Proceedings of the International MultiConference of Engineers and Computer Scientists 2014 Vol I,
IMECS 2014, March 12 - 14, 2014, Hong Kong

Termination Detection for Synchronous Algorithms in P Systems

Huiling Wu

Abstract—**We continue the research on termination detection for synchronous algorithms in P systems [1]. This paper is the first attempt to relate the classical definitions of process status and activation assumptions in the usual distributed computing framework to P systems. We validate our approach by modelling a well-known synchronous termination detection algorithm, the Dijkstra-Feijen-Van Gasteren (DFG) algorithm, and its application to synchronous BFS (SynchBFS) algorithm in P systems. A separation of concerns (SoC) P system design of this application is provided, by using our previous proposal, complex state symbols and parallel composition with interaction [1]. We newly propose semantics that is required for matching variables on components of complex state symbols. Our resulting formal P system achieves the same runtime as the DFG algorithm and shows substantially smaller program size than the high-level informal pseudocodes of the DFG algorithm.**

Index Terms—**termination detection, P systems, synchronous, parallel, complex symbols**

I. Introduction

A P system is a parallel and distributed computational model inspired by the structure and interactions of living cells, introduced by Păun [2]; for a recent overview of the domain, see Păun et al.'s [3] recent monograph. Essentially, a P system is specified by its membrane structure (in this thesis, a digraph), symbols and rules. Each cell transforms its content symbols and sends messages to its neighbours using *formal* rules inspired by rewriting systems. The rules of the same cell can be applied in parallel (where possible) and all cells work in parallel, traditionally in the *synchronous* mode.

The adequacy of P systems to model distributed algorithms has been investigated largely during recent years [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [1]. However, to the best of our knowledge, the classical definitions of process status and activation assumption in the usual distributed computing framework have not been addressed in P systems, which is non-trivial in modelling distributed algorithms, e.g., termination detection algorithms. Here we relate these classical definitions and assumptions in distributed computing to P systems and validate our approach by modelling a *synchronous* termination detection algorithm in P systems.

Termination detection determines whether a distributed algorithm has terminated, which is a common problem in distributed computing. A distributed algorithm terminates when each process is *passive* and all channels are *empty*. Intuitively, this can be clearly detected from outside, by an external powerful observer, who can continuously probe all process and all communication channels. However, can the processes themselves detect this termination? For this purpose, processes can run a termination detection algorithm ideally as a *control layer* over the main algorithm without interfering it.

In this paper, we study the Dijkstra-Feijen-Van Gasteren (DFG) algorithm [15], and apply it to synchronous BFS (SynchBFS) algorithm [16]. We provide a SoC designed P system algorithm (hereafter called P algorithm), by using our previous proposal, complex state symbols and parallel composition with interaction [1]. We further propose semantics that is required for matching variables on components of complex state symbols.

II. Preliminaries

We use a refined version of *simple P systems*, as defined in [17], where all cells share the same state and rule sets, extended with generic rules using complex symbols.

A simple P system with duplex channels is a system $\Pi = (V, E, Q, O, R)$, where V is a finite set of cells; E is a set of structural *parent-child digraph* arcs between cells (functioning as *duplex* channels); Q is a finite set of states; O is a finite non-empty alphabet of symbols; and R is a finite set of multiset rewriting rules.

All components of a P system, i.e. V, E, Q, O and R, are immutable. Each cell, $\sigma_i \in V$, has the initial configuration (S_{i0}, w_{i0}), and the current configuration (S_i, w_i), where $S_{i0} \in Q$ is the initial state; $S_i \in Q$ is the current state; $w_{i0} \in O^*$ is the initial multiset of symbols; and $w_i \in O^*$ is the current multiset of symbols. The general form of a rule in R is:

$$r : S \ x \ \rightarrow_\alpha \ S' \ x' \ (y') \ (y)\beta_\gamma \ldots \ | \ z \neg z',$$

where: $S, S' \in Q$, $x, x', y, z, z' \in O^*$, $\alpha \in \{\mathtt{min}, \mathtt{max}\}$, $\beta \in \{\uparrow, \downarrow, \updownarrow\}$, $\gamma \in V \cup \{\forall\}$ and ellipses $(\ldots)$ indicate possible repetitions of the last parenthesized item; state S is known as the rule's *starting* state and state S' as its *target* state.

For cell σ_i in configuration (S_i, w_i), a rule, r, is *applicable* if $S = S_i$, $xz \subseteq w_i$, $z' \cap w_i = \emptyset$, where multiset z is a *promoter* and z' is an *inhibitor*, which enables and disables the rule respectively, without being consumed [3] and either (a) no other rule was previously applied, in the same step, or (b) all rules previously applied, in the same step, have indicated the same target state, S'.

When applied, the rule consumes multiset x and fixes, if not already fixed, the target state to S'. Multiset x' becomes *immediately available* in the same cell [17]. Message y' is sent to the same cell via a *loopback* channel; message y is *queued* and sent, at the end of the current step, as indicated by the transfer operator β_γ. β's arrow indicates the transfer direction: $\uparrow$—to parents; $\downarrow$—to children; $\updownarrow$—in both directions. γ indicates the distribution form: $\forall$—a broadcast, which is the default distribution form if no γ is specified; a *structural neighbour*, $\sigma_j \in V$—a unicast (to this neighbour).

Manuscript received December 23, 2013; revised January 27, 2014.
H. Wu is with the Department of Computer Science, University of Auckland, New Zealand, e-mail: huiling.wu@auckland.ac.nz.

Operator α describes the rewriting mode: `min` indicates that an applicable rule is applied once; `max` indicates that an applicable rule is applied as many times as possible.

Example 1 explains how a set of rules are considered for applicability and applied in *one* step.

Example 1. Consider the following rules with priorities, $\{r_1, r_2, r_3\}$, in a system where cell σ_1 contains one symbol, a, and has one child cell, σ_2.

$$\begin{array}{l} r_1\colon\ S_0\ a\ \to_{\mathtt{min}}\ S_0\ c\ (b)\ (f)\downarrow_2 \\ r_2\colon\ S_0\ b\ \to_{\mathtt{min}}\ S_1\ d\ (g)\downarrow_2 \\ r_3\colon\ S_0\ c\ \to_{\mathtt{min}}\ S_0\ a\ (h)\downarrow_2 \end{array}$$

- First, rule r_1 is applied: one c becomes immediately available (which can be used by lower priority rules); one b is sent to itself; and one f is sent to σ_2. Also, the target state is fixed to S_0.
- Next, rule r_2 is not applicable, for two distinct reasons: (1) there is no b in the current content (the message b sent to itself by rule r_1 arrives at the end of the step) and (2) it indicates a target state, S_1, different from the one already selected, S_0.
- Finally, rule r_3 is applied: one a becomes available and one h is sent to σ_2.
- At the end of the step, σ_1 contains ab and message fh arrives at σ_2.

We next discuss extended features [1] used in this paper, which provide powerful ingredients for modelling distributed algorithms.

A. Complex symbols

Complex symbols [17] provide complex data structures for complex distributed algorithms and allow the design of *fixed-size* P algorithms, i.e. solutions having a fixed number of rules, which does not depend on the number of cells in the underlying P systems.

Complex symbols can be viewed as complex molecules, consisting of elementary atoms or other molecules, which are compound terms of the form: $t(i, \dots)$, where (1) t is an *elementary symbol* representing the functor; (2) i can be (a) an *elementary* symbol, (b) another *complex* symbol, (c) a *free variable* (open to be bound, according to the cell's current content), (d) a *multiset* of elementary and complex symbols and free variables.

Free variables are used for *pattern matching* on term arguments and typically denoted by lowercase subscripts such as i, j, k, or or uppercase letters such as X, Y, Z. Following are examples of complex symbols: $b(2) = b_2$, $c(i) = c_i$, $d(i,j) = d_{i,j}$, $e(j, c^5) = e_j(c^5)$, $f(j, X) = f_j(X)$.

Here we assume that each cell σ_i is "blessed" with a unique complex *cell ID* symbol, $\iota(i)$, typically abbreviated as ι_i, which is exclusively used as an *immutable promoter*.

B. Generic Rules

To process complex symbols, we use high-level generic rules [12], [17], which are identified by an extended version of the classical rewriting mode, a combined *instantiation.rewriting* mode, where (1) the *instantiation* mode is one of {`min`, `max`} and (2) the *rewriting* mode is one of {`min`, `max`}. Four combinations of the instantiation and rewriting modes are used: `min.min`, `min.max`, `max.min`, `max.max`.

- The instantiation mode indicates how many instance rules are conceptually generated, using free variable matching:
  - `min` indicates that the generic rule is nondeterministically generated only *once*, if possible;
  - `max` indicates that the generic rule is repeatedly generated as *many* times as possible, depending on the actually cell contents, without superfluous instances (i.e. without duplicates).

  Note that the rule instantiation is based on the actual cell content and thus a generated rule is always applicable and applied according to the rewriting mode.
- The rewriting mode indicates how each instantiated rule is applied (as in the classical framework).
  - `min` indicates that the instantiated rule is applied once;
  - `max` indicates that the instantiated rule is applied as many times as possible.

  After the instantiated rule is applied, if the instantiation mode is `max`, then the generic rule repeats the generation process until no new rules can be generated.

Example 2. Consider a system where cell σ_7 contains multiset $f_2 {f_3}^2 v$, and the generic rule ρ_α, where $\alpha \in \{\mathtt{min.min}, \mathtt{min.max}, \mathtt{max.min}, \mathtt{max.max}\}$ and i and j are free variables:

$$(\rho_\alpha)\ S_{20}\ f_j\ \to_\alpha\ S_{20}\ (b_i)\updownarrow_j\ \mid v\ \iota_i$$

1) $\rho_{\mathtt{min.min}}$ nondeterministically generates *one* of the following rule instances:
$$\begin{array}{l} (\rho'_1)\ S_{20}\ f_2\ \to_{\mathtt{min}}\ S_{20}\ (b_7)\updownarrow_2 \\ (\rho''_1)\ S_{20}\ f_3\ \to_{\mathtt{min}}\ S_{20}\ (b_7)\updownarrow_3 \end{array}$$
In the first case, using (ρ'_1), cell σ_7 ends with $f_3^2 v$.
In the second case, using (ρ''_1), cell σ_7 ends with $f_2 f_3 v$.
2) $\rho_{\mathtt{min.max}}$ nondeterministically generates *one* of the following rule instances:
$$\begin{array}{l} (\rho'_2)\ S_{20}\ f_2\ \to_{\mathtt{max}}\ S_{20}\ (b_7)\updownarrow_2 \\ (\rho''_2)\ S_{20}\ f_3\ \to_{\mathtt{max}}\ S_{20}\ (b_7)\updownarrow_3 \end{array}$$
In the first case, using (ρ'_2), cell σ_7 ends with $f_3^2 v$.
In the second case, using (ρ''_2), cell σ_7 ends with $f_2 v$.
3) $\rho_{\mathtt{max.min}}$ nondeterministically generates one of the following *lists* of rule instances:
$$\begin{array}{l} (\rho'_3)\ S_{20}\ f_2\ \to_{\mathtt{min}}\ S_{20}\ (b_7)\updownarrow_2 \\ (\rho''_3)\ S_{20}\ f_3\ \to_{\mathtt{min}}\ S_{20}\ (b_7)\updownarrow_3 \\ \quad\dots \\ (\rho'_4)\ S_{20}\ f_3\ \to_{\mathtt{min}}\ S_{20}\ (b_7)\updownarrow_3 \\ (\rho''_4)\ S_{20}\ f_2\ \to_{\mathtt{min}}\ S_{20}\ (b_7)\updownarrow_2 \end{array}$$
In the first case, using (ρ'_3) and (ρ''_3), cell σ_7 ends with $f_3 v$.
In the second case, using (ρ'_4) and (ρ''_4), cell σ_7 also ends with $f_3 v$.
In both cases, although σ_7 still contains f_3, rule $S_{20}\ f_3\ \to_{\mathtt{min}}\ S_{20}\ (b_7)\updownarrow_3$ can not be generated again because `max` instantiation mode does not allow duplicates.
4) $\rho_{\mathtt{max.max}}$ nondeterministically generates one of the following *lists* of rule instances:
$$\begin{array}{l} (\rho'_5)\ S_{20}\ f_2\ \to_{\mathtt{max}}\ S_{20}\ (b_7)\updownarrow_2 \\ (\rho''_5)\ S_{20}\ f_3\ \to_{\mathtt{max}}\ S_{20}\ (b_7)\updownarrow_3 \end{array}$$

$$\ldots$$
$$(\rho_6') \quad S_{20}\ f_3 \ \rightarrow_{\mathtt{max}} \ S_{20}\ (b_7)\updownarrow_3$$
$$(\rho_6'') \quad S_{20}\ f_2 \ \rightarrow_{\mathtt{max}} \ S_{20}\ (b_7)\updownarrow_2$$

In the first case, using (ρ_5') and (ρ_5''), cell σ_7 ends with v.

In the second case, using (ρ_6') and (ρ_6''), cell σ_7 also ends with v.

III. Cell Status and Activation Assumptions

A distributed system consists of a collection of processes and a *communication subsystem* [15]. Each process performs a collection of discrete *events*, each event being an atomic change. To interact with the communication subsystem, a process has *internal events*, which perform local computations, *receive events*, which receive messages from channels, and *send events*, which queue messages to channels [15].

At any time during the computation of a distributed algorithm, a process is either

- *active*, if an internal or send event is applicable; or
- *passive*, if no internal or send event is applicable: only receive events are applicable [15].

The following *activation assumptions* are usually made:

- internal events can be only activated or disactived by receive events; and
- send events can be only activated by internal events.

As a consequence,

- a message can only be sent by an active process;
- a passive process can only become active when a message is received;
- an active process can only become passive after performing an internal event or a send event.

In P systems, the receipt of messages are automatically done (no rule is needed); a rule application can be considered as *combined* internal+send events, which transform contents and send messages. We relate the above definitions to P systems: at any time during the evolution of a P system, a cell is either

- *active*, if it has at least one applicable rule; or
- *passive*, if it cannot apply any (more) rule.

A cell is a *source cell*, if it is active when the P system starts to evolve.

P systems may not conform to usual activation assumptions: a rule can be applicable at the start of next step without receiving a message. One may need constraints to make a P system conform to the usual activation assumptions, so that a rule can be only be applicable on receiving a message.

In Example 1, one a is immediately available after rule r_3 is applied and makes rule r_1 applicable at the start of the next step without receiving a message. However, if we change rule r_3 to r_3 to r_3' by replacing a with (a), which is treated as a loopback message, then this P system conforms to usual activation assumptions.

In this paper, these usual activation assumptions are made in order to simplify algorithm descriptions and all P systems are constrained conform to the usual activation assumptions.

IV. Termination Detection in P Systems

As discussed before, to detect the algorithm termination, cells can run a termination detection algorithm as a *control layer* over the main algorithm. To differentiate the main algorithm, A, with termination detection algorithm, B, A is called the *basic algorithm* while B is called the *control algorithm*; messages in A are *basic messages* while messages in B are *control messages*. Control algorithm B runs in parallel with basic algorithm A, interacting with A at specific points. Following our previous approach [1], to make a clean separation, we describe this combination as a *parallel composition with interaction* in P systems.

This approach enables separation of concerns (SoC) designs, so that a P algorithm of a complex distributed problem can be divided into smaller rule fragments and the solution is the composition of bigger chunks out of rule fragments.

Our previous proposal parallel composition with interaction [1], is a parallel composition with interaction of two P systems, Π_1 and Π_2. This can be considered as running in parallel Π_1 and Π_2, where Π_1 "feeds" symbols to Π_2. This parallel composition is essential for cleanly adding a separate control layer, Π_2, over any algorithm, Π_1.

Consider two P systems, Π_1 and Π_2, which share the *same membrane structure* and satisfy the following conditions:

- Π_1 and Π_2 use disjoint sets of states (if not, without loss of generality, we can relabel the states to satisfy this condition);
- Π_1 and Π_2 share a set of symbols on three conditions:
  - initially, no left-side symbols or promoter symbols of Π_2 are available;
  - no rule of Π_2 has empty left-side symbols;
  - no rule of Π_2 generates symbols of Π_1 or symbols generated by Π_2 do not affect Π_1.

The parallel composition of Π_1 and Π_2 with interaction, denoted as $\Pi_1 \triangleright \Pi_2$, is constructed in the following way:

1) the structure of $\Pi_1 \parallel \Pi_2$ is the same as Π_1 and Π_2;
2) rules of Π_1 and Π_2 are concatenated, with rules of Π_1 having higher priority than rules of Π_2;
3) each state S_i of Π_1 is replaced by *complex state* $\Theta(S_i, Y)$ and each state S_i' of Π_2 is replaced by complex state $\Theta(X, S_i')$.

We propose *weak binding* for matching variables on components of complex state symbols: during rules' application, the final target state is successively refined according to the current rule's target state; at the end of the step, the unmapped variables are set according to the cell's current state.

Example 3 illustrates an example. Π_1 cycles over three states and each iteration generates one symbol, b, and Π_2 cycles over two states and each iteration transforms one b to one d. In $\Pi_1 \triangleright \Pi_2$, Π_2 transforms symbol b generated by Π_1 in each iteration to symbol d, therefore obtaining one d in each iteration.

Example 3.

- Π_1, has 3 states and 3 rules:

$$S_1\ a \ \rightarrow_{\mathtt{min}} \ S_2\ b\ e$$
$$S_2\ e \ \rightarrow_{\mathtt{min}} \ S_3\ f$$
$$S_3\ f \ \rightarrow_{\mathtt{min}} \ S_1\ a$$

Step-by-step evolution:

ISBN: 978-988-19252-5-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

$S_1\ a \Rightarrow S_2\ b\ e \Rightarrow S_3\ b\ f \Rightarrow S_1\ b\ a \Rightarrow S_2\ b^2\ e \Rightarrow \cdots \Rightarrow S_1\ b^n\ a$ (after n iterations)

- Π_2, has 2 states and 2 rules:

$$\begin{array}{lll} S'_1\ b & \rightarrow_{\mathtt{min}} & S'_2\ c \\ S'_2\ c & \rightarrow_{\mathtt{min}} & S'_1\ d \end{array}$$

Step-by-step evolution:
$S'_1\ b \Rightarrow S'_2\ c \Rightarrow S'_1\ d$

- $\Pi_1 \triangleright \Pi_2$, has 4 $(= 2 + 2)$ states and 5 $(= 3 + 2)$ rules:

$$\begin{array}{lll} \Pi_1 : & & \\ r_1 : \Theta(S_1, Y)\ a & \rightarrow_{\mathtt{min}} & \Theta(S_2, Y)\ b\ e \\ r_2 : \Theta(S_2, Y)\ e & \rightarrow_{\mathtt{min}} & \Theta(S_3, Y)\ f \\ r_3 : \Theta(S_3, Y)\ f & \rightarrow_{\mathtt{min}} & \Theta(S_1, Y)\ a \\ \Pi_2 : & & \\ r_4 : \Theta(X, S'_1)\ b & \rightarrow_{\mathtt{min}} & \Theta(X, S'_2)\ c \\ r_5 : \Theta(X, S'_2)\ c & \rightarrow_{\mathtt{min}} & \Theta(X, S'_1)\ d \end{array}$$

Step-by-step evolution:
$\Theta(S_1, S'_1)\ a \Rightarrow \Theta(S_2, S'_2)\ e\ c \Rightarrow \Theta(S_3, S'_1)\ f\ d \Rightarrow \Theta(S_1, S'_1)ad \Rightarrow \Theta(S_2, S'_2)bed \Rightarrow \cdots \Rightarrow \Theta(S_1, S'_1)ad^n$ (after n iterations)

Now we explain the first step evolution, $\Theta(S_1, S'_1)\ a \Rightarrow \Theta(S_2, S'_2)\ e\ c$, and the mapping of target state by *weak binding*. Consider above rules of $\Pi_1 \triangleright \Pi_2$ in a system where cell σ_1 in state $\Theta(S_1, S'_1)$ contains one symbol a.

1) First, rule r_1 is applied: one a is consumed and be become immediately available; σ_1's target state is temporarily mapped to $\Theta(S_2, Y)$.
2) Next, rule r_4 is applied: one b is consumed and one c becomes immediately available; σ_1's target state is now fixed to $\Theta(S_2, S'_2)$.
3) At the end of the step, σ_1 is in state $\Theta(S_2, S'_2)$ and contains multiset ec.

$\Pi_1 \triangleright \Pi_2$ composition generates 6 states: $\Theta(S_1, S'_1)$, $\Theta(S_2, S'_1)$, $\Theta(S_3, S'_1)$, $\Theta(S_1, S'_2)$, $\Theta(S_2, S'_2)$ and $\Theta(S_3, S'_2)$. However, some are not reachable due to the sharing of symbols; only three states are used: $\Theta(S_1, S'_1)$, $\Theta(S_2, S'_2)$ and $\Theta(S_3, S'_1)$.

In this paper, we use parallel composition with interaction to apply a termination detection algorithm (Π_1) to a basic algorithm (Π_2), obtaining an augmented algorithm, in which the source cell knows when the basic algorithm terminates, but other cells are not aware of the termination.

V. Dijkstra-Feijen-Van Gasteren algorithm

The Dijkstra-Feijen-Van Gasteren (DFG) algorithm assumes *synchronous* messaging and an underlying network containing a Hamiltonian cycle (also called a *ring*). It checks whether all cells are passive by passing a *d-token*, around the ring using a black and white colouring scheme.

This algorithm is *round-based*, which detects termination by *repeated* rounds: each round starts when the source cell sends a d-token and ends when the source cell receives back the d-token and becomes passive.

Assume that the basic algorithm is extended with ingredients for the DFG algorithm: a *white or black* property for each cell and a global *white or black d-token*, the DFG algorithm detects termination of the basic algorithm based on the following *detection rules*.

Additions to the basic algorithm:

(I) Initially, all cells are white.
(II) A (source or non-source) cell that sends a basic message becomes black.
(III) When the source cell becomes *passive* in the basic algorithm, it sends a white d-token to start the first round (this is done only once).

Control layer of the DFG algorithm:

(IV) A non-source cell only forwards the d-token when it is *passive* in the basic algorithm.
(V) When a black non-source cell forwards the d-token, the d-token becomes black (if it is white).
(VI) Each (source or non-source) cell becomes white (if it is black) immediately after forwarding the d-token.
(VII) When the d-token returns to the source cell, the source cell waits until it is *passive* in the basic algorithm:
  (a) if the d-token and the source cell are white, the source cell knows termination;
  (b) otherwise, the source cell sends a white d-token again to start another round.

Rules (IV) and (VII) of the DFG algorithm can be only applied when a cell would become *passive* in the basic algorithm. In P systems, this is achieved by the parallel composition with interaction. The rules of the DFG algorithm have lower priority than the rules of the basic algorithm and thus can be only applied when a cell can not apply any more rules of the basic algorithm, i.e. when a cell would become passive in the basic algorithm.

As a simple illustration, Example 4 shows the most straightforward way to detect termination for synchronous BFS (SynchBFS). SynchBFS produces a BFS spanning tree in the synchronous mode. Initially, the source cell broadcasts a visit token. On receiving the visit token, an unvisited cell marks itself as visited, chooses one of the token sending cells as its parent and sends its visit token to all non-parent neighbours [1]. The algorithm terminates when no more visit tokens are sent; however, no cell knows the algorithm termination. To solve this problem, we apply the DFG algorithm to SynchBFS; the augmented algorithm is called SynchBFS+DFG.

Example 4. Figure 1 shows how to apply the DFG algorithm to SynchBFS in a *synchronous* scenario. Graph G contains a ring, $\sigma_1.\sigma_2.\sigma_3.\sigma_4.\sigma_1$.

(a) At the start, the source cell, σ_1, broadcasts its visit token and becomes black.
(b) On receiving σ_1's visit token, each of the unvisited cells, σ_2, σ_3 and σ_4, marks itself as visited, sets its parent as σ_1, sends its visit tokens to all non-parent neighbours and becomes black.
The source cell, σ_1, sends a white d-token to σ_2 to start round 1.
(c) Visited cells σ_2, σ_3 and σ_4 delete all received visit tokens (no more visit tokens are sent).
On receiving the white d-token, black σ_2 sends a black d-token to σ_3 and becomes white.
(d) On receiving the black d-token, black σ_3 sends a black d-token to σ_4 and becomes white.
(e) On receiving the black d-token, black σ_4 sends a black d-token to σ_1 and becomes white.

ISBN: 978-988-19252-5-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

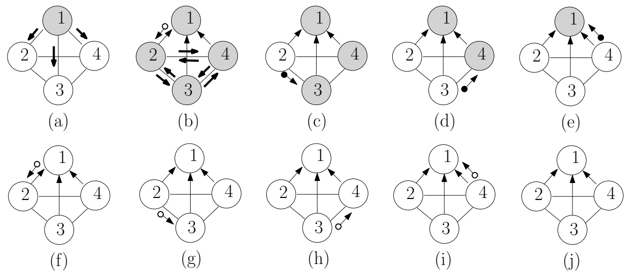

Fig. 1. An example of SynchBFS+DFG in the synchronous mode. Edges with arrows: virtual spanning tree child-parent arcs; thick arrows near edges: basic messages (visit tokens); thin arrows with white or black circles near edges: white or black d-tokens respectively.

(f) The black source cell, σ_1, receives back the black d-token, so it sends a white d-token again to start round 2.
(g)–(i) The white d-token travels around the ring.
(j) Finally, the white source cell, σ_1, receives back a white d-token and thus knows termination.

P Algorithm 1: Control layer of the DFG algorithm

Input: Assumptions of the basic algorithm are made: (I) initially, all cells contain no b'; (II) a cell that sends a basic message generates one b' if it does not contain b'; (III) the source cell, σ_s, sends a loopback message of one t when it starts computation. Additionally, each cell, σ_i, contains its ring successor pointer, r_k.
Output: All ring successor pointer symbols are intact. The source cell, σ_s, contains one g, indicating it knows the algorithm termination.

Symbols and states

Cell σ_i uses the following symbols: r_j indicates a ring successor, σ_j; w is a white d-token; b is a black d-token; b' indicates that it is black (white if it has no b'); t indicates that it must next send a black or white d-token; g indicates that it knows the algorithm termination.

The control layer of DFG algorithm uses one state, S_1.

Rules

1) $S_1\ w\ \rightarrow_{\texttt{min}}\ S_1\ g \mid s \neg b'$
2) $S_1\ w\ \rightarrow_{\texttt{min}}\ S_1\ t$
3) $S_1\ b\ \rightarrow_{\texttt{min}}\ S_1\ t$
4) $S_1\ t\ b'\ \rightarrow_{\texttt{min.min}}\ S_1\ (w) \updownarrow_j \mid s\ r_j$
5) $S_1\ t\ b'\ \rightarrow_{\texttt{min.min}}\ S_1\ (b) \updownarrow_j \mid r_j$
6) $S_1\ t\ \rightarrow_{\texttt{min.min}}\ S_1\ (w) \updownarrow_j \mid r_j \neg b'$

The P rules correspond to the detection rules of the control layer of the DFG algorithm.

(IV) Rules 2–3: when cell σ_i receives a white or black d-token, w or b, it generates one t, indicating it must next send a d-token (rules 2–3).
Then if σ_i is white, $\neg\ b'$, it sends a white d-token, w, to its ring successor, r_j, and deletes t (rule 6).
(V) Rule 5: otherwise (if σ_i is black, b'), it sends a black d-token, b, to its ring successor, r_j, becomes white by erasing b' and deletes t.
(VI) Rule 5: as discussed in (V).
(VII) Rules 1–4, 6: consider the source cell, σ_s, which receives w or b.
  (a) Rule 1: if σ_s, receives w and is white, $\neg\ b'$, then it generates one g, indicating that it knows termination.
  (b) Rules 2–4, 6: otherwise, it generates one t (rules 2–3); then it sends w to start another round and deletes t (rules 4, 6). Also, if σ_s is black, b', it becomes white by erasing b' (rule 4).

VI. SynchBFS+DFG algorithm

The DFG algorithm detect termination based on the usual activation assumptions: it does not work if a cell can become active without receiving any message. Thus, here we ensure that our P algorithm conforms to the usual activation assumptions, denoted as SynchBFS+DFG.

P Algorithm 2: SynchBFS+DFG

Input: All cells start in the same initial state, $\Theta(S_2, S_1')$, and with the same set of rules. Each cell, σ_i, contains an immutable cell ID symbol, ι_i, neighbour pointers, n_j's, and a ring successor pointer, r_k. The source cell, σ_s, is additionally marked with one symbol, s.
Output: All cells end in the same state, $\Theta(S_2, S_1')$; neighbour pointer symbols and cell IDs are intact. Cell σ_s is still marked with one s. Each cell contains a visited mark, v, and a spanning tree parent pointer, p_j. Specifically, the source cell, σ_s, contains one p_s, indicating it is the root of the spanning tree, and contains one g, indicating it knows the algorithm termination.

Table I shows initial and final configurations of P Specification 2 for Figure 1.

TABLE I
Initial and final configurations of xP Specification 2 for Figure 1.

σ_1	σ_2	σ_3	σ_4
$\Theta(S_2, S_1')\ s\ \iota_1$ $r_2\ n_2\ n_3\ n_4$	$\Theta(S_2, S_1')\ \iota_2$ $r_3\ n_1\ n_3\ n_4$	$\Theta(S_2, S_1')\ \iota_3$ $r_4\ n_1\ n_2\ n_4$	$\Theta(S_2, S_1')\ \iota_4$ $r_1\ n_1\ n_2\ n_3$
$\Theta(S_2, S_1')\ s\ \iota_1$ $r_2\ n_2\ n_3\ n_4$ $v\ p_1\ \boldsymbol{g}$	$\Theta(S_2, S_1')\ \iota_2$ $r_3\ n_1\ n_3\ n_4$ $v\ p_1$	$\Theta(S_2, S_1')\ \iota_3$ $r_4\ n_1\ n_2\ n_4$ $v\ p_1$	$\Theta(S_2, S_1')\ \iota_4$ $r_1\ n_1\ n_2\ n_3$ $v\ p_1$

Symbols and states

The modified SynchBFS, Π_1^*, uses the following symbols.

Cell σ_i uses symbols of SynchBFS: n_j indicates its neighbour, σ_j; p_k indicates its BFS parent, σ_k; f indicates that it is a token holding cell; v indicates that it is visited.

Cell σ_i uses specific symbols for the DFG algorithm: r_j indicates a ring successor, σ_j; b' indicates that it is black (white if it has no b'); t indicates that it must next send a black or white d-token.

Rules

Π_1^*: rules of the modified version of SynchBFS by replacing S_2 with $\Theta(S_2, Y)$, where $\boxed{\text{boxed rules}}$ correspond to additions to SynchBFS for the DFG algorithm.

1) $\boxed{\Theta(S_2, Y)\ \rightarrow_{\texttt{min.min}}\ \Theta(S_2, Y)\ (t)\ f_i \mid \iota_i\ s \neg v}$
2) $\Theta(S_2, Y)\ f_j\ \rightarrow_{\texttt{min.min}}\ \Theta(S_2, Y)\ f\ v\ p_j \neg v$
3) $\boxed{\Theta(S_2, Y)\ \rightarrow_{\texttt{min}}\ \Theta(S_2, Y)\ b' \mid f \neg b'}$
4) $\Theta(S_2, Y)\ \rightarrow_{\texttt{max.min}}\ \Theta(S_2, Y)\ (f_i) \updownarrow_j \mid \iota_i\ f\ n_j \neg p_j$
5) $\Theta(S_2, Y)\ f\ \rightarrow_{\texttt{min}}\ \Theta(S_2, Y)$
6) $\Theta(S_2, Y)\ f_j\ \rightarrow_{\texttt{max.max}}\ \Theta(S_2, Y) \mid v$

Π_2: rules of the control layer of the DFG algorithm by replacing S_1 with $\Theta(X, S_1')$.

ISBN: 978-988-19252-5-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

SynchBFS: The source cell, σ_s, generates one token, f_s, which simulates that σ_s receives a visit token from a non-existing cell (rule 1). When unvisited cell σ_i, indicated by $\neg\, v$, receives f_j's, it selects one of the sending cells, by using `min.min` mode, as its parent, p_j, marks itself as visited by v and generates one f, indicating it is being visited (rule 2); next, σ_i sends f_i to all non-parent neighbours, indicated by $n_k \neg\, p_k$ (rule 4), and deletes f (rule 5). Visited cell σ_i, indicated by v, deletes all received f_j's (rule 6).

Additions to SynchBFS: The modification corresponds to the detection rules for the DFG algorithm.

(I) Initially, all cells contain no b'.
(II) Rule 3: symbol f indicates that a cell must next send visit tokens, so rule 1.3 is added, which uses f as a promoter to generate b' if no b' exists.
(III) Rule 1: the source cell, σ_s, sends a loopback message of one t, indicating that it must next send a d-token.

Table II shows partial traces of P Algorithm 2 for cell σ_3 in Figure 1, highlighting the symbols used for the DFG algorithm. Omitted symbols $(\ldots)$ are $\iota_3\ n_1\ n_2\ n_4$.

To explain the cell content evolution at a step, a form $\{r\}\ c \Rightarrow g\ \{l\}\ \{m\}_R \ldots$ is used, where received message r is consumed; multiset c is consumed; multiset g becomes immediately available in the same cell; message l is a loopback message sent to the same cell; message m is sent to neighbours indicated by R.

TABLE II
PARTIAL TRACES OF XP SPECIFICATION 2 FOR CELL σ_3 IN FIGURE 1.

Fig.	Content evolution	Content
(a)		$r_4 \ldots$
(b)	$\{f_1\} \Rightarrow v\ p_1\ \boldsymbol{b'}\ \{f_3\}_{2,4}$	$r_4\ v\ p_1\ \boldsymbol{b'} \ldots$
(c)	$\{f_2\ f_4\} \Rightarrow$	$r_4\ v\ p_1\ \boldsymbol{b'} \ldots$
(d)	$\{\boldsymbol{b}\}\ \boldsymbol{b'} \Rightarrow \{\boldsymbol{b}\}_4$	$r_4\ v\ p_1 \ldots$
(e) (f) (g)		$r_4\ v\ p_1 \ldots$
(h)	$\{\boldsymbol{w}\} \Rightarrow \{\boldsymbol{w}\}_4$	$r_4\ v\ p_1 \ldots$
(i)		$r_4\ v\ p_1 \ldots$

The runtime of a termination detection algorithm is the detection latency of the augmented algorithm; the program size of a termination detection algorithm includes the rules of additions to the basic algorithm and the rules of the control layer of the termination detection algorithm. Thus, these two measures may change when adapted to the specific basic algorithm.

In our example, P Algorithm 2 takes seven steps, achieving the same the runtime complexity of the DFG algorithm as discussed in [15], $O(n)$. The number of rules of P Solution 2 is eight, which is approximately one third of the number of lines of pseudocodes presented in [15].

VII. CONCLUSION

The activation assumptions are non-trivial in usual distributed computing framework, e.g., in termination detection. We address these assumptions in P systems, which we believe is the first attempt relate such distributed concepts in the domain of P systems. This approach is successfully validated by modelling a synchronous termination detection algorithm in P systems. Our P systems use parallel composition with interaction [1] and complex state symbols that are mapped in a newly proposed weak binding way, enabling a high-level SoC design. The resulting P system has a reasonably fixed-size ruleset, achieving the same runtime and substantially smaller program size than standard algorithms.

As future work, we intend to continue this study and make this work more complete, for example, by modelling a termination detection algorithm in the asynchronous setting. We are also interested in investigating a clean and common solution for all P systems to conform to the usual activation assumptions.

ACKNOWLEDGMENT

The author wishes to thank Radu Nicolescu for valuable comments and feedback, and the assistance received via the University of Auckland FRDF grant 9843/3626216.

REFERENCES

[1] R. Nicolescu and H. Wu, "Complex objects and applications," in *Proceedings of the Asian Conference on Membrane Computing (ACMC2013), Southwest Jiaotong University, November 4-7, 2012, Chengdu, China*, 2013, pp. 179–198.
[2] G. Păun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.
[3] G. Păun, G. Rozenberg, and A. Salomaa, *The Oxford Handbook of Membrane Computing*. New York, NY, USA: Oxford University Press, Inc., 2010.
[4] G. Ciobanu, R. Desai, and A. Kumar, "Membrane systems and distributed computing," in *WMC-CdeA*, ser. Lecture Notes in Computer Science, G. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, Eds., vol. 2597. Springer-Verlag, 2002, pp. 187–202.
[5] G. Ciobanu, "Distributed algorithms over communicating membrane systems," *Biosystems*, vol. 70, no. 2, pp. 123–133, 2003.
[6] R. Nicolescu, "Parallel and distributed algorithms in P systems," in *Membrane Computing*, ser. Lecture Notes in Computer Science, M. Gheorghe, G. Păun, G. Rozenberg, A. Salomaa, and S. Verlan, Eds. Springer, Berlin Heidelberg, 2012, vol. 7184, pp. 35–50.
[7] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu, "A faster P solution for the Byzantine agreement problem," in *Conference on Membrane Computing*, ser. Lecture Notes in Computer Science, M. Gheorghe, T. Hinze, and G. Păun, Eds., vol. 6501. Springer-Verlag, Berlin Heidelberg, 2010, pp. 175–197.
[8] R. Nicolescu and H. Wu, "BFS solution for disjoint paths in P systems," in *Unconventional Computation*, ser. Lecture Notes in Computer Science, C. Calude, J. Kari, I. Petre, and G. Rozenberg, Eds. Springer, Berlin Heidelberg, 2011, vol. 6714, pp. 164–176.
[9] T. Bălănescu, R. Nicolescu, and H. Wu, "Asynchronous P systems," *International Journal of Natural Computing Research*, vol. 2, no. 2, pp. 1–18, 2011.
[10] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu, "An adaptive algorithm for P system synchronization," in *Twelfth International Conference on Membrane Computing (CMC12), Fontainebleau/Paris, France, August 23-26, 2011, Proceedings*, 2011, pp. 127–152.
[11] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu, "Faster synchronization in P systems," *Natural Computing*, pp. 1–9, 2011.
[12] R. Nicolescu and H. Wu, "New solutions for disjoint paths in P systems," *Natural Computing*, vol. 11, pp. 637–651, 2012.
[13] H. Wu, "Minimum spanning tree in P systems," in *Proceedings of the Asian Conference on Membrane Computing (ACMC2012), Huazhong University of Science and Technology, October 15-18, 2012, Wuhan, China*, L. Pan, G. Păun, and T. Song, Eds., 2012, pp. 88–104.
[14] H. Wu, "Fast distributed BFS solution for edge-disjoint paths," in *Proceedings of The Eighth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), 2013*, ser. Advances in Intelligent Systems and Computing, Z. Yin, L. Pan, and X. Fang, Eds. Springer Berlin Heidelberg, 2013, vol. 212, pp. 1003–1011.
[15] G. Tel, *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.
[16] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.
[17] H. ElGindy, R. Nicolescu, and H. Wu, "Fast distributed DFS solutions for edge-disjoint paths in digraphs," in *Membrane Computing*, ser. Lecture Notes in Computer Science, E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, and G. Vaszil, Eds. Springer-Verlag, Berlin Heidelberg, 2013, vol. 7762, pp. 173–194.

ISBN: 978-988-19252-5-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)