A Faster P Solution for the Byzantine Agreement Problem

Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu

Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
{mjd,yun,radu}@cs.auckland.ac.nz

Abstract. We propose an improved generic version of P modules, an extensible framework for recursive composition of P systems. We further provide a revised P solution for the Byzantine agreement problem, based on Exponential Information Gathering (EIG) trees, for N processes connected in a complete graph. Each process is modelled by the combination of $N + 1$ modules: one "main" module, plus one "firewall" communication module for each process (including one for itself). The EIG tree evaluation functionality is localized into a "main" single cell P module. The messaging functionality is localized into a three cells communication P module. This revised P solution improves overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds. Most of the running time, $5L$ steps, is spent on the communication overhead. We briefly discuss if single cells can solve the Byzantine agreement without support and protection from additional communication cells; we conjecture that this is not possible, within the currently accepted definitions.

Keywords: P systems, P modules, Byzantine agreement, Distributed algorithms, Modular design.

1 Introduction

Large distributed systems are typically *composed* from smaller building blocks. However, until recently, classical P systems did not offer enough support for effective programmability. Recent papers, such as [18, 17, 15], have started to address these problems. Guided by similar goals, we recently proposed a new modular framework, called *P modules*, that supports *generic objects*, *encapsulation*, *information hiding* and *recursive composition* [7]. Our proposal is compatible with any data structure based on directed arcs, i.e. it covers cell-like P systems (based on trees), hP systems (based on dags) and nP systems (based on digraphs).

In this paper, we extend this previous proposal, with *external definitions* and *external references*, which support safer and more flexible module interconnection facilities. We demonstrate its enhanced expressibility on a couple of simple examples, then we use it to provide a new and improved P systems solution to the Byzantine agreement problem.

The Byzantine agreement problem was first proposed by Pease *et al.* in 1980 [16] and further elaborated in Lamport *et al.*'s seminal paper [9]. This problem addresses a fundamental issue in complex systems: correctly functioning processes must be able to overcome their possible differences and achieve a consensus, despite arbitrarily faulty processes that can give conflicting information to different parts of the system.

The Byzantine agreement has become one of the most studied problems in distributed computing—some even consider it the "crown jewel" of distributed computing. Lynch covers many versions of this problem and their solutions, including a complete description of the classical algorithm, based on Exponential Information Gathering (EIG) trees as a data structure [10].

Recent years have seen revived interest in this problem and its solutions, to achieve higher performance or stronger resilience, in a wide variety of contexts [4, 1, 3, 11], including, for example, solutions for quantum computers [2].

To the best of our knowledge, except our previous work on Byzantine agreement problem [7], no other complete solution for P systems has been published. In the context of P systems, this problem was briefly mentioned, without solutions [6, 5]. Our solution was based on the classical EIG-based algorithm, where each EIG node was implemented by a distinct cell.

In this paper, we provide a revised P solution for the Byzantine agreement problem, based on EIG trees, for N processes connected in a complete graph. Each process is modelled by the combination of $N+1$ modules: one "main" module, plus one "firewall" communication module for each process (including one for itself). The EIG tree evaluation functionality is localized into a "main" single cell P module. The messaging functionality is localized into a communication P module with three cells. This revised P solution uses only duplex channels, uses fewer cells and rules, and improves overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds.

The rest of the paper is organized as follows. Section 2 covers a few basic preliminaries, then introduces a combinatorial definition of the EIG data structure. We describe the Byzantine agreement problem in detail in Section 3, which also includes a small case study with four processes. An extended version of P modules is formally introduced in Section 4. In Section 5, using our new modular framework, we model and develop the structure of a P systems implementation of the Byzantine agreement problem. The rules used in our design are described in Section 6, where we also discuss the correctness of our design. Finally, in Section 7, we summarize our results and discuss related open problems.

2 Preliminaries

We assume that the reader is familiar with the basic terminology and notations: functions, relations, graphs, nodes (vertices), arcs, directed graphs, dags, trees, alphabets, strings and multisets [12]. Given two sets, A, B, a subset f of their cartesian product, $f \subseteq A \times B$, is a *functional relation* if $\forall (x, y_1), (x, y_2) \in f \Rightarrow y_1 = y_2$. Obviously, any function $f : A \to B$ can be viewed a functional relation,

$\{(x, f(x)) \mid x \in A\}$, and, vice-versa, any functional relation can be viewed as a function.

We now recall a few basic concepts from combinatorial enumerations. The *integer range* from m to n is denoted by $[m, n]$, i.e. $[m, n] = \{m, m+1, \ldots, n\}$, if $m \leq n$, and $[m, n] = \emptyset$, if $m > n$. The set of *permutations* of n of length m is denoted by $P(n, m)$, i.e. $P(n, m) = \{\pi : [1, m] \to [1, n] \mid \pi \text{ is injective}\}$. A permutation π is represented by the sequence of its values, i.e. $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$, and we will often abbreviate this further as the sequence $\pi = \pi_1.\pi_2 \ldots \pi_m$. The sole element of $P(n, 0)$ is denoted by $()$, or by λ, if the context removes any possible ambiguity. Given a subrange $[p, q]$ of $[1, m]$, we define a *subpermutation* $\pi(p : q) \in P(n, q-p+1)$ by $\pi(p : q) = (\pi_p, \pi_{p+1}, \ldots, \pi_q)$. The *image* of a permutation π, denoted by $Im(\pi)$, is the set of its values, i.e. $Im(\pi) = \{\pi_1, \pi_2, \ldots, \pi_m\}$. The *concatenation* of two permutations is denoted by $\oplus$, i.e. given $\pi \in P(n, m)$ and $\tau \in P(n, k)$, such that $Im(\pi) \cap Im(\tau) = \emptyset$, $\pi \oplus \tau = (\pi_1, \pi_2, \ldots, \pi_m, \tau_1, \tau_2, \ldots, \tau_k) \in P(n, m+k)$.

An *Exponential Information Gathering* (EIG) tree $T_{N,L}$, $N \geq L \geq 0$, is a labelled (ordered) rooted tree of height L that is defined recursively as follows. The tree $T_{N,0}$ is a rooted tree with just one node, its root, labelled λ. For $L \geq 1$, $T_{N,L}$ is a rooted tree with $1 + N|T_{N-1,L-1}|$ nodes (where $|T|$ is the size of tree T), root λ, having N subtrees, where each subtree is isomorphic with $T_{N-1,L-1}$ and each node, except the root, is labelled by the least element of $[1, N]$ that is different from any ancestor node or any left sibling node. Alternatively, $T_{N,L-1}$ is isomorphic and identically labelled with the tree obtained from $T_{N,L}$ by deleting all its leaves. It is straightforward to see that there is a bijective correspondence between the permutations of $P(N, L)$ and the sequences (concatenations) of labels on all paths from root to the leaves of $T_{N,L}$. Thus, each node σ in an EIG tree $T_{N,L}$ is uniquely identified by a permutation $\pi_\sigma \in P(N, l)$, where $l \in [0, L]$ is also σ's depth, and, vice-versa, each such permutation π has a corresponding node σ_π. We will further use this node-permutation identification, while referring to nodes.

Given EIG tree $T_{N,L}$, an attribute is a function $\aleph : T_{N,L} \to V$, for some value set V; alternatively, $\aleph$ can be given as a functional subset of $\{\pi \in P(N, t) \mid t \in [0, L]\} \times V$.

See Figure 1 for an example of the EIG tree, $T_{4,2}$. Level 0 corresponds to permutation set $\{\lambda\}$. Level 1 corresponds to permutation set $\{(1), (2), (3), (4)\}$. Level 2 corresponds to permutation set $\{(1,2), (1,3), (1,4), (2,1), (2,3), (2,4), (3,1), (3,2), (3,4), (4,1), (4,2), (4,3)\}$. This tree is decorated with two attributes, α and β. Using an alternate notation for permutations (to avoid embedded parentheses), attribute α corresponds to the functional relation $\{(\lambda, 1), (1.0), (2.0), (3.1), (4.1), (1.2.0), (1.3.0), (1.4.1), (2.1.0), (2.3.0), (2.4.0), (3.1.0), (3.2.1), (3.4.1), (4.1.1), (4.2.1), (4.3.1)\}$.

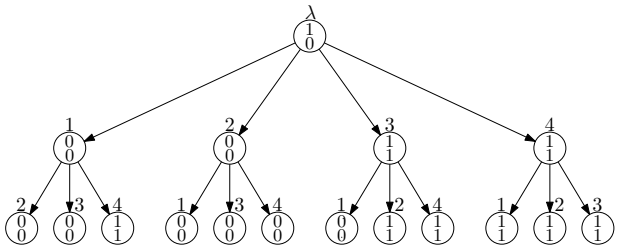


Fig. 1. A sample EIG tree, $T^3_{4,2}$, completed with two attributes, α and β. The node labels appear besides the node blob. Each node blob contains its two attribute values: the α value at the top, and the β value at the bottom.

3 The EIG-based Byzantine agreement algorithm

Each process starts with its own initial decision choice (typically different). At the end, all non-faulty processes must take the same final decision, even if the faulty processes attempt to disrupt the agreement, accidentally or intentionally.

The classical EIG-based algorithm solves the Byzantine agreement problem in the *binary decision* case ($no = 0, yes = 1$), for N *processes*, connected in a *complete graph* (where edges indicate *reliable duplex communication lines*), provided that $N \geq 3F+1$, where F is the maximum number of faulty processes. This is a *synchronous* algorithm; celebrated results (see for example [10]) show that the Byzantine agreement is *not* possible if $N \leq 3F$, in the *asynchronous* case or when the communication links are *not* reliable.

Without providing a complete description, we provide a sketch of the classical algorithm, *reformulated* on the basis of the theoretical framework introduced in Section 2. For a more complete and verbose description of this algorithm, including correctness and complexity proofs, we refer the reader to Lynch [10].

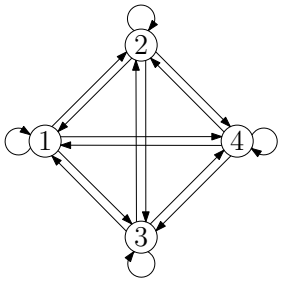
Each non-faulty process, h, has its own copy of an EIG tree, $T^h_{N,L}$, where $L = F + 1$. This tree is decorated with two attributes, $\alpha^h, \beta^h : \{\pi \in P(N,t) \mid t \in [0,L]\} \rightarrow \{0, 1, \texttt{null}\}$, where `null` designates undefined items (not yet evaluated). Attributes α^h and β^h are also known as val_h and $newval_h$ [10], or *top-down* and *bottom-up* [7]. As their alternative names suggest, α^h is first evaluated, in a top-down tree traversal, in increasing level order; next, β^h is evaluated, in a bottom-up traversal, in decreasing level order.

The algorithm works in two phases. Its *first phase* is a *messaging* phase which completes the evaluation of the top-down attribute α^h. Initially, $\alpha^h(\lambda) = v^h$, the initial choice of process h; all the other α^h and β^h values are still undefined. Next, there are L messaging rounds. At round $t \in [1, L]$, h broadcasts to all processes (including self), a reversibly encoded message which identifies its α^h values at level $t-1$, i.e. the set $\{(\pi, \alpha^h(\pi)) \mid \pi \in P(N, t-1)\}$. All other non-faulty processes broadcast messages, in a similar way. Process h decodes and processes the messages that it receives. From process f, $f \in [1, N]$, process h receives the set $\{(\pi, \alpha^f(\pi)) \mid \pi \in P(N, t-1)\}$. Each item $(\pi, \alpha^f(\pi))$, where $f \notin Im(\pi)$, is

used to assign further α^h values, to the next level down the EIG tree, by setting $\alpha^h(\pi \oplus f) = \alpha^f(\pi)$; items where $f \in Im(\pi)$ are silently discarded. As this formula suggests, it is indeed *critical* that h "knows" the origin f of each received message and that this origin mark cannot be faked by faulty processes. Wrong or missing values are replaced by the value of a predefined default parameter, $W \in \{0, 1\}$. Thus, there are L messaging rounds and, after the last round, all nodes are decorated with values of attribute α. In fact, only the last level α values are actually needed, to start the next phase, a practical implementation can choose to discard the other α values.

Next, the algorithm switches to its *second phase*, the evaluation of the bottom-up attribute β^h. First, for leaves, $\beta^h(\pi) = \alpha^h(\pi), \pi \in P(N, L)$. Next, given β^h values for level $t \in [1, L]$, each β^h value for the next level up, $\beta^h(\pi), \pi \in P(N, t-1)$, is evaluated on the basis of the β^h values of node π's children, i.e. on the multiset $\{\beta^h(\pi \oplus f) \mid f \in [1, N] \setminus Im(\pi)\}$, using a local majority voting scheme: $\beta^h(\pi) = 0$, if a strict majority of the above multiset values are 0; or, $\beta^h(\pi) = 1$, if a strict majority of the above multiset values are 1; or, $\beta^h(\pi) = W$ (the same default parameter mentioned above), if there is a tie. At the end, the β^h value for the EIG root, $\beta^h(\lambda)$, is process h's final decision. All non-faulty processes will simultaneously reach the same final decision; any decision taken by faulty nodes is not relevant.

Example 1 (Sample Byzantine scenario). Consider a Byzantine scenario with $N = 4$ and $F = 1$, thus $L = 2$. Assume that processes 1, 2, 3 and 4 start with initial choices 0, 0, 1, and 1, respectively. Further, assume that process 1 is faulty and these four processes exchange the messages described in Figure 2. For a more verbose description of this example, please see [7].



Process	1	2	3	4
Initial choice	0	0	1	1
Faulty	Yes	No	No	No
Round 1 messages	(λ, x)	$(\lambda, 0)$	$(\lambda, 1)$	$(\lambda, 1)$
Round 2 messages	$(1,0)(2,0)$ $(3,y)(4,1)$	$(1,0)(2,0)$ $(3,1)(4,1)$	$(1,0)(2,0)$ $(3,1)(4,1)$	$(1,1)(2,0)$ $(3,1)(4,1)$
Final decision	?	0	0	0

Fig. 2. A sample Byzantine scenario, $N = 4$, $F = 1$, where process 1 is faulty. Each of the non-faulty processes, 1, 2 and 3, broadcasts identical messages to each of the four processes. The faulty process 1 sends conflicting messages. In our scenario, $x = 0$, in the message sent to 1, 2 and 3, but $x = 1$, in the message sent to 4. Also, $y = 1$, in the message sent to 1, 2 and 4, but $y = 0$, in the message sent to 3. The second phase is not detailed here, except the common final decisions (the question mark indicates an irrelevant value).

The second phase is illustrated in Figure 1, for process 3. The EIG tree owned by process 3, $T^3_{4,2}$, is shown completed with all attribute values. The α^3 values are filled from messages received in the two messaging rounds, as indicated in Figure 2. The β^3 values are evaluated as required by the algorithm, by a majority voting scheme. The evaluation of $\beta^3(\lambda)$ reaches a tie, on multiset $\{0, 0, 1, 1\}$, which has two 0's and two 1's; this tie is broken using the default value, here we assume $W = 0$. Thus, $\beta^3(\lambda) = 0$ is the final decision of process 3, which is different from its initial choice, $\alpha^3(\lambda) = 1$. A similar argument shows that all other non-faulty processes, 2 and 4, end with the same final decision, 0, thereby achieving the required agreement, despite starting with different initial choices and the conflicting messages sent by faulty process 1.

4 P modules

For this section, and the rest of the paper, we assume familiarity with P systems [13, 14], nP systems [14] and hP systems [12]. We will also use a terminology inspired from standard modular design.

Intuitively, a P module is one of the above P systems, with additional features, required for its further assembly into a larger P module. A P module exposes the following additional features, collectively called *generic parameters*, which can be further *instantiated*, when the current module is *combined* with other modules:

- Besides objects of the initial alphabet, rules can also use *generic symbols*. A generic symbol, abbreviated as `sym`, can be instantiated to one of the already existing objects or as a new object (thereby extending the initial alphabet).
- Cells can be designated as *external definitions*. An external definition indicates either the start or the end of a potential arc and is abbreviated as $\texttt{def}_{\uparrow}$ or as $\texttt{def}_{\downarrow}$, respectively.
- Arcs can be designated as *external arcs*, indicating potential arcs between existing cells and external definitions of other modules. An external arc has one uninstantiated start or end cell, called an *external reference*, which is abbreviated as $\texttt{ref}_{\uparrow}$ or as $\texttt{ref}_{\downarrow}$, respectively.
- An *external arc* can be instantiated by identifying its external reference to a matching external definition from another module, i.e. either its $\texttt{ref}_{\downarrow}$ reference to a $\texttt{def}_{\downarrow}$ definition, or its $\texttt{ref}_{\uparrow}$ reference to a $\texttt{def}_{\uparrow}$ definition.

Figure 3 illustrates these intuitive ideas. Module Π_1 offers a $\texttt{def}_{\downarrow}$ definition, τ_1, and uses a $\texttt{ref}_{\downarrow}$ reference, Z_1. Module Π_2, which can be viewed as a copy of Π_1, offers a $\texttt{def}_{\downarrow}$ definition, τ_2, and uses a $\texttt{ref}_{\downarrow}$ reference, Z_2. Module Π_3 is the result of their composition, after instantiating $Z_1 = \tau_2$ and $Z_2 = \tau_1$, thereby instantiating arcs (θ_1, τ_2) and (θ_2, τ_1).

Definition 1 (P module). *A* P module *is a system* $\Pi = (O, K, \delta, S, D_{\uparrow}, D_{\downarrow}, R_{\uparrow}, R_{\downarrow})$*, where:*

1. *O is a finite non-empty alphabet of* objects*;*

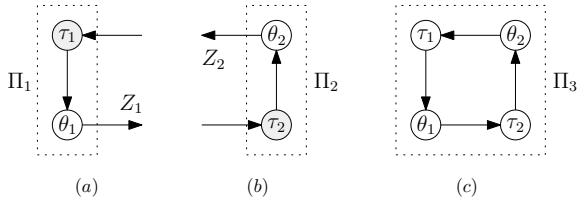


Fig. 3. Composing two simple P modules. Module Π_3 is the composition of modules Π_1 and Π_2, after instantiating $Z_1 = \tau_2$, $Z_2 = \tau_1$. External references are indicated by labels on outgoing arcs, Z_1 and Z_2, and external definitions by shaded cells, τ_1 and τ_2.

2. *K is a finite set of* cells*;*
3. *δ is a subset of $(K \times K) \cup (K \times R_\downarrow) \cup (R_\uparrow \times K)$, i.e. a set of parent-child structural arcs, representing* duplex *or* simplex *communication channels, between two existing cells or between an existing cell and an external reference;*
4. *S is a finite alphabet, disjoint of O, of generic* `sym` *objects;*
5. *$D_\uparrow$ is a subset of K, representing* $\texttt{def}_\uparrow$ *definitions;*
6. *$D_\downarrow$ is a subset of K, representing* $\texttt{def}_\downarrow$ *definitions;*
7. *$R_\uparrow$ is a finite set, disjoint of K, representing* $\texttt{ref}_\uparrow$ *references;*
8. *$R_\downarrow$ is a finite set, disjoint of K, representing* $\texttt{ref}_\downarrow$ *references.*

Let $\bar{O} = O \cup S$ be the original alphabet extended with the generic symbols. Each cell, $\sigma \in K$, has the form $\sigma = (Q, s_0, w_0, R)$, where:

- *Q is a finite set of* states*;*
- *$s_0 \in Q$ is the* initial state*;*
- *$w_0 \in \bar{O}^*$ is the* initial multiset *of objects;*
- *R is a finite* ordered *set of multiset rewriting* rules *of the general form: $s\ x \rightarrow_\alpha s'\ x'\ (u)_{\beta_\gamma}$, where $s, s' \in Q$, $x, x' \in \bar{O}^*$, $u \in \bar{O}^*$, $\alpha \in \{\texttt{min}, \texttt{max}\}$, $\beta \in \{\uparrow, \downarrow, \updownarrow, \leftrightarrow\}$, $\gamma \in \{\texttt{one}, \texttt{spread}, \texttt{repl}\} \cup K \cup R_\uparrow \cup R_\downarrow$. If $u = \lambda$, this rule can be abbreviated as $s\ x \rightarrow_\alpha s'\ x'$. The semantics of the rules and of the α, β, γ operators are further described in the rest of this section.*

Remark 1. This definition of P module subsumes several earlier definitions of P systems, hP systems and nP systems. If δ is a *tree*, then Π is essentially a tree-based P system (which can also be interpreted as a cell-like P system). If δ is a *dag*, then Π is essentially an hP system. If δ is a *digraph*, then Π is essentially an nP system.

Remark 2. Most often, our P systems are introduced semi-formally, where the objects, cells, arcs and rules are inferred from diagrams and listings. In this case, we use angular brackets to emphasize generic parameters, together with their type. For example, the generic parameters of module Π_1 of Figure 3 can be indicated as $\langle \texttt{def}_\downarrow\ \tau_1, \texttt{ref}_\downarrow\ \theta_1 \rangle$, and the whole module can be emphasized as $\Pi_1 \langle \texttt{def}_\downarrow\ \tau_1, \texttt{ref}_\downarrow\ \theta_1 \rangle$.

The rules given by the ordered set R are attempted in *weak priority* order [14]. If a rule is *applicable*, then it is *applied* and then the next rule is attempted (if any). If a rule is not applicable, then the next rule is attempted (if any). Note that state-based rules introduce an extra requirement for determining rule applicability, namely the target state indicated on the right-hand side must be the same as the previously chosen target state (if any) [13, 12]. Rules are applied under the usual immediate ("eager") evaluation of their left-hand sides and deferred ("lazy") evaluation of their right-hand sides [13].

With these conventions, one cell's ordered set of rules becomes a sequence of programming statements for a hypothetical P machine, where each rule includes a simple if-then-fi conditional test for applicability and, as we see below, some while-do-od looping facilities (`max` and `repl` operators), with some potential for in-cell parallelism, in addition to the more obvious inter-cell parallelism. State compatibility introduces another intra-cell if-then-fi conditional test, this time between rules.

The *rewriting* operator $\alpha = \texttt{min}$ indicates that the rewriting is applied once, if the rule is applicable; and $\alpha = \texttt{max}$ indicates that the rewriting is applied as many times as possible, if the rule is applicable.

The *transfer* operator $\beta = \uparrow$ indicates that the multiset u is sent "up", to the parents; $\beta = \downarrow$ indicates that the multiset u is sent "down", to the children; $\beta = \updownarrow$ indicates that the multiset u is sent both "up" and "down", to both parents and children; and $\beta = \leftrightarrow$, indicates "lateral" transfer, to the siblings (this $\leftrightarrow$ operator is not used in this paper).

The additional transfer operator $\gamma = \texttt{one}$ indicates that the multiset u is sent to one recipient (parent or child, according to the direction indicated by β). The operator $\gamma = \texttt{spread}$ indicates that the multiset u is spread among an arbitrary number of recipients (parents, children or parents and children, according to the direction indicated by β). The operator $\gamma = \texttt{repl}$ indicates that the multiset u is replicated and broadcast to all recipients (parents, children or parents and children, according to the direction indicated by β). The operator $\gamma = \sigma \in K \cup R_\uparrow \cup R_\downarrow$ indicates that the multiset u is sent to σ, if cell σ is in the direction indicated by β; otherwise, the multiset u is "lost". By convention, if cells have unique indices or are labelled and labels are locally unique, we can abbreviate $\gamma = \sigma$ by $\gamma = i$, where i is the index or label of σ.

The following examples illustrate the behaviour of these operators. Consider a cell σ, in state s and containing aa. Consider the potential application of a rule $s\ a \rightarrow_\alpha s'\ b\ (c)_{\beta_\gamma}$, by looking at specific values for α, β, γ operators:

- The rule $s\ a \rightarrow_{\texttt{min}} s'\ b\ (c)_{\uparrow_{\texttt{repl}}}$ can be applied and, after its application, cell σ will contain ab and a copy of c will be sent to each of σ's parents.
- The rule $s\ a \rightarrow_{\texttt{max}} s'\ b\ (c)_{\uparrow_{\texttt{repl}}}$ can be applied and, after being applied twice, cell σ will contain bb and a copy of cc will be sent to each of σ's parents.
- The rule $s\ a \rightarrow_{\texttt{min}} s'\ b\ (c)_{\downarrow_{\sigma'}}$ (where $\sigma' \in K$), can be applied and, after its application, cell σ will contain ab and a copy of c will be sent to σ', if σ' appears among the children of σ, otherwise, this c will be lost.

– The rule $s\ a \rightarrow_{\texttt{max}} s'\ b\ (c)_{\downarrow_{\sigma'}}$ (where $\sigma' \in K$) can be applied and, after being applied twice, cell σ will contain bb and a copy of cc will be sent to σ', if σ' appears among the children of σ, otherwise, this cc will be lost.

In this paper, we are only interested in *deterministic solutions*, and we will exclusively use the `min`, `max`, `repl`, and K operators, and avoid operators with a higher potential for non-determinism, such as `par`, `one`, `spread`.

By default, the channels are *duplex*, allowing simultaneous transmissions from both ends. Although we do not use them here, *simplex* channels are also available in our model; a simplex channel indicates a single open direction, either from parent to child, or from child to parent (thus there is no necessary relation between the structural directions and communication direction); messages sent in the other direction are silently "lost".

Given an arbitrary finite set of P modules, we can construct a higher level P module by instantiating some of their external references to some of their external definitions, which implicitly instantiates some new arcs, and by instantiating some of their unspecified symbols. This construction requires that the original P modules are disjoint, in the sense specified below.

Consider a finite family of n P modules, $\mathcal{P} = \{\Pi_i \mid i \in [1,n]\}$, where $\Pi_i = (O_i, K_i, \delta_i, S_i, D_{\uparrow i}, D_{\downarrow i}, R_{\uparrow i}, R_{\downarrow i})$, $i \in [1,n]$. This family $\mathcal{P}$ is *cell-disjoint*, if their cell sets disjoint, i.e. $K_i \cap K_j = \emptyset$, for $i, j \in [1,n]$. If required, any such family can be made cell-disjoint, by a *deep copy* process, which clones all cells and, as a convenience, automatically allocates successive indices to cloned cells (e.g., starting from cell σ, the first cloned cell is σ_1, the second is σ_2, etc). However, a good practice is to systematically index all cells of a P module, by labels related to the generic parameters, such that distinct copies of the same generic module are automatically cell-disjoint. We will generally follow this convention.

Given a family $\mathcal{P}$, the result of a composition depends on the actual instantiations, i.e. which unspecified symbols are instantiated and which external references and definitions are matched. Symbol instantiation is specified by a partial mapping $\omega : \bigcup_{i\in[1,n]} S_i \rightarrow \Omega$, where Ω is a universal alphabet, covering all alphabets used in a given application. The symbols that have been instantiated are defined by the domain of ω, i.e. $\mathrm{Dom}(\omega)$, and their assigned objects by the image of ω, i.e. $\mathrm{Im}(\omega)$. External references are similarly matched to external definitions by two partial mappings, $\rho_\uparrow : \bigcup_{i\in[1,n]} R_{\uparrow i} \rightarrow \bigcup_{i\in[1,n]} D_{\uparrow i}$, $\rho_\downarrow : \bigcup_{i\in[1,n]} R_{\downarrow i} \rightarrow \bigcup_{i\in[1,n]} D_{\downarrow i}$. A previously uninstantiated arc (σ, x), $\sigma \in K_i$, $x \in R_{\downarrow i}$, $i \in [1,n]$, is instantiated as $(\sigma, \rho_\downarrow(x))$, and a previously uninstantiated arc (x, σ), $\sigma \in K_i$, $x \in R_{\uparrow i}$, $i \in [1,n]$, is instantiated as $(\rho_\uparrow(x), \sigma)$.

Definition 2 (P modules composition). *The P module* $\Psi = (O, K, \delta, S, D_\uparrow, D_\downarrow, R_\uparrow, R_\downarrow)$ *is a* composition *of the P module family* $\mathcal{P}$*, if:*

– $\mathcal{P}$ *is cell-disjoint;*
– ω*,* $\rho_\uparrow$*,* $\rho_\downarrow$ *are the partial mappings which define the instantiation (as previously introduced);*
– $O = \bigcup_{i\in[1,n]} O_i \cup \mathrm{Im}(\omega)$*;*
– $K = \bigcup_{i\in[1,n]} K_i$*;*

– $\delta = \{(\hat{\rho}_\uparrow(\sigma), \hat{\rho}_\downarrow(\sigma)) \mid (\sigma, \tau) \in \bigcup_{i\in[1,n]} \delta_i\}$, *where* $\hat{\rho}_\uparrow$ *and* $\hat{\rho}_\downarrow$ *are defined by* $\hat{\rho}_\uparrow(\sigma) = \sigma \in \mathrm{Dom}(\rho_\uparrow)\ ?\ \rho_\uparrow(\sigma) : \sigma$, $\hat{\rho}_\downarrow(\sigma) = \sigma \in \mathrm{Dom}(\rho_\downarrow)\ ?\ \rho_\downarrow(\sigma) : \sigma$;
– $S = \bigcup_{i\in[1,n]} S_i \setminus \mathrm{Dom}(\omega)$;
– $D_\uparrow \subseteq \bigcup_{i\in[1,n]} D_{\uparrow i}$, $D_\downarrow \subseteq \bigcup_{i\in[1,n]} D_{\downarrow i}$;
– $R_\uparrow = \bigcup_{i\in[1,n]} R_{\uparrow i} \setminus \mathrm{Dom}(\rho_\uparrow)$, $R_\downarrow = \bigcup_{i\in[1,n]} R_{\downarrow i} \setminus \mathrm{Dom}(\rho_\downarrow)$.

In this case, the P modules in $\mathcal{P}$ *are called* components *of* Ψ. *We omit here the straightforward but lengthy details of the required translations of the rule-sets. Note that we can keep any of the previous external definitions, even those matched by external references (for further matches), thus the instantiations alone do not completely define the composition result.*

This modular approach provides encapsulation, information hiding and recursive composition, facilitating the design of P programs for complex algorithms. We now give a more elaborated example, where we use modules both to build a more complex system and to argue about its properties.

Example 2 (A composite P module for computing the GCD).

The left diagram of Figure 4 illustrates a logical flowchart for computing the standard Euclidean *Greatest Common Divisor* (GCD) algorithm, where, initially, x and y are the two positive integer inputs, and, on termination, the final value of y is the resulting GCD. For illustrative purposes, this design is recursively built, starting from four elementary blocks, Π_1, Π_2, Π_3 and Π_4. We first combine Π_1, Π_2 and Π_3 into a high-level block Π_5, and then combine Π_5 and Π_4 into a higher-level block Π_6. Our correctness proofs can start from the elementary blocks and then follow up this recursive composition.

The right diagram of Figure 4 illustrates a closely related recursive composition of four elementary P modules, Π_1, Π_2, Π_3 and Π_4, which solves the same problem (modulo a straightforward encoding).

Module $\Pi_1\langle \mathtt{def}_\downarrow\ \sigma_1, \mathtt{ref}_\downarrow\ Y_1, \mathtt{ref}_\downarrow\ N_1\rangle$ contains a single cell, σ_1, which also appears as an external $\mathtt{def}_\downarrow$ definition, and makes external $\mathtt{ref}_\downarrow$ references to two unspecified cells, Y_1 and N_1. Using the ruleset which follows this paragraph, a straightforward argument shows that, if cell σ_1 receives x copies of a and y copies of b, then Π_1 ends by sending x copies of a and y copies of b; to Y_1, if $x \geq y$; or to N_1, otherwise.

1. $s_0\ ab \rightarrow_{\mathtt{max}}\ s_1\ cde$
2. $s_1\ a \rightarrow_{\mathtt{max}}\ s_2\ c$
3. $s_1\ b \rightarrow_{\mathtt{max}}\ s_3\ d$
4. $s_1\ e \rightarrow_{\mathtt{max}}\ s_2$
5. $s_1\ e \rightarrow_{\mathtt{max}}\ s_3$
6. $s_2\ c \rightarrow_{\mathtt{max}}\ s_0\ (a)_{\downarrow_{Y_1}}$
7. $s_2\ d \rightarrow_{\mathtt{max}}\ s_0\ (b)_{\downarrow_{Y_1}}$
8. $s_3\ c \rightarrow_{\mathtt{max}}\ s_0\ (a)_{\downarrow_{N_1}}$
9. $s_3\ d \rightarrow_{\mathtt{max}}\ s_0\ (b)_{\downarrow_{N_1}}$

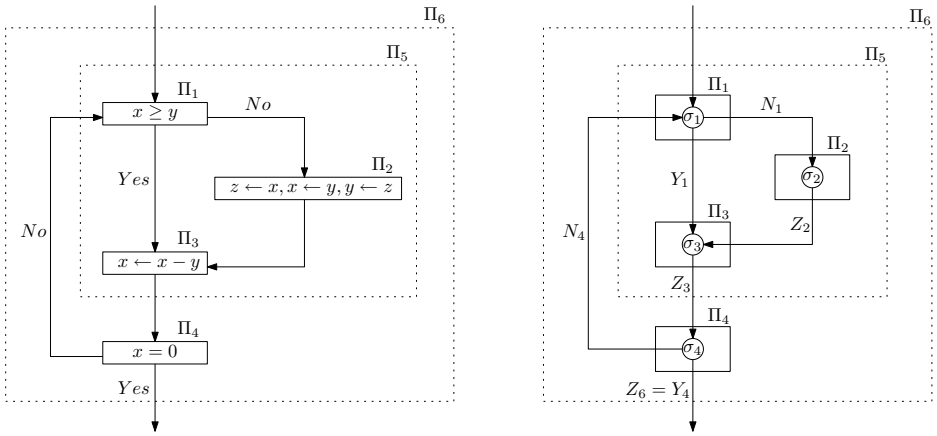


Fig. 4. Diagrams for computing GCD: left, a logical flowchart; right, a recursive composition of P modules.

Module $\Pi_2\langle \mathtt{def}_\downarrow\ \sigma_2, \mathtt{ref}_\downarrow\ Z_2\rangle$ contains a single cell σ_2, which also appears as an external $\mathtt{def}_\downarrow$ definition, and makes external $\mathtt{ref}_\downarrow$ references to one unspecified cell, Z_2. Using the ruleset which follows this paragraph, a straightforward argument shows that, if cell σ_2 receives x copies of a and y copies of b, then Π_2 ends by sending, to Z_2, y copies of a and x copies of b.

1. $s_0\ a \rightarrow_{\mathtt{max}}\ s_0\ (b)_{\downarrow_{Z_2}}$
2. $s_0\ b \rightarrow_{\mathtt{max}}\ s_0\ (a)_{\downarrow_{Z_2}}$

Module $\Pi_3\langle \mathtt{def}_\downarrow\ \sigma_3, \mathtt{ref}_\downarrow\ Z_3\rangle$ contains a single cell σ_3, which also appears as an external $\mathtt{def}_\downarrow$ definition, and makes external $\mathtt{ref}_\downarrow$ references to one unspecified cell, Z_3. Using the ruleset which follows this paragraph, a straightforward argument shows that, if cell σ_3 receives x copies of a and y copies of b, then Π_3 ends by sending, to Z_3, x' copies of a and y' copies of b, where $x' = x - \min(x, y)$, $y' = \min(x, y)$.

1. $s_0\ ab \rightarrow_{\mathtt{max}}\ s_0\ (b)_{\downarrow_{Z_3}}$
2. $s_0\ a \rightarrow_{\mathtt{max}}\ s_0\ (a)_{\downarrow_{Z_3}}$

Module $\Pi_4\langle \mathtt{def}_\downarrow\ \sigma_4, \mathtt{ref}_\downarrow\ Y_4, \mathtt{ref}_\downarrow\ N_4\rangle$ contains a single cell σ_4, which also appears as an external $\mathtt{def}_\downarrow$ definition, and makes external $\mathtt{ref}_\downarrow$ references to two unspecified cells, Y_4 and N_4. Using the ruleset which follows this paragraph, a straightforward argument shows that, if cell σ_4 receives x copies of a and y copies of b, then, if $x = 0$, Π_4 ends by sending, to Y_4, y copies of c; or, if $x \neq 0$, Π_4 ends by sending, to N_4, x copies of a and y copies of b.

1. $s_0\ a \rightarrow_{\mathtt{max}}\ s_1\ a$
2. $s_0\ b \rightarrow_{\mathtt{max}}\ s_0\ (c)_{\downarrow_{Y_4}}$
3. $s_1\ a \rightarrow_{\mathtt{max}}\ s_0\ (a)_{\downarrow_{N_4}}$
4. $s_1\ b \rightarrow_{\mathtt{max}}\ s_0\ (b)_{\downarrow_{N_4}}$

We first combine Π_1, Π_2 and Π_3, using the following generic instantiations: $Y_1 = \sigma_3, N_1 = \sigma_2, Z_2 = \sigma_3$; this instantiates the connecting arcs (σ_1, σ_3), (σ_1, σ_2) and (σ_2, σ_3). The result is a composite module with two generic parameters, an external definition and an external reference, $\Pi_5\langle \mathtt{def}_\downarrow\ \sigma_1, \mathtt{ref}_\downarrow\ Z_3\rangle$. Module Π_5's behaviour can be inferred from the behaviour of its constituent modules, Π_1, Π_2 and Π_3. If cell σ_1 receives x copies of a and y copies of b, then Π_5 ends by sending, to Z_3, x' copies of a and y' copies b, where $x' = \max(x, y) - \min(x, y)$, $y' = \min(x, y)$.

We further combine Π_5 and Π_4, using the following generic instantiations: $Z_3 = \sigma_4, N_4 = \sigma_1$; this instantiates the connecting arcs (σ_3, σ_4) and (σ_4, σ_1). The result is another composite module with two generic parameters, an external definition and an external reference, $\Pi_6\langle \mathtt{def}_\downarrow\ \sigma_1, \mathtt{ref}_\downarrow\ N_4\rangle$, which can also be renamed as $\Pi_6\langle \mathtt{def}_\downarrow\ \sigma_1, \mathtt{ref}_\downarrow\ Z_6\rangle$. Module Π_6's behaviour can be inferred from the behaviour of its constituent modules, Π_5 and Π_4. If cell σ_1 receives x copies of a and y copies of b, then Π_6 ends by sending, to Z_6, z copies of c, where $z = \gcd(x, y)$.

5 Revised Byzantine agreement solution

Our revised P solution for the Byzantine agreement problem is still based on Exponential Information Gathering (EIG) trees, for N processes connected in a complete graph, with "hardcoded" parameters L, the EIG tree height, and W, the default value (for missing or wrong messages). Each non-faulty process h, $h \in [1, N]$, is modelled by a "process" module, Π_h, which is a combination of $N + 1$ modules: one "main" module, Ψ_h, which provides the main EIG functionality; plus one "firewall" communication module, Γ_{hf}, for each process f, $f \in [1, N]$. Compared to the previous solution, this revised P solution uses fewer cells and rules and only duplex channels.

Elementary modules are illustrated in Figure 5. Module Ψ_h has a single cell, ψ_h, which is also offered as an external definition, $\langle \mathtt{def}_\downarrow\ \psi_h\rangle$. Module Γ_{hf} has three cells, γ_{hf}, γ'_{hf}, γ''_{hf}, offers one external definition $\langle \mathtt{def}_\downarrow\ \gamma'_{hf}\rangle$, and uses two external references $\langle \mathtt{ref}_\downarrow\ \psi_h, \mathtt{ref}_\downarrow\ \gamma'_{fh}\rangle$.

As mentioned, process module Π_h is composed from modules $\{\Psi_h\} \cup \{\Gamma_{hf} \mid f \in [1, N]\}$; this composition instantiates arcs $\{(\gamma_{hf}, \psi_h) \mid f \in [1, N]\}$. Module Π_h offers N external definitions, $\langle \mathtt{def}_\downarrow\ \gamma'_{hf} \mid f \in [1, N]\rangle$, and uses N external references, $\langle \mathtt{ref}_\downarrow\ \gamma'_{fh} \mid f \in [1, N]\rangle$. Composite module Π_h, for $N = 4$, $L = 2$, is illustrated in Figure 6.

Any arbitrary module can play the role of a *faulty* module; however, to provide maximal adversity, it needs connection facilities similar to the expected facilities of non-faulty module. Therefore, without loss of generality, we model faulty processes by arbitrary modules Θ_h, $h \in [1, N]$, which offer N $\mathtt{def}_\downarrow$ definitions and use N $\mathtt{ref}_\downarrow$ references.

The final system is the composition of N modules from $\{\Pi_h \mid h \in [1, N]\} \cup \{\Theta_h \mid h \in [1, N]\}$, that instantiates arcs $\{(\gamma''_{hf}, \gamma'_{fh}) \mid h, f \in [1, N]\}$. The Byzantine agreement problem can be solved if at least $\lfloor 2(N-1)/3 \rfloor$ of these modules

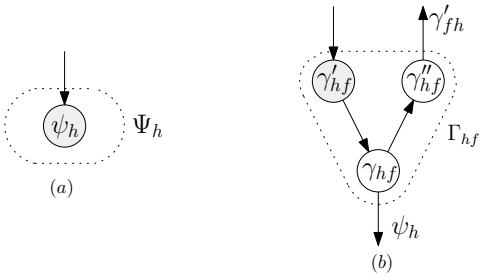


Fig. 5. Elementary P modules for Byzantine agreement: (a) main module Ψ_h, (b) communication module Γ_{hf}. The $\texttt{ref}_\downarrow$ references are indicated by labels on outgoing arcs and the $\texttt{def}_\downarrow$ definitions are indicated by shaded cells.

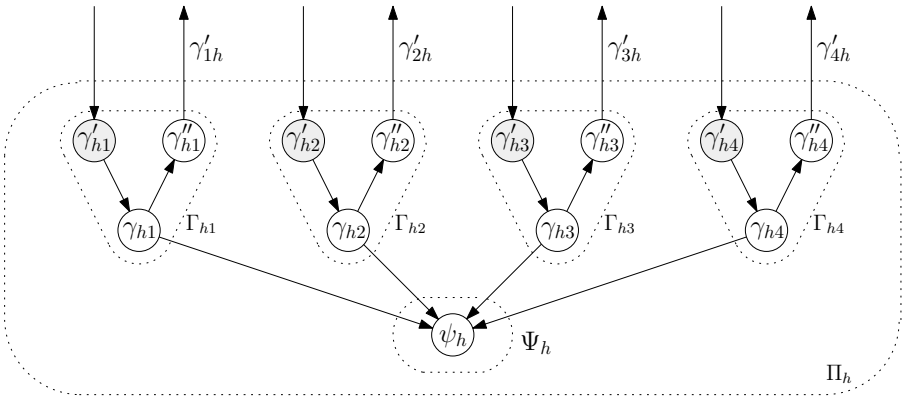


Fig. 6. The process module Π_h, for $N = 4$, $L = 2$. Its $\texttt{ref}_\downarrow$ references are indicated by labels on outgoing arcs (γ'_{1h}, γ'_{2h}, γ'_{3h}, γ'_{4h}) and its $\texttt{def}_\downarrow$ definitions are indicated by shaded cells (γ'_{h1}, γ'_{h2}, γ'_{h3}, γ'_{h4}).

are non-faulty, i.e. from the Π_h family. Figure 7 shows a fragment with two modules, Π_2 and Π_3, of the four process modules of the final system, for the case $N = 4$, $L = 2$.

6 Rules and correctness

The following objects are used by all non-faulty processes: $\Omega = \{x^v_\pi \mid v \in \{0, 1, ?, *\}, t \in [0, L-1], \pi \in P(N, t)\} \cup \{x^v_\pi \mid v \in \{0, 1\}, \pi \in P(N, L)\}$. Object x^v_π is viewed as an *encoding* of pair (π, v), which associates a permutation π, i.e. an EIG node, with a value v; object x^v_λ can be abbreviated as v. Encodings with binary digit values, x^v_π, $v \in \{0, 1\}$, are called *value objects* and represent EIG nodes with known α^h or β^h values. Where process number h is clearly inferred from the context, we will use α and β instead of α^h and β^h, respectively. Encodings with asterisks, x^*_π, are called *placeholder objects*, and represent EIG

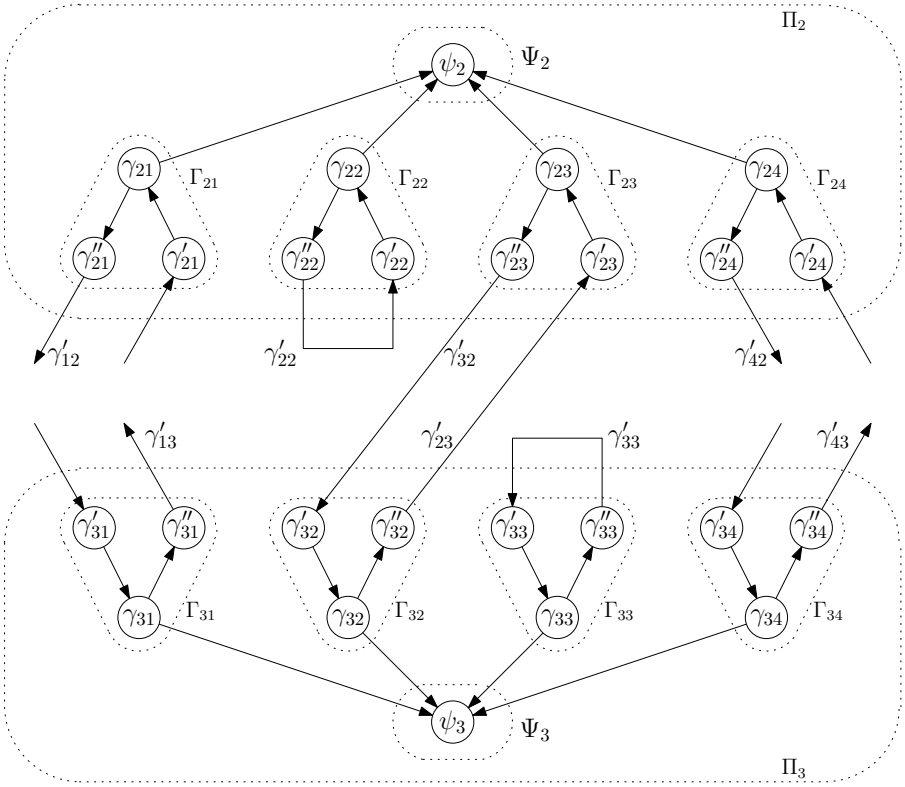


Fig. 7. Interconnection details between process modules Π_2 and Π_3, for $N = 4$, $L = 2$.

nodes with still undefined β^h values. Encodings with question marks, $x^?_\pi$, are called *template objects*, and are used to filter incoming messages which are not well-formed.

Besides encodings in Ω, faulty process can send any other available *objects*. The set of all possible objects is denoted by a universal set $\mho$, i.e. $\mho = \Omega \cup \{$*all other objects which can be sent by a faulty process*$\}$.

The following sets of objects, which appear in several of the subsequent sections, are defined here:

$$
\begin{aligned}
R_t^{\alpha^h} &= \{x_\pi^{\alpha^h(\pi)} \mid \pi \in P(N,t)\}, t \in [0,L] \\
R_t^{\beta^h} &= \{x_\pi^{\beta^h(\pi)} \mid \pi \in P(N,t)\}, t \in [0,L] \\
R_t^{v} &= \{x_\pi^{v} \mid \pi \in P(N,t)\}, t \in [0,L], v \in \{0,1,?,*\} \\
R_t^{0,1} &= \{x_\pi^{v} \mid \pi \in P(N,t), v \in \{0,1\}\}, t \in [0,L]
\end{aligned}
$$

The first three sets, $R_t^{\alpha^h}$, $R_t^{\beta^h}$, R_t^{v}, describe functional relations, on the underlying permutation π; the last one, $R_t^{0,1}$, is not. Where h can be unambigu-

ously inferred from the context, the superscript h can be dropped from attribute names, α and β, i.e. in such cases, $R_t^{\alpha} = R_t^{\alpha^h}$, $R_t^{\beta} = R_t^{\beta^h}$.

6.1 Rule sequence for Ψ_h's cell ψ_h

According to the following rules, and as illustrated in the diagram of Figure 8, cell ψ_h progresses through states s_0 (start state), s_1, ..., s_L, e_L, e_{L-1}, ..., e_0 (final state).

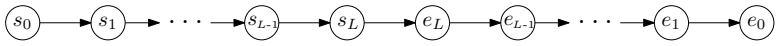


Fig. 8. State diagram for module Ψ_h, i.e. cell ψ_h.

1. $s_t\ x_\pi^v \rightarrow_{\texttt{min}} s_{t+1}\ x_\pi^*\ (x_\pi^v x_\pi^?)_{\uparrow_{\texttt{repl}}}$, for $v \in \{0,1\}$, $t \in [0, L-1]$, $\pi \in P(N,t)$
2. $s_L\ x_\pi^v \rightarrow_{\texttt{min}} e_L\ x_\pi^v$, for $v \in \{0,1\}$, $\pi \in P(N,L)$
3. $e_{t+1}\ x_{\pi\oplus k}^0 x_{\pi\oplus l}^1 \rightarrow_{\texttt{max}} e_t$, for $v \in \{0,1\}$, $t \in [0, L-1]$, $\pi \in P(N,t)$, $k,l \in [1,N] \setminus Im(\pi)$, $k \neq l$
4. $e_{t+1}\ x_{\pi\oplus k}^v x_\pi^* \rightarrow_{\texttt{min}} e_t\ x_\pi^v$, for $v \in \{0,1\}$, $t \in [0, L-1]$, $\pi \in P(N,t)$, $k \in [1,N] \setminus Im(\pi)$
5. $e_{t+1}\ x_{\pi\oplus k}^v \rightarrow_{\texttt{max}} e_t$, for $v \in \{0,1\}, t \in [0, L-1], \pi \in P(N,t), k \in [1,N] \setminus Im(\pi)$
6. $e_{t+1}\ x_\pi^* \rightarrow_{\texttt{min}} e_t\ x_\pi^W$, for $t \in [0, L-1]$, $\pi \in P(N,t)$

The following is an itemized expansion for the case $L = 2$.

1. $s_0\ v \rightarrow_{\texttt{min}} s_1\ *\ (v?)_{\uparrow_{\texttt{repl}}}$, for $v \in \{0,1\}$
   $s_1\ x_j^v \rightarrow_{\texttt{min}} s_2\ x_j^*\ (x_j^v x_j^?)_{\uparrow_{\texttt{repl}}}$, for $v \in \{0,1\}$, $j \in [1,N]$
2. $s_2\ x_{jk}^v \rightarrow_{\texttt{min}} e_2\ x_{jk}^v$, for $v \in \{0,1\}$, $j,k \in [1,N]$, $j \neq k$
3. $e_2\ x_{jk}^0 x_{jl}^1 \rightarrow_{\texttt{max}} e_1$, for $j,k,l \in [1,N]$, $j \neq k$, $j \neq l$, $k \neq l$
   $e_1\ x_j^0 x_k^1 \rightarrow_{\texttt{max}} e_0$, for $j,k \in [1,N]$, $j \neq k$
4. $e_2\ x_{jk}^v x_j^* \rightarrow_{\texttt{min}} e_1\ x_j^v$, for $v \in \{0,1\}$, $j,k \in [1,N]$, $j \neq k$
   $e_1\ x_j^v * \rightarrow_{\texttt{min}} e_0\ v$, for $v \in \{0,1\}$, $j \in [1,N]$
5. $e_2\ x_{jk}^v \rightarrow_{\texttt{max}} e_1$, for $v \in \{0,1\}$, $j,k \in [1,N]$, $j \neq k$
   $e_1\ x_j^v \rightarrow_{\texttt{max}} e_0$, for $v \in \{0,1\}$, $j \in [1,N]$
6. $e_2\ x_j^* \rightarrow_{\texttt{min}} e_1\ x_j^W$, for $j \in [1,N]$
   $e_1\ * \rightarrow_{\texttt{min}} e_0\ W$

Initially, cell ψ_h is in state s_0 and contains a value object describing its initial choice, v^h. Cell ψ_h is a $\texttt{def}_\downarrow$ definition, thus, if properly connected, is able to receive and send objects from/to one or more parent cells, belonging to one or more "parent" modules. Cell ψ_h works in two phases, which roughly correspond to the two phases of the classical EIG-based algorithm: first, a *messaging phase*, implemented by rule groups 1 and 2, and, secondly, a bottom-up phase, implemented by rule groups 3, 4, 5 and 6.

The *external* behaviour of cell ψ_h's messaging phase, and therefore of module Ψ_h, is governed by the external contract described by the following paragraph.

Module Ψ_h's messaging phase takes $L+1$ rounds, indexed by $[0, L]$. The first round, 0, starts immediately, triggered by the presence of the initial value object. Each other round t, $t \in [1, L]$, starts after receiving, collectively from its parent modules, the set $R_t^{\alpha^h}$. Each round t, except the last, $t \in [0, L-1]$, ends by sending up, to each parent module, by replication, the set $R_t^{\alpha^h} \cup R_t^?$. Each round t, $t \in [0, L]$, is completed in exactly one P step. Module Ψ_h is idle between successive rounds.

Each $x_\pi^{\alpha(\pi)}$ sent up is accompanied by a corresponding template object, $x_\pi^?$, which is used, by cell ψ_h's parents, to build a filter, for next round value objects.

Messaging rounds here have different granularity and boundaries than in Section 3: in the classical EIG algorithm, a round starts by sending and continues by receiving messages; here, a round is triggered by receiving objects (except the first round, which is triggered by the initial choice) and continues by processing and sending objects (except the last round, which does only processing). This explains why, here, this messaging phase has $L+1$ rounds, but the messaging phase of Section 3 has L rounds.

The *internal* behaviour of cell ψ_h is very important, during both phases. At all steps, cell ψ_h's contents can be viewed as forming a *virtual EIG tree*, similar, but not identical, to the classical EIG tree of process h. The nodes of this virtual tree are represented by objects from the sets R_t^α, R_t^β, R_t^*, $t \in [0, L]$.

This virtual tree is gradually built, in increasing level order, during the messaging phase, and it gradually shrinks, in decreasing level order, during the bottom-up phase. Messaging round t, $t \in [0, L]$, is triggered by receiving value objects R_t^α. Just before receiving R_t^α, the virtual tree is given by $\cup_{l \in [0,t-1]} R_l^*$. Then, receiving R_t^α extends this virtual tree to $(\cup_{l \in [0,t-1]} R_l^*) \cup R_t^\alpha$. Next, if $t \in [0, L-1]$, messaging round t transforms the tree by "replacing" α value objects by placeholders, i.e. the virtual tree is now given by $(\cup_{l \in [0,t]} R_l^*)$ (which maintains the invariant); otherwise, if $t = L$, the virtual tree is not changed.

Using rule groups 1 and 2, an induction argument on round number t, $t \in [0, L]$, shows that cell ψ_h maintains its external contract for round t, gradually builds its virtual tree as mentioned, and transits from state s_t to state s_{t+1}, except for round $t = L$, when it transits from state s_L to state e_L.

As an example, consider that cell ψ_3 corresponds to process 3 in the scenario of Example 1. Figures 9 (a,b,c,d,e) illustrate the gradual completion of the virtual EIG tree, via the three messaging rounds of cell ψ_3.

These trees are represented with the help of the following sets; $R_0^* = \{x_\lambda^*\}$, $R_1^* = \{x_1^*, x_2^*, x_3^*, x_4^*\}$, $R_0^\alpha = \{x_\lambda^1\}$, $R_1^\alpha = \{x_1^0, x_2^0, x_3^1, x_4^1\}$, $R_2^\alpha = \{x_{12}^0, x_{13}^0, x_{14}^1, x_{21}^0, x_{23}^0, x_{24}^0, x_{31}^0, x_{32}^1, x_{34}^1, x_{41}^1, x_{42}^1, x_{43}^1\}$. $R_2^\beta = R_2^\alpha$, $R_1^\beta = \{x_1^0, x_2^0, x_3^1, x_4^1\}$, $R_0^\beta = \{x_\lambda^0\}$. For a comparison, see also Figure 1.

There are $L+1$ messaging rounds. Cell ψ_h completes each round in one P step and stays idle between rounds. Thus, the messaging phase will take $L(1+\theta)+1$ steps in total, where θ is the interval when cell ψ_h is idle, assumed fixed. (As we will see later, this interval is indeed fixed, $\theta = 4$, making a total of five P steps,

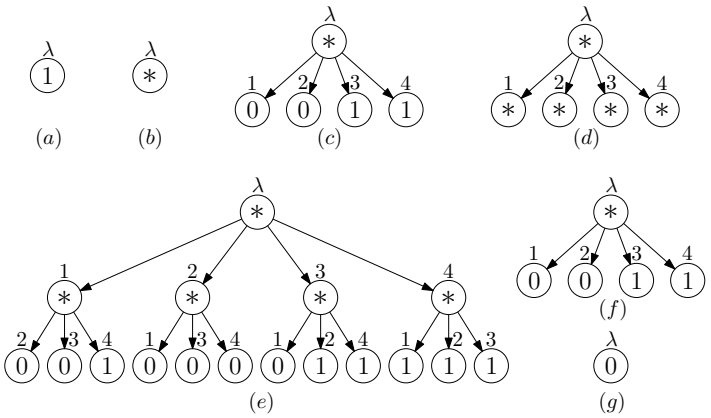


Fig. 9. The evolution and involution of cell ψ_3's virtual EIG tree. Message round 0: (a) ⇒ (b); Message round 1: (c) ⇒ (d); Message round 2: (d) ⇒ (e); Bottom-up step 1 : (e)⇒ (f); Bottom-up step 2 : (f) ⇒ (g).

required for sending objects from cell ψ_h to another cell ψ_f, $f \in [1, N]$, and vice-versa).

State e_L triggers the start of the *bottom-up evaluation* of attribute β, on the virtual EIG tree, in decreasing level order. Each level evaluation takes exactly one P step. Because $\beta(\pi) = \alpha(\pi)$, for $\pi \in P(N, L)$, the virtual tree at state e_L can be alternatively viewed as $(\cup_{t\in[0,L-1]} R_t^*) \cup R_L^\beta$. An induction argument on $t = L, L-1, \ldots, 1, 0$, shows that, after $L-t$ steps since it has reached state e_L, cell ψ^h transits to state e_t and the virtual tree "shrinks" to $(\cup_{u\in[0,t-1]} R_u^*) \cup R_t^\alpha$.

Intuitively, at each transition from e_{t+1} to e_t, the β value objects for level $t+1$, R_{t+1}^β, are removed, and the placeholder objects for level t, R_t^*, are "replaced" by β value objects for level t, R_t^β.

Rule groups 3, 4, 5 and 6 run the required strict majority voting scheme, transiting from state e_{t+1} to state e_t. For each EIG sibling group at level $t+1$,

- Rule group 3 cancels pairs of β value objects at level $t+1$ with opposite binary values, until there either remains only objects with the same binary value, or no β value objects at all.
- Rule group 4 takes one of the remaining β value objects at level $t+1$, if there is a strict majority, and creates a corresponding β value object at level t.
- Rule group 5 removes all superfluous remaining β value objects at level $t+1$.
- Rule group 6 is activated in the tie case and creates a β value objects at level t with the default value W.

The last step of the bottom-up iteration evaluates R_λ^β, which contains a single β value object, $x_\lambda^{\beta(\lambda)}$, where $\beta(\lambda)$ is the final decision of process h. At the same time, cell ψ_h stops, because it reaches the final state e_0.

Figures 9 (e,f,g) continue the previous example, based on process 3 of the scenario of Example 1, and illustrate how the virtual tree "shrinks" after each step of the bottom-up evaluation. Note the tie breaker required for the last bottom-up step.

Including the root, the EIG tree has $L + 1$ levels. Therefore, after receiving and processing the last round objects, which records the $\beta(\pi) = \alpha(\pi)$ values, for the leaves π, $\pi \in P(N, L)$, cell ψ_h needs L more P steps to reach the final state e_0 and evaluate the final decision value.

6.2 Rule sequences for Γ_{hf}

Conceptually, module Γ_{hf} belongs to process h and stands as a local firewall, between its main module Ψ_h and a corresponding firewall module Γ_{fh}, belonging to untrusted remote process f.

Module Γ_{hf} contains three distinct cells, γ'_{hf}, γ''_{hf}, γ_{hf}, each having its own rule sequence and states. As indicated before, on one side (the "home" side), module Γ_{hf} is connected to the main module Ψ_h, and, on the other side (the "foreign" side), module Γ_{hf} expects to be connected to module Γ_{fh} (part of a "friend-or-foe" process f). Specifically, cell γ_{hf} is connected as parent of main cell ψ_h (which is given as an external reference), cell γ''_{hf} is connected as parent of foreign cell γ'_{fh} (which is given as an external reference), and cell γ'_{hf} (which is given as an external definition) is connected as child of foreign cell γ''_{fh}.

As will be shown in the next three subsections, cells γ'_{hf}, γ''_{hf}, γ_{hf}, work in lockstep, cycling continuously through a five P steps period, each period corresponding to a complete messaging round. As shown in Section 6.1, cell ψ_h completes its messaging related tasks in short one P step activity bursts, thus module Ψ does not have its own time constraints for the messaging phase. Therefore, (a) the overall progress of module Ψ_h, during the messaging phase, is also determined by module Γ_{hf}, and (b) between successive messaging rounds, cell ψ_h stays idle for exactly four P steps, and the parameter θ, used in Section 6.1, is 4).

If counter s designates the global step number, $s = 1, 2, \ldots$, then the messaging round number is given by counter $t = (s - 1)/5$, and counter $u \in [0, 4]$, defined by $u = s - 1 \pmod 5$, indicates the current substep inside the five steps period, which is also indicated by their current state index (e.g., cell γ'_{hf}'s states are indexed as c_u, $u \in [0, 4]$). Module Γ_{hf}'s cells and their external connections are expected to switch their responsibilities according to this counter u. Provided that both processes, h and f, are *non-faulty*, the expected messaging workflow of module Γ_h can be summarized as follows:

1. when $u = 0$, external cell ψ_h is expected to send up, to cell γ_{hf}, the object set $R_t^{\alpha^h} \cup R_t^?$;
2. when $u = 1$, cell γ_{hf} is expected to send down, to cell γ''_{hf}, the object set $R_t^{\alpha^h}$;

3. when $u = 2$, cell γ''_{hf} is expected to send down, to external cell γ'_{fh}, the object set $R_t^{\alpha^h}$, and, vice-versa, external cell γ''_{fh} is expected to send down, to cell γ'_{hf}, the object set $R_t^{\alpha^f}$;
4. when $u = 3$, cell γ'_{hf} is expected to send down, to cell γ_{hf}, the object set $R_t^{\alpha^f}$;
5. when $u = 4$, cell γ_{hf} is expected to send down, to cell ψ_h, the object set $R_t^{\alpha^f}$.

If these expectations are not met, i.e. if the foreign process f is faulty, module Γ_{hf} works as firewall, protecting its associated main module Ψ_h against bad, wrongly timed and missing messages. The message flow will not stop and, instead of bad or expected but missing objects, module Ψ_h will timely receive objects recreated with the default value W. A faulty process f might receive back some of the wrong messages it has itself sent to h, but this does not harm the algorithm.

Figure 10 illustrates a fragment of this workflow, by tracing the actual messages between cells ψ_2, ψ_3, the main cells associated to processes 2 and 3, respectively, in the Byzantine scenario of Example 1.

Module Γ_{hf} is hardwired for given level L. After L messaging phases, cell γ_{hf} stops working and enters its final state. The other two cells, γ'_{hf} and γ''_{hf} continue to loop over their five step cycles. This can be easily fixed, if this is not desired. However, this will involve a state-based countdown counter, because these two cells are on the frontline towards an untrusted process f, which potentially can alter their contents at any time.

The following three subsections detail the rules of module Γ_{hf}'s three cells and discuss their behaviour.

6.3 Rule sequence for Γ_{hf}'s cell γ'_{hf}

According to the following rules, and as illustrated in Figure 11 (a), cell γ'_{hf} cycles continuously through states c_0 (start state), c_1, c_2, c_3, c_4.

1. $c_0 \rightarrow_{\texttt{min}} c_1$
2. $c_1 \rightarrow_{\texttt{min}} c_2$
3. $c_2 \rightarrow_{\texttt{min}} c_3$
4. $c_2\ o \rightarrow_{\texttt{max}} c_3$, for $o \in \mho$
5. $c_3 \rightarrow_{\texttt{min}} c_4$
6. $c_3\ x_\pi^v \rightarrow_{\texttt{min}} c_4\ (x_\pi^v)_{\downarrow_{\gamma_{hf}}}$, for $v \in \{0,1\}$, $t \in [0, L-1]$, $\pi \in P(N,t)$
7. $c_4 \rightarrow_{\texttt{min}} c_0$

The following is an itemized expansion for the case $L = 2$.

1. $c_0 \rightarrow_{\texttt{min}} c_1$
2. $c_1 \rightarrow_{\texttt{min}} c_2$
3. $c_2 \rightarrow_{\texttt{min}} c_3$
4. $c_2\ o \rightarrow_{\texttt{max}} c_3$, for $o \in \mho$
5. $c_3 \rightarrow_{\texttt{min}} c_4$

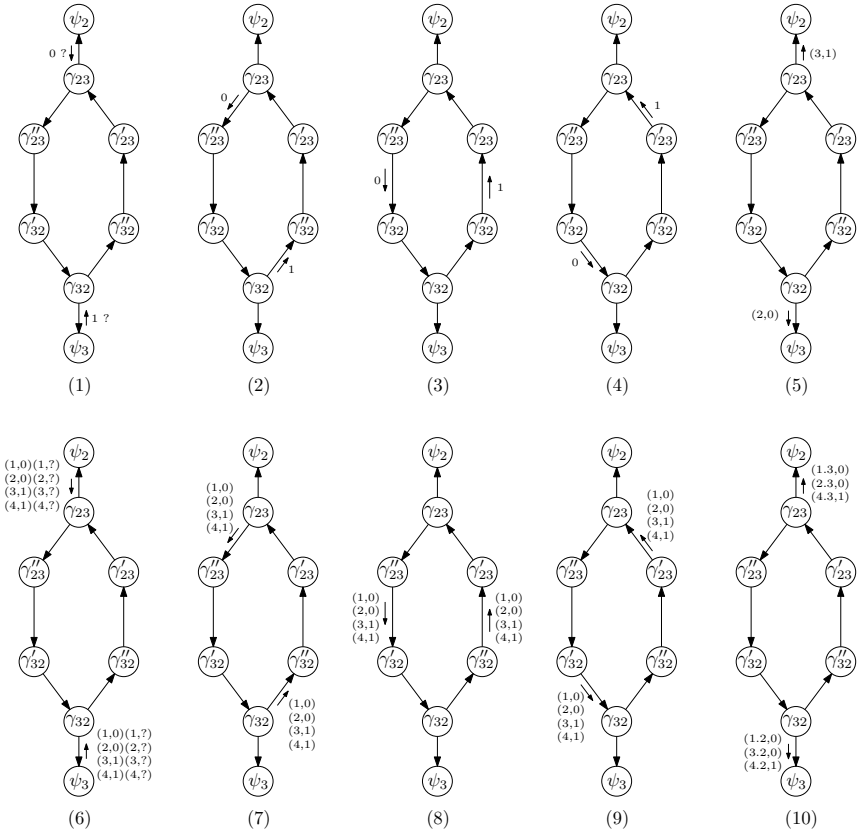


Fig. 10. Traces of the messaging phase between main cells ψ_2 and ψ_3, in the Byzantine scenario of Example 1.

6. $c_3\ v \rightarrow_{\texttt{min}} c_4\ (v)_{\downarrow_{\gamma_{hf}}}$, for $v \in \{0,1\}$
   $c_3\ x_j^v \rightarrow_{\texttt{min}} c_4\ (x_j^v)_{\downarrow_{\gamma_{hf}}}$, for $v \in \{0,1\}$, $j \in [1,N]$
7. $c_4 \rightarrow_{\texttt{min}} c_0$

Assume that s, t and u, are the correlated counters, defined in Section 6.2. When $u = 2$, cell γ'_{hf} is in state c_2, clears its contents by rule group 4 (practically useful, but not strictly needed) and expects $R_t^{\alpha^f}$ from γ''_{fh}, if process f is non-faulty; however, a faulty process f may send, at any time, any objects in $\mho$. Next, when $u = 3$, cell γ'_{hf} is in state c_3 and, by rule group 6, forwards, to γ_{hf}, $R \cap (\cup_{l\in[0,L-1]} R_l^{0,1})$, a filtered subset of the actually received object set, R. This filter is "good-enough", but not complete, because it does not ensure that the forwarded objects form a functional relation on π or that π matches the current message round t. However, this mechanism protects cell γ_{hf} against wrongly timed objects or bad objects which could corrupt its internal consistency. As

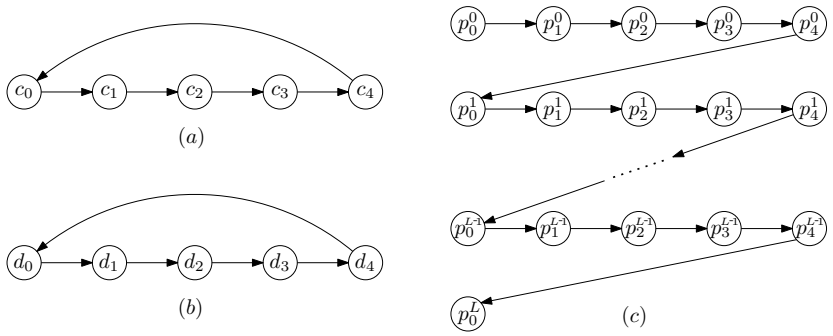


Fig. 11. State diagrams for module Γ_{hf}: (a) for cell γ'_{hf}, (b) for cell γ''_{hf}, (c) for cell γ_{hf}.

we will see, cell γ_{hf} is able to solve the remaining formatting issues. For other values of u, cell γ'_{hf} keeps cycling, without doing any other significant work.

6.4 Rule sequence for Γ_{hf}'s cell γ''_{hf}

According to the following rules, and as illustrated in Figure 11 (b), cell γ''_{hf} cycles continuously through states d_0 (start state), d_1, d_2, d_3, d_4.

1. $d_0 \rightarrow_{\texttt{min}} d_1$
2. $d_1 \rightarrow_{\texttt{min}} d_2$
3. $d_1\ o \rightarrow_{\texttt{max}} d_2$, for $o \in \mho$
4. $d_2\ x_\pi^v \rightarrow_{\texttt{max}} d_3\ (x_\pi^v)_{\downarrow_{\gamma'_{fh}}}$, for $v \in \{0,1\}$, $t \in [0, L-1]$, $\pi \in P(N,t)$
5. $d_3 \rightarrow_{\texttt{min}} d_4$
6. $d_4 \rightarrow_{\texttt{min}} d_0$

The following is an itemized expansion for the case $L = 2$.

1. $d_0 \rightarrow_{\texttt{min}} d_1$
2. $d_1 \rightarrow_{\texttt{min}} d_2$
3. $d_1\ o \rightarrow_{\texttt{max}} d_2$, for $o \in \mho$
4. $d_2\ v \rightarrow_{\texttt{min}} d_3\ (v)_{\downarrow_{\gamma'_{fh}}}$, for $v \in \{0,1\}$
   $d_2\ x_j^v \rightarrow_{\texttt{min}} d_3\ (x_j^v)_{\downarrow_{\gamma'_{fh}}}$, for $v \in \{0,1\}$, $j \in [1,N]$
5. $d_3 \rightarrow_{\texttt{min}} d_4$
6. $d_4 \rightarrow_{\texttt{min}} d_0$

Assume that s, t and u, are the correlated counters, defined in Section 6.2. When $u = 1$, cell γ''_{hf} is in state d_1, clears its contents by rule group 3 (practically useful, but not strictly needed) and expects $R_t^{\alpha^h}$ from γ_{hf}. Next, when $u = 2$, cell γ''_{hf} is in state d_2 and, by rule group 4, forwards down, to γ'_{fh}, the previously received objects, i.e. $R_t^{\alpha^h}$. For other values of u, cell γ''_{hf} keeps cycling, without doing any other significant work.

6.5 Rule sequence for Γ_{hf}'s cell γ_{hf}

According to the following rules, and as illustrated in Figure 11 (c), cell γ_{hf} progresses through states p_0^0 (start state), p_1^0, p_2^0, p_3^0, p_4^0, p_0^1, p_1^1, p_2^1, p_3^1, p_4^1, $\ldots$, p_0^L (final state).

1. $p_0^t \rightarrow_{\texttt{min}} p_1^t$, for $t \in [0, L-1]$
2. $p_1^t \rightarrow_{\texttt{min}} p_2^t$, for $t \in [0, L-1]$
3. $p_1^t\ x_\pi^v \rightarrow_{\texttt{min}} p_2^t\ (x_\pi^v)_{\downarrow_{\gamma''_{hf}}}$, for $v \in \{0,1\}$, $t \in [0, L-1]$, $\pi \in P(N,t)$
4. $p_2^t \rightarrow_{\texttt{min}} p_3^t$, for $t \in [0, L-1]$
5. $p_3^t \rightarrow_{\texttt{min}} p_4^t$, for $t \in [0, L-1]$
6. $p_4^t \rightarrow_{\texttt{min}} p_0^{t+1}$, for $t \in [0, L-1]$
7. $p_4^t\ x_\pi^? \rightarrow_{\texttt{min}} p_0^{t+1}\ (x^0_{\pi\oplus f})_{\downarrow_{\psi_h}}$, for $t \in [0, L-1]$, $\pi \in P(N,t)$, $f \notin Im(\pi)$
8. $p_4^t\ x_\pi^? \rightarrow_{\texttt{min}} p_0^{t+1}\ (x^1_{\pi\oplus f})_{\downarrow_{\psi_h}}$, for $t \in [0, L-1]$, $\pi \in P(N,t)$, $f \notin Im(\pi)$
9. $p_4^t\ x_\pi^? \rightarrow_{\texttt{min}} p_0^{t+1}\ (x^W_{\pi\oplus f})_{\downarrow_{\psi_h}}$, for $t \in [0, L-1]$, $\pi \in P(N,t)$, $f \notin Im(\pi)$
10. $p_4^t\ o \rightarrow_{\texttt{max}} p_0^{t+1}$, for $t \in [0, L-1]$, $o \in \Omega$

The following is an itemized expansion for the case $L = 2$.

1. $p_0^0 \rightarrow_{\texttt{min}} p_1^0$
   $p_0^1 \rightarrow_{\texttt{min}} p_1^1$
2. $p_1^0 \rightarrow_{\texttt{min}} p_2^0$
   $p_1^1 \rightarrow_{\texttt{min}} p_2^1$
3. $p_1^0\ v \rightarrow_{\texttt{min}} p_2^0\ (v)_{\downarrow_{\gamma''_{hf}}}$, for $v \in \{0,1\}$
   $p_1^1\ x_j^v \rightarrow_{\texttt{min}} p_2^1\ (x_j^v)_{\downarrow_{\gamma''_{hf}}}$, for $v \in \{0,1\}$, $j \in [1, N]$
4. $p_2^0 \rightarrow_{\texttt{min}} p_3^0$
   $p_2^1 \rightarrow_{\texttt{min}} p_3^1$
5. $p_3^0 \rightarrow_{\texttt{min}} p_4^0$
   $p_3^1 \rightarrow_{\texttt{min}} p_4^1$
6. $p_4^0 \rightarrow_{\texttt{min}} p_0^1$
   $p_4^1 \rightarrow_{\texttt{min}} p_0^2$
7. $p_4^0\ ?\ 0 \rightarrow_{\texttt{min}} p_0^1\ (x_f^0)_{\downarrow_{\psi_h}}$
   $p_4^1\ x_j^?\ x_j^0 \rightarrow_{\texttt{min}} p_0^2\ (x_{jf}^0)_{\downarrow_{\psi_h}}$, for $j \in [1, N]$, $f \neq j$
8. $p_4^0\ ?\ 1 \rightarrow_{\texttt{min}} p_0^1\ (x_f^1)_{\downarrow_{\psi_h}}$
   $p_4^1\ x_j^?\ x_j^1 \rightarrow_{\texttt{min}} p_0^2\ (x_{jf}^1)_{\downarrow_{\psi_h}}$, for $j \in [1, N]$, $f \neq j$
9. $p_4^0\ ? \rightarrow_{\texttt{min}} p_0^1\ (x_f^W)_{\downarrow_{\psi_h}}$
   $p_4^1\ x_j^? \rightarrow_{\texttt{min}} p_0^2\ (x_{jf}^W)_{\downarrow_{\psi_h}}$, for $j \in [1, N]$, $f \neq j$
10. $p_4^0\ o \rightarrow_{\texttt{max}} p_0^1$, $o \in \Omega$
    $p_4^1\ o \rightarrow_{\texttt{max}} p_0^2$, $o \in \Omega$

Assume that s, t and u, are the correlated counters, defined in Section 6.2. When $u = 0$, cell γ_{hf} is in state p_0^t, is empty, and expects $R_t^{\alpha^h} \cup R_t^?$ from ψ_h. Next, when $u = 1$, cell γ_{hf} is in state p_1^t and, by rule group 3, forwards down, to γ''_{hf}, the value objects destined to process f, $R_t^{\alpha^h}$ but keeps the template objects,

$R_t^?$. Next, when $u = 2$, cell γ_{hf} is in state p_2^t and keeps cycling, by rule group 4, without doing any other significant work. Next, when $u = 3$, cell γ_{hf} is in state p_3^t and expects, from cell γ'_{hf}, a set of good-enough value objects, ideally $R_t^{\alpha^f}$, but could be any subset of $(\cup_{l\in[0,L-1]} R_l^{0,1})$.

Case $u = 4$ describes a critical step, where cell γ_{hf} uses the process number f to tag the objects which are forwarded to ψ_h. Consider the partial function $\mathcal{S}_f : \Omega \to \Omega$, defined by $\mathcal{S}_f(x_\pi^v) = x_{\pi\oplus f}^v$, for $\pi \in (\cup_{l\in[0,L-1]} P(N,l))$, $v \in \{0,1\}$. Cell γ_{hf} is in state p_4^t and forwards down, to cell ψ_h, either (a) the set $\mathcal{S}_f(R_t^{\alpha^f})$, if process f is non-faulty; or, (b) a reconstructed version, where unavailable objects are replaced by value objects, reconstructed with the default value W.

The correct format is checked by matching received value objects against template objects $R_t^?$. Rule group 7 applies when a 0 valued object can be matched against a template. Rule group 8 applies when a 1 valued object can be matched against a template. Rule group 9 applies when no value object matched the template and recreates a missing value object based on the default value W. After this, cell γ_{hf} clears all its contents, by rule group 10, preparing itself for the next cycle.

6.6 Module Π_h

Collecting the above results, we can now complete the last item required by the contract between the main module, Ψ_h, with its firewall modules, Γ_{hf}, $f \in [1, N]$. Assume again that s, t and u, are the correlated counters, defined in Section 6.2 and $t \in [0, L-1]$. As shown in Section 6.5, when $u = 4$, for each $h, f \in [1, N]$, cell Γ_{hf} sends down, to cell ψ_h, either (a) the set $\mathcal{S}_f(R_t^{\alpha^f})$, if process f is non-faulty; or, (b) a reconstructed version, where unavailable objects are replaced by value objects, reconstructed with the default value W.

Therefore, cell ψ_h receives, collectively from its parent modules, the set $S = \cup_{f\in[1,N]} \mathcal{S}_f(R_t^{\alpha^f})$. A straightforward argument shows that sets $\mathcal{S}_f(R_t^{\alpha^f})$, $f \in [1, N]$, are disjoint, and their union S is $S = R_{t+1}^{\alpha^h}$. This triggers the next messaging round $t + 1$, with the required new set of value objects, which completes the argument.

6.7 Complexity

As indicated by the next theorem, this new version of the Byzantine algorithm improves the runtime of the previous version [7], from $9L + 6$ to $6L + 1$, where, typically, $L = \lceil N/3 \rceil$.

Theorem 1. *This revised EIG-based Byzantine algorithm takes $6L + 1$ steps, where L is the number of messaging rounds.*

Proof. As seen above, it takes $5L$ P steps from start until the last message is received by the main cells, ψ_h, $h \in [1, N]$, One additional P step is required to transit from ψ_h's state s_L to state e_L, i.e. to transit from from the messaging

phase to the bottom-up phase. Finally, cell ψ_h needs L more P steps for its bottom-up phase, to evaluate its final decision value and reach the final state. Thus, the revised Byzantine algorithm takes a total of $5L+1+L = 6L+1$ steps.

The new version reduces the total number of cells required, from super-exponential, $O(N!)$, to a small polynomial, $O(N^2)$. However, some other static complexity measures are still very large. The new version does not change the message complexity of the previous version, which is mostly determined by the EIG algorithm itself. Table 12 summarizes these complexity measures.

Table 12. Summary of complexity measures (where, typically, $L = \lceil N/3 \rceil$).

Complexity measure	**Previous version**	**Current version**
Number of steps	$9L + 6$	$6L + 1$
Number of cells per Π module	$2N + 1 + O(N!)$	$3N + 1$
Number of objects	$O(N!)$	$O(N!)$
Maximum number of states per elementary module	$O(L)$	$O(L)$
Maximum number of rules per elementary module	$O(N!)$	$O(N!)$
Number of messages exchanged between Π modules	N^2L	N^2L

7 Conclusions and open problems

In this paper, we proposed an improved generic version of P modules, an extensible framework for recursive composition of P systems, and used it to provide a faster P solution for the Byzantine agreement algorithm, based on Exponential Information Gathering (EIG) trees.

Our modular framework offers three types of generic parameters: generic objects, external definitions and external references and supports encapsulation, information hiding and modular composition.

Our revised P solution uses only duplex channels, fewer cells and fewer rules, while improving overall running time from $9L + 6$ to $6L + 1$, where L is the number of messaging rounds.

We proved that modules, i.e. cell clusters, can solve the classical Byzantine agreement problem. Our design uses $3N + 1$ cells for each module, with one "main" cell and $3N$ ancillary cells, which enclose the main cell inside a "firewall". Can we solve the Byzantine agreement directly between individual cells, without help from any additional firewall?

In our case, firewall cells have a complex role. They protect the main cell against badly formatted, wrongly timed and missing messages. If they reach the main cell, wrongly timed bad messages have the potential to corrupt the internal structures, required by the internal cell logic. Additionally, our firewall cells tag incoming messages with unforgeable origin marks (a feature that current passive

channels do not offer). This is a critical feature of the EIG-based algorithm itself (not of the cell implementing it). If the originator is not guaranteed, this algorithm will fail.

We believe that some of these firewall tasks can be retrofitted into the main cell itself, but not all required critical features. Thus, it seems that it is not possible to achieve a Byzantine agreement between individual cells, if we rely on the classical EIG-based algorithm.

However, there are many other algorithms for the Byzantine agreement, thus, our question is more general. Is there any other algorithm able to solve the Byzantine agreement at the cell level, still using passive channels, as in the current framework? We conjecture that the answer is negative. If this is indeed the case, what is the minimal size of one firewall component, one, two, three?

One can make a parallel to the history of the Internet. "The Internet protocols were originally designed for openness and flexibility, not for security. The ARPA researchers needed to share information easily, so everyone needed to be an unrestricted insider on the network" [8]. It seems that this was also the case in the early development of P systems. Are there simple ways to enhance our passive channels to provide more safety? Besides distributed computing, would this be useful in other modelling scenarios? A related problem, are there real-life biological scenarios, which need sophisticated fault-tolerant mechanisms, similar to the Byzantine agreement algorithms used in distributed computing, and, if yes, how do these really work?

Besides the above conjecture, our investigation leaves open a number of other interesting and challenging problems. Can we extend our P system solution to cover $2F + 1$ connected graphs, but not necessarily complete? Can we design P system solutions for other Byzantine agreement algorithms, not EIG-based, for example using reliable broadcasts? Will other solutions work "better", e.g., faster or with smaller communication overhead? Is it possible to solve the Byzantine agreement problem with a fixed number of P rules? If not, which is likely the case, can we solve this problem by proposing simple "natural" extensions to current rule system? Finally, can we provide solutions for some types of asynchronous P systems and what additional constraints will be needed in this case?

References

1. M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable Byzantine fault-tolerant services. In A. Herbert and K. P. Birman, editors, *SOSP*, pages 59–74. ACM, 2005.
2. M. Ben-Or and A. Hassidim. Fast quantum Byzantine agreement. In H. N. Gabow and R. Fagin, editors, *STOC*, pages 481–485. ACM, 2005.
3. C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography. *J. Cryptology*, 18(3):219–246, 2005.
4. M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.
5. G. Ciobanu. Distributed algorithms over communicating membrane systems. *Biosystems*, 70(2):123–133, 2003.

6. G. Ciobanu, R. Desai, and A. Kumar. Membrane systems and distributed computing. In G. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *WMC-CdeA*, volume 2597 of *Lecture Notes in Computer Science*, pages 187–202. Springer-Verlag, 2002.
7. M. J. Dinneen, Y.-B. Kim, and R. Nicolescu. P systems and the Byzantine agreement. *The Journal of Logic and Algebraic Programming*, In Press, Corrected Proof, 2010.
8. F. E. Froehlich and A. Kent. *Encyclopedia of Telecommunications, Volume 15*. CRC Press, 1997.
9. L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
10. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
11. J.-P. Martin and L. Alvisi. Fast Byzantine consensus. *IEEE Trans. Dependable Sec. Comput.*, 3(3):202–215, 2006.
12. R. Nicolescu, M. J. Dinneen, and Y.-B. Kim. Towards structured modelling with hyperdag P systems. *International Journal of Computers, Communications and Control*, 2:209–222, 2010.
13. G. Păun. *Membrane Computing: An Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.
14. G. Păun. Introduction to membrane computing. In G. Ciobanu, M. J. Pérez-Jiménez, and G. Păun, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 1–42. Springer, 2006.
15. G. Păun and M. J. Pérez-Jiménez. Solving problems in a distributed way in membrane computing: dP systems. *International Journal of Computers, Communications and Control*, 5(2):238–252, 2010.
16. M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.
17. F. J. Romero-Campero, J. Twycross, M. Cámara, M. Bennett, M. Gheorghe, and N. Krasnogor. Modular assembly of cell systems biology models using P systems. *Int. J. Found. Comput. Sci.*, 20(3):427–442, 2009.
18. T. Serbanuta, G. Stefanescu, and G. Rosu. Defining and executing P systems with structured data in K. In D. W. Corne, P. Frisco, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Workshop on Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 374–393. Springer, 2008.