma$, i.e., α maximum-length chains $c^{Z_1}, \ldots, c^{Z_\alpha}$ which have all been obtained by introducing new edges for the multi-edges making up the *single* multi-chain c. W.l.o.g., we can write

$$M'_j[\Gamma] = v_1^{Z_1} c^{Z_1} v_2^{Z_1} \ \ldots v_1^{Z_2} c^{Z_2} v_2^{Z_2} \ldots v_1^{Z_\alpha} c^{Z_\alpha} v_2^{Z_\alpha} \ldots v_1^{Z_1}.$$

Consequently, we have the following chains in $M'_j[\Gamma]$:

$$\begin{aligned}
&v_1^{Z_1} c^{Z_1} v_2^{Z_1} \ldots v_1^{Z_2} c^{Z_2} = M_{j_1} c^{Z_2},\\
&v_1^{Z_2} c^{Z_2} v_2^{Z_2} \ldots v_1^{Z_3} c^{Z_3} = M_{j_2} c^{Z_3},\\
&\vdots\\
&v_1^{Z_\alpha} c^{Z_\alpha} v_2^{Z_\alpha} \ldots v_1^{Z_1} c^{Z_1} = M_{j_\alpha} c^{Z_1}.
\end{aligned}$$

Now, if we rejoin all the edges in $c^{Z_1}, \ldots, c^{Z_\alpha}$ to form the multi-chain c again, that is, if we make a transition from $\mathcal{C}[\Gamma]$ to $\mathcal{C}[\Gamma \backslash c]$, then all instances of $c^{Z_1}, \ldots, c^{Z_\alpha}$ in the above chains collapse to the single multi-chain c. Consequently, in $\mathcal{C}[\Gamma \backslash c]$, every of the M_{j_ℓ}, $1 \leqslant \ell \leqslant \alpha$, above actually forms a *cycle* $M_{j_\ell}[\Gamma \backslash c]$ — note that the vertices $v_1^{Z_\ell}$ and $v_1^{Z_{\ell+1}}$ also collapse to a single vertex. Since $M'_j[\Gamma]$ is i-consistent w.r.t. every $Z_i[\Gamma]$, $1 \leqslant i \leqslant n^\Gamma$, every $M_{j_\ell}[\Gamma \backslash c]$ must be i-consistent w.r.t. every $Z_i[\Gamma \backslash c]$, $1 \leqslant i \leqslant n^{\Gamma \backslash c}$, as well. Furthermore, according to the construction above, every edge of $M'_j[\Gamma]$ (except the edges in $c^{Z_1}, \ldots, c^{Z_\alpha}$, of course) is contained in exactly one cycle $M_{j_\ell}[\Gamma \backslash c]$, and no $M_{j_\ell}[\Gamma \backslash c] \cap M_{j_{\ell'}}[\Gamma \backslash c]$ can contain oppositely oriented edges. Finally, no $M_{i_\ell}[\Gamma \backslash c] \cap M_{j_{\ell'}}[\Gamma \backslash c]$, $i \neq j$, can contain oppositely oriented edges either, since $M'_j[\Gamma]$ and $M'_i[\Gamma]$ are disjoint. Hence, taking all the sets $M_{j_\ell}[\Gamma \backslash c]$ (or, if $\alpha = 0$ for $M'_j[\Gamma]$, then $M_j[\Gamma \backslash c] := M'_j[\Gamma]$) provides the sought set

$$M_1[\Gamma \backslash c], \ldots, M_k[\Gamma \backslash c] \in \mathcal{C}[\Gamma \backslash c],$$

thereby completing our proof. □

Corollary 1. *Let $C \in \mathcal{C}$ be such that $C = Z_1 \oplus \cdots \oplus Z_n$, for relevant cycles $Z_1, \ldots, Z_n$. Then, $\frac{|C^-|}{|C^+|} < \Xi$.*

Proof. Applying Theorem 11 yields cycles $M_1, \ldots, M_l$ such that

$$C = Z_1 \oplus \cdots \oplus Z_n = M_1 \oplus \cdots \oplus M_l,$$

which do not contain oppositely oriented messages that would cancel when summing up. In order to prove $\frac{|C^-|}{|C^+|} < \Xi$, it hence suffices to show $\frac{|M_i^-|}{|M_i^+|} < \Xi$ for every M_i. There are only two possibilities:

1. $M_i^- \subseteq C^-, M_i^+ \subseteq C^+$: If M_i is relevant, then obviously $\frac{|M_i^-|}{|M_i^+|} < \Xi$. Assume, for the sake of contradiction, that M_i is non-relevant. Then there is a local edge $\kappa \in M_i$ that is traversed forward (causally) in M_i, and hence in C. Since $C = Z_1 \oplus \cdots \oplus Z_n$, there must be some Z_j with $\kappa \in Z_j$ where κ is traversed in the same way as in C and, hence, in M_i. This contradicts Z_j being a relevant cycle, however.
2. $M_i^+ \subseteq C^-, M_i^- \subseteq C^+$: By (1), it holds that $|M_i^-| \geqslant |M_i^+|$ and hence $\frac{|M_i^+|}{|M_i^-|} \leqslant 1 < \Xi$. Since M_i^- resp. M^+ correspond to edges in C^+ resp. C^-, it follows that M_i contributes properly to $\frac{|C^-|}{|C^+|} < \Xi$ as required. Note that this holds independently of whether M_i is relevant or not.

This completes the proof of Corollary 1. □

As a consequence, we finally get the desired proof of (9) for the relevant part:

Lemma 11 (*Relevant Sum Property*)**.** *Let $\mathbf{z}_1, \ldots, \mathbf{z}_\ell$ be cycle vectors representing relevant cycles and let $\mathbf{s}_R$ be the vector corresponding to a non-negative integer linear combination of $\mathbf{z}_1, \ldots, \mathbf{z}_\ell$. Then, it holds that $\Xi \mathbf{s}_R^+ + \mathbf{s}_R^- < 0$.*

Proof. Corollary 1 does not require the Z_i to be different. Since $S_R = \lambda_1 Z_1 \oplus \cdots \oplus \lambda_\ell Z_\ell$ for non-negative integer coefficients λ_i, we can hence invoke Corollary 1 with λ_i instances of the same relevant cycle Z_i, $1 \leqslant i \leqslant \ell$. □

Combining Lemmas 7 and 11 immediately proves that every canonical solution $\bar{\mathbf{y}}$ satisfies $\bar{\mathbf{y}}^T\mathbf{b} > 0$. It only remains to extend this result to arbitrary solution vectors $\mathbf{y}$, which is done in the following Theorem 12.

Theorem 12 (*Existence of a Normalized Assignment*)**.** *The system $\mathbf{Ax} < \mathbf{b}$ corresponding to a finite execution graph has always a solution, and hence a normalized assignment.*

Proof. The statement follows immediately from Theorem 10, if we can show that every $\mathbf{y} > \mathbf{0}$ with coefficients $y_j \in \mathbb{Q}$ satisfying $\mathbf{y}^T A = 0$ also fulfills $\mathbf{y}^T\mathbf{b} > 0$. If $\mathbf{y}$ is a canonical solution $\bar{\mathbf{y}}$, then $\bar{\mathbf{y}}^T\mathbf{b} > 0$ follows from adding the results of Lemmas 7 and 11, recall (9) in conjunction with $\Xi\mathbf{s}^+ + \mathbf{s}^- = \Xi\mathbf{s}_R^+ + \mathbf{s}_R^- + \Xi\mathbf{s}_N^- + \mathbf{s}_N^+$. Otherwise, we can derive a canonical solution $\bar{\mathbf{y}}$ from a non-canonical solution $\mathbf{y}$ as follows:

1. For all upper coefficients $1 \leqslant j \leqslant k$ of $\mathbf{y}$: If $y_j > y_{k+j}$, then $\bar{y}_j = y_j - y_{k+j}$ and $\bar{y}_{k+j} = 0$; otherwise, $\bar{y}_{k+j} = y_{k+j} - y_j$ and $\bar{y}_j = 0$.
2. For all cycle coefficients $2k + 1 \leqslant i \leqslant 2k + l + m$ of $\mathbf{y}$: $\bar{y}_i = y_i$.
3. Finally, multiply every $\bar{y}_j$ by the least common multiple of $\bar{y}_1, \ldots, \bar{y}_{2k+l+m}$ to get integer coefficients.

Since $\mathbf{y}^T\mathbf{A} = 0$, it follows immediately from the above definition of $\bar{\mathbf{y}}$ that $\bar{\mathbf{y}}^T\mathbf{A} = 0$. Hence, $\bar{\mathbf{y}}^T\mathbf{b} > 0$. Now consider $(\mathbf{y}^T - \bar{\mathbf{y}}^T)\mathbf{b}^T$; after cancelling the common parts of $\bar{\mathbf{y}}$ and $\mathbf{y}$, according to our construction, we get

$$(\mathbf{y}^T - \bar{\mathbf{y}}^T)\mathbf{b}^T = \sum_{j:y_{k+j} \geqslant y_j} (\Xi - 1)\, y_j + \sum_{j:y_j > y_{k+j}} (\Xi - 1)\, y_{k+j}.$$

This term is non-negative, since $\mathbf{y}$ is non-negative and $\Xi > 1$. Hence, $\mathbf{y}^T\mathbf{b} \geqslant \bar{\mathbf{y}}^T\mathbf{b} > 0$ and we are done. □

Theorem 12 immediately implies the sought Theorem 7.

5. Related work and practical aspects

In this section, we briefly relate the ABC model to the existing partially synchronous models (1)–(7) listed in Section 1 and discuss some observations concerning the practical applicability of the model.

The fact that we will primarily discuss aspects where the ABC model surpasses alternative models should not be taken as a claim of general superiority, however: A fair model comparison is difficult and also highly application dependent; it almost always leads to the conclusion that any two models are incomparable, in the sense that model A is better than B in aspect X but worse in aspect Y, cp. [40]. We start with a brief account of the major features of those models.

The classic partially synchronous models introduced in Dwork et al. [17] and the semi-synchronous models from [36,7] incorporate a bound Φ on the relative speed of processes, in addition to a transmission delay bound Δ. All those models allow a process to timeout messages: The semi-synchronous models assume that local real-time clocks are available, the classic partially synchronous models use the computing step time of the fastest process as the units of "real time"; a spin loop with loop count Δ is hence sufficient for timing out the maximum message delay here. An even older partially synchronous model is the Archimedean model introduced in [39], which assumes a bounded ratio $s \geqslant u/c$ of the minimum computing step time c and the maximum computing step time + transmission delay u. Again, any process can timeout a message by means of a local spin loop with loop count s here.

An even more relaxed way of adding synchrony properties to asynchronous systems underlies the chase for the weakest system model for implementing the Ω failure detector in systems with crash failures, see [3,2,4,34,6,30,26,27]; an extension to Byzantine failures can be found in [1]. These Weak Timely Link (WTL) models can be viewed as a "spatial" relaxation of the classic partially synchronous or semi-synchronous models: The currently weakest of these models, introduced in [26,

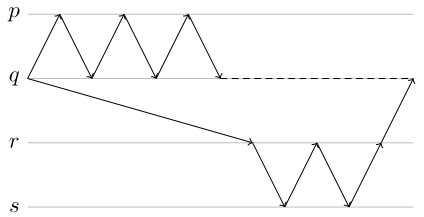

Fig. 8. A relevant cycle, valid for any $\Xi > 1$. Note that r takes no step while p and q can make progress only bounded by $|Z^-|$.

27], requires just a single process p (a timely f-source) with f eventual timely outgoing links, connecting p to a time-variant set of f receiver processes. Note that communication over all the other links in the system can be totally asynchronous.

A model that is very close to pure asynchrony is the Finite Average Response time (FAR) model introduced in [21,22]. The properties added to the FLP model are an unknown lower bound for the computing step time, and an unknown finite average of the round-trip delays between any pair of correct processes. The latter allows round-trip delays to be increasing without bound, provided that there are sufficiently many short round trips in between that compensate for the resulting increase of the average. Due to the computing step time lower bound, any process can implement a weak bounded drift clock via a local spin loop here, which allows to safely timeout messages by using timeout values learned at runtime.

The remaining models MCM and MMR will be described in Section 5.2.

5.1. Relation to the classic partially synchronous model

In this section, we relate the ABC model to the classic partially synchronous model (abbreviated ParSync[14]) introduced in Dwork et al. [17]. ParSync stipulates a bound Φ on relative computing speeds and a bound Δ on message delays, relative to an (external) discrete "global clock", which ticks whenever a process takes a step: During Φ ticks of the global clock, every process takes at least one step, and if a message m was sent at time k to a process p that subsequently performs a receive step at or after time $k + \Delta$, p is guaranteed to receive m.

First of all, we note that the ABC model and ParSync are equivalent in terms of solvability of timing-independent problems in fully connected networks. In [40], it was shown that the Θ-Model and ParSync are equivalent in this regard: Since the synchrony parameters Φ, Δ of the ParSync model imply bounded (and non-zero) end-to-end delays, any Θ-algorithm can be run in a ParSync system if $\Theta = \Theta(\Phi, \Delta)$ is chosen sufficiently large. Conversely, using the lock-step round simulation for the Θ-Model provides a "perfect" ParSync system ($\Phi = 1$ and $\Delta = 0$), which obviously allows to execute any ParSync algorithm atop of it. The claimed equivalence thus follows from the model indistinguishability of the ABC model and the Θ-Model established in Section 4.

This problem equivalence does not imply that the models are indeed equivalent, however. First, as shown below, there are problems that can be solved in the ABC model but not in ParSync in case of not fully connected networks and/or distributed algorithms where processes communicate only with a subset of the other processes. Moreover, whereas we can choose Ξ such that every execution of a message-driven algorithm in ParSync with Φ, Δ is also admissible in the ABC model for some $\Xi > \Theta(\Phi, \Delta)$, we can even conclude from $\mathcal{M}_{\text{ABC}} \supset \mathcal{M}_{\Theta}$ that some ABC executions cannot be modeled in ParSync. Actually, in [40], it has been shown that there are Θ-executions that cannot be modeled in ParSync.

To investigate this issue also from a different perspective, it is instructive to embed the ABC model in the taxonomy of partially synchronous models introduced in [16]: In this seminal work, the exact border between consensus solvability and impossibility has been determined. It distinguishes whether (c) communication is synchronous (Δ holds) or asynchronous, whether (p) processes are synchronous (Φ holds) or asynchronous, whether (s) steps are atomic (send + receive in a single step) or non-atomic (separate send and receive steps), whether (b) send steps can broadcast or only unicast, and whether (m) message delivery is (globally) FIFO ordered or out-of-order.

We will argue below that, within this taxonomy, the ABC model model must be mapped to the case of asynchronous communication, asynchronous processes, atomic steps, broadcast send and out-of-order delivery. Using the corresponding "binary encoding" ($c = 0, p = 0, s = 1, b = 1, m = 0$) in [16, Table 1], it turns out that consensus is not solvable in the resulting ParSync model. The apparent contradiction to the solvability of consensus in the ABC model is due to the ABC synchrony condition, which (weakly) restricts the asynchrony of processes and communication. Since this restriction is not expressible in the taxonomy of [16], the ABC model must be "overapproximated" by totally asynchronous processes and communication here.

Asynchronous communication and asynchronous processes:

Consider a 2-player game where the Prover first chooses Ξ and the Adversary, knowing Ξ, chooses a pair (Φ, Δ). Finally, the Prover has to choose an execution satisfying (2) for Ξ; the Prover wins iff this execution violates the adversary-chosen

[14] We consider only the perpetual partially synchronous model here ($t_{\text{GST}} = 0$); some issues related to the eventual variants ($t_{\text{GST}} > 0$) will be discussed in Section 6. A detailed relation of all variants of ParSync models to the corresponding variants of the Θ-Model can be found in [40].

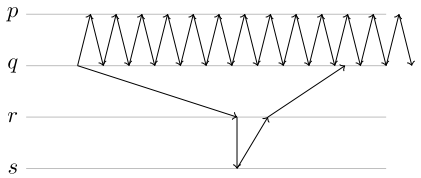

Fig. 9. The long delay on the link between q and r is compensated by the fast delay on the link between r and s.

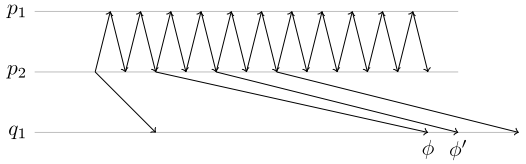

Fig. 10. A system implementing bounded-size FIFO channels. If the order of ϕ and ϕ' is changed, there would be a relevant cycle violating (2) if $\Xi = 4$.

parameters (Φ, Δ). The Prover has a winning strategy: It suffices to choose any execution containing a relevant cycle as shown in Fig. 8, which respects (2) but lets $|Z^-|$ be greater than both Φ and Δ: While the (slow) message from q to r is in transit, process q executes more than Δ steps. Moreover, neither process r nor s execute a step during the more than Φ steps of q. As a consequence, both communication and processes must be considered asynchronous ($c = 0$, $p = 0$).

Atomic steps and broadcast:

Whereas it is clear that out-of-order delivery ($m = 0$) makes it more difficult to solve problems, one may be wondering whether the "favorable" choices $s = 1$ and $b = 1$, rather than the ABC synchrony condition, make consensus solvable in the ABC model. Dolev et al. [16, Table 1] reveals that this is not the case, however: All the entries corresponding to $p = 0, c = 0, m = 0$ are the same (consensus impossible), irrespectively of the choice of b and s. And indeed, the assumption of atomic send + receive steps with broadcast in the ABC model's definition in Section 2 is just a simplifying abstraction: Every non-atomic unicast execution can be mapped to a causally equivalent atomic send + receive + broadcast step execution with appropriately adjusted end-to-end delays. The ABC model can hence also be used for making classic distributed algorithms results applicable to non-atomic models like the Real-Time Model introduced in [33].

Another major difference between ParSync and the ABC model results from the cumulative and dynamic, non-system-wide character of the ABC synchrony condition. Since (2) needs to hold only in relevant cycles, which are in fact defined by the message patterns of the specific algorithm employed, the ABC model is particularly suitable for modeling systems with not fully connected communication graphs: For choosing Ξ, only the cumulative end-to-end delay ratio over certain paths counts.

Consider the execution shown in Fig. 9, for example, which corresponds to a system where process q exchanges messages directly with p (over a 1-hop path P_{qpq}), and indirectly with s (over a 2-hop path P_{qrsrq} via r). As long as the sum of the delays along P_{qrsrq} is less than the cumulative delay of Ξ instances of P_{qpq}, the individual delays along the links between q, r and r, s are totally irrelevant. In the VLSI context, for example, this gives more flexibility for place-and-route, as well as some robustness against dynamic delay variations. By contrast, in ParSync, very conservative values of Φ, Δ would be needed to achieve a comparable flexibility; obviously, this would considerably degrade the achievable performance system-wide.

In case of not fully connected networks, there are even situations which cannot be modeled in ParSync at all. Consider the message pattern given in Fig. 10 in a system with $\Xi = 4$, for example: The ABC synchrony condition ensures FIFO order of the messages sent from p_2 to q_1, even when their delay is unbounded (and may even continuously grow, as e.g. in a formation of fixed-constellation clusters of spacecraft that move away from each other): If there was a reordering of ϕ and ϕ', a relevant cycle with $\Xi = 5$ would be formed, which is not admissible for $\Xi = 4$ and hence cannot occur. Note that processes p_1, p_2 make unbounded progress while a message to q_1 is in transit here. Hence, as in the example of Fig. 8 mentioned before, the problem cannot be solved in ParSync. Clearly, such message ordering capabilities are very useful in practice, e.g., for implementing stable identifiers, bounded message size, single source FIFOs etc.

5.2. Relation to other partially synchronous models

In this section, we will briefly relate the ABC model to the remaining partially synchronous models MCM and MMR listed in Section 1.

Whereas most existing partially synchronous models refer to message delays and computing step times, the ABC model only constrains the ratio of the number of backward vs. forward messages in relevant cycles. Moreover, the ABC model does not assume any relation between computing step times and message delays, and, in contrast to the models considered in Section 5.1, can also be used to model systems where no hardware clock is available. Note that this is also true for the FAR Model by Fetzer et al. [22], which requires a bounded-speed clock and hence introduces some (weak) dependency

between average message delays and computing steps as well. The only exceptions are the Θ-Model, which constrains the maximal/minimal end-to-end delay ratio, the MCM model by Fetzer [20] that assumes a classification of received messages in "slow" and "fast" ones, and the MMR model of Mostefaoui et al. [35], which restricts the message order in round-trip communication patterns.

The MCM model applies to systems with process crash failures. It assumes that all received messages are correctly flagged as either "slow" or "fast", with the requirement that the end-to-end delay of any slow message is more than two times the end-to-end delay of any fast message. To prohibit flagging all messages as "slow", Fetzer [20] assumes the existence of at least one correct process p that can eventually communicate bidirectionally with all correct processes via "fast" messages. Except for messages sent/received by this eventual bidirectional n-source p (in the terminology of the WTL models), the MCM model also allows message loss and even transient partitioning. Since the message classification assumption facilitates a time-free timeout mechanism for "fast" round trips (by means of local messages, sent from a process to itself, which are always delivered as "slow"), the MCM model allows the implementation of the eventually strong failure detector $\diamond S$ and, hence, to solve consensus.

Like the ABC model, the MCM model is totally time-free, yet allows to reliably timeout certain messages. However, whereas the MCM uses local "slow" messages to timeout a round-trip of "fast" ones, the ABC model uses "fast" message chains to timeout "slow" ones.[15] The message classification assumption is hence more demanding than the ABC condition, since no two messages with delay ratio in the interval $(1, 2]$ may ever be in transit simultaneously (unless they are both "slow").

The MMR model by Mostefaoui et al. [35] also applies to systems with at most f process crash failures. It assumes that, in every round-trip of process p_i with all its peers, there is a fixed set of processes Q_i the responses of which are among the first $n - f$ responses received by p_i. This property turned out to be sufficient for implementing the eventually strong failure detector $\diamond S$ and, hence, to solve consensus.

Like the ABC condition, the MMR condition enforces a certain order of events. Although this condition cannot be expressed explicitly in the ABC model, it can be interpreted as a special instance of the (undefined[16]) situation $\Xi = 1$ for certain messages. Due to its order-based synchrony condition, the MMR model shares several advantages of the ABC; on the down side, it is restricted to a specific communication pattern and has a quite demanding synchrony requirement (albeit for certain messages only). Moreover, the ABC model is superior w.r.t. solvability of problems, since the MMR model does not allow to reliably timeout messages. It is hence impossible to implement uniform lock-step rounds, for example: If a process q sends a round message to p and then immediately crashes, p cannot distinguish this from the scenario where q has crashed before sending the round message. Consequently, neither Lemma 4, which gives a bound on the failure detection time, nor the bounded progress condition in Theorem 4 could be derived in the MMR model. Actually, the same is true for any model that does not provide stronger synchrony properties than provided by a perfect failure detector, i.e., for any model where the Strongly Dependent Decision Problem cannot be solved, see [12].

A particularly attractive property of the ABC model is its ability to deal with unbounded message delays. Among the existing partially synchronous models, this is also true for the MCM model, the MMR model, the FAR Model and the dynamic Θ-Model introduced in [40]. The FAR Model actually surpasses the ABC model here, in the sense that it does not require any correlation between the delays of messages exchanged by different processes. On the other hand, the FAR Model is inferior to the ABC model due to its requirement of finite average message delays, which rules out continuously growing delays (occurring e.g. in a formation of spacecraft that continuously move away from each other). The remaining partially synchronous models also allow growing delays, although all message delays must change roughly at the same rate. By contrast, as already mentioned in Section 5.1, the ABC model does not force unrelated messages to meet any constraints, and even allows messages with zero delay.

5.3. Practical aspects

Since the ABC model shares many features with the Θ-Model, most of the applicability and model coverage aspects discussed in [40] apply a fortiori to the ABC model. In particular, as a message-driven model, the ABC model suffers from the problem that the entire system may become mute in case of excessive message loss and/or partitioning, and that the overhead of continuously sending messages may become significant. Although there are ways of mitigating these problems, they cannot be ruled out completely. At the same time, ABC algorithms are easily portable and benefit from the ABC model's good coverage in real systems.

Moreover, the non-system-wide scope of the ABC synchrony condition (2), which only constrains messages in relevant cycles on a per-cycle basis, makes the ABC model also applicable to in systems that cannot be modeled by the existing partially synchronous models. Consider a formation of spacecraft, for example, where clusters of spacecraft continuously move away from each other but stay close within a cluster: If an algorithm generates only message chains that span multiple

[15] Note that, in contrast to the MCM model, in the ABC model there is no global classification of messages into "fast" and "slow" messages, as a "fast" (i.e. *backward*, see Definition 3) message in one cycle Z might actually be a "slow" (i.e. *forward*) message in another cycle Y.

[16] Since this would make the definition of forward and backward edges (and hence of relevant and irrelevant cycles) superfluous, the ABC model does not allow $\Xi = 1$.

clusters in relevant cycles, a properly chosen ABC synchrony condition in the corresponding execution graph will always be maintained. No existing partially synchronous model can adequately model such systems: The DLS Model Dwork et al. [17], the Archimedean Model Vitányi [38] and the WTL Models Aguilera et al. [4,30,26] assume some (possibly unknown) global delay bound for all timely messages. The Θ-Model [40], on the other hand, suffers from the problem that *all* messages simultaneously in transit within the whole system must obey the global delay ratio Θ. Somewhat an exception is the FAR Model, which does not require any correlation between the delays of messages exchanged by different processes; it fails to model the above example just because of the ever-growing delays, however.

Interestingly, the ABC model is also a promising candidate for modeling distributed algorithms in *very large scale integration* (VLSI) circuits[17]: Due to continuously shrinking feature sizes and increasing clock speeds, today's deep sub-micron VLSI have much in common with loosely coupled asynchronous distributed systems studied in distributed computing for decades, see e.g. [37,18]. Given that the delays observed in a particular chip depend heavily on the VLSI implementation technology, as well as on the actual place-and-route of the components and wires, it is definitely sub-optimal to compile time values into a distributed algorithm here—in particular, when those values affect its internal structure (message-buffer sizes, for example): Re-using such an algorithm in conjunction with a different implementation technology or within a different application would be difficult.

By contrast, when an ABC algorithm is used in the VLSI context, there is a very good chance that the algorithm can be re-used without a change. In particular, when a design is migrated to a, say, faster implementation technology, both minimum and maximum delay paths are usually sped up in a similar way. Hence, the algorithm's Ξ is likely[18] to continue to hold. Similarly, if an ABC algorithm is employed within a different VLSI application, one can usually guarantee its Ξ by setting suitable constraints during place-and-route. Thanks to the ABC model's weak properties, these constraints concern (1) *cumulative* delays, and (2) timing *ratios* only. They are hence much easier to satisfy than explicit timing constraints put on individual components and wires.

6. Weaker variants of ABC models

Analogous to Dwork et al. [17] and Widder and Schmid [40], it is possible to define 4 variants of the ABC model:

- ABC model: Ξ is known and holds perpetually (the model introduced in Section 2)
- ?ABC model: Ξ is unknown and holds perpetually
- $\diamond$ABC model: Ξ is known and holds eventually
- $\diamond$?ABC model: Ξ is unknown and holds eventually

In case of the latter two models, we assume that only relevant cycles starting at or after some (unknown) consistent cut $\mathcal{C}_{GST}$ (replacing the "global stabilization time" in Dwork et al. [17]) in the execution graph satisfy (2).

Due to the "indistinguishability" of the ABC model and the Θ-Model for Θ-algorithms, established in Section 4, one can immediately use the algorithms proposed in [40] for providing eventual lock-step rounds in the ?Θ-, $\diamond\Theta$-, and $\diamond$?Θ-Models to achieve the same in the ?ABC model, $\diamond$ABC model, and $\diamond$?ABC model.

In case of the ?ABC model and the $\diamond$?ABC model, the resulting algorithms are not particularly efficient, however, since they double the round duration with every round. A more clever algorithm could exploit the ABC synchrony condition to eventually learn a feasible value for Ξ: Suppose p's current estimate $\hat{\Xi}$ of Ξ is 2 in the execution depicted in Fig. 3. In the computing step ψ, process p finds out that either $\hat{\Xi} = 2$ is wrong or p_{slow} has crashed; it can hence increase its estimate $\hat{\Xi}$ as soon as the slow message from p_{slow} arrives. The definition and analysis of such refined algorithms is a subject of our current research.

An orthogonal way of weakening the ABC model is to drop all cycles from the space–time diagram that exceed a certain length. For example Algorithm 1 will work correctly even in an ABC model where only cycles with at most 2 forward messages are considered.

A related, particularly interesting topic of our current research is consensus solvability in the ABC model: Obviously, on top of the lock-step round simulation in the ABC model, any Byzantine fault-tolerant synchronous consensus algorithm can be used for solving consensus. Similarly, on top of the eventual lock-step round simulations in the ?ABC model, the $\diamond$ABC model, and the $\diamond$?ABC model, consensus can be solved with the Byzantine resilient algorithms introduced in Dwork et al. [17].

It is still an open question, however, whether the ABC model can contribute another step beyond the currently weakest model by Hutle et al. [27] for solving consensus in the presence of crash failures. Both the time freeness and the non-system-wide scope of the ABC synchrony condition make it a promising candidate. However, answering this question boils down to finding "minimum" execution graphs (recall Section 2), i.e., ways of dropping almost all messages (and hence exempting

17 Which is an emerging field, see e.g. the Dagstuhl seminar *Fault-Tolerant Distributed Algorithms in VLSI Chips* organized by Charron-Bost et al. [13].

18 For example, we succeeded to migrate the DARTS fault-tolerant clock generation algorithm for Systems-on-Chip (see [24,19]), which is based on Algorithm 1, from an FPGA to a high-speed ASIC without a change.

them from the ABC synchrony condition) except for a minimal set that cannot be dropped without running into the FLP impossibility.

For example, one could just adopt the idea underlying the simple Ω failure detector Chandra and Toueg [11] for the Θ-Model [9]: The ABC synchrony condition could be restricted to a fixed subset of $f + 2$ processes in the system, which elect a leader among themselves and disseminate its id to the remaining processes in the system. By virtue of our "model indistinguishability" result, it immediately follows that the algorithms of Biely and Widder [9] are correct Ω-implementation in the ABC model as well. Similarly, using a straightforward extension of the ABC model to time-driven systems, it would also possible to adapt the Ω-implementations developed for the WTL models to the ABC model.

7. Conclusions and future work

We have introduced a novel partially synchronous system model, the ABC model, which is completely time-free and thus rests on a causality-based synchrony condition only. We showed that it is sufficiently strong for implementing lock-step rounds and, hence, for solving important distributed computing problems, including consensus. We also proved that algorithms designed for the Θ-Model also work correctly in the ABC model. Our results thus reveal that explicit timing constraints are not essential for a synchrony-based approach, even if failure detector oracles or randomization are not available. Part of our future work is devoted to fully exploiting the ABC model in the chase for the weakest system model for solving consensus, and to the analysis of the ABC model's coverage in real systems, in particular, VLSI Systems-on-Chip.

Acknowledgements

We are most indebted to Martin Biely, Josef Widder, Martin Hutle, Matthias Függer, Heinrich Moser, and Bernadette Charron-Bost for their contributions to the development of the ABC model.

References

[1] Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, Sam Toueg, Consensus with Byzantine failures and little system synchrony, in: DSN '06: Proceedings of the International Conference on Dependable Systems and Networks, IEEE Computer Society, Washington, DC, USA, 2006, pp. 147–155. doi:10.1109/DSN.2006.22. ISBN 0-7695-2607-1.
[2] Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, Sam Toueg, On implementing omega in systems with weak reliability and synchrony assumptions, Distributed Computing 21 (4) (2008).
[3] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, Sam Toueg, Stable leader election, in: DISC'01: Proceedings of the 15th International Conference on Distributed Computing, Springer-Verlag, 2001, pp. 108–122. ISBN 3-540-42605-1.
[4] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, Sam Toueg, Communication-efficient leader election and consensus with limited link synchrony, in: Proceedings of the 23th ACM Symposium on Principles of Distributed Computing, PODC'04, ACM Press, St. John's, Newfoundland, Canada, 2004, pp. 328–337. doi:10.1145/1011767.1011816. ISBN 1-58113-802-4.
[5] Bowen Alpern, Fred B. Schneider, Defining liveness, Information Processing Letters 21 (4) (1985) 181–185.
[6] Emmanuelle Anceaume, Antonio Fernández, Achour Mostéfaoui, Gil Neiger, Michel Raynal, A necessary and sufficient condition for transforming limited accuracy failure detectors, Journal of Computer and System Sciences 68 (1) (2004) 123–133. ISSN 0022-0000.
[7] Hagit Attiya, Cynthia Dwork, Nancy Lynch, Larry Stockmeyer, Bounds on the time to reach agreement in the presence of timing uncertainty, Journal of the ACM 41 (1) (1994) 122–152. doi:10.1145/174644.174649. ISSN 0004-5411.
[8] Martin Biely, Josef Widder, Optimal message-driven implementation of Omega with mute processes, in: Proceedings of the Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS 2006, in: LNCS, vol. 4280, Springer Verlag, Dallas, TX, USA, 2006, pp. 110–121.
[9] Martin Biely, Josef Widder, Optimal message-driven implementations of omega with mute processes, ACM Transactions on Autonomous and Adaptive Systems 4 (1) (2009) Article 4, 22 pages.
[10] Walter B. Carver, Systems of linear inequalities, Annals of Mathematics 23 (1921) 212–220.
[11] Tushar Deepak Chandra, Sam Toueg, Unreliable failure detectors for reliable distributed systems, Journal of the ACM 43 (2) (1996) 225–267.
[12] Bernadette Charron-Bost, Rachid Guerraoui, André Schiper, Synchronous system and perfect failure detector: solvability and efficiency issues, in: Proceedings of the IEEE Int. Conf. on Dependable Systems and Networks, DSN'00, IEEE Computer Society, New York, USA, 2000, pp. 523–532.
[13] Bernadette Charron-Bost, Shlomi Dolev, Jo Ebergen, and Ulrich Schmid (Eds.), Fault-Tolerant Distributed Algorithms on VLSI Chips, Schloss Dagstuhl, Germany, September 2008. http://drops.dagstuhl.de/opus/frontdoor.php?source_opus=1927.
[14] Reinhard Diestel, Graph Theory, 3rd ed., Springer, 2006.
[15] Danny Dolev, Joseph Y. Halpern, H. Raymond Strong, On the possibility and impossibility of achieving clock synchronization, Journal of Computer and System Sciences 32 (1986) 230–250.
[16] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus, Journal of the ACM 34 (1) (1987) 77–97.
[17] Cynthia Dwork, Nancy Lynch, Larry Stockmeyer, Consensus in the presence of partial synchrony, Journal of the ACM 35 (2) (1988) 288–323.
[18] Jo C. Ebergen, A formal approach to designing delay-insensitive circuits, Distributed Computing 5 (1991) 107–119.
[19] Markus Ferringer, Gottfried Fuchs, Andreas Steininger, Gerald Kempf, VLSI implementation of a fault-tolerant distributed clock generation, in: IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, DFT2006, October 2006, pp. 563–571.
[20] Christof Fetzer, The message classification model, in: Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, ACM Press, Puerto Vallarta, Mexico, 1998, pp. 153–162. doi:10.1145/277697.277726. ISBN 0-89791-977-7.
[21] Christof Fetzer, Ulrich Schmid, Brief announcement: on the possibility of consensus in asynchronous systems with finite average response times, in: Proceedings of the 23th ACM Symposium on Principles of Distributed Computing, PODC'04, Boston, Massachusetts, 2004, p. 402.
[22] Christof Fetzer, Ulrich Schmid, Martin Süßkraut, On the possibility of consensus in asynchronous systems with finite average response times, in: Proceedings of the 25th International Conference on Distributed Computing Systems, ICDCS'05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 271–280.
[23] Michael J. Fischer, Nancy A. Lynch, M.S. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM 32 (2) (1985) 374–382.

[24] Matthias Fuegger, Ulrich Schmid, Gottfried Fuchs, Gerald Kempf, Fault-tolerant distributed clock generation in VLSI systems-on-chip, in: Proceedings of the Sixth European Dependable Computing Conference, EDCC-6, IEEE Computer Society Press, 2006, pp. 87–96. doi:10.1109/EDCC.2006.11.
[25] Vassos Hadzilacos, Sam Toueg, Fault-tolerant broadcasts and related problems, in: Sape Mullender (Ed.), Distributed Systems, 2nd ed., Addison-Wesley, 1993, pp. 97–145 (Chapter 5).
[26] Martin Hutle, Dahlia Malkhi, Ulrich Schmid, Lidong Zhou, Brief announcement: chasing the weakest system model for implementing omega and consensus, in: Proceedings Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS 2006, in: LNCS, Springer Verlag, Dallas, USA, 2006, pp. 576–577.
[27] Martin Hutle, Dahlia Malkhi, Ulrich Schmid, Lidong Zhou, Chasing the weakest system model for implementing omega and consensus, IEEE Transactions on Dependable and Secure Computing 6 (October–December) (2009) 269–281.
[28] Leslie Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of ACM 21 (7) (1978) 558–565. doi:10.1145/359545.359563. ISSN 0001-0782.
[29] Gérard Le Lann, Ulrich Schmid, How to implement a timer-free perfect failure detector in partially synchronous systems. Technical Report 183/1-127, Department of Automation, Technische Universität Wien, January 2003 (Replaced by Research Report 28/2005, Institut für Technische Informatik, TU Wien, 2005).
[30] Dahlia Malkhi, Florin Oprea, Lidong Zhou, Ω meets paxos: leader election and stability without eventual timely links, in: Proceedings of the 19th Symposium on Distributed Computing, DISC'05, in: LNCS, vol. 3724, Springer Verlag, Cracow, Poland, 2005, pp. 199–213.
[31] Friedemann Mattern, Virtual time and global states of distributed systems, in: Parallel and Distributed Algorithms, North-Holland, 1989, pp. 215–226.
[32] Friedemann Mattern, On the relativistic structure of logical time in distributed systems, in: Parallel and Distributed Algorithms, Elsevier Science Publishers B.V, 1992, pp. 215–226.
[33] Heinrich Moser, Ulrich Schmid, Optimal clock synchronization revisited: upper and lower bounds in real-time systems, in: Proceedings of the International Conference on Principles of Distributed Systems, OPODIS, in: LNCS, vol. 4305, Springer Verlag, Bordeaux & Saint-Emilion, France, 2006, pp. 95–109.
[34] Achour Mostéfaoui, Michel Raynal, Solving consensus using Chandra–Toueg's unreliable failure detectors: a general quorum-based approach, in: P. Jayanti (Ed.), Distributed Computing: 13th International Symposium, DISC'99, in: Lecture Notes in Computer Science, vol. 1693, Springer-Verlag, Bratislava, Slovak Republic, 1999, pp. 49–63.
[35] Achour Mostefaoui, Eric Mourgaya, Michel Raynal, Asynchronous implementation of failure detectors, in: Proceedings of the International Conference on Dependable Systems and Networks, DSN'03, San Francisco, CA, June 22–25, 2003.
[36] Stephen Ponzio, Ray Strong, Semisynchrony and real time, in: Proceedings of the 6th International Workshop on Distributed Algorithms, WDAG'92, Haifa, Israel, November 1992, pp. 120–135.
[37] Ivan E. Sutherland, Jo Ebergen, Computers without clocks, Scientific American 287 (2) (2002) 62–69.
[38] Paul M.B. Vitányi, Distributed elections in an archimedean ring of processors, in: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, ACM Press, 1984, pp. 542–547. ISBN 0-89791-133-4.
[39] Paul M.B. Vitányi, Time-driven algorithms for distributed control. Report CS-R8510, C.W.I., May 1985.
[40] Josef Widder, Ulrich Schmid, The theta-model: achieving synchrony without clocks, Distributed Computing 22 (1) (2009) 29–47.
[41] Josef Widder, Gérard Le Lann, Ulrich Schmid, Failure detection with booting in partially synchronous systems, in: Proceedings of the 5th European Dependable Computing Conference, EDCC-5, in: LNCS, vol. 3463, Springer Verlag, Budapest, Hungary, 2005, pp. 20–37.