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. \( V, E, Q, O, R \) and \( R \), are immutable. Each cell, \( \sigma_i \in V \), has the initial configuration \((S_0,w_{i0})\), and the current configuration \((S_i,w_i)\), where is the initial state; \( S_i \in Q \) 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:

\[
\alpha : S x_1 \rightarrow_{\gamma} S' y_1 \cdots z_1 = z' \\\text{where:}
\]

\[
S, S' \in Q, x, x', y, z, z' \in O^*, \alpha \in \{\min, \max\},
\]

\[
\beta \in \{\uparrow, \downarrow, \} \gamma \in V \cup \{\}\text{ and ellipses (\ldots) indicate possible repetitions of the last parenthesized item; state } S \text{ is known as the rule’s starting state and state } S' \text{ as its target state.}
\]

For cell \( \sigma_i \) in configuration \((S_i,w_i)\), a rule, \( r \), is applicable if \( S = S_i, x z \subseteq w_i, z' \cap w_i = \emptyset \), 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 \( y_1 \) 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 \( \beta \), \( \beta \)’s arrow indicates the transfer direction: \( \uparrow \)—to parents; \( \downarrow \)—to children; \( \leftrightarrow \)—in both directions. \( \gamma \) indicates the distribution form: \( \forall \) —a broadcast, which is the default distribution form if no \( \gamma \) is specified; a structural neighbour, \( \alpha_j \in V \)—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 $\alpha$ 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 $\sigma_1$ contains one symbol, $a$, and has one child cell, $\sigma_2$.

\[
\begin{align*}
    r_1: & \quad S_0 \ a \rightarrow_{\min} S_0 \ (b \ (f)) \downarrow_2 \\
    r_2: & \quad S_0 \ b \rightarrow_{\min} S_1 \ d \ (g) \downarrow_2 \\
    r_3: & \quad S_0 \ c \rightarrow_{\min} S_0 \ (h) \downarrow_2 
\end{align*}
\]

- 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 $\sigma_2$. 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, $\sigma_0$.
- Finally, rule $r_3$ is applied: one $a$ becomes available and one $h$ is sent to $\sigma_2$.
- At the end of the step, $\sigma_1$ contains $ab$ and message $fh$ arrives at $\sigma_2$.

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: \( t(i, \ldots), \) 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 uppercase letters such as \( X, Y, Z. \) Following are examples of complex symbols: \( b(2) = b_2, c(i) = c_i, d(i, j) = d_{i,j}, e(j, c^2) = e_j(c^2), f(j, X) = f_j(X). \)

Here we assume that each cell $\sigma_1$ is “blessed” with a unique complex cell ID symbol, \( \iota(i), \) typically abbreviated as \( \iota, \) 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 $\sigma_7$ contains multiset $f_2 f_3^2 v$, and the generic rule $\rho$, where $\rho \in \{\min, \min, \max, \max\}$ and $i$ and $j$ are free variables:

\[
(\rho) \ S_20 \ f_j \rightarrow_{\alpha} S_20 \ (b_i \downarrow_j | v \ i)
\]

1) $\rho_{\min, \min}$ nondeterministically generates one of the following rule instances:

\[
\begin{align*}
    (\rho_1^1) & \ S_{20} \ f_2 \rightarrow_{\min} S_{20} \ (b_7 \downarrow_2) \\
    (\rho_1^2) & \ S_{20} \ f_3 \rightarrow_{\min} S_{20} \ (b_7 \downarrow_3)
\end{align*}
\]

In the first case, using $(\rho_1^1)$, cell $\sigma_7$ ends with $f_2 f_3^2 v$.

In the second case, using $(\rho_1^2)$, cell $\sigma_7$ ends with $f_2 f_3^2 v$.

2) $\rho_{\min, \max}$ nondeterministically generates one of the following rule instances:

\[
\begin{align*}
    (\rho_2^1) & \ S_{20} \ f_2 \rightarrow_{\max} S_{20} \ (b_7 \downarrow_2) \\
    (\rho_2^2) & \ S_{20} \ f_3 \rightarrow_{\max} S_{20} \ (b_7 \downarrow_3)
\end{align*}
\]

In the first case, using $(\rho_2^1)$, cell $\sigma_7$ ends with $f_2 f_3^2 v$.

In the second case, using $(\rho_2^2)$, cell $\sigma_7$ ends with $f_2 f_3^2 v$.

3) $\rho_{\max, \min}$ nondeterministically generates one of the following lists of rule instances:

\[
\begin{align*}
    (\rho_3^1) & \ S_{20} \ f_2 \rightarrow_{\min} S_{20} \ (b_7 \downarrow_2) \\
    (\rho_3^2) & \ S_{20} \ f_3 \rightarrow_{\min} S_{20} \ (b_7 \downarrow_3) \\
    \ldots
\end{align*}
\]

In the first case, using $(\rho_3^1)$ and $(\rho_3^2)$, cell $\sigma_7$ ends with $f_2 f_3^2 v$.

In the second case, using $(\rho_3^1)$ and $(\rho_3^2)$, cell $\sigma_7$ also ends with $f_2 f_3^2 v$.

In both cases, although $\sigma_7$ still contains $f_3$, rule $S_{20} \ f_3 \rightarrow_{\min} S_{20} \ (b_7 \downarrow_3)$ can not be generated again because max instantiation mode does not allow duplicates.

4) $\rho_{\max, \max}$ nondeterministically generates one of the following lists of rule instances:

\[
\begin{align*}
    (\rho_4^1) & \ S_{20} \ f_2 \rightarrow_{\max} S_{20} \ (b_7 \downarrow_2) \\
    (\rho_4^2) & \ S_{20} \ f_3 \rightarrow_{\max} S_{20} \ (b_7 \downarrow_3)
\end{align*}
\]
In P systems, the receipt of messages are automatically
• an active process can only become passive after per-
• a passive process can only become active when a
As a consequence,
• a message can only be sent by an active process;
• a passive process can only become active when a
• an active process can only become passive after per-
In P systems, the receipt of messages are automatically
done (no rule is needed); a rule application can be considered
• internal events can be only activated or disactivated by
• send events can be only activated by internal events. 

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 differentiatate 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, Π1 and Π2. This can be considered as running in parallel Π1 and Π2, where Π1 “feeds” symbols to Π2. This parallel composition is essential for cleanly adding a separate control layer, Π2, over any algorithm, Π1.

Consider two P systems, Π1 and Π2, which share the same membrane structure and satisfy the following conditions:

- Π1 and Π2 use disjoint sets of states (if not, without loss of generality, we can relabel the states to satisfy this condition);
- Π1 and Π2 share a set of symbols on three conditions:
  - initially, no left-side symbols or promoter symbols of Π2 are available;
  - no rule of Π2 has empty left-side symbols;
  - no rule of Π2 generates symbols of Π1 or symbols generated by Π2 do not affect Π1.

The parallel composition of Π1 and Π2 with interaction, denoted as Π1 ∥ Π2, is constructed in the following way:
1) the structure of Π1 ∥ Π2 is the same as Π1 and Π2;
2) rules of Π1 and Π2 are concatenated, with rules of Π1 having higher priority than rules of Π2;
3) each state Si of Π1 is replaced by complex state Θ(Si, Y) and each state S′i of Π2 is replaced by complex state Θ(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. Π1 cycles over three states and each iteration generates one symbol, b, and Π2 cycles over two states and each iteration transforms one b to one d. In Π1 ∥ Π2, Π2 transforms symbol b generated by Π1 in each iteration to symbol d, therefore obtaining one d in each iteration.

Example 3.
- Π1, has 3 states and 3 rules:

\[
\begin{align*}
S_1 & \rightarrow_{\text{ex}} S_2 b \ e \\
S_2 & \rightarrow_{\text{ex}} S_3 f \\
S_3 & \rightarrow_{\text{ex}} S_1 a
\end{align*}
\]

Step-by-step evolution:
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 cannot 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 (SynchronBFS). SynchronBFS 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 SynchronBFS; the augmented algorithm is called SynchronBFS+DFG.

Example 4. Figure 1 shows how to apply the DFG algorithm to SynchronBFS 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, \( \sigma_1 \), broadcasts its visit token and becomes black.

(b) On receiving \( \sigma_1 \)'s visit token, each of the unvisited cells, \( \sigma_2, \sigma_3 \) and \( \sigma_4 \), marks itself as visited, sets its parent as \( \sigma_1 \), sends its visit tokens to all non-parent neighbours and becomes black.

The source cell, \( \sigma_1 \), sends a white d-token to \( \sigma_2 \) to start round 1.

(c) Visited cells \( \sigma_2, \sigma_3 \) and \( \sigma_4 \) delete all received visit tokens (no more visit tokens are sent). On receiving the white d-token, black \( \sigma_2 \) sends a black d-token to \( \sigma_3 \) and becomes black.

(d) On receiving the black d-token, black \( \sigma_3 \) sends a black d-token to \( \sigma_4 \) and becomes white.

(e) On receiving the black d-token, black \( \sigma_4 \) sends a black d-token to \( \sigma_1 \) and becomes white.
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, $\sigma_s$, sends a loopback message of one $t$ when it starts computation. Additionally, each cell, $\sigma_i$, contains its ring successor pointer, $r_s$.

Output: All ring successor pointer symbols are intact. The source cell, $\sigma_s$, contains one $g$, indicating it knows the algorithm termination.

Symbols and states

Cell $\sigma_i$ uses the following symbols: $r_j$ indicates a ring successor, $\sigma_j$; $w$ 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. $S_1 \ x w \rightarrow_{\text{min}} S_1 \ g \ | \ s \rightarrow b’$
2. $S_1 \ x w \rightarrow_{\text{min}} S_1 \ t$
3. $S_1 \ b \rightarrow_{\text{min}} S_1 \ t$
4. $S_1 \ t b’ \rightarrow_{\text{min}, \text{min}} S_1 \ (w) \rightarrow_{j} \ | \ s \rightarrow_{j} r_j$
5. $S_1 \ t b’ \rightarrow_{\text{min}, \text{min}} S_1 \ (b) \rightarrow_{j} \ | \ r_j$
6. $S_1 \ t \rightarrow_{\text{min}, \text{min}} S_1 \ (w) \rightarrow_{j} \ | \ r_j \rightarrow b’$

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

(V) Rules 2–3: when cell $\sigma_i$ 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 $\sigma_i$ is white, $\rightarrow b’$, it sends a white d-token, $w$, to its ring successor, $r_j$, and deletes $t$ (rule 6).

(V) Rule 5: otherwise (if $\sigma_i$ 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, $\sigma_s$, which receives $w$ or $b$.

(a) Rule 1: if $\sigma_s$, receives $w$ and is white, $\rightarrow b’$, 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 $\sigma_s$ is black, $b’$, it becomes white by erasing $b’$ (rule 4).

VI. SYNCHBFS+DFG ALGORITHM

The DFG algorithm detects termination based on usual activation assumptions: it does not work if a cell cannot 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, $\sigma_i$, contains an immutable cell ID symbol, $i$, neighbour pointers, $n_j$’s, and a ring successor pointer, $r_j$. The source cell, $\sigma_s$, 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 $\sigma_s$ 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, $\sigma_s$, 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.

Symbols and states

The modified SynchBFS, $\Pi_1'$, uses the following symbols. Cell $\sigma_i$ uses symbols of SynchBFS: $n_j$ indicates its neighbour, $\sigma_j$; $p_k$ indicates its BFS parent, $\sigma_k$; $f$ indicates that it is a token holding cell; $v$ indicates that it is visited.

Cell $\sigma_i$ uses specific symbols for the DFG algorithm: $r_j$ indicates a ring successor, $\sigma_j$; $b’$ 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

$\Pi_1'$: rules of the modified version of SynchBFS by replacing $S_2$ with $\Theta(S_2, Y)$, where boxed rules correspond to additions to SynchBFS for the DFG algorithm.

1. $\Theta(S_2, Y) \rightarrow_{\text{min}, \text{min}} \Theta(S_2, Y) (t) f_j | t_i \ s \rightarrow v$
2. $\Theta(S_2, Y) f_j \rightarrow_{\text{min}, \text{min}} \Theta(S_2, Y) f v p_j \rightarrow v$
3. $\Theta(S_2, Y) \rightarrow_{\text{min}} \Theta(S_2, Y) b’ | f \rightarrow b’$
4. $\Theta(S_2, Y) \rightarrow_{\text{max}, \text{min}} \Theta(S_2, Y) (f_i) \rightarrow_{j} t_i | t_i f n_j \rightarrow p_j$
5. $\Theta(S_2, Y) f \rightarrow_{\text{min}} \Theta(S_2, Y)$
6. $\Theta(S_2, Y) f_j \rightarrow_{\text{max}, \text{max}} \Theta(S_2, Y) | v$

$\Pi_2$: rules of the control layer of the DFG algorithm by replacing $S_1$ with $\Theta(X, S_1')$. 
SynchBFS: The source cell, $\sigma_0$, generates one token, $f_s$, which simulates that $\sigma_0$ receives a visit token from a non-existing cell (rule 1). When unvisited cell $\sigma_i$, indicated by $\neg v$, receives $f'_s$, 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, $\sigma_i$ sends $f_j$ to all non-parent neighbours, indicated by $n_k \not= p_k$ (rule 4), and deletes $f$ (rule 5). Visited cell $\sigma_i$, indicated by $v$, deletes all received $f'_s$’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, $\sigma_0$, 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 $\sigma_3$ in Figure 1, highlighting the symbols used for the DFG algorithm. Omitted symbols (…) are $\iota_4 \iota_1 \iota_2 \iota_4$. 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>
<thead>
<tr>
<th>Fig.</th>
<th>Content evolution</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>$f \Rightarrow v p_1 b' f_4 s_{2,4}$</td>
<td>$r_4 v p_1 b' \ldots$</td>
</tr>
<tr>
<td>(b)</td>
<td>$f_2 f_4 \Rightarrow r_4 v p_1 b' \ldots$</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>$g b' \Rightarrow r_4 v p_1 \ldots$</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>$l g$</td>
<td>$r_4 v p_1 \ldots$</td>
</tr>
<tr>
<td>(e)</td>
<td>$w \Rightarrow w s_{4,4}$</td>
<td>$r_4 v p_1 \ldots$</td>
</tr>
</tbody>
</table>

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 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


