CDMTCS
Research
Report
Series

Structured Modelling with Hyperdag P Systems: Part B

Radu Nicolescu
Michael J. Dinneen
Yun-Bum Kim

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

CDMTCS-373
October 2009

Centre for Discrete Mathematics and
Theoretical Computer Science

Structured Modelling with Hyperdag P Systems: Part B

Radu Nicolescu, Michael J. Dinneen and Yun-Bum Kim
Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand

October 2009

Abstract

In an earlier paper, we presented an extension to the families of P systems, called hyperdag P systems (hP systems), by proposing a new underlying topological structure based on the hierarchical dag structure (instead of trees or digraphs).

In this paper, we develop building-block membrane algorithms for discovery of the global topological structure from the local cell point of view. In doing so, we propose more convenient operational modes and transfer modes, that depend only on each of the individual cell rules.

Additionally, we propose two uniform solutions to an open problem: the Firing Squad Synchronization Problem (FSSP), for hyperdag and symmetric neural P systems, with anonymous cells. Our 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. The first and fast solution is based on a novel proposal, which dynamically extends P systems with mobile channels. The second solution is solely based on classical rules and static channels. In contrast to the previous solutions, which work for tree-based P systems, our solutions synchronize any subset of the underlying digraph, and do not require membrane polarizations or conditional rules, but require states, as typically used in hyperdag and neural P systems.

Finally, by extending our initial work on the visualization of hP system membranes with interconnections based on dag structures without transitive arcs, we propose several ways to represent structural relationships, that may include transitive arcs, by simple-closed planar regions, which are folded (and possibly twisted) in three dimensional space.

1 Introduction

Following [NDK09a], we develop basic building blocks that are relevant for network discovery (see also [Lyn96]): broadcast, convergecast, flooding, determine shortest paths and other basic metrics (such as, the number of nodes, descendants, paths) which highlight the versatility of the dag structure underlying hyperdag P systems (hP systems)

[NDK08]. For Algorithms 2 and 6, we extend to dags the approach pioneered by Ciobanu *et al.* in [Cio03, CDK02]. We also provide explicit rewriting and transfer rules, as a replacement for pseudo-code. In this process, we identify areas where our initial model was not versatile enough and we propose corresponding adjustments, that can also be retrofitted to other models of the P family, such as the refinement of the rewriting and transfer modes. We also advocate the weak policy for priority rules [Pău06], which we believe is closer to the actual task scheduling in operating systems.

Further, we continue the study of FSSP in the framework of P systems, by providing solutions for hP systems and for neural P systems [Pău02] with symmetric communication channels (snP systems). Following [DKN09], we propose two deterministic solutions to a variant of FSSP [Szw82], in which there is a single commander, at an arbitrary position. We further generalize this problem by synchronizing a subset of cells of the considered hyperdag or neural P system. The first solution is based on a novel proposal, which dynamically extends P systems with mobile channels. The second solution is substantially longer, but is solely based on classical rules and static channels. In contrast to the previous FSSP solutions for tree-based P systems [BGMV08, AMV08], our solutions synchronize any subset of the underlying digraph, and do not require membrane polarizations or conditional rules, but require states, as typically used in hyperdag and neural P systems.

We have earlier proposed an algorithm to visually represent hP systems, where the underlying cell structure was restricted to a canonical dag (i.e., without transitive arcs) [NDK09b]. Nodes were represented as simple closed regions on the plane (with possible nesting or overlaps) and channels were represented by direct containment relationships of the regions. Following [NDK09a], we extend this planar representation by presenting several plausible solutions that enable us to visualize any hP system, modelled as an arbitrary dag, in the plane (or almost). Additionally, we discuss advantages and limitations of these solutions, and provide a new algorithm for representing general hP systems, where transitive arcs are not excluded.

2 Preliminaries

A (binary) *relation* R over two sets X and Y is a subset of their Cartesian product, $R \subseteq X \times Y$. For $A \subseteq X$ and $B \subseteq Y$, we set $R(A) = \{y \in Y \mid \exists x \in A, (x, y) \in R\}$, $R^{-1}(B) = \{x \in X \mid \exists y \in B, (x, y) \in R\}$.

A *digraph* (directed graph) G is a pair (X, A), where X is a finite set of elements called *nodes* (or *vertices*), and A is a binary relation $A \subseteq X \times X$, of elements called *arcs*. A length $n-1$ *path* is a sequence of n distinct nodes $x_1, \ldots, x_n$, such that $\{(x_1, x_2), \ldots, (x_{n-1}, x_n)\} \subseteq A$. A *cycle* is a path $x_1, \ldots, x_n$, where $n \geq 1$ and $(x_n, x_1) \in A$. A digraph is *symmetric* if its relation A is symmetric, i.e., $(x_1, x_2) \in A \Leftrightarrow (x_2, x_1) \in A$. By default, all digraphs considered in this paper, and all structures from digraphs (dag, rooted tree, see below) will be *weakly connected*, i.e., each pair of nodes is connected via a chain of arcs, where the arc direction is not relevant.

A *dag* (directed acyclic graph) is a digraph (X, A) without cycles. For $x \in X$, $A^{-1}(x)$

are x's *parents*, $A(x)$ are x's *children*, and $A(A^{-1}(x))\backslash\{x\}$ are x's *siblings*.

A *rooted tree* is a *special case of dag*, where each node has exactly one parent, except a distinguished node, called *root*, which has none.

Throughout this paper, we will use the term *graph* to denote a *symmetric digraph* and *tree* to denote a *rooted tree*.

For a given tree, dag or digraph, we define e_c, the *eccentricity* of a node c, as the maximum length of a shortest path between c and any other reachable node in the corresponding 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 δ and node 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 $G = (X, A)$, we set $Neighbor(x) = A(x) = \{y \mid (x, y) \in A\}$. 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 digraph 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 *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*. We define $maxlevel_c = max\{level_c(x) \mid x \in X\}$ and we note $e_c = maxlevel_c$. A *level-preserving path* from c to a node y is a sequence $x_0, \dots, x_k$, 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 further define $count_c(y)$ as the number of distinct level-preserving paths from c to y.

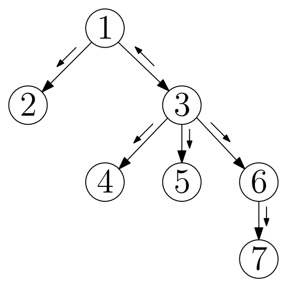
The level of a node and number of level-preserving paths to it can be determined by a standard breadth-first-search, as shown in Algorithm 1. Intuitively, this algorithm defines a *virtual dag* based on successor relation and, if the original structure is a tree, this algorithm will "reset" the root at another node in that tree.

Algorithm 1: Determine levels and count level-preserving paths.

- INPUT: A tree, dag or digraph, with nodes $\{1, \dots, n\}$ and a commander $c \in \{1, \dots, n\}$.
- OUTPUT: The arrays $level_c[\,]$ and $count_c[\,]$ of shortest distances and number of level-preserving paths from c to each node in the structure, over the $Neighbor$ relation.

```
**array** $level_c[1,\ldots,n] = [-1,\ldots,-1]$; $count_c[1,\ldots,n] = [0,\ldots,0]$
**queue** $Q = ()$
$Q \Leftarrow c$
$level_c[c] = 0$; $count_c[c] = 1$
**while** $Q \neq ()$ **do**
        $x \Leftarrow Q$
        **for each** $y \in Neighbor(x)$ **do**
            **if** $level_c[y] = -1$ **then**
                $Q \Leftarrow y$
                $level_c[y] = level_c[x] + 1$
            **if** $level_c[y] = level_c[x] + 1$ **then**
                $count_c[y] = count_c[y] + count_c[x]$
**return** $level_c$
```

Example 1 Figures 1, 2 and 3 show $level_c$, *predecessors*, *successors*, *peers* and $count_c$, for a tree, a dag and a digraph structure, respectively. Small side-arrows indicate the arcs traversed while computing the levels, over the induced $Neighbor$ relation, as described in Algorithm 1.

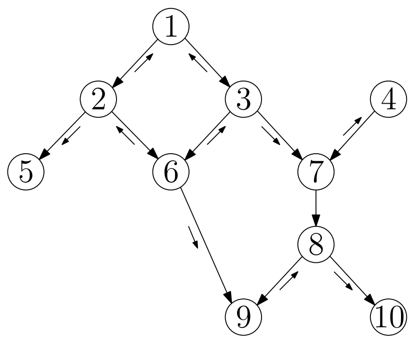


Node	$level_c$	*predecessors*	*successors*	*peers*	$count_c$
1	1	3	2	–	1
2	2	1	–	–	1
3	0	–	1, 4, 5, 6	–	1
4	1	3	–	–	1
5	1	3	–	–	1
6	1	3	7	–	1
7	2	6	–	–	1

Figure 1: Left: a tree (taken from Bernardini *et al* [BGMV08]), with commander $c = 3$, $e_3 = 2$; Right: table with node levels, predecessors, successors, peers and $count_c$'s.

Definition 2 (Hyperdag P systems) An *hP system* of order n is a system $\Pi = (O, \sigma_1, \ldots, \sigma_n, \delta, I_{out})$, where:

1. O is an ordered finite non-empty alphabet of *objects*;
2. $\sigma_1, \ldots, \sigma_n$ are *cells*, of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i)$, $1 \leq i \leq n$, where:
    - Q_i is a finite set of *states*;
    - $s_{i,0} \in Q_i$ is the *initial state*;
    - $w_{i,0} \in O^*$ is the *initial multiset* of objects;



Node	$level_c$	*predecessors*	*successors*	*peers*	$count_c$
1	2	$2, 3$	–	–	2
2	1	6	$1, 5$	–	1
3	1	6	$1, 7$	–	1
4	3	7	–	–	1
5	2	2	–	–	1
6	0	–	$2, 3, 9$	–	1
7	2	3	4	8	1
8	2	9	10	7	1
9	1	6	8	–	1
10	3	8	–	–	1

Figure 2: Left: a dag with commander $c = 6$, $e_6 = 3$ (siblings excluded); Right: table with node levels, predecessors, successors, peers and $count_c$'s.

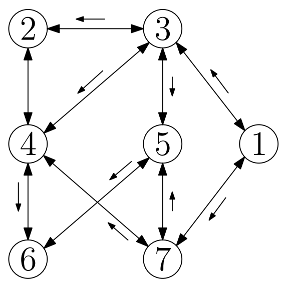
- P_i is a finite set of multiset rewriting *rules* of the form $sx \to s'x'u_{\uparrow}v_{\downarrow}w_{\leftrightarrow} y_{go}z_{out}$, where $s, s' \in Q_i$, $x, x' \in O^*$, $u_{\uparrow} \in O^*_{\uparrow}$, $v_{\downarrow} \in O^*_{\downarrow}$, $w_{\leftrightarrow} \in O^*_{\leftrightarrow}$, $y_{go} \in O^*_{go}$ and $z_{out} \in O^*_{out}$, with the restriction that $z_{out} = \lambda$ for all $i \in \{1, \ldots, n\} \backslash I_{out}$;

3. δ is a set of dag parent-child arcs on $\{1, \ldots, n\}$, i.e., $\delta \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\}$, representing *duplex* channels between cells;
4. $I_{out} \subseteq \{1, \ldots, n\}$ indicates the *output cells*, the only cells allowed to send objects to the "environment".

A *symmetric* nP system, (here) in short, a *snP system*, is an nP system where the underlying digraph *syn* is symmetric (i.e., a graph). For further definitions describing the evolution of hP and nP systems, such as *configuration*, *rewriting modes*, *transfer modes*, *transition steps*, *halting* and *results*, see our previous work [NDK08]. For all structures, we also utilize the *weak policy* for applying *priorities* to rules, as defined by Păun [Pău06].

Remark 3 Most of the P systems considered here (i.e., nP systems, snP systems, hP systems with siblings and hP systems without siblings) define a tag *go* that sends a multiset of objects along the previously defined *Neighbor* relation. Traditional tree-based P systems do not directly provide this facility, however, it can be easily provided by the union of *out* and *in*! target indications, that represent sending "to parent" and "to all children", respectively. That is, $(w, go) \equiv (w, out)(w, in!)$.

The dynamic operations of hP systems, i.e., the configuration changes via object rewriting and object transfer, are a natural extension of similar operations used by transition and neural P systems. Our earlier paper, [NDK09b], describes the dynamic behavior of hP systems, in more detail.



Node	$level_c$	*predecessors*	*successors*	*peers*	$count_c$
1	0	–	$3, 7$	–	1
2	2	3	–	4	1
3	1	1	$2, 4, 5$	–	1
4	2	$3, 7$	6	2	2
5	2	$3, 7$	6	–	2
6	3	$4, 5$	–	–	4
7	1	1	$4, 5$	–	1

Figure 3: Left: a graph with commander $c = 1$, $e_1 = 3$; Right: table with node levels, predecessors, successors, peers and $count_c$'s.

We measure the *runtime complexity* of a P system in terms of *P-steps*, where a P-step corresponds to a transition on a parallel P machine. If no more transitions are possible, the hP system halts. For halted hP systems, the *computational result* is the multiset of objects emitted *out* (to the "environment"), over all the time steps, from the output cells I_{out}. The *numerical result* is the set of vectors consisting of the object multiplicities in the multiset result. Within the family of P systems, two systems are *functionally equivalent* if they yield the same computational result.

Example 4 Figure 4 shows the structure of an hP system that models a computer network. Four computers are connected to "Ethernet Bus 1", the other four computers are connected to "Ethernet Bus 2", while two of the first group and two of the second group are at the same time connected to a wireless cell. In this figure we also suggest that "Ethernet Bus 1" and "Ethernet Bus 2" are themselves connected to a higher level communication hub, in a generalized hypergraph.

We have already shown, [NDK09b], that our hP systems can simulate any transition P system [Pău06] and any snP system [Pău02], with the same number of steps and object transfers. To keep the arguments simple, we have only considered systems without additional features, such as dissolving membranes, priorities or polarities. However, our definition of hP systems can also be extended, as needed, with additional features, in a straightforward manner, and we do so in this paper.

Model refinements

- As initially defined [NDK09b], the rules are applied according to the current cell state s, in the rewriting mode $\alpha(s) \in \{min, par, max\}$, and the objects are sent out in the transfer mode $\beta(s) \in \{one, spread, repl\}$. In this paper, we propose

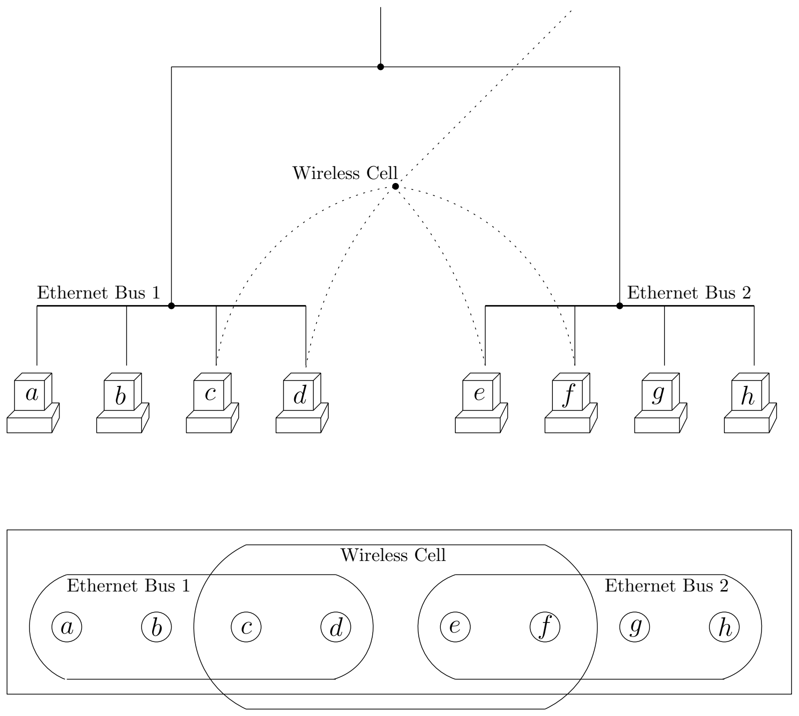


Figure 4: A computer network and its corresponding hypergraph representation.

a refinement to these modes and allow that *the rewriting and transfer modes to depend on the rule used* (instead of the state), as long as there are no conflicting requirements. We will highlight the cases where this mode extension is essential.

- We also consider rules with *priorities*, in their *weak* interpretation [Pău06]. In the current paper, *lower numbers* (i.e., first enumerated) indicate *higher priority*. In the *weak* interpretation of the priority, rules are applied in decreasing order of their priorities—where a lower priority rule can only applied after all higher priority rules have been applied (as required by the rewriting modes). In contrast, in the *strong* interpretation, a lower priority rule cannot be applied at all, if a higher priority rule was applied. We will highlight the cases where the weak interpretation is required.

3 Basic algorithms for network discovery—Without IDs

In this section and the following, we study several basic distributed algorithms for network discovery, adapted to hP systems. Essentially, all cells start in the same state and with the same or similar (set of) rules, but there are several different scenarios:

1. Initially, cells know nothing about the structure in which they are linked, and must even discover their local neighborhood (i.e., their parents, children, siblings), as

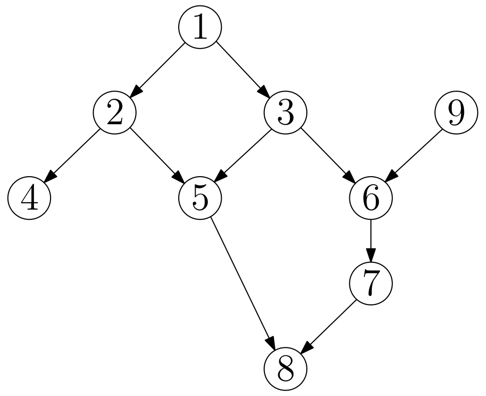


Figure 5: A dag for illustrating our network discovery algorithms.

well as some global model topology characteristics (such as various dag measures or shortest paths).

2. As above, but each cell has its own ID (identifier) and is allowed to have custom rules for this ID.

3. As above, each cell has its own ID and also knows the details of its immediate neighbors (parents, children and, optionally, siblings).

Algorithm 2: Broadcast to all descendants.

Precondition: Cells do not need any inbuilt knowledge about the network topology. All cells start in state s_0, with the same rules. The initiating cell has an additional object a, that is not present in any other cell.

Postcondition: All descendant cells are eventually visited and enter state s_1.

Rules:

1. $s_0 a \to s_1 a_\downarrow$, with $\alpha = min$, $\beta = repl$.

2. $s_1 a \to s_1$, with $\alpha = par$.

Proof. This is a *deterministic* algorithm. Rule 1 is applied exactly once, when a cell is in state s_0 and it contains an a. This a is consumed, the cell enters state s_1 and another a is sent to all the children, replicated as necessary. Additional a's may appear in a cell, because, in a dag structure, a cell may have more than one parent. Rule 2 is applicable in state s_1 and silently discards any additional a's, without changing the state and without interacting with other cells. All a's will eventually disappear from the system—however, cells themselves may never know that the algorithm has completed and no other a's will come from their parents. By induction, all descendants will receive an a and enter state s_1. $\square$

Remarks 5

- This broadcast algorithm can be initiated anywhere in the dag. However, it is probably most useful when initiated on a dag source, or on all sources at the same time (using the same object a or a different object for each source).
- This algorithm completes after $h + 1$ P-steps, where h is the *height* of the initiating node.
- State s_1 may be reached before the algorithm completes and cannot be used as a termination indicator.
- Several other broadcasting algorithms can be built in a similar manner, such as *broadcast to all ancestors* or *broadcast to all reachable cells* (ancestors and descendants).
- This algorithm family follows the approach used by Ciobanu *et al.* [Cio03, CDK02], for tree based algorithms, called *skin membrane broadcast* and *generalized broadcast.*

Example 6 We illustrate the algorithm for broadcasting to all descendants, for the hP system shown in Figure 5.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	s_0a	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1	s_0a	s_0a	s_0	s_0	s_0	s_0	s_0	s_0
2	s_1	s_1	s_1	s_0a	s_0aa	s_0a	s_0	s_0	s_0
3	s_1	s_1	s_1	s_1	s_1a	s_1	s_0a	s_0a	s_0
4	s_1	s_1	s_1	s_1	s_1	s_1	s_1	s_1a	s_0
5	s_1	s_1	s_1	s_1	s_1	s_1	s_1	s_1	s_0

Algorithm 3: Counting all paths from a given ancestor.

Precondition: Cells do not need any inbuilt knowledge about the network topology. All cells start in state s_0 and with the same rules. The initiating cell has an additional object a, not present in any other cell.

Postcondition: All descendant cells are eventually visited, enter state s_1 and will have a number of b's equal to the number of distinct paths from the initiating cell.

Rules:

1. $s_0a \to s_1ba_{\downarrow}$, with $\alpha = par$, $\beta = repl$.

2. $s_1a \to s_1ba_{\downarrow}$, with $\alpha = par$, $\beta = repl$.

Proof. This is a *deterministic* algorithm. Rule 1 is applied when the cell is in state s_0 and an a is available. This a is consumed, the cell enters state s_1, a b is generated and another a is sent to all its children, replicated as necessary. Additional a's may appear in a cell, because, in a dag structure, a cell may have more than one parent. Rule 2 is similar to rule 1. State s_1 is similar to state s_0 and is not essential here, it appears here only to mark visited cells. The number of generated b's is equal to the number of received a's, which eventually will be equal to the number of distinct paths from the initiating cell. All a's will eventually disappear from the system—however, cells themselves may never know that the algorithm has completed, that no other a's will come from their parents and all paths have been counted. A more rigorous proof will proceed by induction. $\square$

Remarks 7

- This algorithm completes after $h+1$ P-steps, where h is the *height* of the initiating node.

- State s_1 may be reached before the algorithm completes and cannot be used as a termination indicator.

- Several other path counting algorithms can be built in a similar manner, such as the number of *paths to a given descendant.*

Example 8 We illustrate the algorithm for counting all paths from a given ancestor, for the hP system shown in Figure 5.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	s_0a	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1b	s_0a	s_0a	s_0	s_0	s_0	s_0	s_0	s_0
2	s_1b	s_1b	s_1b	s_0a	s_0aa	s_0a	s_0	s_0	s_0
3	s_1b	s_1b	s_1b	s_1b	s_1bb	s_1b	s_0a	s_0aa	s_0
4	s_1b	s_1b	s_1b	s_1b	s_1bb	s_1b	s_1b	s_1abb	s_0
5	s_1b	s_1b	s_1b	s_1b	s_1bb	s_1b	s_1b	s_1bbb	s_0

Algorithm 4: Counting the children of a given cell.

Precondition: Cells do not need any inbuilt knowledge about the network topology. The initiating cell and its children start in state s_0 and with the same rules. The initiating cell has an additional object a, not present in any other cell.

Postcondition: The initiating cell ends in state s_1 and will contain a number of c's equal to its child count. The child cells end in state s_1. As a side effect, other parents (if any) of these children will receive superfluous c's—however, these c's can be discarded, if needed (rules not shown here).

Rules:

1. $s_0 a \to s_1 p_\downarrow$, with $\alpha = \mathit{min}$, $\beta = \mathit{repl}$.
2. $s_0 p \to s_1 c_\uparrow$, with $\alpha = \mathit{min}$, $\beta = \mathit{repl}$.

Proof. This is a *deterministic* algorithm with a straightforward proof, not given here. □

Remarks 9

- This algorithm completes after two P-steps.
- Several other algorithms that enumerate the immediate neighborhood can be built in a similar manner, such as *counting parents*, *counting siblings*, *counting neighbors*.

Algorithm 5: Broadcast for counting all children.

Precondition: Cells do not need any inbuilt knowledge about the network topology. All cells start in state s_0 and with the same rules. The initiating cell has an additional object a, not present in any other cell.

Postcondition: Each descendant cell enters state s_1 and, eventually, will contain a number of c's equal to its child count.

Rules:

0. For state s_0:

    1) $s_0 a \to s_1 p_\downarrow$, with $\alpha = min$, $\beta = repl$.

    2) $s_0 p \to s_1 p_\downarrow c_\uparrow$, with $\alpha = min$, $\beta = repl$.

1. For state s_1:

    1) $s_1 p \to s_1$, with $\alpha = par$.

Proof. This is a *deterministic* algorithm: the proof combines those from the broadcast algorithm (Algorithm 2) and the child counting algorithm (Algorithm 4). □

Remarks 10

- This algorithm runs in $h+1$ P-steps, where h is the *height* of the initiating cell.
- State s_1 may be reached before the algorithm completes its cleanup phase and cannot be used as a termination indicator.
- As a side effect, any parent of the visited children that is not a descendant of the initiating node will receive superfluous c's.
- Several other algorithms that broadcast a request to count the immediate neighborhood can be built in a similar manner, such as *broadcast for counting all parents*, *broadcast for counting all siblings*, *broadcast for counting all neighbors*.

Example 11 We illustrate the algorithm for counting all children via broadcasting, for the hP system shown in Figure 5.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	$s_0 a$	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1	$s_0 p$	$s_0 p$	s_0	s_0	s_0	s_0	s_0	s_0
2	$s_1 cc$	s_1	s_1	$s_0 p$	$s_0 pp$	$s_0 p$	s_0	s_0	s_0
3	$s_1 cc$	$s_1 cc$	$s_1 cc$	s_1	$s_1 p$	s_1	$s_0 p$	$s_0 p$	$s_0 c$
4	$s_1 cc$	$s_1 cc$	$s_1 cc$	s_1	$s_1 c$	$s_1 c$	$s_1 c$	$s_1 p$	$s_0 c$
5	$s_1 cc$	$s_1 cc$	$s_1 cc$	s_1	$s_1 c$	$s_1 c$	$s_1 c$	s_1	$s_0 c$

$\boxed{\textbf{Algorithm 6: Counting heights by flooding.}}$

Precondition: Cells do not need any inbuilt knowledge about the network topology. All cells start in state s_0, with the same rules and have no initial object.

Postcondition: All cells end in state s_2. The number of t's in each cell equals the distance from a furthest descendant.

Rules:

0. For state s_0:

   1) $s_0 \rightarrow s_1 a c_{\uparrow}$, $\alpha = min$, $\beta = repl$.

1. For state s_1, the rules will run under the following *priorities*, under the *weak interpretation*:

   1) $s_1 ac \rightarrow s_1 atc_{\uparrow}$, $\alpha = max$, $\beta = repl$.
   2) $s_1 c \rightarrow s_1$, $\alpha = max$.
   3) $s_1 a \rightarrow s_2$, $\alpha = min$.

Proof. Each cell emits a single object c to each of its parents in the first step. During successive active steps, a cell either: (a) uses rule 1.3 to enter the terminating state s_2 or (b) continues via rule 1.1 to forward one c up to each of its parents. In the latter case, since we have $\alpha = max$, and as enabled by the weak interpretation of priorities, rule 1.2 is further used to remove all remaining c's (if any), in the same step. The cell safely enters the end state s_2 when no more c's appear. Induction shows that the set of times that c's appear is consecutive: if a cell at $k > 1$ links away emitted a c, then there must be another cell at $k - 1$ links away emitting another c. Finally, the number of times rule 1.1 is applied is the number of times a cell receives at least one new c from below. These steps are tallied by occurrences of the object t. □

Remarks 12

- This algorithm, like other distributed flooding based algorithms, requires that all cells start at the same time. Achieving this synchronization is a nontrivial task—in Section 5, we suggest a simple and fast algorithm that achieves this synchronization.

- The time complexity of this quick algorithm is $h + 2$ P-steps, where h is the height of the dag. The two extra P-steps correspond to the initial step and the step to detect no more c's.

- This algorithm follows the approach by Ciobanu *et al.* [Cio03, CDK02], for the tree based algorithm called *convergecast*. Here we prefer to use the term *flooding*, and use the term *convergecast* for a result accumulation triggered by an initial broadcast.
- This algorithm makes critical use of the *weak interpretation* for *priorities*.

Example 13 We illustrate the algorithm for counting heights by flooding, for the hP system shown in Figure 5.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1acc	s_1acc	s_1acc	s_1a	s_1ac	s_1ac	s_1ac	s_1a	s_1ac
2	s_1acct	s_1act	s_1acct	s_2	s_1at	s_1act	s_1at	s_2	s_1act
3	s_1acctt	s_1att	s_1actt	s_2	s_2t	s_1att	s_2t	s_2	s_1actt
4	s_1act^3	s_2tt	s_1at^3	s_2	s_2t	s_2tt	s_2t	s_2	s_1at^3
5	s_1at^4	s_2tt	s_2t^3	s_2	s_2t	s_2tt	s_2t	s_2	s_2t^3
6	s_2t^4	s_2tt	s_2t^3	s_2	s_2t	s_2tt	s_2t	s_2	s_2t^3

Algorithm 7: Counting nodes in a single-source dag.

Precondition: Cells do not need any inbuilt knowledge about the network topology. All cells start in state s_0, with the same rules. The initiating cell is the source of a single-source dag and has an additional object a, not present in any other cell.

Postcondition: Eventually, the initiating cell will contain a number of c's equal to the number of all its descendants, including itself, which is also the required node count.

Rules:

0. For state s_0:

   1) $s_0a \to s_3p_{\downarrow}c$, with $\alpha = min$, $\beta = repl$.

   2) $s_0p \to s_1p_{\downarrow}$, with $\alpha = min$, $\beta = repl$.

1. For state s_1:

   1) $s_1 \to s_2c_{\uparrow}$, with $\alpha = min$, $\beta = one$.

2. For state s_2:

   1) $s_2c \to s_2c_{\uparrow}$, with $\alpha = max$, $\beta = one$.

   2) $s_2p \to s_2$, with $\alpha = max$.

Proof. We prove that the source will eventually contain k copies of object c, where k is the order of the single-source dag. The source cell will produce a copy of c following rule 0.1. A non-source cell σ_i will send one c to a parent σ_j, where $j \in \delta^{-1}(i)$, because a node is at state s_1 during at most one P-step, by rule 1.1. A cell σ_i will forward up, using rule 2.1, additional c's to one of its parents, which will eventually arrive at the source. $\square$

Remarks 14

- This algorithm takes up to $2h$ P-steps, where h is the *height* of the initiating cell.
- The end state s_3 is not halting, may be reached before the algorithm completes and cannot be used as a termination indicator.

Example 15 We illustrate the algorithm for counting nodes in a single-source dag via convergecast, for the hP system shown in Figure 5, after removing node 9.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
0	s_0a	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_3c	s_0p	s_0p	s_0	s_0	s_0	s_0	s_0
2	s_3c	s_1	s_1	s_0p	s_0pp	s_0p	s_0	s_0
3	s_3c^3	s_2	s_2	s_1	s_1p	s_1	s_0p	s_0p
4	s_3c^3	s_2c	s_2cc	s_2	s_2p	s_2	s_1	s_1p
5	s_3c^6	s_2	s_2	s_2	s_2	s_2c	s_2c	s_2p
6	s_3c^6	s_2	s_2c	s_2	s_2	s_2c	s_2	s_2
7	s_3c^7	s_2	s_2c	s_2	s_2	s_2	s_2	s_2
8	s_3c^8	s_2	s_2	s_2	s_2	s_2	s_2	s_2

4 Basic algorithms for network discovery—With IDs

In this section we assume each cell has an unique ID and each cell only knows its own ID. Objects may be tagged with IDs to aid in communication.

Algorithm 8: Counting descendants by convergecast—With cell IDs.

Precondition: Cells do not need any inbuilt knowledge about the network topology. For each cell with index i, $1 \leq i \leq m$, the alphabet includes special ID objects c_i and $\bar{c_i}$. All cells start in state s_0 and have the same rules, except several similar, but custom specific, rules to process the IDs. The initiating cell has an additional object a, not present in any other cell.

Postcondition: All visited cells enter state s_1 and, eventually, each cell will contain exactly one $\bar{c}_i$ for each descendant cell with index i, including itself: the number of these objects is the descendant count.

Rules:

0. For state s_0 and cell σ_i (these are custom rules, specific for each cell):
   1) $s_0 a \rightarrow s_1 p_{\downarrow} \bar{c}_i$, with $\alpha = min$, $\beta = repl$.
   2) $s_0 p \rightarrow s_1 p_{\downarrow} c_{i\uparrow} \bar{c}_i$, with $\alpha = min$, $\beta = repl$.
1. For state s_1, the rules will run under the following *priorities*:
   1) $s_1 c_j \bar{c}_j \rightarrow s_1 \bar{c}_j$, for $1 \leq j \leq m$, with $\alpha = max$.
   2) $s_1 \bar{c}_j \bar{c}_j \rightarrow s_1 \bar{c}_j$, for $1 \leq j \leq m$, with $\alpha = max$.
   3) $s_1 c_j \rightarrow s_1 c_{j\uparrow} \bar{c}_j$, for $1 \leq j \leq m$, with $\alpha = max$, $\beta = repl$.
   4) $s_1 p \rightarrow s_1$, with $\alpha = max$.

Proof. Assume that δ is the underlying dag relation. For each cell σ_i, consider the sets $C_i = \{c_j \mid j \in \delta^*(i)\}$, $\bar{C}_i = \{\bar{c}_j \mid j \in \delta^*(i)\}$, which consist of ID objects matching σ_i's children. By induction on the dag height, we prove that each visited cell σ_i will eventually contain the set $\bar{C}_i$, and, if it is not the initiating cell, will also send up all elements of the set C_i, possibly with some duplicates (up to all its parents). The base case, height $h = 0$, is satisfied by rule 0.1, if σ_i is the initiator, or by rule 0.2, otherwise. For cell σ_i at height $h+1$, by induction, each child cell σ_k sends up C_k, possibly with some duplicates. By rules 0.1 and 0.2, cell σ_i further acquires one $\bar{c}_i$ and, if not the initiator, sends up one c_i. From its children, cell σ_i acquires the multiset C_i', consisting of all the elements of the set $\bigcup_{k \in \delta(i)} C_k = C_i \setminus c_i$, possibly with some duplications. Rule 1.3 sends up one copy of each element of multiset C_i' and records a barred copy of it. Rule 1.2 halves the number of duplicates in multiset C_i'. Rule 1.1 filters out duplicates in multiset C_i', if a barred copy already exists. Rule 1.4 clears all p's, which are not needed anymore. □

Remarks 16

- Other counting algorithms can be built in a similar manner, such as *counting ancestors*, *counting siblings*, *counting sources* or *counting sinks.*
- The end state s_1 is not halting, it may be reached before the algorithm completes and cannot be used as a termination indicator.
- As a side effect, any parent of the visited children that is not a descendant of the initiating node may receive superfluous c_i's.

- This algorithm works under both *strong* and *weak* interpretation of *priorities*.

Example 17 We illustrate the algorithm for counting descendants via convergecast using cell IDs, for the hP system shown in Figure 5.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	s_0a	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1 $\bar{c_1}$	s_0p	s_0p	s_0	s_0	s_0	s_0	s_0	s_0
2	$s_1c_2c_3$ $\bar{c_1}$	s_1 $\bar{c_2}$	s_1 $\bar{c_3}$	s_0p	s_0pp	s_0p	s_0	s_0	s_0
3	s_1 $\bar{c_1}\bar{c_2}\bar{c_3}$	$s_1c_4c_5$ $\bar{c_2}$	$s_1c_5c_6$ $\bar{c_3}$	s_1 $\bar{c_4}$	s_1p $\bar{c_5}$	s_1 $\bar{c_6}$	s_0p	s_0p	s_0c_6
4	$s_1c_4c_5c_5c_6$ $\bar{c_1}\bar{c_2}\bar{c_3}$	s_1 $\bar{c_2}\bar{c_4}\bar{c_5}$	s_1 $\bar{c_3}\bar{c_5}\bar{c_6}$	s_1 $\bar{c_4}$	s_1c_8 $\bar{c_5}$	s_1c_7 $\bar{c_6}$	s_1c_8 $\bar{c_7}$	s_1p $\bar{c_8}$	s_0c_6
5	s_1 $\bar{c_1}\bar{c_2}\bar{c_3}\bar{c_4}\bar{c_5}\bar{c_5}\bar{c_6}$	s_1c_8 $\bar{c_2}\bar{c_4}\bar{c_5}$	$s_1c_7c_8$ $\bar{c_3}\bar{c_5}\bar{c_6}$	s_1 $\bar{c_4}$	s_1 $\bar{c_5}\bar{c_8}$	s_1c_8 $\bar{c_6}\bar{c_7}$	s_1 $\bar{c_7}\bar{c_8}$	s_1 $\bar{c_8}$	$s_0c_6c_7$
6	$s_1c_7c_8c_8$ $\bar{c_1}\bar{c_2}\bar{c_3}\bar{c_4}\bar{c_5}\bar{c_6}$	s_1 $\bar{c_2}\bar{c_4}\bar{c_5}\bar{c_8}$	s_1c_8 $\bar{c_3}\bar{c_5}\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_4}$	s_1 $\bar{c_5}\bar{c_8}$	s_1 $\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_7}\bar{c_8}$	s_1 $\bar{c_8}$	$s_0c_6c_7c_8$
7	s_1 $\bar{c_1}\bar{c_2}\bar{c_3}\bar{c_4}\bar{c_5}\bar{c_6}\bar{c_7}\bar{c_8}\bar{c_8}$	s_1 $\bar{c_2}\bar{c_4}\bar{c_5}\bar{c_8}$	s_1 $\bar{c_3}\bar{c_5}\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_4}$	s_1 $\bar{c_5}\bar{c_8}$	s_1 $\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_7}\bar{c_8}$	s_1 $\bar{c_8}$	$s_0c_6c_7c_8$
8	s_1 $\bar{c_1}\bar{c_2}\bar{c_3}\bar{c_4}\bar{c_5}\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_2}\bar{c_4}\bar{c_5}\bar{c_8}$	s_1 $\bar{c_3}\bar{c_5}\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_4}$	s_1 $\bar{c_5}\bar{c_8}$	s_1 $\bar{c_6}\bar{c_7}\bar{c_8}$	s_1 $\bar{c_7}\bar{c_8}$	s_1 $\bar{c_8}$	$s_0c_6c_7c_8$

Algorithm 9: Shortest paths from a given cell.

Precondition: Cells do not need any inbuilt knowledge about the network topology. For each cells with indices i, j, $1 \leq i, j \leq m$, the alphabet includes special ID objects: p_i, $\bar{p_i}$, $\bar{c_i}$, x_{ij}. All cells start in state s_0 and have the same rules, except several similar but custom specific rules to process the IDs. The initiating cell has an additional object a, not present in any other cell.

Postcondition: This algorithm builds a *shortest paths* spanning tree, that is a breadth-first tree rooted at the initiating cell and preserving this dag's relation δ. Each visited cell σ_i, except the initiating cell, will contain one $\bar{p_k}$, indicating its parent σ_k in the spanning tree. Each visited cell σ_i will also contain one $\bar{c_j}$ for each σ_j that is a child of σ_i in the spanning tree, i.e., it will contain all elements of the set $\{\bar{c_j} \mid (i, j) \in \delta, \sigma_j \text{ contains } \bar{p_i}\}$.

Rules:

0. For state s_0 and cell σ_i (custom rules, specific for cell σ_i):

   1) $s_0a \to s_1p_{i\downarrow}$, with $\alpha = min$, $\beta = repl$.
   2) $s_0p_j \to s_1\bar{p_j}p_{i\downarrow}x_{ji\uparrow}$, for $1 \leq j \leq m$, with $\alpha = min$, $\beta = repl$.

3) $s_0 x_{kj} \to s_0$, for $1 \le k, j \le m, k \neq i$, with $\alpha = max$.

1. For state s_1 and cell σ_i (custom rules, specific for cell σ_i):

   1) $s_1 x_{ij} \to s_1 \bar{c_j}$, for $1 \le j \le m$, with $\alpha = max$.
   2) $s_1 p_j \to s_1$, for $1 \le j \le m$, with $\alpha = max$.
   3) $s_1 x_{kj} \to s_1$, for $1 \le k, j \le m, k \neq i$, with $\alpha = max$.

Proof. It is clear that every visited cell σ_i, except the initiating cell, contains one $\bar{p_k}$ where $k \in \delta^{-1}(i)$ from rule 0.2. By a node's height, we prove that a cell σ_i will contain the set $C_i = \{\bar{c_j} \mid (i,j) \in \delta, \sigma_j \text{ contains } \bar{p_i}\}$. For height 0, $C_i = \emptyset$ is true since a sink σ_i does not have any children to receive an x_{ji}—see rule 0.2. For a cell σ_i of height greater than 0, first observe that rule 1.1 is only applied if rule 0.2 has been applied for a child cell σ_j. Thus, C_i contains all $\bar{c_j}$ such that (i,j) is in the spanning tree. Those x_{kj}'s are removed by rule 0.3, and x_{ij}'s that are not converted to $\bar{c_j}$ are removed by rule 1.3. $\square$

Remarks 18

- For this algorithm, cells need additional symbols, see the precondition.
- This algorithm takes $h+1$ P-steps, where h is the *height* of the initiating cell.
- The end state s_1 is not halting, it may be reached before the algorithm completes and cannot be used as a termination indicator.
- As a side effect, any parent of the visited children that is not a descendant of the initiating node will receive superfluous x_{ij}'s, but they are removed by rule 0.3.
- The rules for state s_0 make effective use of our rewriting mode refinement: rules 0.1 and 0.2 use $\alpha = min$, while rule 0.3 uses $\alpha = max$.
- Provided that arcs are associated with weights, this algorithm can be extended into a distributed version of the *Bellman-Ford algorithm* [Lyn96].

Example 19 We illustrate the algorithm for counting nodes in a single-source dag via convergecast, for the hP system shown in Figure 5. The thick arrows in Figure 6 show the resulting spanning tree.

Step\Cell	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
0	$s_0 a$	s_0	s_0	s_0	s_0	s_0	s_0	s_0	s_0
1	s_1	$s_0 p_1$	$s_0 p_1$	s_0	s_0	s_0	s_0	s_0	s_0
2	$s_1 x_{12} x_{13}$	$s_1 \bar{p_1}$	$s_1 \bar{p_1}$	$s_0 p_2$	$s_0 p_2 p_3$	$s_0 p_3$	s_0	s_0	s_0
3	$s_1 \bar{c_2} \bar{c_3}$	$s_1 \bar{p_1} x_{24} x_{25}$	$s_1 \bar{p_1} x_{25} x_{36}$	$s_1 \bar{p_2}$	$s_1 p_3 \bar{p_2}$	$s_1 \bar{p_3}$	$s_0 p_6$	$s_0 p_5$	$s_0 x_{36}$
4	$s_1 \bar{c_2} \bar{c_3}$	$s_1 \bar{p_1} \bar{c_4} \bar{c_5}$	$s_1 \bar{p_1} \bar{c_6}$	$s_1 \bar{p_2}$	$s_1 \bar{p_2} x_{58}$	$s_1 \bar{p_3} x_{67}$	$s_1 \bar{p_6} x_{58}$	$s_1 p_7 \bar{p_5}$	s_0
5	$s_1 \bar{c_2} \bar{c_3}$	$s_1 \bar{p_1} \bar{c_4} \bar{c_5}$	$s_1 \bar{p_1} \bar{c_6}$	$s_1 \bar{p_2}$	$s_1 \bar{p_2} \bar{c_8}$	$s_1 \bar{p_3} \bar{c_7}$	$s_1 \bar{p_6}$	$s_1 \bar{p_5}$	s_0

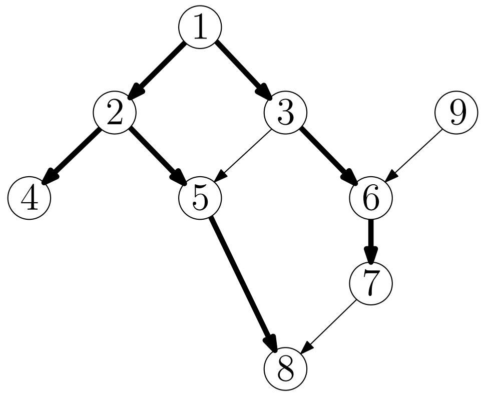


Figure 6: A spanning tree created by the shortest paths algorithm (Algorithm 9).

5 The Firing Squad Synchronization Problem

The *Firing Squad Synchronization Problem* (FSSP) [KG05, Maz87, Nog04, SW04, Szw82, UMF02] 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 [SW04, Szw82], have been proposed and studied. Studies of these variations mainly focus on finding a solution with as few states as possible and possibly running in optimum time.

The synchronization problem has recently been studied in the framework of P systems. A P systems version of the FSSP is described below.

We are given a P, hP, snP or nP system with n cells, $\{\sigma_1, \ldots, \sigma_n\}$, where all cells have the *same states set* and *same rules set.* Two states are distinguished: an *initial* state s_0 and a *firing* state s_ϕ. We select an arbitrary *commander* cell σ_c and an arbitrary subset of *squad* cells, $F \subseteq \{\sigma_1, \ldots, \sigma_n\}$ (possibly the whole set), that we wish to synchronize; the commander itself may or may not be part of the firing squad. At startup, all cells start in the initial state s_0; the commander and the squad cells may contain specific objects, but all other cells are empty. Initially, all cells, except the commander, are idle, and will remain idle until they receive a message. The commander sends one or more orders, to one or more of its neighbors, to start and control the synchronization process. Idle cells may become active upon receiving a first message. Notifications may be relayed to all cells, as necessary. Eventually, all cells in the squad set F will enter the designated firing state s_ϕ, *simultaneously* and for the *first time.* At that time, all the other cells have reached a different state, typically s_0 or s_1, without ever passing through the firing state s_ϕ. Optionally, at that time, all cells should be empty.

Using tree-based P systems, Bernardini *et al* [BGMV08] provided a non-deterministic 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* [AMV08] 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*, algorithmic techniques with a high potential for parallelism.

In this paper, we continue the study of FSSP in the framework of P systems, by providing deterministic solutions for dag-based hP and graph-based snP systems. Our deterministic solutions are a variant of FSSP [Szw82], in which there is a single commander, at an arbitrary position. We further generalize this problem by synchronizing a subset of cells of the considered hP or snP system.

These more complex structures pose additional challenges, not considered in the previous FSSP papers on tree-based P systems, such as multiple network sources (no single root) and multiple paths between cells. Additionally, by allowing an arbitrary position for the commander, we cannot anymore take advantage of the sense of direction between adjacent cells; practically, our structures need to be treated as undirected graphs.

Our first solution uses simple rules, but requires *dynamical structures.* In this paper we propose a *novel extension*, which supports the creation of dynamical structures, by allowing *mobile channels.* This solution works for hP systems and snP systems; it will also work for tree-based P systems, but only if we reconsider them as dag-based P systems, because the resulting structures will be dags, not trees. This solution takes $e_c + 5$ steps, where e_c is the eccentricity of the commander cell of the underlying dag or graph. The relative simplicity and the speed of this solution supports our hypothesis that basing P systems on dag, instead of tree, structures allows more natural expressions of some fundamental distributed algorithms [NDK08, NDK09a].

Our second solution is more traditional and does not require dynamical structures, but is substantially more complex, combining a *breadth-first-search*, a *broadcast* and a *convergecast.* This solution works for tree-based P systems, hP systems and snP systems and takes $6e_c + 7$ steps. When restricted to P systems, our algorithm takes more steps than Alhazov *et al* [AMV08], if the commander is the root node, but comparable to this, when the commander is a central node of an unbalanced rooted tree.

Our two solutions do not require *polarities* or *conditional* rules, but require *priorities* and *states.* Both hP systems and snP systems already have states, by definition. However, it seems that traditional tree-based P systems have not used states so far, or not much.

5.1 FSSP—Dynamic Structures via Mobile Channels

In this section, we further refine our solution given in an earlier paper [NDK09a]. A natural solution is possible when we are allowed to extend the cell structure of the given hP or snP system. We achieve this by supporting the dynamic creation and manipulation of *mobile* channels. The endpoints of our mobile channels appear in the rules like all other objects and are subject to usual rewriting and transfer rules. Our mobile channels move together with their endpoints and are thus able to "grow" step-by-step, not unlike nerves which extend in a regenerating tissue, or threads extended by spiders.

As endpoints for mobile channels, we use the objects from the set $\{\alpha, \theta, \omega\}$:

- The object α (here we use only *one* α) marks the containing cell as the starting endpoint of all mobile channels.
- Each object θ or ω marks the final endpoint of a mobile channel.
- In the graphical representation, a mobile channel will be represented by a dotted arrow, spanning from the cell containing α to the cell containing θ or ω.

Example 20 Although not strictly needed, we can imagine that Figures 7(a), (b), (c) correspond to successive hP system configurations (resulting from rewriting and transfer rules).

- Figure 7(a) shows one mobile channel spanning from σ_1 to σ_3.
- Figure 7(b) shows two mobile channels, spanning from σ_1 to σ_3 and σ_4, respectively.
- Figure 7(c) shows two mobile channels, spanning from σ_1 to σ_3 and σ_5, respectively.

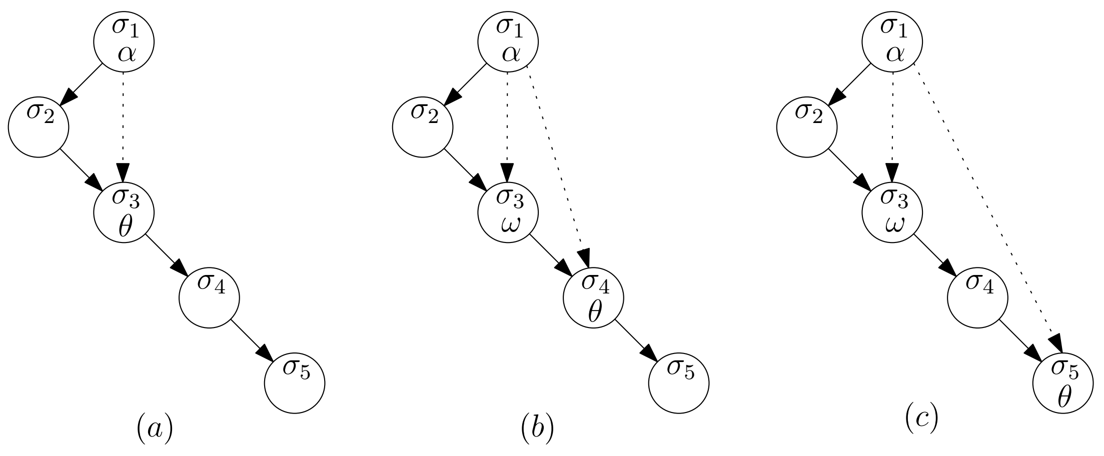


Figure 7: Mobile channels, indicated by dotted arrows.

In our algorithm, we further assume that the dag was already extended by an external cell, which will be called the *sergeant*. We leave out the details of this extension, but we note that the existing P system framework provides several facilities, which can achieve this, e.g., cell division [Pău06].

Next, this sergeant will send a self-replicating θ endpoint, that will be repeatedly broadcasted, until all cells are reached. A θ endpoint will leave an ω endpoint in a squad cell and will disappear without trace from the other cells. In the end, the structure will be extended with new channels which will link the sergeant with all squad cells. Finally, when there are no more structural changes, the sergeant, will send a firing command to all squad cells, prompting these cells to enter the firing state, all at the same time.

Algorithm 10: FSSP—Dynamic structures via mobile channels.

Precondition: An hP or snP system, with n cells $\sigma_1, \ldots, \sigma_n$, a commander cell σ_c and a set of squad cells F to be synchronized. Additionally, we already have a sergeant cell, σ_{n+1}, linked to the commander by one arc, (σ_{n+1}, σ_c), for hP systems, or by two arcs, (σ_{n+1}, σ_c), (σ_c, σ_{n+1}), for snP systems.

All cells start in state s_0 and have the same rules. The state s_ϕ is the firing state. Initially, the sergeant σ_{n+1} is marked by one object α, and each squad cell is marked by one object f (this can include the commander σ_c or the sergeant σ_{n+1}, or both); all other cells have no objects.

Postcondition: All cells in the set F enter state s_ϕ, simultaneously and for the first time, after $e_c + 5$ steps, where e_c is the commander's eccentricity in the underlying graph. All other cells enter state s_1, without ever passing through state s_ϕ.

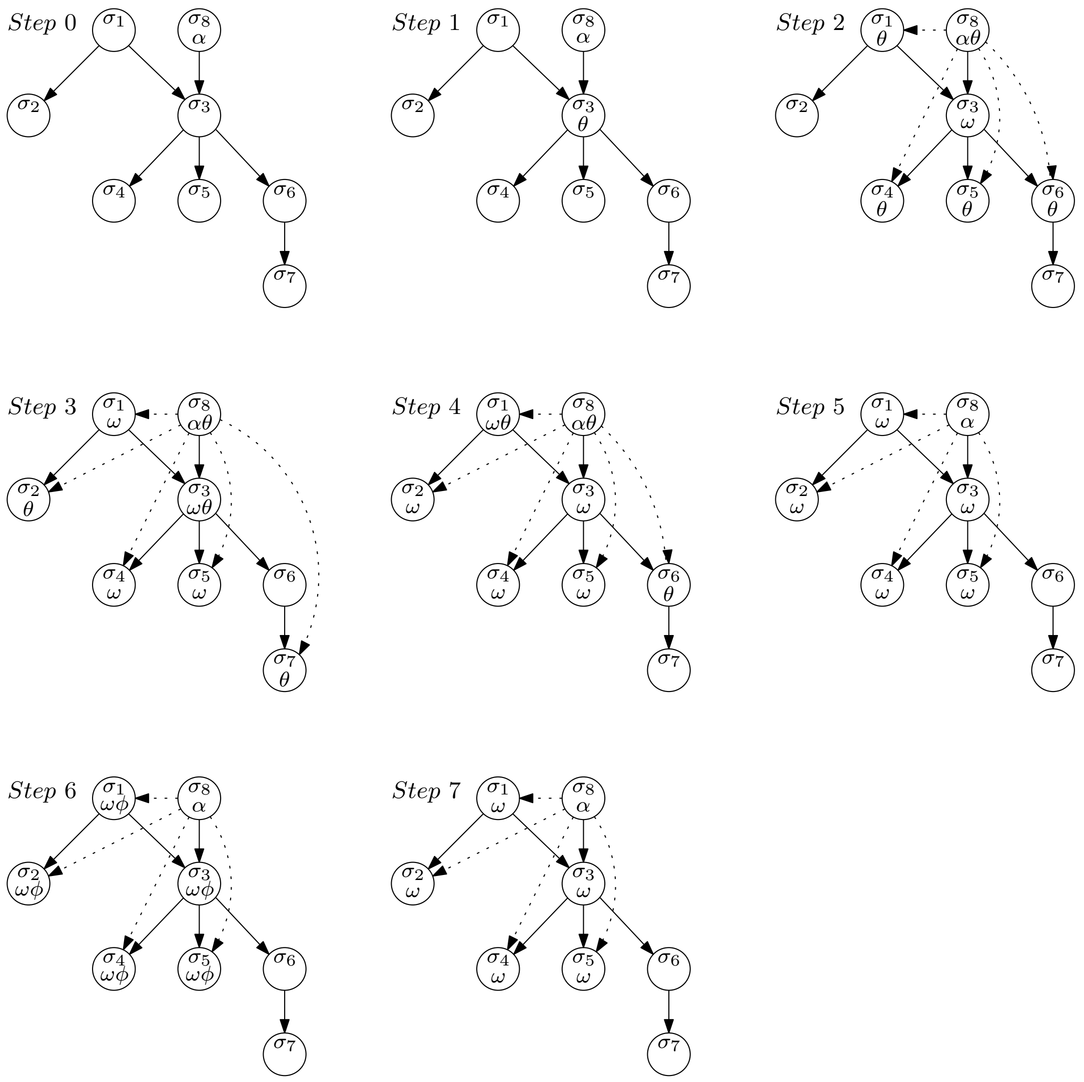
Rules: (rules are applied under the weak interpretation of priorities, in the rewriting mode $\alpha = max$ and transfer mode $\beta = repl$):

1. $s_0\alpha \to s_2\alpha\theta_{go}$
2. $s_0f\theta \to s_4\omega\theta_{go}$
3. $s_0\theta \to s_1\theta_{go}$
4. $s_1\theta \to s_1$
5. $s_2\alpha \to s_3\alpha$
6. $s_3\theta \to s_3$
7. $s_3f\alpha \to s_4f\alpha\phi\phi_{go}$
8. $s_3\alpha \to s_1\alpha\phi_{go}$
9. $s_4f\phi \to s_\phi$
10. $s_4\theta \to s_4$

Example 21 Figure 8 and Table 1 illustrate this algorithm for an hP system based on the dag of Figure 1. Here, the commander cell is σ_3, the squad set is $F = \{\sigma_1, \ldots, \sigma_5\}$ and this system's structure has already been extended by the sergeant cell σ_8 and the arc (σ_8, σ_3). The mobile channels are represented by dotted arrows.

Table 1: Traces for Algorithm 10 on an hP system based on the dag of Figure 1.

Step\Cell	σ_8	σ_3	σ_1	σ_4	σ_5	σ_6	σ_2	σ_7
0	$s_0\alpha$	s_0f	s_0f	s_0f	s_0f	s_0	s_0f	s_0
1	$s_2\alpha$	$s_0f\theta$	s_0f	s_0f	s_0f	s_0	s_0f	s_0
2	$s_3\alpha\theta$	$s_4f\omega$	$s_0f\theta$	$s_0f\theta$	$s_0f\theta$	$s_0\theta$	s_0f	s_0
3	$s_3\alpha\theta^4$	$s_4f\omega\theta^4$	$s_4f\omega$	$s_4f\omega$	$s_4f\omega$	s_1	$s_0f\theta$	$s_0\theta$
4	$s_3\alpha\theta^2$	$s_4f\omega$	$s_4f\omega\theta$	$s_4f\omega$	$s_4f\omega$	$s_1\theta$	$s_4f\omega$	s_1
5	$s_3\alpha$	$s_4f\omega$	$s_4f\omega$	$s_4f\omega$	$s_4f\omega$	s_1	$s_4f\omega$	s_1
6	$s_1\alpha$	$s_4f\omega\phi$	$s_4f\omega\phi$	$s_4f\omega\phi$	$s_4f\omega\phi$	s_1	$s_4f\omega\phi$	s_1
7	$s_1\alpha$	$s_\phi\omega$	$s_\phi\omega$	$s_\phi\omega$	$s_\phi\omega$	s_1	$s_\phi\omega$	s_1



Step 0
σ_1 σ_8 α σ_2 σ_3 σ_4 σ_5 σ_6 σ_7
Step 1
σ_1 σ_8 α σ_2 σ_3 θ σ_4 σ_5 σ_6 σ_7
Step 2
σ_1 θ σ_8 $\alpha\theta$ σ_2 σ_3 ω σ_4 θ σ_5 θ σ_6 θ σ_7
Step 3
σ_1 ω σ_8 $\alpha\theta$ σ_2 θ σ_3 $\omega\theta$ σ_4 ω σ_5 ω σ_6 σ_7 θ
Step 4
σ_1 $\omega\theta$ σ_8 $\alpha\theta$ σ_2 ω σ_3 ω σ_4 ω σ_5 ω σ_6 θ σ_7
Step 5
σ_1 ω σ_8 α σ_2 ω σ_3 ω σ_4 ω σ_5 ω σ_6 σ_7
Step 6
σ_1 $\omega\phi$ σ_8 α σ_2 $\omega\phi$ σ_3 $\omega\phi$ σ_4 $\omega\phi$ σ_5 $\omega\phi$ σ_6 σ_7
Step 7
σ_1 ω σ_8 α σ_2 ω σ_3 ω σ_4 ω σ_5 ω σ_6 σ_7

Figure 8: Running Algorithm 10 on an hP system based on the dag of Figure 1.

5.2 FSSP—Static Structures and Rules

Here we consider a second scenario, where we are allowed to modify the rules of the given hP or snP system, but not its original structure. A brief description of this solution is as follows. The commander intends to send an order to all cells in the set F, which will prompt them to synchronize by entering the designated firing state. However, in general, the commander does not have direct channels with all the cells. In this case, the process of sending a command to the destination cell will cause delays (some steps), as the command is relayed through intermediate cells. Hence, to ensure all firing squad cells enter the firing state simultaneously, each firing squad cell determines the number of steps it needs to wait before entering the firing state.

As in our earlier paper [NDK09a], cells have no built-in knowledge of the network topology. Additionally, cells are anonymous, i.e., not identified by cell IDs, and not implicitly named by membrane polarization techniques. The cells are initially empty, except the commander, which is initially marked by one a, and the squad cells, which are initially marked by one f each. All cells start with the same set of rules, applied under the *weak* interpretation of priorities, in the rewriting mode $\alpha = max$ and the transfer mode $\beta = repl$.

Each cell *independently* progresses through *four phases*, called FSSP-I, FSSP-II, FSSP-III and FSSP-IV, which are detailed in Algorithms 11, 12, 13 and 14, respectively. An overview of these four phases is as follows:

- Phase FSSP-I is a *broadcast* from the commander, that follows the virtual dag defined by $level_c$. This phase starts in state s_0 and ends in state s_2. Also, the commander starts a counter, which, at the end of Phase FSSP-II, will determine its eccentricity.

- Phase FSSP-II is a subsequent *convergecast* from terminal cells, that follows the same virtual dag. This phase starts in state s_2 and ends when the commander enters state s_6. At the end of this phase, the commander's counter determines its eccentricity.

- Phase FSSP-III is a second *broadcast*, initiated from the commander, that follows the same virtual dag. This phase starts in state s_6 and ends in state s_8. The commander sends out its eccentricity, which is successively decremented at each level.

- Phase FSSP-IV is a *timing* (countdown) for entering the firing state. This phase starts in state s_8 and continues with a countdown, until squad cells simultaneously enter the firing state s_9, and all other cells enter state s_0.

The statechart in Figure 9 illustrates the combined flow of these four phases. The nodes represent the states of the hP or snP system and the arcs are labelled with numbers of the rules that match the corresponding transitions. The rest of this section describes these four phases, proving their correctness and time complexities. A sample run of our algorithm will follow at the end of this section, in Example 27.

FSSP: The initial configuration

- $\Gamma = \{\sigma_1, \dots, \sigma_n\}$, $n > 1$, is the set of all cells, σ_c is the commander, and the firing squad is $F \subseteq \Gamma$;

- $O = \{a, b, c, d, e, f, g, h, k, l, p, q\}$;

- $Q_i = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$, for $i \in \{1, \dots, n\}$, which is "allocated" to four phases as follows: FSSP-I contains rules for states $\{s_0, s_1\}$; FSSP-II contains rules for states $\{s_2, s_3, s_4, s_5, s_6\}$; FSSP-III contains rules for states $\{s_6, s_7\}$; FSSP-IV contains rules for states $\{s_8, s_9\}$;

- $s_\phi = s_9$ is the firing state;

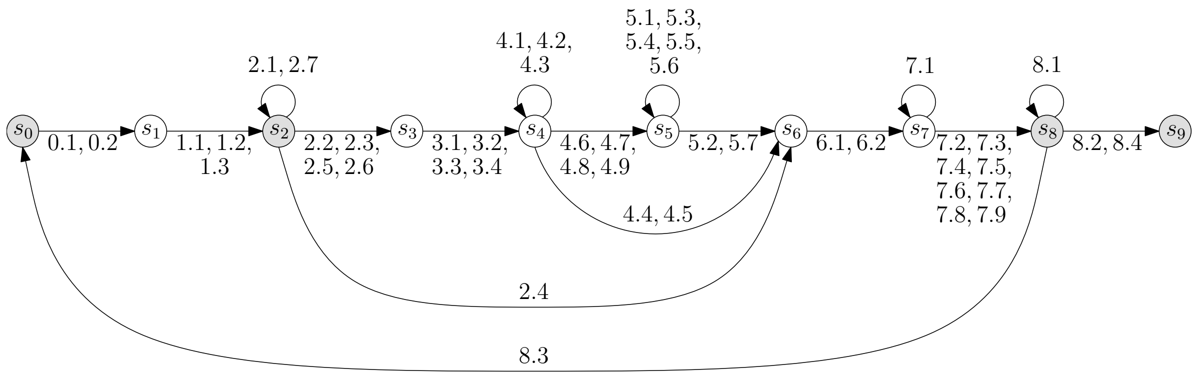


Figure 9: Statechart view of the combined FSSP algorithm phases.

- $s_{i,0} = s_0$, for $i \in \{1, \ldots, n\}$;
- $w_{c,0} = \{a\}$, if $\sigma_c \notin F$, or $\{a, f\}$, otherwise; $w_{i,0} = \{f\}$ for all $\sigma_i \in F \setminus \sigma_c$; $w_{i,0} = \emptyset$, for all $\sigma_i \in \Gamma \setminus (F \cup \{\sigma_c\})$;
- The following rules are applied under the weak interpretation of priorities, in the rewriting mode $\alpha = max$ and transfer mode $\beta = repl$:

0. For state s_0:
   1) $s_0 a \to s_1 a e d_{go}$
   2) $s_0 d \to s_1 a d_{go}$
1. For state s_1:
   1) $s_1 a e \to s_2 a e e k$
   2) $s_1 a \to s_2 a k$
   3) $s_1 d \to s_2 l$
2. For state s_2:
   1) $s_2 k \to s_2$
   2) $s_2 a e \to s_3 a e e$
   3) $s_2 d \to s_3 d$
   4) $s_2 a \to s_6 a c_{go}$
   5) $s_2 l \to s_3 l g_{go}$
   6) $s_2 g \to s_3$
   7) $s_2 a e \to s_2 a e e$
3. For state s_3:
   1) $s_3 a e \to s_4 a e e$
   2) $s_3 a \to s_4 a$
   3) $s_3 g \to s_4 p$
   4) $s_3 c \to s_4$

4. For state s_4:
   1) $s_4cd \to s_4$
   2) $s_4ade \to s_4adee$
   3) $s_4d \to s_4d$
   4) $s_4aeeeee \to s_6aeee$
   5) $s_4eeeee \to s_6e$
   6) $s_4a \to s_5ak$
   7) $s_4l \to s_5lh_{go}$
   8) $s_4h \to s_5$
   9) $s_4q \to s_5$
   10) $s_4c \to s_6$
   11) $s_4g \to s_6$
   12) $s_4h \to s_6$
   13) $s_4q \to s_6$

5. For state s_5:
   1) $s_5k \to s_5$
   2) $s_5a \to s_6ac_{go}$
   3) $s_5hp \to s_5p$
   4) $s_5pq \to s_5$
   5) $s_5p \to s_5kp$
   6) $s_5l \to s_5lh_{go}$
   7) $s_5l \to s_6q_{go}$

6. For state s_6:
   1) $s_6ae \to s_7ak$
   2) $s_6e \to s_7be_{go}$
   3) $s_6c \to s_6$
   4) $s_6g \to s_6$
   5) $s_6h \to s_6$
   6) $s_6p \to s_6$
   7) $s_6q \to s_6$

7. For state s_7:
   1) $s_7k \to s_7$
   2) $s_7a \to s_8a$
   3) $s_7e \to s_8$

8. For state s_8:
   1) $s_8ab \to s_8a$
   2) $s_8af \to s_9$
   3) $s_8a \to s_0$
   4) $s_8a \to s_9$

Algorithm 11 (FSSP-I: First broadcast from the commander)

Precondition: The initial configuration as specified earlier.
Postcondition:

- The end state is s_2.
- A cell σ_i has
  - $count_c(i)$ copies of a and $count_c(i)$ copies of k;
  - u copies of l, where u is the total number of a's in σ_i's *peers*;
  - v copies of d, where v is the total number of a's in σ_i's *successors*;
  - two copies of e, if $\sigma_i = \sigma_c$;

  - one copy of f, if $\sigma_i \in F$.

Proof. This phase of the algorithm is a *broadcast* that follows the virtual dag created by the *levels* determined by Algorithm 1.

Consider a cell σ_i. By induction:

- At step $level_c(i)$, σ_i (except the commander) receives a total of $count_c(i)$ copies of d from its *predecessors*.
- At step $level_c(i)+1$, σ_i broadcasts $count_c(i)$ copies of d to each of its *neighbors* and transits to state s_1. At the same time, σ_i accumulates one local copy of a for each sent d, for a total count of $count_c(i)$ of a's. Also, σ_i receives u copies of d, similarly sent by its *peers*, where u is equal to the total number of a's similarly accumulated, at the same time step, by σ_i's peers.
- At step $level_c(i) + 2$, σ_i receives v copies of d, sent back by its *successors*; and transits to state s_2, where v is equal to the total number of a's created, at the same time step, by σ_i's successors;

The commander, by initially having one a, creates two copies of e. Finally, the rules associated with this phase do not change the number of f's, thus, each cell in the firing squad still ends with one f. □

Corollary 22 (FSSP-I: Number of steps) *For each cell σ_i, the phase FSSP-I takes $level_c(i) + 2$ steps.*

Proof. As indicated in the proof of the Algorithm 11, the total number of steps is $level_c(i) + 2$. □

Algorithm 12 (FSSP-II: Convergecasts from terminal nodes)

Precondition: As described in the postcondition of Algorithm 11.
Postcondition:

- This phase ends when the commander enters state s_6.
- A cell 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 $\sigma_i \in F$.

Proof. Briefly, this phase of the algorithm is a *convergecast* of c's, starting from terminal cells, and further relayed up, on the virtual dag, until the commander is reached.

For the purpose of this phase, the non-commander cells can be organized in the following three groups: TC cells = terminal cells; NTC-NTP cells = non-terminal cells without non-terminal peers (i.e., cells without peers or cells with terminal peers only); NTC+NTP cells = non-terminal cells with non-terminal peers (these cells may also have terminal peers).

During this phase, these cells will make transitions between the following three conceptual stages: WCS = waiting for convergecasts from successors (state s_4); RTC = ready to convergecast (state s_5); HC = have convergecasted (state s_6). Specifically, the following transitions will be made: the TC cells will transit immediately from the WCS stage to the HC stage; the NTC-NTP cells will linger in the WCS stage until they receive convergecasts from all their successors, after which they will transit directly to the HC stage; the NTC+NTP cells will linger in the WCS stage until they receive convergecasts from all their successors, subsequently they will linger in the RTC stage until all their non-terminal peers reach the RTC stage as well, after which they will transit to the HC stage.

During this process, cells will exchange c-notifications, which are messages consisting of number of c's and h-notifications, which are messages consisting of number of h's. The actual numbers depend on network topology and take into account the multiple paths that appear in the virtual dag.

The c-notification broadcasted by cell σ_i consists of $count_c(i)$ copies of c and is only sent once when σ_i transits into the HC stage.

The h-notification broadcasted by cell σ_i consists of u copies of h, where u is the number defined in the precondition. This notification is sent repeatedly, while σ_i remains in the RTC stage, until σ_i transits into the HC stage. The h-notifications synchronize the non-terminal peers that cannot transit to the HC stage until all of them have reached the RTC stage. This avoids the potential confusion that could otherwise arise when a non-terminal cell receives an "ambiguous" c-notification, i.e., a c-notification that could come both from a successor or from a non-terminal peer.

Without loss of generality, we illustrate our solution on the dag from Figure 10. This figure shows a typical sub-dag of the virtual dag created by Algorithm 1, where cells σ_1, σ_2, σ_3, σ_4 and σ_5 are at the same level, and the horizontal lines indicate peer relations.

Cell σ_1 is a TC cell and will transit from stage WCS to stage HC, immediately after detecting that it has no successors (there is no need for any synchronization with its peer σ_2).

Cell σ_5 is a NTC-NTP cell and will transit from stage WCS to stage HC, after receiving c-notifications from all its successors.

Cells σ_2, σ_3 and σ_4 are NTC+NTP cells. Each of these cells will linger in stage WCS until it receives c-notifications from all its successors, when it will enter stage RTC. Cells σ_2 and σ_3 are peers, therefore none of them will be allowed to transit to stage HC, until

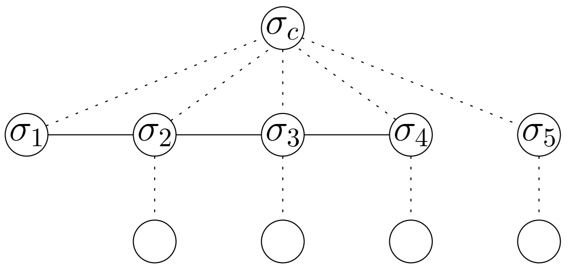


Figure 10: A typical sub-dag of the virtual dag.

both of them have reached the RTC stage. Similarly, cells σ_3 and σ_4 are peers, therefore none of them will be allowed to transit to stage HC, until both of them have reached the RTC stage. Assume that cells σ_2, σ_3 and σ_4 will reach stage RTC in this order. Then, cell σ_2 will wait in stage RTC until σ_3 also reaches the same stage. When this eventuates, σ_2 will transit to stage HC, while σ_3 will still linger in stage RTC until σ_4 reaches the same stage. When this eventuates, both σ_3 and σ_4 will transit at the same time to stage HC.

A TC cell σ_i enters this phase $level_c(i)$ steps after the commander, idles one step in state s_2, then starts its role in the convergecast, by broadcasting $count_c(i)$ copies of c to its predecessors and peers (it does not have successors) and transits to state s_6. This cell further idles in state s_6 until it receives e's from its predecessors. The convergecast takes four steps at each level.

The total run-time is dominated by e_c, the length of the longest level-preserving path from commander. Therefore, the convergecast wave will complete at commander after $e_c + 4e_c - 2 = 5e_c - 2$ steps after the commander starts this phase. When the commander receives the convergecast from all its successors, it takes two steps to transit to state s_6. Therefore, the commander enters state s_6, $5e_c$ steps after it starts this phase. □

Corollary 23 (FSSP-II: Number of steps) *For each cell σ_i, the phase FSSP-II takes $5e_c - level_c(i)$ steps.*

Proof. As indicated in the proof of Algorithm 12, this phase takes $5e_c$ steps. □

Algorithm 13 (FSSP-III: Second broadcast from the commander)

Precondition: As described in the postcondition of Algorithm 12.

Postcondition:

- The end state is s_8.
- A cell σ_i has
  - $count_c(i)$ copies of a;
  - $(e_c + 1 - level_c(i))count_c(i)$ copies of b;
  - one copy of f, if $\sigma_i \in F$.

Proof. In this phase, commander starts its second *broadcast*, by sending $e_c + 1$ copies of e's to all its successors. By induction on level, a cell σ_i receives a total of $(e_c + 2 - level_c(i))count_c(i)$ copies of e's from its predecessors, reduces this count by $count_c(i)$ (i.e., the count of a's), forwards the remaining $(e_c + 1 - level_c(i))count_c(i)$ copies of e's to all its successors and creates for itself $(e_c + 1 - level_c(i))count_c(i)$ copies of b's. A more detailed description will be given in the final version.

All rules of this phase do not change the number of a's or the number of f's; therefore, the corresponding postcondition holds. $\square$

Corollary 24 (FSSP-III: Number of steps) *For each cell σ_i, the phase FSSP-III takes $level_c(i) + 3$ steps.*

Proof. As indicated in the proof of Algorithm 13, this phase takes $level_c(i) + 3$ steps. $\square$

Algorithm 14 (FSSP-IV: Timing for entering the firing state)

Precondition: As described in the postcondition of Algorithm 13.
Postcondition:

- The end state is s_9 for cells in the firing squad, or s_0, otherwise.
- Each cell is empty.

Proof. As long as b's are present, a cell σ_i performs a transition step that decreases the number of b's by $count_c(i)$ (i.e., the number of a's). This step will be repeated $(e_c + 1 - level_c(i))$ times, as given by the initial ratio between the number of b's, $(e_c + 1 - level_c(i))count_c(i)$, and the number of a's, $count_c(i)$. This is the delay every cell needs to wait, before entering either the firing state s_9 or the initial state s_0.

Finally, in the last step, cell σ_i enters s_9, if σ_i has one f, or s_0, otherwise. At the same time, all existing objects are removed. $\square$

Corollary 25 (FSSP-IV: Number of steps) *For each cell σ_i, the phase FSSP-IV takes $e_c + 2 - level_c(i)$ steps.*

Proof. As indicated in the proof of Algorithm 14, this phase takes $(e_c+1-level_c(i))+1 = e_c + 2 - level_c(i)$. □

Theorem 26 *For each cell σ_i, the combined running time of the four phases Algorithm 11, 12, 13 and 14 is $6e_c + 7$, 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 Corollaries 22, 23, 24 and 25: $(level_c(i) + 2) + (5e_c - level_c(i)) + (level_c(i) + 3) + (e_c + 2 - level_c(i)) = 6e_c + 7$. □

Example 27 We present in Table 2 the traces of the FSSP algorithm for an hP system with the dag in Figure 2 as its underlying structure. All cells are ordered according to their *levels* and the starting states of phases FSSP-II, FSSP-III and FSSP-IV are highlighted.

6 Planar representation

We define a *simple region* as the interior of a simple closed curve (Jordan curve). By default, all our regions will be delimited by simple closed curves that are also smooth, with the possible exception of a finite number of points. This additional assumption is not strictly needed, but simplifies our arguments.

A simple region R_j is *directly contained* in a simple region R_i, if $R_j \subset R_i$ and there is no simple region R_k, such that $R_j \subset R_k \subset R_i$ (where $\subset$ denotes strict inclusion).

It is well known that any transition P system has a planar Venn-like representation, with a 1:1 mapping between its tree nodes and a set of hierarchically nested simple regions. Conversely, any single rooted set of hierarchically nested simple regions can be interpreted as a tree, which can further form the structural basis of a number of transition P systems.

We have already shown that this planar representation can be generalized for hP systems based on canonical dags (i.e., without transitive arcs) and arbitrary sets of simple regions (not necessarily nested), while still maintaining a 1:1 mapping between dag nodes and simple regions [NDK09b].

Specifically, any hP system structurally based on a canonical dag can be intensionally represented by a set of simple regions, where direct containment denotes a parent-child

Table 2: The traces of the FSSP algorithm for an hP system with the dag in Figure 2 as its underlying structure, where $c = 6$, $e_6 = 3$, $F = \{\sigma_1, \sigma_4, \sigma_5, \sigma_7, \sigma_9, \sigma_{10}\}$.

	σ_6	σ_2	σ_3	σ_9	σ_1	σ_5	σ_7	σ_8	σ_4	σ_{10}
0	s_0af	s_0	s_0	s_0f	s_0	s_0f	s_0f	s_0	s_0f	s_0f
1	s_1aef	s_0d	s_0d	s_0df	s_0	s_0f	s_0f	s_0	s_0f	s_0f
2	$s_2ad^3e^2fk$	s_1a	s_1a	s_1af	s_0d^2	s_0df	s_0df	s_0d	s_0f	s_0f
3	$s_2ad^3e^3f$	s_2ad^3k	s_2ad^3k	s_2adfk	s_1a^2	s_1af	s_1adf	s_1ad	s_0df	s_0df
4	$s_3ad^3e^4f$	s_2ad^3	s_2ad^3	s_2adf	$s_2a^2k^2$	s_2afk	s_2adfkl	s_2adkl	s_1af	s_1af
5	$s_4ad^3e^5f$	s_3ad^3	s_3ad^3	s_3adf	s_2a^2	s_2af	s_2adfl	s_2adl	s_2afk	s_2afk
6	$s_4ad^3e^6f$	$s_4ac^3d^3$	$s_4ac^2d^3g$	s_4adfg	s_6a^2	s_6af	s_3adfgl	s_3adgl	s_2afg	s_2afg
7	$s_4ad^3e^7f$	s_4a	s_4adg	s_4adfg	s_6a^2	s_6af	$s_4acdflp$	s_4acdlp	s_6afg	s_6afg
8	$s_4ad^3e^8f$	s_5ak	s_4adg	s_4adfg	s_6a^2	s_6af	s_4aflp	s_4alp	s_6af	s_6af
9	$s_4ad^3e^9f$	s_5a	s_4adgh	s_4adfgh	s_6a^2	s_6af	$s_5afhklp$	s_5ahklp	s_6afh	s_6afh
10	$s_4acd^3e^{10}f$	s_6a	s_4adgh^2	s_4adfgh^2	s_6a^2c	s_6acf	s_5afhlp	s_5ahlp	s_6afh	s_6afh
11	$s_4ad^2e^{11}f$	s_6a	s_4acdgh^2q	$s_4acdfgh^2q$	s_6a^2	s_6af	$s_6acfhpq$	s_6achpq	s_6acfq	s_6acfq
12	$s_4ad^2e^{12}f$	s_6a	s_4agh^2q	s_4afgh^2q	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
13	$s_4ad^2e^{13}f$	s_6a	s_5agk	s_5afgk	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
14	$s_4ad^2e^{14}f$	s_6a	s_5ag	s_5afg	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
15	$s_4ac^2d^2e^{15}f$	s_6a	s_6ag	s_6afg	s_6a^2c	s_6af	s_6acf	s_6ac	s_6af	s_6af
16	$s_4ae^{15}f$	s_6a	s_6a	s_6af	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
17	s_6ae^5f	s_6a	s_6a	s_6af	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
18	s_7ab^4fk	s_6ae^4	s_6ae^4	s_6ae^4f	s_6a^2	s_6af	s_6af	s_6a	s_6af	s_6af
19	$s_7ab^4e^9f$	s_7ab^3k	s_7ab^3k	s_7ab^3fk	$s_6a^2e^6$	s_6ae^3f	s_6ae^3f	s_6ae^3	s_6af	s_6af
20	s_8ab^4f	$s_7ab^3e^6$	$s_7ab^3e^6$	$s_7ab^3e^2f$	$s_7a^2b^4k^2$	s_7ab^2fk	$s_7ab^2e^2fk$	$s_7ab^2e^2k$	s_6ae^2f	s_6ae^2f
21	s_8ab^3f	s_8ab^3	s_8ab^3	s_8ab^3f	$s_7a^2b^4$	s_7ab^2f	$s_7ab^2e^3f$	$s_7ab^2e^3$	s_7abfk	s_7abfk
22	s_8ab^2f	s_8ab^2	s_8ab^2	s_8ab^2f	$s_8a^2b^4$	s_8ab^2f	s_8ab^2f	s_8ab^2	s_7abf	s_7abf
23	s_8abf	s_8ab	s_8ab	s_8abf	$s_8a^2b^2$	s_8abf	s_8abf	s_8ab	s_8abf	s_8abf
24	s_8af	s_8a	s_8a	s_8af	s_8a^2	s_8af	s_8af	s_8a	s_8af	s_8af
25	s_9	s_0	s_0	s_9	s_0	s_9	s_9	s_0	s_9	s_9

relation. The converse is also true, any set of simple regions can be interpreted as a canonical dag, which can further form the structural basis of a number of hP systems.

We will now provide several solutions to our open question [NDK09b]: How to represent the other dags, that do contain transitive arcs? First, we discuss a negative result. First, a counter-example that appeals to the intuition, and then a theorem with a brief proof.

Example 28 Consider the dag (a) of Figure 11, where nodes $1, 2, 3$ are to be represented by simple regions R_1, R_2, R_3, respectively. We consider the following three candidate representations: (e), (f) and (g). However, none of them properly match the dag (a), they only match dags obtained from (a) by removing one of its arcs:

(e) represents the dag (b), obtained from (a) by removing the arc $(1, 3)$;

(f) represents the dag (c), obtained from (a) by removing the arc $(1, 2)$;

(g) represents the dag (d), obtained from (a) by removing the arc $(2, 3)$.

Theorem 29 *Dags with transitive arcs cannot be planarly represented by simple regions, with a 1:1 mapping between nodes and regions.*

Proof. Consider again the counter-example in Example 28. The existence of arcs $(2, 3), (1, 2)$ requires that $R_3 \subset R_2 \subset R_1$. This means that R_3 cannot be directly contained in R_1, as required by the arc $(1, 3)$. □

It is clear, in view of this negative result, that we must somehow relax the requirements, if we want to obtain meaningful representations for general hP systems, based on dag structure that may contain transitive arcs. We consider in turn five tentative solutions.

6.1 Solution I: Self-intersecting curves

We drop the requirement of mapping nodes to simple regions delimited by simple closed curves. We now allow self-intersecting closed curves with inward folds. A node can be represented as the union of *subregions*: first, a base simple region, and, next, zero, one or more other simple regions, which are delimited by inward folds of base region's contour (therefore included in the base region). For this solution, we say that there is an arc (i, j) in the dag if and only if a subregion of R_i directly contains region R_j, where regions R_i, R_j represent nodes i, j in the dag, respectively.

Example 30 The region R_1 in Figure 12 is delimited by a self-intersecting closed curve with an inward fold that defines the inner R_1'' subregion. Note the following relations:

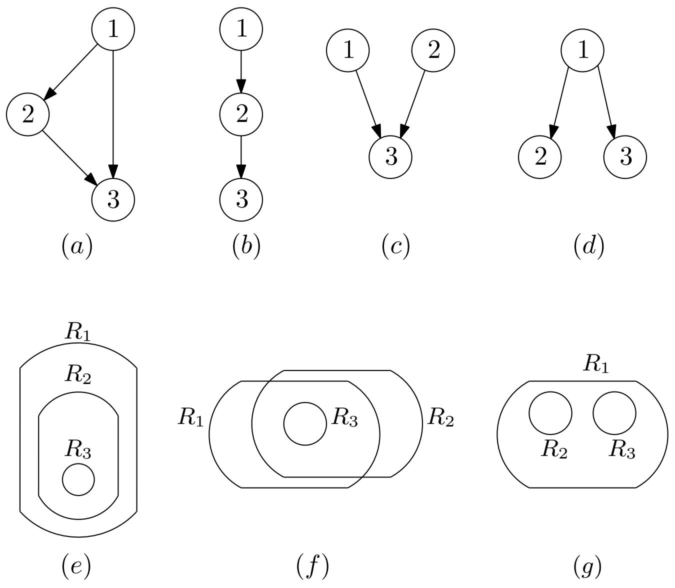


Figure 11: A counter-example for planar representation of non-canonical dags.

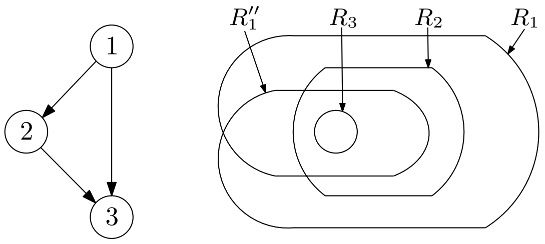


Figure 12: Solution I: R_1 is delimited by a self-intersecting closed curve.

- $R_1 = R_1 \cup R_1''$, thus R_1'' is a subregion of R_1;
- R_1 directly contains R_2, which indicates the arc $(1, 2)$;
- R_2 directly contains R_3, which indicates the arc $(2, 3)$;
- R_1'' directly contains R_3, which indicates the transitive arc $(1, 3)$, because R_1'' is a subregion of R_1.

Remark 31 It is difficult to visualize a cell that is modelled by a self-intersecting curve. Therefore, this approach does not seem adequate.

6.2 Solution II: Distinct regions

We drop the requirement of a 1:1 mapping between dag nodes and regions. Specifically, we accept that a node may be represented by the union of one or more distinct simple

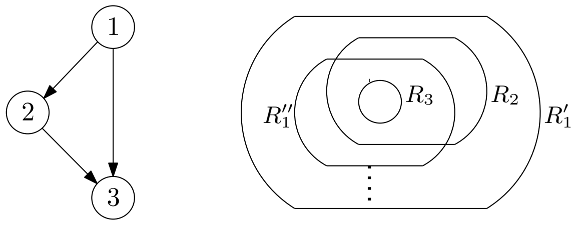


Figure 13: Solution II: R_1 is the union of two simple regions, R_1' and R_1''.

regions, here called *subregions*. Again, as in Solution I, an arc (i, j) is in the dag if and only if a subregion of R_i directly contains region R_j.

Example 32 In Figure 13, the simple region R_1 is the union of two simple regions, R_1' and R_1'', connected by a dotted line. Note the following relations:

- $R_1 = R_1' \cup R_1''$, thus R_1' and R_1'' are subregions of R_1;
- R_1' directly contains R_2, which indicates the arc $(1, 2)$, because R_1' is a subregion of R_1;
- R_2 directly contains R_3, which indicates the arc $(2, 3)$;
- R_1'' directly contains R_3, which indicates the transitive arc $(1, 3)$, because R_1'' is a subregion of R_1.

Remark 33 In Example 32, a dotted line connects two regions belonging to the same node. It is difficult to see the significance of such dotted lines in the world of cells. Widening these dotted lines could create self-intersecting curves—a solution which we have already rejected. Two distinct simple regions should represent two distinct cells, not just one. Therefore, this approach does not seem adequate either.

6.3 Solution III: Flaps

We again require simple regions, but we imagine that our representation is an infinitesimally thin "sandwich" of several superimposed layers, up to one distinct layer for each node (see Figure 14b). Initially, each region is a simple region that is conceptually partitioned into a *base subregion* (at some bottom layer) and zero, one or more other *flap subregions*, that appear as flaps attached to the base. These flaps are then folded, in the three-dimensional space, to other "sandwich" layers (see Figure 14c). The idea is that orthogonal projections of the regions corresponding to destinations of transitive arcs, which cannot be contained directly in the base region, will be directly contained in such subregions (or vice-versa). Because the thin tethered strip that was used for flapping is not relevant, it is represented by dots (see Figure 14d). As in the previous solutions, an arc (i, j) is in the dag if and only if a subregion S_k of R_i directly contains region R_j.

Superficially, this representation looks similar to Figure 13. However, its interpretation is totally different, it is now a flattened three-dimensional object. We can visualize this by imagining a living organism that has been totally flattened by a roller-compactor (apologies for the "gory" image).

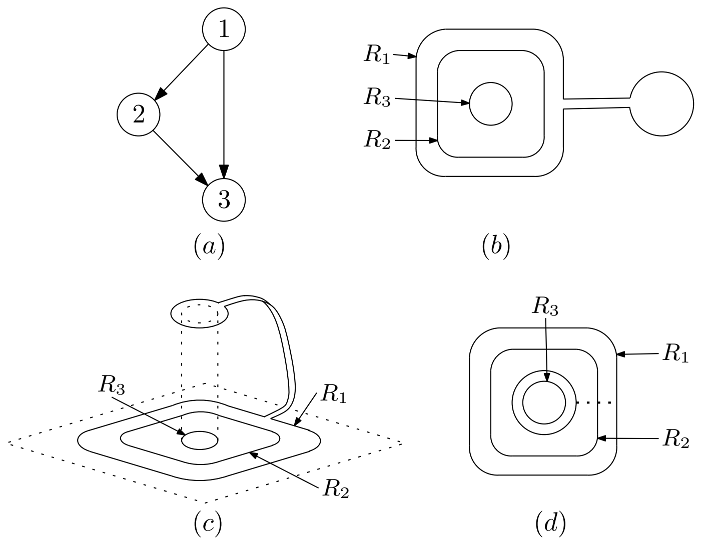


Figure 14: The process described in Solution III.

We next give a constructive algorithm that takes as input a dag (X, δ) and produces a set of overlapping regions $\{R_k \mid k \in X\}$, such that $(i, j) \in \delta$ if and only if a subregion of R_i directly contains R_j.

Algorithm 15: A dag to regions.

Input: dag (X, δ).
Output: flattened regions $\{R_k \mid k \in X\}$.

```
Step 1: Reorder the nodes of the dag $(X, \delta)$ to be in reverse topological order.
        (That is, sink nodes come before source nodes.)

Step 2: For each node $i$ in $\delta$ ordered as in step 1 do:
        If $i$ is a sink:
            Create a new region $R_i$ disjoint from all previous regions.
        Otherwise:
            Create a base region of $R_i$ by creating a simple closed re-
            gion properly containing the union of all regions $R_j$ such
            that $(i, j) \in \delta$. Further, for any transitive arc $(i, j)$ create
            a flap subregion that directly contains $R_j$ and attach it
            with a strip to the edge of the base region.
```

Remark 34 In the set constructed by this algorithm, if two or more transitive arcs are incident to a node j then the respective flaps (without tethers) may share the same projected subregion directly containing region R_j.

Example 35 Figure 15 shows an input dag with six nodes, three transitive arcs and its corresponding planar region representation. Note the reverse topological order is $6, 5, 4, 3, 2, 1$ and the regions R_1 and R_2 use the same flap subregions containing the region R_6.

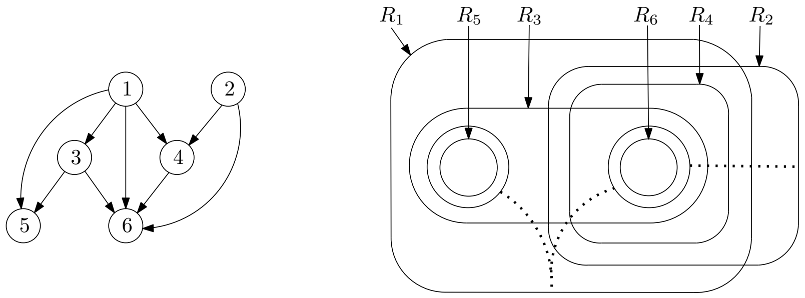


Figure 15: Illustration of Algorithm 15.

Theorem 36 *Every dag with transitive arcs can be represented by a set of regions with folded flaps, with a 1:1 mapping between nodes and regions.*

Proof. We show by induction on the order of the dags that we can always produce a corresponding planar representation. First, note that any dag can be recursively constructed by adding a new node i and arcs incident from i to existing nodes. Note that Algorithm 15 builds planar representations from sink nodes (induction base case) to source nodes (inductive case). Hence, any dag has at least one folded planar representation, depending on the topological order used. We omit the details of how to ensure non-arcs; this can be easily achieved by adding "spikes" to the regions—see our first paper for representing non-transitive dags [NDK09b]. □

Theorem 37 *Every set of regions with folded flaps can be represented by a dag with transitive arcs, with a 1:1 mapping between nodes and regions.*

Proof. We show how to produce a unique dag from a folded planar representation. The first step is to label each region R_k, which will correspond to node $k \in X$ of a dag (X, δ). We add an arc (i, j) to δ if an only if a subregion of R_i directly contains the region R_j. □

Remark 38 One could imagine an additional constraint, that nodes, like cells, need to differentiate between its outside and inside or, in a planar representation, between up and down. We can relate this to membrane polarity, but we refrain from using this idea here, because it can conflict with the already accepted role of polarities in P systems. It is clear that, looking at our example, this solution does not take into account this *sense of direction.*

For example, considering the scenario of Figure 15, regions R_3, R_2 and R_1' (the base subregion of R_1) can be stacked "properly", i.e., with the bottom side of R_3 on the top side of R_2 and the bottom side of R_2 on the top side of R_1'. However, the top side of R_1'' (the flap of R_1) will improperly sit on the top side of R_3, or, vice-versa, the bottom side of R_1'' will improperly sit on the bottom side of R_3.

Can we improve this? The answer follows.

6.4 Solution IV: Flaps with half-twists

This is a variation of Solution III, that additionally takes proper care of the outside/inside (or up/down) directions. We achieve this by introducing half-twists (as used to build Moebius strips), of which at most one half-twist is needed for each simple region.

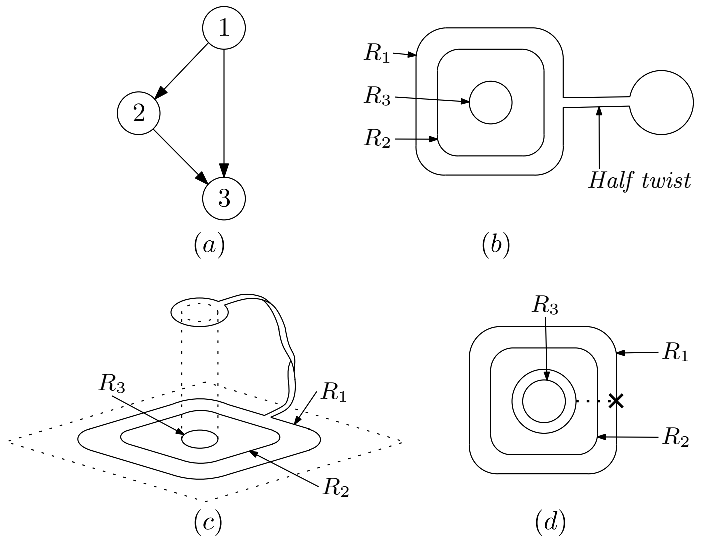


Figure 16: The process described in Solution IV.

Example 39 Figure 16 describes this process.

(a) a given dag with three nodes, $1, 2$ and 3;

(b) three simple regions, R_1, R_2 and R_3, still in the same plane;

(c) R_1 flapped and half-twisted in three-dimensional space;

(d) final "roller-compacted" representation, where dots represent the thin strip that was flapped, and the mark $\times$ a possible location of the half-twist.

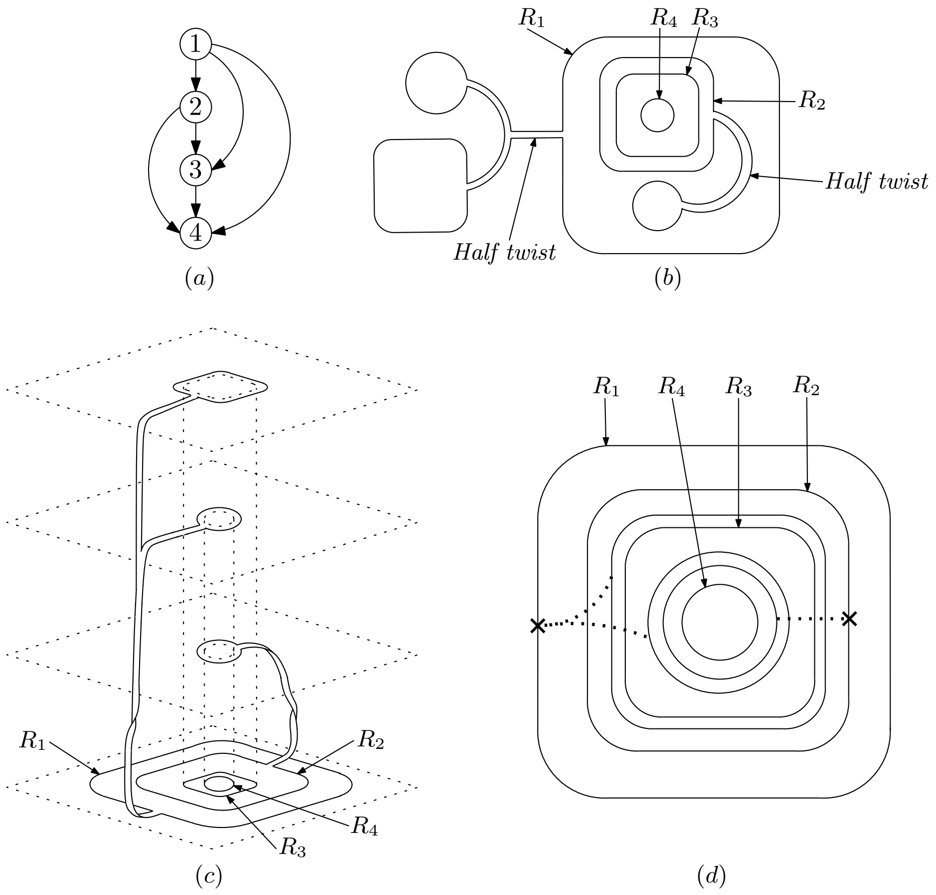


Figure 17: The process described in Solution IV.

Corollary 40 *Dags with transitive arcs can be represented by regions with half-twisted flaps, with a 1:1 mapping between nodes and regions.*

Proof. Since half-twisted flaps are folded flaps, the projection of the boundary of the base and flaps used for a region is the same region as given in the proof of Theorems 36 and 37, provided we always twist a fold above its base. □

Remark 41 This solution solves all our concerns here and seems the best, taking into account the impossibility result (Theorem 29).

6.5 Solution V: Moebius strips

To be complete, we mention another possible solution, which removes any distinction between up and down sides. This representation can be obtained by representing membranes by (connected) Moebius strips.

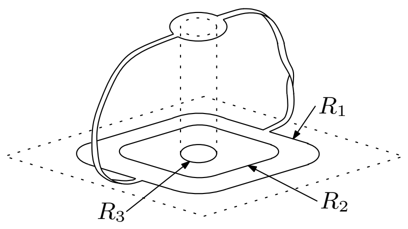


Figure 18: The process described in Solution IV.

Perhaps interestingly, Solutions IV and V seem to suggest links (obviously superficial, but still links) to modern applications of topology (Moebius strips and ladders, knot theory) to molecular biology, for example, see [Fla00].

7 Conclusions

We presented several concrete examples of hP systems for the discovery of basic membrane structure. Our primary goal was to show that, with the correct model in terms of operational and transfer modes, we could present simple algorithms. Our secondary goal was to obtain reasonably efficient algorithms. We first started with cases, where the cells could be anonymous, and showed, among other things, how an hP system could (a) broadcast to descendants, (b) count paths between cells, (c) count children and descendants, and (d) determine cell heights. We then provided examples where we allowed each cell to know its own ID and use it as a communication marker. This model is highlighted by our algorithm that computes all the shortest paths from a given source cell—a simplified version of the distributed Bellman-Ford algorithm, with all unity weights. For each of our nontrivial algorithms, we illustrated the hP system computations on a fixed dag, providing step-by-step traces.

We presented two deterministic solutions FSSP for hP systems and neural P systems with symmetric channels. In contrast to the previous FSSP solutions for tree-based P systems [BGMV08, AMV08], our solutions are a variant of FSSP [Szw82], in which there is one commander, at an arbitrary position. We further generalized the problem by synchronizing a subset of (possibly all) cells of hP systems and neural P systems with symmetric channels. The first solution is based on a novel proposal, which dynamically extends P systems with mobile channels. The second solution is substantially longer, but is solely based on classical rules and static channels. Our solutions do not require membrane polarizations or conditional rules, but require states, as typically used in hyperdag and neural P systems.

We focused on visualizing hP systems in the plane. We presented a natural model, using folded simple closed regions to model the membrane interconnections, including the transitive arcs, as specified by an arbitrary dag structure of an hP system.

As with most ongoing projects, there are several open problems regarding practical computing using P systems and their extended models. We end by mentioning just a few, closely related to the development of fundamental algorithms for discovery of membrane topology.

- In terms of using membrane computing as a model for realistic networking, is there a natural way to route a message between cells (not necessarily connected directly) using messages, tagged by addressing identifiers, in analogy to the way messages are routed on the internet, with dynamically created routing information?
- What are the system requirements to model fault tolerant computing? The tree structure seems to fail here, because a single node failure can disconnect the tree and make consensus impossible. Is the dag structure versatile enough?
- Do we have the correct mix of rewriting and transfer modes for membrane computing? For example, in which situations can we exploit parallelism and in which scenarios are we forced to sequentially apply rewriting rules?
- Can we find simpler and more efficient solutions for hP systems based on single-source dags? Can we find simpler and more efficient solutions for hP or snP systems using named cells (unique cell IDs)? Can we find a solution for arbitrary strongly-connected (not necessarily symmetric) nP systems?
- What is the relation between the mobile channels we have presented here for P systems and the support for mobile channels in the π-calculus?

Acknowledgements

The authors wish to thank John Morris for detailed comments and feedback that helped us improve the paper.

References

[AMV08] 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.

[BGMV08] 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.

[CDK02] Gabriel Ciobanu, Rahul Desai, and Akash Kumar. Membrane systems and distributed computing. In Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *WMC-CdeA*, volume 2597 of *Lecture Notes in Computer Science*, pages 187–202. Springer, 2002.

[Cio03] Gabriel Ciobanu. Distributed algorithms over communicating membrane systems. *Biosystems*, 70(2):123–133, 2003.

[DKN09] 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.

[Fla00] Erica Flapan. *When Topology Meets Chemistry: A Topological Look at Molecular Chirality.* Cambridge University Press, 2000.

[KG05] Kojiro Kobayashi and Darin Goldstein. On formulations of firing squad synchronization problems. In Cristian Calude, Michael J. Dinneen, Gheorghe Păun, Mario J. Pérez-Jiménez, and Grzegorz Rozenberg, editors, *UC*, volume 3699 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 2005.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[Maz87] Jacques Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theor. Comput. Sci.*, 50:183–238, 1987.

[NDK08] Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Structured modelling with hyperdag P systems: Part A. Report CDMTCS-342, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland, New Zealand, December 2008.

[NDK09a] Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Discovering the membrane topology of hyperdag P systems. In *Membrane Computing, Tenth International Workshop, WMC 2009, Curtea de Argeş, Romania, August 24-27, 2009*, pages 426–451, 2009.

[NDK09b] Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Structured modelling with hyperdag P systems: Part A. In *Membrane Computing, Seventh Brainstorming Week, BWMC 2009, Sevilla, Spain, February 2-6, 2009*, volume 2, pages 85–107, 2009.

[Nog04] Kenichiro Noguchi. Simple 8-state minimal time solution to the firing squad synchronization problem. *Theor. Comput. Sci.*, 314(3):303–334, 2004.

[Pău02] Gheorghe Păun. *Membrane Computing-An Introduction.* Springer-Verlag, 2002.

[Păun06] 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.

[SW04] 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.

[Szw82] 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.*, 19:305–320, 1982.

[UMF02] Hiroshi Umeo, Masashi Maeda, and Norio Fujiwara. An efficient mapping scheme for embedding any one-dimensional firing squad synchronization algorithm onto two-dimensional arrays. In Stefania Bandini, Bastien Chopard, and Marco Tomassini, editors, *ACRI*, volume 2493 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2002.