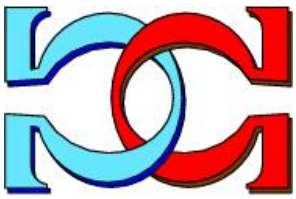
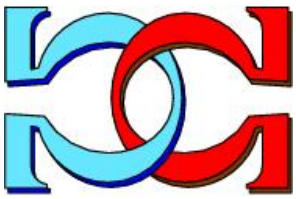
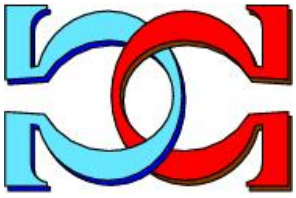


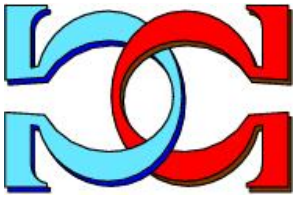
CDMTCS
Research
Report
Series



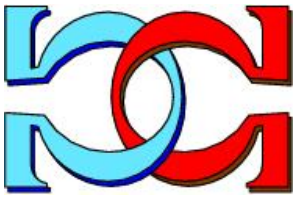
Synchronization in P Modules



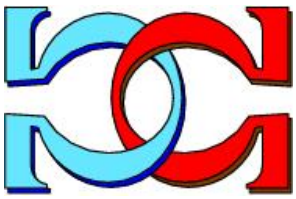
Michael J. Dinneen
Yun-Bum Kim
Radu Nicolescu



Department of Computer Science,
University of Auckland,
Auckland, New Zealand



CDMTCS-378
February 2010



Centre for Discrete Mathematics and
Theoretical Computer Science

Synchronization in P Modules

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

In the field of molecular computing, in particular P systems, synchronization is an essential requirement for composing or sequentially linking together congenial P system activities. We provide an improved deterministic algorithm based on static structures and traditional rules, which runs in $4e + 13$ steps, where e is the eccentricity of the initiating cell. Using the same model, extended with the support of cell IDs, we provide another deterministic algorithm, which runs in $3e+13$ steps. Our algorithms use a convenient framework, called P modules, which embraces the essential features of many popular types of P systems.

Keywords: P systems, P modules, synchronization, cellular automata.

1 Introduction

The *Firing Squad Synchronization Problem* (FSSP) [17, 11, 8, 15, 9, 18] is one of the best studied problems for cellular automata. The problem involves finding a cellular automaton, such that, after a command is given, all the cells, after some finite time, enter a designated *firing* state *simultaneously* and *for the first time*. Several variants of FSSP have been proposed and studied [15, 17]. Studies of these variations mainly focus on finding a solution with as few states as possible and possibly running in optimum time.

There are several applications that require synchronization. We list just a few here. At the biological level, cell synchronization is a process by which cells at different stages of the cell cycle (division, duplication, replication) in a culture are brought to the same phase. There are several biological methods used to synchronize cells at specific cell phases [7]. Once synchronized, monitoring the progression from one phase to another allows us to calculate the timing of specific cells' phases. A second example relates to operating systems [16], where process synchronization is the coordination of simultaneous threads or processes to complete a task without race conditions. In distributed computing, in particular consensus problems, such as the Byzantine agreement problem,

processes need to be in a common state at certain fixed times, see Lynch [10]. Finally, in telecommunication networks [6], we often want to synchronize computers to the same time, i.e., primary reference clocks should be used to avoid clock offsets.

The synchronization problem has recently been studied in the framework of P systems. Using tree-based P systems, Bernardini *et al.* [2] provided a non-deterministic solution with time complexity $3h$ and a deterministic solution with time complexity $4n + 2h$, where h is the height of the tree structure underlying the P system and n is the number of membranes of the P system. The deterministic solution requires membrane *polarization* techniques and uses a *depth-first-search*.

More recently, Alhazov *et al.* [1] described an improved deterministic algorithm for tree-based P systems, that runs in $3h + 3$ steps. This solution requires conditional rules (promoters and inhibitors) and combines a *breadth-first-search*, a *broadcast* and a *convergecast*.

We continued the study of FSSP for digraph-based P systems [4]. We proposed two uniform deterministic solutions to a variant of FSSP [17], in which there is a single commander, at an arbitrary position, for hyperdag P systems [12] and for neural P systems [13] with symmetric communication channels. We further generalized this problem by synchronizing a subset of cells (or membranes) of the considered hyperdag or neural P system. Our first solution dynamically extends P systems with mobile channels. Our second solution is solely based on classical rules and static channels. In contrast to the previous FSSP solutions for tree-based P systems, our solutions do not require membrane polarizations or conditional rules, but require states, as typically used in hyperdag and neural P systems. These solutions take $e_c + 5$ and $6e_c + 7$ steps, respectively, where e_c is the eccentricity of the commander cell of the digraph underlying these P systems.

We continue our work on FSSP for digraph-based P systems (neural and hyperdag P systems), using a single framework, called P modules [5], which embraces the computational functionality of many popular types of P systems.

In Section 3 of this paper, we improve our previous results for a natural four-phase FSSP algorithm (classical rules and static structures) by reducing the multiplicative factor from 6 to 4, such that the overall running time is $4e_c + 13$. Further, in Section 4, by exploiting the notion of cell IDs, we provide a slightly faster algorithm of running time $3e_c + 13$. In Section 2, we first define a virtual structure for a given P system structure, based on a neighboring relationship from a commanding node and formally define a P module. Finally, in Section 5, we summarize our results and conclude with some open problems.

2 Preliminary

We assume that the reader is familiar with the basic terminology and notations, such as relations, graphs, nodes (vertices), arcs, directed graphs (digraphs), directed acyclic graphs (dags), trees, alphabets, strings and multisets [12].

For a given tree, connected dag or (weakly) connected digraph, we define the *eccentricity* of a node c, e_c, as the maximum length of a shortest path between c and any other node in the corresponding underlying undirected structure.

For a tree, the set of *neighbors* of a node x, $Neighbor(x)$, is the union of x's parent and x's children. For a dag (X, δ) and a node $x \in X$, we define $Neighbor(x) = \delta(x) \cup \delta^{-1}(x) \cup \delta(\delta^{-1}(x)) \backslash \{x\}$, if we want to include the siblings, or, $Neighbor(x) = \delta(x) \cup \delta^{-1}(x)$, otherwise. For a graph (X, A) and a node $x \in X$, we define $Neighbor(x) = A(x) = \{y \mid (x, y) \in A\}$. Note that, as defined, $Neighbor$ is always a symmetric relation.

A special node c of a structure will be designated as the *commander*. For a given commander c, we define the *level* of a node x, $level_c(x) \in \mathbb{N}$, as the length of a shortest path between the c and x, over the $Neighbor$ relation.

For a given tree, dag or graph and commander c, for nodes x and y, if $y \in Neighbor(x)$ and $level_c(y) = level_c(x) + 1$, then x is a *predecessor* of y and y is a *successor* of x. Similarly, a node z is a *peer* of a node x, if $z \in Neighbor(x)$ and $level_c(z) = level_c(x)$. Note that, for a given node x, the set of peers and the set of successors are disjoint. A node without a successor will be referred to as a *terminal*. The *eccentricity* of a node c is $e_c = max\{level_c(x) \mid x \in X\}$. A *level-preserving path* from c to a node y is a sequence $x_0, \ldots, x_k$, such that $x_0 = c$, $x_k = y$, $x_i \in Neighbor(x_{i-1})$, $level_c(x_i) = i$, $1 \leq i \leq k$. We define $count_c(y)$ as the number of distinct level-preserving paths from c to y. Also define $span_c(y) = max\{level_c(z) \mid z$ is in a level-preserving path that contains $y\}$.

Definition 1 (P module [5]). A *P module* is a system $\Pi = (O, K, \delta, P)$, where:

1. O is a finite non-empty alphabet of *objects*;
2. K is a finite set of *cells* and each cell $\sigma \in K$ is represented as $\sigma = (Q, s, w, R)$, where:
   - Q is a finite set of *states* for the cell;
   - $s \in Q$ is the cell's current state;
   - $w \in O^*$ is the cell's current multiset of objects;
   - R is the cell's finite *ordered* set of multiset rewriting *rules* of the form: $s\ x \rightarrow_\alpha s'\ x'\ (u)_{\beta_\gamma}$, where $s, s' \in Q$, $x, x' \in O^*$, $u \in O^*$, $\alpha \in \{min, max\}$ is the *rewriting* operator, $\beta \in \{\uparrow, \downarrow, \updownarrow\}$ and $\gamma \in \{one, spread, repl\} \cup K$ are the *transfer* operators (see below for details). If $u = \lambda$ (the emptyset), this rule can be abbreviated as $s\ x \rightarrow_\alpha s'\ x'$.
3. δ is a binary relation on K, i.e., a set of structural arcs, representing *duplex* or *simplex* communication channels between cells;
4. P is a subset of K, indicating the *port* cells, i.e., the only cells can be connected to other modules.

The rules given by the ordered set(s) R are applied in the *weak priority* order [14]. For a cell $\sigma = (Q, t, w, R)$, a rule $s\ x \rightarrow_\alpha s'\ x'\ (u)_{\beta_\gamma} \in R$ is *applicable* if $t = s$ and $x \subseteq w$. Additionally, if $s\ x \rightarrow_\alpha s'\ x'\ (u)_{\beta_\gamma}$ is the first applicable rule, then each subsequent applicable rule's target state (i.e., state indicated in the right-hand side) must be s'.

The rewriting operator $\alpha = max$ indicates that the applicable rewriting rule is applied as many times as possible. The transfer operators $\beta = \updownarrow$ and $\gamma = repl$ (presented as

$(u)_{\uparrow_{repl}}$ in a rewriting rule) indicate that the multiset u is replicated and sent to all neighbor cells. The other rewriting and (non-deterministic) transfer operators are not used in this paper and are described in [5] for those interested.

3 Deterministic FSSP solution

In the FSSP, the commander sends an order to all (firing) squad cells, which will prompt them to synchronize by entering the designated firing state. However, in general, the commander does not have direct communication channels to all squad cells. Relaying the order through intermediate cells results in some squad cells receiving the order before other squad cells. In this case, to ensure that all squad cells enter the firing state simultaneously, each squad cell needs to wait until all other squad cells receive the order.

For convenience and ease of understanding, we present the algorithmic steps into four conceptual phases. The details of these phases are given first, for the solution with a finite set of alphabet objects and later in Section 4, for the solution with cell ID objects.

The initial configuration of a P module Π with $n > 1$ cells, as qualified by δ below, for our FSSP solution with a finite set of alphabet objects is as follows:

1. $O = \{a, b, c, d, e, f, g, h, u, v, w\}$.
2. $K = \{\sigma_1, \ldots, \sigma_n\}$ is the set of all cells, $\sigma_c \in K$ is the commander and $F \subseteq K$ is the set of squad cells. For each cell $\sigma_i = (Q_i, s_i, w_i, R_i) \in \{\sigma_1, \ldots, \sigma_n\}$:
   - $Q_i = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\}$, where s_{10} is the firing state.
   - $s_i = s_0$.
   - $w_i = \begin{cases} \{a\} & \text{if } \sigma_i = \sigma_c \text{ and } \sigma_i \notin F, \\ \{a, f\} & \text{if } \sigma_i = \sigma_c \text{ and } \sigma_i \in F, \\ \{f\} & \text{if } \sigma_i \in F \setminus \sigma_c, \\ \emptyset & \text{if } \sigma_i \in K \setminus (F \cup \{\sigma_c\}). \end{cases}$
   - R_i is given in Phases 1, 2, 3 and 4.
3. δ is either a symmetric digraph with simplex channels or is a (weakly) connected digraph with duplex channels.
4. $P = \emptyset$.

For the proofs below, we define, for a cell σ_i, $\phi_i \in \mathbb{N}$ as the total number of an object a in all of σ_i's peers (i.e., the number of distinct level-preserving paths from σ_c to all σ_i's peers), and $\tau_i \in \mathbb{N}$ is the total number of an object a in all of σ_i's successors (i.e., the number of distinct level-preserving paths from σ_c to all σ_i's successors).

Phase 1 (FSSP-I: First broadcast from the commander).
Overview: The commander σ_c initiates a *broadcast*, which relays a broadcast message from the predecessors to their successors. The commander starts a counter, which is incremented by one in each step.

Each cell relays the broadcast message as follows. If a cell σ_i receives a broadcast message from its predecessors, σ_i becomes *do-broadcast* cell and sends a broadcast message to all its successors. All *do-broadcast* cells ignore any subsequently received broadcast messages.

Precondition: The initial configuration of the P module Π.

Postcondition: Each cell σ_i ends in state s_4 and σ_i has: $count_c(i)$ copies of a; $count_c(i)$ copies of b; τ_i copies of d; four copies of e, if $\sigma_i = \sigma_c$; one copy of f, if $\sigma_i \in F$; ϕ_i copies of v.

Number of steps: For each cell σ_i, this phase takes $level_c(i) + 4$ steps.

Rules:

0. For state s_0:
   1) $s_0\ a \rightarrow_{max} s_1\ abe\ (d)_{\updownarrow_{repl}}$
   2) $s_0\ d \rightarrow_{max} s_1\ ab\ (d)_{\updownarrow_{repl}}$
1. For state s_1:
   1) $s_1\ be \rightarrow_{max} s_2\ bee$
   2) $s_1\ a \rightarrow_{max} s_2\ a\ (u)_{\updownarrow_{repl}}$
   3) $s_1\ u \rightarrow_{max} s_2$
   4) $s_1\ d \rightarrow_{max} s_2$
2. For state s_2:
   1) $s_2\ be \rightarrow_{max} s_3\ bee$
   2) $s_2\ a \rightarrow_{max} s_3\ a$
   3) $s_2\ u \rightarrow_{max} s_3\ v$
3. For state s_3:
   1) $s_3\ be \rightarrow_{max} s_4\ bee$
   2) $s_3\ a \rightarrow_{max} s_4\ a$
   3) $s_3\ u \rightarrow_{max} s_4$

Proof. Consider a cell σ_i at $level_c(i)$. Note that, an object d (the broadcast message) needs $level_c(i)$ steps to relay down to a cell σ_i at level $level_c(i)$.

- At step $level_c(i) + 1$: if σ_i is the commander, it transits to state s_1, broadcasts one d to each of its neighbors and produces one b. If σ_i is not the commander, σ_i transits to state s_1, broadcasts $count_c(i)$ copies of d to each of its neighbors and produces $count_c(i)$ copies of a and b by accumulating one copy of a and b for each sent d.
- At step $level_c(i)+2$: σ_i transits to state s_2, broadcasts $count_c(i)$ copies of u, receives ϕ_i copies of u from its *peers* and receives τ_i copies of d from its successors. At the same time, σ_i eliminates all superfluous copies of u and d.
- At step $level_c(i)+3$: σ_i transits to state s_3 and converts τ_i copies of u into τ_i copies of v.
- At step $level_c(i) + 4$: σ_i transits to state s_4 and eliminates all superfluous copies of u.

In this phase, each cell σ_i is idle in steps $0, \ldots, level_c(i)$ and is active in steps $level_c(i)+1$, $level_c(i)+2$, $level_c(i)+3$, $level_c(i)+4$ as stated above. Thus, each σ_i takes $level_c(i)+4$ steps.

The commander σ_c initially has one a and produces four copies of e by accumulating one e in each step, $level_c(i) + 1, \ldots, level_c(i) + 4$. In this phase, f is not produced or consumed, thus, $\sigma_i \in F$ has one f. The number of each object a, b, d, u and v, for a cell σ_i, is verified in the steps $level_c(i) + 1, \ldots, level_c(i) + 4$. □

Phase 2 (FSSP-II: Convergecasts from terminal cells).
Overview: Immediately after the first broadcast (Phase 1), each terminal cell σ_t initiates a *convergecast*, which relays a convergecast message from the successors to their predecessors, for all distinct level-preserving paths from σ_c to σ_t.

Each non-terminal cell relays the convergecast message as follows. If a cell σ_i receives a convergecast message from all its successors, σ_i becomes a *ready-to-convergecast* (RTC) cell and sends a RTC-notification to all its peers. If a RTC cell σ_j receives RTC-notifications from all its peers, σ_j becomes a *do-convergecast* (DC) cell and sends a convergecast message to all its predecessors.

After σ_c receives convergecast messages from all its successors, σ_c stops the counter (which was started in Phase 1) and uses this counter to compute its eccentricity, where the counter is the number of steps for σ_c to send a message to a farthest cell σ_z and receive a reply message from σ_z.
Precondition: As described in the postcondition of Phase 1.
Postcondition: This phase ends when the commander enters state s_7. Each cell σ_i ends in state s_7 and σ_i has: $count_c(i)$ copies of a; $e_c + 2$ copies of e, if $\sigma_i = \sigma_c$; one copy of f, if $\sigma_i \in F$.
Number of steps: For each cell σ_i, this phase takes $3e_c - level_c(i) + 4$ steps.
Rules:

4. For state s_4:
   1) $s_4\ h \rightarrow_{max} s_5$
   2) $s_4\ cd \rightarrow_{max} s_4\ cd$
   3) $s_4\ ad \rightarrow_{max} s_4\ ad$
   4) $s_4\ a \rightarrow_{max} s_4\ ah\ (w)_{\updownarrow_{repl}}$
   5) $s_4\ w \rightarrow_{max} s_4$
   6) $s_4\ cd \rightarrow_{max} s_5$
   7) $s_4\ vw \rightarrow_{max} s_5\ v$
   8) $s_4\ cv \rightarrow_{max} s_5\ cv$
   9) $s_4\ av \rightarrow_{max} s_5\ av\ (w)_{\updownarrow_{repl}}$
   10) $s_4\ a \rightarrow_{max} s_5\ ah\ (c)_{\updownarrow_{repl}}$
   11) $s_4\ be \rightarrow_{max} s_4\ bee$
   12) $s_4\ be \rightarrow_{max} s_5\ ee$

5. For state s_5:
   1) $s_5\ h \rightarrow_{max} s_6$
   2) $s_5\ vw \rightarrow_{max} s_5\ v$
   3) $s_5\ cv \rightarrow_{max} s_5\ cv$
   4) $s_5\ av \rightarrow_{max} s_5\ av\ (w)_{\updownarrow_{repl}}$
   5) $s_5\ a \rightarrow_{max} s_5\ ah\ (c)_{\updownarrow_{repl}}$
   6) $s_5\ eee \rightarrow_{max} s_6\ e$

6. For state s_6:
   1) $s_6\ cv \rightarrow_{max} s_6$
   2) $s_6\ bc \rightarrow_{max} s_6$
   3) $s_6\ b \rightarrow_{max} s_6\ b$
   4) $s_6\ a \rightarrow_{max} s_7\ a$
   5) $s_6\ v \rightarrow_{max} s_7$
   6) $s_6\ w \rightarrow_{max} s_7$

Proof. To establish the correctness of this phase, we categorize a cell σ_i into one of four groups.

(1) $span_c(i) = e_c$.

(2) $span_c(i) < e_c$ and σ_i has no peers.

(3) $span_c(i) < e_c$ and for each σ_i's peer σ_j, $span_c(j) \leq span_c(i)$.

(4) $span_c(i) < e_c$ and σ_i has a peer σ_j, $span_c(j) > span_c(i)$.

A cell σ_i in group (1), (2) or (3) progresses through two *convergecast* steps, $t_1 = span_c(i) + 5 + 2(span_c(i) - level_c(i)) = 3span_c(i) - 2level_c(i) + 5$, $t_2 = 3span_c(i) - 2level_c(i) + 6$, to send $count_c(i)$ copies of c (the convergecast message).

- At step t_1: σ_i remains in state s_4, broadcasts $count_c(i)$ copies of w and produces $count_c(i)$ copies of h by accumulating one h for each sent w. At the same time, σ_i eliminates all superfluous copies of w.

- At step t_2: σ_i transits to state s_5 and consumes $count_c(i)$ copies of h, ϕ_i copies of w, τ_i copies of c and τ_i copies of d. At the same time, σ_i broadcasts $count_c(i)$ copies of c and produces $count_c(i)$ copies of h by accumulating one h for each sent c.

Note that, if $span_c(i) = e_c$, step $t_1 = 3e_c - 2level_c(i) + 5$ and step $t_2 = 3e_c - 2level_c(i) + 6$. Hence, each other cell σ_j in level $level_c(i)$ sends $count_c(j)$ copies of c, before or at step $3e_c - 2level_c(i) + 6$.

A cell σ_i in group (4), where $0 < level_c(i) < e_c$, progresses through three *convergecast* steps, $t'_1, t'_2, t'_3 \in \{level_c(i) + 5, \ldots, 3e_c - 2level_c(i) + 6\}$, where $t'_1 < t'_2 < t'_3$, to send $count_c(i)$ copies of c. Note that, for $level_c(i) = 1$, steps $t'_1, t'_2, t'_3 \in \{6, \ldots, 3e_c + 4\}$ and for $level_c(i) = e_c - 1$, steps $t'_1, t'_2, t'_3 \in \{e_c + 4, \ldots, e_c + 8\}$.

- At step t'_1: σ_i remains in state s_4, broadcasts $count_c(i)$ copies of w and produces $count_c(i)$ copies of h by accumulating one h for each sent w. At the same time, σ_i eliminates all superfluous copies of w.

- At step t'_2: σ_i transits to state s_5, broadcasts $count_c(i)$ copies of w and consumes $count_c(i)$ copies of h, τ_p copies of c and τ_p copies of d.

- At step t'_3: σ_i remains in state s_5, broadcasts $count_c(i)$ copies of c and produces $count_c(i)$ copies of h by accumulating one h for each sent c.

Each cell progresses through the convergecast steps, after all its successors progress through the convergecast steps. Hence, the commander is the last cell to apply the convergecast steps and this phase ends when it enters state s_7 (the commander belongs to group (1), hence it transits to state s_6 after it progresses through the convergecast steps).

Note that, $count_c(i)$ copies of c, sent from σ_i, is received by σ_i's neighbors, which include σ_i's successors. In general, the commander transits from state s_6 to s_7 in one step and a non-commander cell σ_k transits from state s_6 to s_7, after σ_k receives $count_c(j)$ copies of c from each predecessor σ_j. Thus, when σ_c transits to state s_7, all other cells are already in state s_7.

To compute the running time of this phase for each cell σ_i, let us first compute the step, in which the commander transits to state s_7. Consider a level-preserving path, $\sigma_c, \ldots, \sigma_t$, where $level_c(c) = 0$ and $level_c(t) = e_c$. The terminal cell σ_t ends its last phase at step $e_c + 4$ and each cell σ_j in this path progresses through two steps, $3e_c - 2level_c(j) + 5$, $3e_c - 2level_c(j) + 6$, to send $count_c(j)$ copies of c. Further, the commander

takes one step to transit from state s_6 to s_7. Thus, commander transits to state s_7 at step $(e_c + 5) + 2(e_c + 1) + 1 = 3e_c + 8$, i.e., all other cells end this phase at step $3e_c + 8$.

Each cell σ_i ends its Phase 1 at step $level_c(i)+4$, thus, σ_i takes $(3e_c+8)-(level_c(i)+4) = 3e_c - level_c(i) + 4$ steps in this phase.

For a cell σ_i, the rules in this phase do not produce or consume copies of a and f, thus, σ_i still has $count_c(i)$ copies of a and still has one f, if $\sigma_i \in F$. Additionally, σ_i consumes all copies of b, c, d, h, v and w by rules 4.12, 5.1, 6.1, 6.2, 6.5 and 6.6.

In any level-preserving path, $\sigma_c, \ldots, \sigma_t$, where $level_c(c) = 0$ and $level_c(t) = e_c$, the terminal cell σ_t starts its convergecast steps at step $level_c(t) + 5 = e_c + 5$. After $2(e_c + 1)$ steps, all cells in this path progress through their convergecast steps. In this phase, from step $level_c(c) + 5$, the commander accumulates one e in each step, until it progresses through all its convergecast steps. The commander initially has four copies of e from Phase 1 and in this phase, the commander accumulates $(e_c+5)-(level_c(c)+5)+2(e_c+1) = 3e_c + 2$ copies of e. Hence, the commander has $4 + (3e_c + 2) = 3(e_c + 2)$ copies of e. The commander σ_c then consumes two thirds copies of e by rule 5.6, thus, σ_c ends this phase with $e_c + 2$ copies of e. □

Remark 2. The purpose of designating cells as DC cells or RTC cells is to ensure that the commander correctly computes its eccentricity. To highlight this need, consider the case when a RTC cell sends a convergecast message, without receiving RTC notifications from all its peers. A scenario for this considered case assumes the following: cell σ_i has one successor σ'_i and one peer σ_j; cell σ_j has one successor σ'_j and one peer σ_i; $count_c(i) = count_c(j)$; $span_c(i)+2 = span_c(j)$. When σ_i receives a convergecast message from σ'_i, σ_i sends a convergecast message to its neighbors, which includes σ_i's predecessors, σ_i's successor σ'_i and σ_i's peer σ_j. One step later when σ_j receives σ_i's convergecast message, σ_j sends a convergecast message. In this case, the commander may compute a value, which is less than its actual eccentricity.

Phase 3 (FSSP-III: Second broadcast from the commander).
Overview: Note that, the commander σ_c computes the eccentricity e_c in Phase 2. In this phase, σ_c initiates a second *broadcast* by sending the broadcast message e^{e_c}. This broadcast message is relaid from predecessors to successors.

In this phase, each cell relays a broadcast message as follows. If a cell σ_i receives a broadcast message e^k, where $k = e_c - level_c(i)$, from its predecessors, σ_i becomes a *do-broadcast* cell and sends a broadcast message e^{k-1} to all its successors. All *do-broadcast* cells ignore any subsequent broadcast messages.
Precondition: As described in the postcondition of Phase 2.
Postcondition: Each cell σ_i ends in state s_9 and σ_i has: $count_c(i)$ copies of a; one copy of f, if $\sigma_i \in F$; $(e_c - level_c(i) + 1)count_c(i)$ copies of g.
Number of steps: For each cell σ_i, this phase takes $level_c(i) + 3$ steps.
Rules:

7. For state s_7:
   1) $s_7\ ae \rightarrow_{max} s_8\ ah$
   2) $s_7\ e \rightarrow_{max} s_8\ g\ (e)_{\updownarrow_{repl}}$
8. For state s_8:
   1) $s_8\ h \rightarrow_{max} s_8$
   2) $s_8\ a \rightarrow_{max} s_9\ a$
   3) $s_8\ e \rightarrow_{max} s_9$

Proof. Consider a cell σ_i. Note that, the object e needs $level_c(i)$ steps to propagate down to a cell σ_i at level $level_c(i)$. Note that, if σ_i is not the commander, σ_i receives $(e_c - level_c(i) + 2)count_c(i)$ copies of e at step $3e_c + level_c(i) + 7$.

- At step $3e_c + level_c(i) + 8$: σ_i transits to state s_8, consumes $count_c(i)$ copies of e and produces $count_c(i)$ copies of h. At the same time, σ_i broadcasts the remaining $(e_c - level_c(i) + 1)count_c(i)$ copies of e and produces $(e_c - level_c(i) + 1)count_c(i)$ copies of g by accumulating one g for each sent e.

- At step $3e_c + level_c(i) + 9$: σ_i remains in state s_8 and consumes $count_c(i)$ copies of h.

- At step $3e_c + level_c(i) + 10$: σ_i transits to state s_9 and eliminates all superfluous copies of e.

In this phase, each cell σ_i is idle in steps $3e_c + 7, \ldots, 3e_c + level_c(i) + 7$ and is active in steps $3e_c + level_c(i) + 8$, $3e_c + level_c(i) + 9$, $3e_c + level_c(i) + 10$ as stated above. Thus, each σ_i takes $level_c(i) + 3$ steps.

The rules in this phase do not produce or consume f, thus, each squad cell still ends with one f. The number of each object a, e, g and h, for a cell σ_i, is verified in the three steps $3e_c + level_c(i) + 8, 3e_c + level_c(i) + 9, 3e_c + level_c(i) + 10$. □

Phase 4 (FSSP-IV: Counting down for entering the firing state).
Overview: In this phase, each cell performs a *countdown* as follows. At each step a cell σ_i decrements the counter k of the broadcast message e^k (from Phase 3) by one, until $k = 0$. If $k = 0$, σ_i becomes a *ready-to-synchronize* cell. A squad *ready-to-synchronize* cell enters the firing state and a non-squad *ready-to-synchronize* cell enters the initial state.
Precondition: As described in the postcondition of Phase 3.
Postcondition: Firing squad cells end in state s_{10} and other cells end in state s_0. All cells are empty.
Number of steps: For each cell σ_i, this phase takes $e_c - level_c(i) + 2$ steps.
Rules: 9. For state s_9:

1) $s_9\ ag \rightarrow_{max} s_9\ a$
2) $s_9\ af \rightarrow_{max} s_{10}$
3) $s_9\ a \rightarrow_{max} s_0$
4) $s_9\ a \rightarrow_{max} s_{10}$

Proof. For a cell σ_i, from the postcondition of Phase 3, the number of g's is a multiple of the number of a's, where the multiplier is $(e_c - level_c(i) + 1)$.

- Between step $3e_c + level_c(i) + 11$ and step $4e_c + 12$: σ_i remains in state s_9 and consumes $count_c(i)$ copies of g at each step.

- At step $4e_c + 13$: if σ_i has one f, it transits to state s_{10} and consumes one a and one f, otherwise, it transits to state s_0 and consumes one a.

In this phase, each cell σ_i progresses through steps $3e_c + level_c(i) + 11, \ldots, 4e_c + 12, 4e_c + 13$ as stated above. Thus, each σ_i takes $e_c - level_c(i) + 2$ steps.

The number of each object a, f and g, for a cell σ_i, is verified in steps $3e_c + level_c(i) + 11, \ldots, 4e_c + 13$. □

Theorem 3. *With finite set of alphabet objects, we can solve the FSSP in* $4e_c + 13$ *steps, where* e_c *is the eccentricity of the commander* σ_c*.*

Proof. The result is obtained by summing the individual running times of the four phases, as given by Phases 1, 2, 3 and 4: $(level_c(i) + 4) + (3e_c - level_c(i) + 4) + (level_c(i) + 3) + (e_c - level_c(i) + 2) = 4e_c + 13$. □

In Table 1 on Page 14, we present the traces of the FSSP algorithm for the P module shown in Figure 1. Note, for convenience, the phase boundaries are shaded in Table 1.

4 Improved deterministic FSSP solution using cell IDs

The difficulties discussed in Remark 2 are naturally resolved, if a cell is able to determine the convergecast message sender, i.e., to distinguish between its successors or peers. In this section, we provide a revised FSSP solution, which allows each cell to determine the message sender by having the sender's label (cell IDs) attached in the received message, e.g., a cell σ_i sends an object c_i.

Since a cell σ_i is now able to distinguish the messages sent from successors and peers, σ_i does not need to communicate with all its peers during the convergecast phase. Thus, the number of steps σ_i takes in Phase 2 (convergecasts from terminal cells) is reduced by e_c, which contributes to the improvement of Theorem 4.

The initial configuration of a P module $\Pi' = (O', K', \delta', P')$ (with n cells) for this revised FSSP solution is same as the initial configuration of the previous FSSP solution in Section 3, with the following adjustments: $O' = \{a, b, e, f, g, h\} \cup \{c_i, \bar{c}_i, p_i, \bar{p}_i \mid i \in \{1, \ldots, n\}\}$. For each cell σ_i, the set of multiset rewriting rules R'_i is given in Phases $1'$, $2'$, $3'$ and $4'$.

This improved FSSP solution is essentially the FSSP solution given in Section 3, with reduced running time in Phase 2. Using cell IDs, this revised FSSP solution enables each cell to avoid steps that were needed to distinguish the message sender. That is, all aspects of the previous FSSP solution remain the same. Thus, the overviews and the correctness proofs of all four phases are omitted.

Phase $1'$ (FSSP-I: First broadcast from the commander).
Precondition: The initial configuration of P module Π'.
Postcondition: Each cell σ_i ends in state s_4 and σ_i has: $count_c(i)$ copies of a; one b, if $\sigma_i = \sigma_c$; $count_c(i)$ copies of $\bar{c}_j$, for each σ_i's *successor* σ_j; three copies of e, if $\sigma_i = \sigma_c$; one f, if $\sigma_i \in F$.
Number of steps: For each cell σ_i, this phase takes $level_c(i) + 4$ steps.

Rules:

0. For state s_0:
   1) $s_0\ a \rightarrow_{max} s_1\ abe\ (p_i)_{\updownarrow_{repl}}$
   2) $s_0\ p_j \rightarrow_{max} s_1\ a\bar{p_j}\ (c_i p_i)_{\updownarrow_{repl}}$
1. For state s_1:
   1) $s_1\ a \rightarrow_{max} s_2\ a$
   2) $s_1\ c_j \rightarrow_{max} s_2$
   3) $s_1\ p_j \rightarrow_{max} s_2$
   4) $s_1\ be \rightarrow_{max} s_2\ bee$
2. For state s_2:
   1) $s_2\ a \rightarrow_{max} s_3\ a$
   2) $s_2\ c_j \rightarrow_{max} s_3\ \bar{c}_j$
   3) $s_2\ p_j \rightarrow_{max} s_3$
   4) $s_2\ be \rightarrow_{max} s_3\ bee$
3. For state s_3:
   (a) $s_3\ a \rightarrow_{max} s_4\ a$

Phase 2′ (FSSP-II: Convergecasts from terminal cells).
Precondition: As described in the postcondition of Phase 1′.
Postcondition: This phase ends when the commander enters state s_7. Each cell σ_i ends in state s_7 and σ_i has: $count_c(i)$ copies of a; e_c+2 copies of e, if $\sigma_i = \sigma_c$; one f, if $\sigma_i \in F$.
Number of steps: For each cell σ_i, this phase takes $2e_c - level_c(i) + 4$ steps.
Rules:

4. For state s_4:
   1) $s_4\ h \rightarrow_{max} s_5$
   2) $s_4\ c_j\bar{c}_j \rightarrow_{max} s_4$
   3) $s_4\ a\bar{c}_j \rightarrow_{max} s_4\ a\bar{c}_j$
   4) $s_4\ a \rightarrow_{max} s_4\ ah\ (c_i)_{\updownarrow_{repl}}$
   5) $s_4\ be \rightarrow_{max} s_4\ bee$
   6) $s_4\ be \rightarrow_{max} s_5$
   7) $s_4\ ee \rightarrow_{max} s_5\ e$
5. For state s_5:
   1) $s_5\ a \rightarrow_{max} s_6\ a$
6. For state s_6:
   1) $s_6\ c_j\bar{p_j} \rightarrow_{max} s_6$
   2) $s_6\ \bar{p_j} \rightarrow_{max} s_6\ \bar{p_j}$
   3) $s_6\ c_k \rightarrow_{max} s_6$
   4) $s_6\ a \rightarrow_{max} s_7\ a$

Phase 3′ (FSSP-III: Second broadcast from the commander).
Precondition: As described in the postcondition of Phase 2′.
Postcondition: Each cell σ_i ends in state s_9 and σ_i has: $count_c(i)$ copies of a; one f, if $\sigma_i \in F$; $(e_c - level_c(i) + 1)count_c(i)$ copies of g.
Number of steps: For each cell σ_i, this phase takes $level_c(i) + 3$ steps.
Rules:

7. For state s_7:
   1) $s_7\ ae \rightarrow_{max} s_8\ ah$
   2) $s_7\ e \rightarrow_{max} s_8\ g\ (e)_{\updownarrow_{repl}}$
8. For state s_8:
   1) $s_8\ h \rightarrow_{max} s_8$
   2) $s_8\ a \rightarrow_{max} s_9\ a$
   3) $s_8\ e \rightarrow_{max} s_9$

Phase 4′ (FSSP-IV: Counting down for entering the firing state).
Precondition: As described in the postcondition of Phase 3′.
Postcondition: Firing squad cells end in state s_{10} and other cells end in state s_0. All cells are empty.
Number of steps: For each cell σ_i, this phase takes $e_c - level_c(i) + 2$ steps.

Rules: 9. For state s_9:

1) $s_9\ ag \rightarrow_{max} s_9\ a$ 3) $s_9\ a \rightarrow_{max} s_0$

2) $s_9\ af \rightarrow_{max} s_{10}$ 4) $s_9\ a \rightarrow_{max} s_{10}$

Theorem 4. *Extending the set of alphabet objects with cell IDs, we can solve the FSSP in $3e_c + 13$ steps, where e_c is the eccentricity of the commander σ_c.*

Proof. The result is obtained by summing the individual running times of the four phases, as given by Phases 1′, 2′, 3′ and 4′: $(level_c(i) + 4) + (2e_c - level_c(i) + 4) + (level_c(i) + 3) + (e_c - level_c(i) + 2) = 3e_c + 13$. □

5 Conclusion

We have proposed improved deterministic FSSP solutions in the framework of P systems, expressed using P modules. Both of our solutions are based on static structures and traditional rules. Our first FSSP algorithm runs in $4e_c + 13$ steps, where e_c is the eccentricity of the initiating cell. Our second FSSP algorithm, which is extended with the facility of using cell IDs, runs in $3e_c + 13$ steps. Note that, the former algorithm benefits by using a finite set of alphabet objects, while the latter algorithm requires a linear number of alphabet objects.

We end this paper with a couple of open problems. First, is the multiplier of the running times optimal for our two solutions? Note, we can slightly reduce the number of states and run-time steps (additive constants). Another question relates to the type of channels. Are there FSSP solutions for an arbitrary P module that uses *simplex* channels and its structure is also strongly connected?

References

[1] Artiom Alhazov, Maurice Margenstern, and Sergey Verlan. Fast synchronization in P systems. In David W. Corne, Pierluigi Frisco, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Workshop on Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 2008.

[2] Francesco Bernardini, Marian Gheorghe, Maurice Margenstern, and Sergey Verlan. How to synchronize the activity of all components of a P system? *Int. J. Found. Comput. Sci.*, 19(5):1183–1198, 2008.

[3] Cristian S. Calude, Michael J. Dinneen, Gheorghe Păun, Mario J. Pérez-Jiménez, and Grzegorz Rozenberg, editors. *Unconventional Computation, Fourth International Conference, UC 2005, Sevilla, Spain, October 3-7, 2005, Proceedings*, volume 3699 of *Lecture Notes in Computer Science*. Springer, 2005.

[4] Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. New solutions to the firing squad synchronization problem for neural and hyperdag P systems. In *Membrane Computing and Biologically Inspired Process Calculi, Third Workshop, MeCBIC 2009, Bologna, Italy, September 5, 2009*, pages 117–130, 2009.

[5] Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. P systems and the Byzantine agreement. In *The Journal of Logic and Algebraic Programming*, pages 1–31, to appear. (Also see CDMTCS-375 research report `http://www.cs.auckland.ac.nz/CDMTCS/researchreports/375Byzantine.pdf`).

[6] Roger L. Freeman. *Fundamentals of Telecommunications, 2nd Edition*. Wiley-IEEE Press, 2005.

[7] Tim Carter Humphrey. *Cell Cycle Control: Mechanisms and Protocols*. Humana Press, 2005.

[8] Katsunobu Imai, Kenichi Morita, and Kenji Sako. Firing squad synchronization problem in number-conserving cellular automata. *Fundam. Inform.*, 52(1-3):133–141, 2002.

[9] Kojiro Kobayashi and Darin Goldstein. On formulations of firing squad synchronization problems. In Calude et al. [3], pages 157–168.

[10] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[11] Jacques Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theor. Comput. Sci.*, 50:183–238, 1987.

[12] Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Structured modelling with hyperdag P systems: Part A. In Miguel Angel Martínez del Amor, Enrique Francisco Orejuela-Pinedo, Gheorghe Păun, Ignacio Pérez-Hurtado, and Agustin Riscos-Núñez, editors, *Membrane Computing, Seventh Brainstorming Week, BWMC 2009, Sevilla, Spain, February 2-6, 2009*, volume 2, pages 85–107. Universidad de Sevilla, 2009.

[13] Gheorghe Păun. *Membrane Computing: An Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[14] Gheorghe Păun. Introduction to membrane computing. In Gabriel Ciobanu, Mario J. Pérez-Jiménez, and Gheorghe Păun, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 1–42. Springer, 2006.

[15] Hubert Schmid and Thomas Worsch. The firing squad synchronization problem with many generals for one-dimensional CA. In Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell, editors, *IFIP TCS*, pages 111–124. Kluwer, 2004.

[16] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts, Seventh Edition*. Wiley, 2004.

[17] Helge Szwerinski. Time-optimal solution of the firing-squad-synchronization-problem for n-dimensional rectangles with the general at an arbitrary position. *Theor. Comput. Sci.*, 19(3):305–320, 1982.

[18] Hiroshi Umeo, Masaya Hisaoka, and Shunsuke Akiguchi. A twelve-state optimum-time synchronization algorithm for two-dimensional rectangular cellular arrays. In Calude et al. [3], pages 214–223.

Appendix

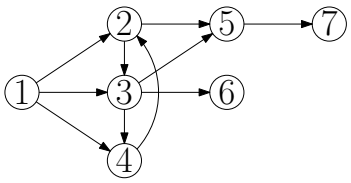


Figure 1: A digraph structure for a P module with duplex channels.

Table 1: The FSSP trace of the P module shown in Figure 1, where $c = 1$, $e_1 = 3$, $F = \{\sigma_1, \sigma_3, \sigma_5, \sigma_7\}$.

	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7
0	$s_0\ af$	s_0	$s_0\ f$	s_0	$s_0\ f$	s_0	$s_0\ f$
1	$s_1\ abef$	$s_0\ d$	$s_0\ df$	$s_0\ d$	$s_0\ f$	s_0	$s_0\ f$
2	$s_2\ abd^3e^2f$	$s_1\ abd^2u$	$s_1\ abd^2fu$	$s_1\ abd^2u$	$s_0\ d^2f$	$s_0\ d$	$s_0\ f$
3	$s_3\ abd^3e^3fu^3$	$s_2\ abd^2u^2$	$s_2\ abd^3fu^2$	$s_2\ abu^2$	$s_1\ a^2b^2fu^2$	$s_1\ abu$	$s_0\ d^2f$
4	$s_4\ abd^3e^4f$	$s_3\ abd^2u^2v^2$	$s_3\ abd^3fu^3v^2$	$s_3\ abv^2$	$s_2\ a^2b^2d^2f$	$s_2\ ab$	$s_1\ a^2b^2fu^2$
5	$s_4\ abd^3e^5f$	$s_4\ abd^2v^2$	$s_4\ abd^3fv^2$	$s_4\ abv^2$	$s_3\ a^2b^2d^2fu^2$	$s_3\ ab$	$s_2\ a^2b^2f$
6	$s_4\ abd^3e^6fw$	$s_4\ abd^2v^2w$	$s_4\ abd^3fv^2w$	$s_4\ abhv^2$	$s_4\ a^2b^2d^2f$	$s_4\ ab$	$s_3\ a^2b^2f$
7	$s_4\ abd^3e^7fw$	$s_4\ abd^2v^2w$	$s_4\ abd^3fv^2w^2$	$s_5\ abv^2$	$s_4\ a^2b^2d^2f$	$s_4\ abh$	$s_4\ a^2b^2f$
8	$s_4\ abd^3e^8fw$	$s_4\ abd^2v^2w$	$s_4\ abcd^3fv^2w$	$s_5\ abv^2$	$s_4\ a^2b^2d^2fw^2$	$s_5\ abh$	$s_4\ a^2b^2fh^2$
9	$s_4\ abd^3e^9fw$	$s_4\ abd^2v^2w$	$s_4\ abcd^3fv^2w$	$s_5\ abv^2$	$s_4\ a^2b^2c^2d^2f$	$s_6\ ab$	$s_5\ a^2b^2fh^2$
10	$s_4\ abd^3e^{10}fw$	$s_4\ abd^2v^2w^3$	$s_4\ abcd^3fv^2w^3$	$s_5\ abv^2$	$s_4\ a^2b^2c^2d^2fh^2$	$s_6\ ab$	$s_6\ a^2b^2fw^2$
11	$s_4\ abd^3e^{11}fw$	$s_4\ abc^2d^2v^2w$	$s_4\ abc^3d^3fv^2w$	$s_5\ abv^2$	$s_5\ a^2b^2fh^2$	$s_6\ ab$	$s_6\ a^2b^2c^2fw^2$
12	$s_4\ abd^3e^{12}fw^3$	$s_4\ abc^2d^2hv^2w^2$	$s_4\ abc^3d^3fhv^2w^2$	$s_5\ abv^2w^2$	$s_6\ a^2b^2fw^2$	$s_6\ abw$	$s_6\ a^2fw^2$
13	$s_4\ abc^3d^3e^{13}f$	$s_5\ abc^2hv^2$	$s_5\ abc^2fhv^2$	$s_5\ abc^2hv^2$	$s_6\ a^2b^2c^2fw^2$	$s_6\ abcw$	$s_7\ a^2f$
14	$s_4\ abc^3d^3e^{14}fh$	$s_6\ abc^2v^2w$	$s_6\ abc^2fv^2w$	$s_6\ abc^2v^2w$	$s_6\ a^2fw^2$	$s_6\ aw$	$s_7\ a^2f$
15	$s_5\ ae^{15}fh$	$s_6\ abcw$	$s_6\ abcfw$	$s_6\ abcw$	$s_7\ a^2f$	$s_7\ a$	$s_7\ a^2f$
16	$s_6\ ae^5f$	$s_6\ aw$	$s_6\ afw$	$s_6\ aw$	$s_7\ a^2f$	$s_7\ a$	$s_7\ a^2f$
17	$s_7\ ae^5f$	$s_7\ a$	$s_7\ af$	$s_7\ a$	$s_7\ a^2f$	$s_7\ a$	$s_7\ a^2f$
18	$s_8\ afg^4h$	$s_7\ ae^4$	$s_7\ ae^4f$	$s_7\ ae^4$	$s_7\ a^2f$	$s_7\ a$	$s_7\ a^2f$
19	$s_8\ ae^9fg^4$	$s_8\ ae^6g^3h$	$s_8\ ae^6fg^3h$	$s_8\ ae^6g^3h$	$s_7\ a^2e^6f$	$s_7\ ae^3$	$s_7\ a^2f$
20	$s_9\ afg^4$	$s_8\ ae^{10}g^3$	$s_8\ ae^{12}fg^3$	$s_8\ ae^6g^3$	$s_8\ a^2fg^4h^2$	$s_8\ ag^2h$	$s_7\ a^2e^4f$
21	$s_9\ afg^3$	$s_9\ ag^3$	$s_9\ afg^3$	$s_9\ ag^3$	$s_8\ a^2e^2fg^4$	$s_8\ ag^2$	$s_8\ a^2fg^2h^2$
22	$s_9\ afg^2$	$s_9\ ag^2$	$s_9\ afg^2$	$s_9\ ag^2$	$s_9\ a^2fg^4$	$s_9\ ag^2$	$s_8\ a^2fg^2$
23	$s_9\ afg$	$s_9\ ag$	$s_9\ afg$	$s_9\ ag$	$s_9\ a^2fg^2$	$s_9\ ag$	$s_9\ a^2fg^2$
24	$s_9\ af$	$s_9\ a$	$s_9\ af$	$s_9\ a$	$s_9\ a^2f$	$s_9\ a$	$s_9\ a^2f$
25	s_{10}	s_0	s_{10}	s_0	s_{10}	s_0	s_{10}