Computer Science • 17 (1) 2016 http://dx.doi.org/10.7494/csci.2016.17.1.45

Akos Balasko

ON A WORKFLOW MODEL BASED ON GENERALIZED COMMUNICATING P SYSTEMS

Abstract *This paper introduces a new formal mathematical model for investigating workflows from dynamical and behavioural point of view. The model is designed on the basis of a special variant of the biology-inspired formal computational model called membrane systems, where the jobs or services are represented by membrane objects whose behaviour is defined by communication and generalization rules. The model supports running computations in a massive parallel manner, which makes it ideal to model high throughput workflow interpreters. Among the variants introduced in the literature, we have selected the Generalized Communicating P Systems, as it focuses on the communication among the membranes.*

Most of the workflow languages, based on different formal models like Petri nets or Communicating Sequential Processes, support several predefined structures – namely workflow patterns – to control the workflow interpretation such as conditions, loops etc. In this paper we show how these patterns are adapted into the membrane environment which, taking into account that membrane systems can be used to study complex dynamic systems' runtime behaviour, makes this model a relevant alternative for the current models.

Keywords workflow languages, workflow patterns, control patterns, membrane computing, Generalized Communicating P Systems

Citation Computer Science 17 (1) 2016: 45–68

1. Introduction

At present, distributed and parallel computing plays a key role in supporting research communities. On top of technical infrastructure issues, researchers organize their algorithms, applications as workflows by connecting them together. In this term a workflow is represented by directed graphs where nodes are the applications, and arcs represent dependency among them, which could be interpreted as data transfer or some kind of interactions as well. Considering the requirement of re-usage of workflows including the possibility to replace components, a workflow management system must not affect the application itself.

In most cases workflow languages have formal mathematical bases such as Petri nets or Communicating Sequential Processes, therefore, to design a flexible and adaptive workflow language, the most clear way is if the formal model itself supports dynamism and adaptability. Furthermore, it enables the workflow developer to simulate, validate and verify his/her workflow in a formal mathematical environment. *Generalized Communication Membrane Systems* (*GCPS*), as they are constructed with attention to the communication among membranes disregarding what is going on inside a membrane, is ideal as a formal model for dynamic and massively parallel systems.

In this paper we narrow the set of control patterns identified by the observations of van der Aalst, then we define this smallest set of control patterns as a membrane construct to show that a modified *GCPS* can act as a formal model for workflow management systems.

Commonly used workflow structures are identified and classified as control and data flow patterns [16, 17] as a result of investigation by van der Aalst et al. in various workflow languages. A major outcome of their work is that all of the possible workflow structures can be created by connecting these patterns together. Moreover, a workflow language is as strong in aspect of its descriptive power as much control or data patterns are supported. Hence, if a formal model is capable to describe these patterns, it is capable to describe all the possible workflow structures built on the patterns.

This paper is organized as follows. Section 2 introduces *Membrane Systems* in general, section 3 details how *Membrane System* can model workflow enactment and vice versa, what kind of formal tools are available in the literature for modelling workflow enactment. Then section 4 introduces the *Generalized Communication P Systems* in general. The method we have used to select mandatory workflow patterns is shown in Section 5. The subsequent section introduces our proposed modification of the general *GCPS* model. Section 8 is on the adaptation of these patterns giving informal descriptions and illustrations for them. Conclusions and acknowledgements close the paper.

2. On membrane computing

Concept of Membrane Systems (or P systems) and its variants have been introduced by Gheorghe Păun in [14] as a theoretic computation model that offers a mathemat-

ical framework supported with generative grammatical systems for investigation of massively parallel algorithms in a formal way.

The fundamental idea has come from biology namely from the investigation of cell (membrane) systems; the entity used for computation is analogous with a living cell contains chemical elements, such as oxygen, hydrogen etc. which are represented by symbols of a given alphabet. As many symbols may be in cells, they may be represented by strings, chains of symbols. But these strings are just for representation, the theory defines that symbols compose a multiset in a cell, and the computation is done on all combinations of the symbols in parallel. The computation itself is formalized using generative grammatical rules (named evolution rules) on them as well as the communication between cells (named communication rules).

The membrane system introduced by Georghe Păun contains a set of cells in a tree-structure within an environment. The environment differs from membranes in that it has no evolution rules, but it may be involved in the communication rules of the cells, e.g. a cell can send a symbol out to the environment. In addition, an environment may contain infinite instances of a symbol, while a cell can have a finite number of symbol instances. A computation is controlled by synchronized discrete time steps, on where generative and communication rules are applied. In the field of general theory of membrane systems the applicable rules are selected in a non-deterministic maximal parallel manner, which means that the rules are chosen randomly from the set of the applicable rules, and they must be applied as many times in parallel as many suitable pair of symbols are considered as well, such as *minimal parallel manner*, etc.

Literature considers several variants for the original definition of a membrane system, for instance, the structure may be generalized (investigated in the field of tissue-based membrane system [10]), new types of rules (for instance membrane dissolution) are introduced to investigate dynamic behaviour of the cell structure (in [13]). Mainly they are inspired from biology such as introducing permeability as a property of a membrane, and allowing rule-creation during the computation.

3. Related work

Many formal models may be used for investigating workflows in different aspects such as communicational (Communicating Sequence Processes, CSP [6] or Calculus for Communicating Systems, CCS [11]), and functional or dynamic behaviour (Petri nets [19]). However CSP and CCS offer low-level formal methods, which encumber the focused investigation by increasing the complexity of the workflow structure. Hence, and as it enables to focus on dynamic aspects, Petri nets have become a widely used formalism for workflows as it is discussed in [19] and in [20] compared with Pi-Calculus [12].

Nevertheless, classical Petri nets cannot be used for investigating dynamic changes of the workflow structure, therefore several extensions have been introduced such as coloured, timed or hierarchical Petri nets. These types are introduced in [5],

and their usage in modelling a workflow management system is detailed in [19]; moreover complete workflow systems based on *Timed* and *Coloured Petri nets* are shown in [8, 9].

The relationship between Petri nets and generic P Systems has been studied in [14] and among others by Zhengwei Qi et al., who compared the models in [15] and proposed a hybrid model called MP-nets which has methods motivated by *Active P Systems* (where the rules allow dissolution, or move of membranes) and can be analysed by formal tools coming from the field of Petri nets (for instance, property of boundedness or liveness are defined).

Conversion between Petri nets and P systems is studied by Kleijn et.al. in [7], where they take the dynamical changes of the membrane structure into account. It looks an appropriate approach, but the type of P system they use relies on evolution (or reaction) rules, which must be left out of consideration in our case. Since, if we investigate the model as a workflow management system, evolution rules – as they are applied within the membrane – would represent the computation itself within the node. A workflow management system handles the nodes as black-box applications, therefore in our aspect using of evolution rules is not allowed.

Using membrane systems as formal method to express workflow managements is investigated by Verma et al. [22], nevertheless they use the defaut definition of a membrane system, which has some disadvantages such as they have to deal with evolution rules defining what happens inside the node, what does not fit to the approach of workflow management, on where the nodes are used to be considered as black-box applications.

Generalized Communication P System [21] focuses on the communication between certain cells or between cells and the environment by-passing all the evolution rules. The original concept became a fruitful field of P systems many of its variants proposed and investigated such as working in fair sequential model [18], with minimal interaction [4] or in combination with classical automata [1] in aspect generative power and size [3]. Moreover it is shown that *GCP* systems remain remain computationally complete if they are given with a singleton alphabet of objects and with only one of the restricted types of rules [2].

In the aspect of P systems, in this paper we propose an other variant of *Generalized Communicating P Systems* called *Fine-tuned Communicating P System*. In the next section we give a brief introduction to this formal model.

4. Generalized Communicating P Systems

Generalized Communicating P Systems (*GCPS*) originally has been conceived by Verlan et al. [21] as a focused model on the field of membrane computing, where the intercell communication is in focus leaving the evolution of the symbols within cells out of consideration.

The default model of *GCPS* of degree n, where $n \geq 1$, is an $(n+4)$-tuple

$$\Pi = (O, E, w_1, \ldots, w_n, R, o)$$

where:

- O is a finite alphabet, called the *set of symbols* of Π;
- $E \subseteq O$; called the *the set of environmental symbols* of Π;
- $w_i \in O^*$, for all $1 \leq i \leq n$, are strings which represent the multiset of objects *initially associated with cell* i;
- R is a finite set of *interaction rules* of the form $(a,i)(b,j) \to (a,k)(b,l)$, where $a, b \in O$, $0 \leq i, j, k, l \leq n$, and if $i = 0$ and $j = 0$, then $\{a, b\} \cap (O \setminus E) \neq \emptyset$; i.e., $a \notin E$ and/or $b \notin E$.
- $o \in \{1, 2, \ldots, n\}$ specifies the output cell.

The P system consists of n cells, numbered from 1 to n. Each cell contain multisets of symbols over O; initially cell i contains the multiset w_i. The cell numbered by 0 is called the *environment*. The environment can be considered as a specific cell which allows to contain symbols of $E \subseteq O$ in an infinite number of copies. The cells interact with each other by means of the rules in R having the form $(a,i)(b,j) \to (a,k)(b,l)$, with $a, b \in O$ and $0 \leq i, j, k, l \leq n$. This kind of rule form gives a condition, namely, symbols a and b must be in membranes i and j, respectively, and the effect of performing the rule is that a moves to membrane k and in parallel b moves to membrane l.

The computation starts on those symbols which are available initially $(w_1, \ldots, w_n)$ in the cells, and performs a set of interaction rules selected nondetereministically from the possible ones. According to the rules, symbols move from a cell to an other one, remain in their cell or leave to the environment. When none of the rules can be applied, the system halts, and the number of the symbols (or without loss of generality we can limit the output to the number of a specified symbol) contained by the output cell can be considered as a result. Empty cells are prohibited, these cells are "dead", they cannot receive further symbols anymore.

Interpretation of a node in terms of workflow management can be modelled as the existence of a specified symbol in a cell, hence moving this symbol into the cell means its execution, and moving it out from the cell means that the execution is finished.

The original model operates on *maximal parallel* semantics meaning that once an interaction rule has been selected to be used, it must be used on as many symbol pairs as it is possible. Nevertheless the paper mentions that other rule semantics can be considered depending on the number of the symbols processed by each application of such a rule. Let us cite some of them:

1. One application of a rule $(a,i)(b,j) \to (a,k)(b,l)$ affects $\min(|w_i|_a, |w_j|_b)$ objects (i.e. $\min(|w_i|_a, |w_j|_b)$ occurrences of a are moved from cell 1 to cell 2, and $\min(|w_i|_a, |w_j|_b)$ occurrences of b are moved from cell 3 to cell 4) ($\langle h : h \rangle$ semantics).

2. One application of a rule $(a,i)(b,j) \to (a,k)(b,l)$ affects *one* occurrence of a and *all* occurrences of b ($\langle 1 : all \rangle$ semantics).
3. One application of a rule $(a,i)(b,j) \to (a,k)(b,l)$ affects h_1 occurrences of a and h_2 occurrences of b, with h_1, h_2 arbitrary values such that $1 \leq h_1 \leq |w_i|_a$, $1 \leq h_2 \leq |w_j|_b$ ($\langle : \langle$ semantics).

In this paper we give a variant of *GCPS*, where a rule semantics can be defined on each communication rules in the system, in a fine-grain level, instead of defining the application strategy globally for all rules. Moreover, to the best of our knowledge there is no publication that investigates *GCP* systems modified in this way.

5. Control patterns

Workflow patterns were defined by van der Aalst et al. [17] after investigating many commonly used scientific and business workflow languages in many aspects such as control structures, data dependency or exception-handling techniques.

However, several workflow patterns are not independent from each other. A pattern can be derived from another by generalizing it, or from two others by composing them, as these relations are depicted in Figure 1 by solid and dashed edges, respectively. Edges are considered to be directed from the specified or component patterns to the pattern which is created using them.

Hence, we do not need all patterns to be formalized in *GCPS*, it is sufficient if we implement those ones which meet the following requirements, all the others can be created from them. We implement a pattern if and only if:

- it is the most generic;
- it is not composed from others;
- it is supported by at least two workflow languages

First and second condition follow from the construct of the patterns, the third is to guarantee to focus on the "commonly used" patterns, and exclude the exotic ones. The following patterns satisfy the requirements (marked by circles in Figure 1): *Exclusive Choice, Parallel Split, Generalized AND-Join, Multiple Merge, Acyclic Synchronizing Merge, Blocking Partial Join, Interleaved Routing, Cancel Region, Multiple Instances with A-priori runtime knowledge.*

In section 8 we describe them one by one informally and define them as a certain *GCPS*.

6. Power of GCPS in correlation with control patterns

Regarding the functional behaviour of the selected patterns, it can be observed that some of them cannot be defined in the default *GCPS* model because of the manner of the interaction rules. In this section we illustrate this statement with *Generalized AND-Join* pattern.

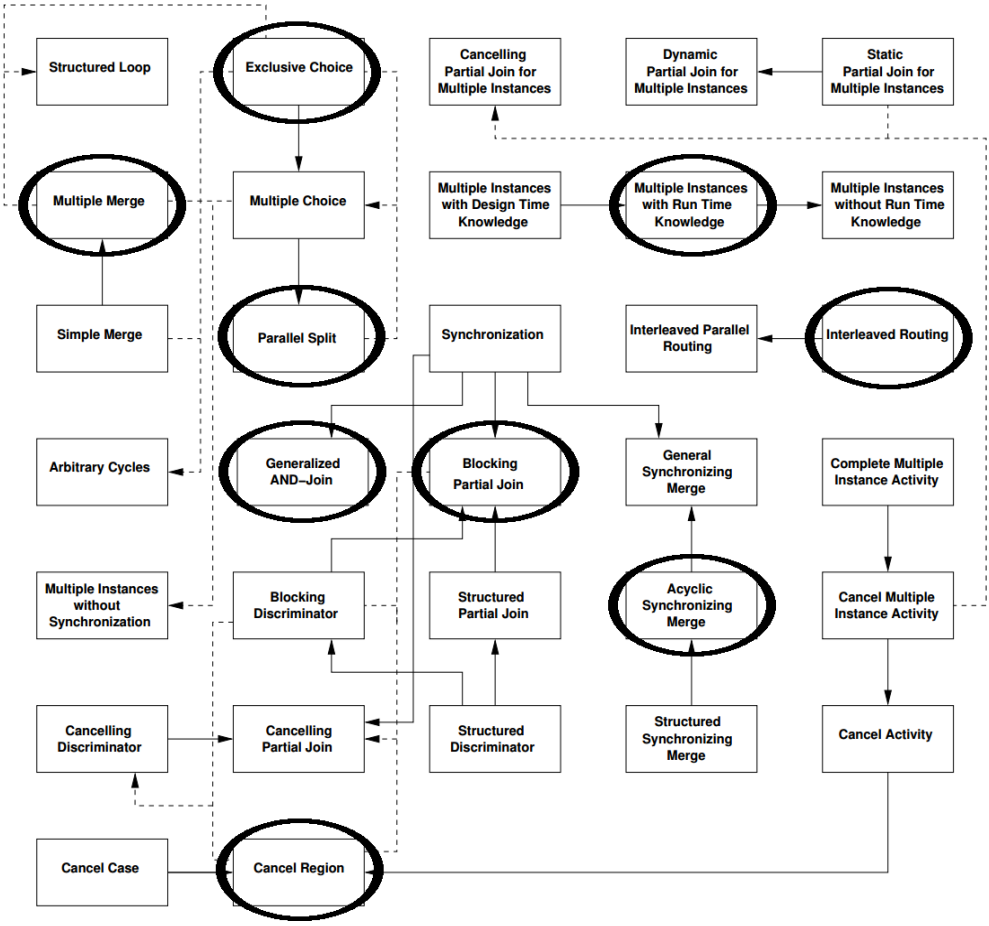

Figure 1. Control patterns (taken from [17]).

This pattern controls parallel branches of execution and waits till all of them are executed successfully, then a single subsequent branch will be interpreted. Figure 2 shows an example structure of the Generalized AND-Join pattern specified for n parallel branches, nodes $A_1, \ldots, A_n$ represent the parallel branches, while node B is to be executed after *all the others*.

Assuming that k symbols of a from n cells have already been moved to cell B. This case shows the key point of this pattern, namely the system must wait for the other $n - k$ symbols before moving a symbol to the subsequent $(n + 1)$th cell. In other words, we must guarantee that a rule must *not* be applied before *all* of the symbols are available in the cell. Nevertheless the default semantics of an interaction rule $(r\colon (a, i)(x, j) \to (a, n + 1)(x, l))$ makes this behaviour impossible, because the semantics defines the followings:

- the multiplicity of the symbols involved in the application of a rule is limited to 1;
- once a rule is selected, it *must* be applied right during that step.

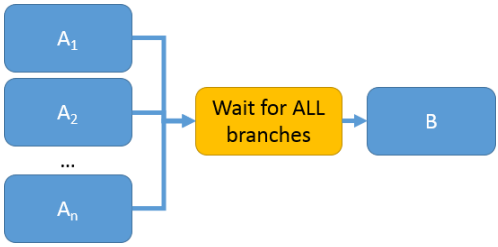

Figure 2. Generalized And-Join pattern.

Moreover, adding more applicable rules would just decrease the probability of applying r, but would not block r till all of the symbols arrive. Other option if the multiplicity of a symbol would be encoded into a set of temporary cells (for instance appearing symbol a in c_i would represent a in i times in the origin), but this technique would increase the number of cells significantly.

Consequently Generalized AND-Join cannot be defined in *GCPS* model, hence the basic model must be extended by defining more semantics for the interaction rules. Some of them were mentioned by Verlan et al. as interesting variants of the core model [21] nevertheless, the descriptive power of the modified versions still has not been shown, and to the best of our knowledge there is no paper in the literature showing these alternatives in action. On the grounds of these arguments in the next section we introduce a variant of *GCPS* which defines independent application strategy for each communication rule.

7. Fine-tuned Communicating P Systems

Since our proposed variant of Communicating P Systems (Fine-tuned Communicating P Systems, *FtCPS* in what follows) is based on *GCPS* model introduced in Section 4, we would not repeat the complete formal definition, but would focus on the differences of the models.

FtCPS of degree n, where $n \geq 1$, is an $(n+4)$-tuple,

$$\Pi = (O, E, w_1, \ldots, w_n, R, h)$$

where:

- O is a finite alphabet, called the *set of symbols* of Π;
- $E \subseteq O$; called the *the set of environmental symbols* of Π;
- $w_i \in O^*$, for all $1 \leq i \leq n$, are strings which represent the multiset of objects *initially associated with cell* i;

- R is a finite set of *interaction rules* of the form $(a,i)(b,j) \xrightarrow{\varrho} (a,k)(b,l)$, where $a,b \in O$, $0 \leq i,j,k,l \leq n$, and if $i = 0$ and $j = 0$, then $\{a,b\} \cap (O \setminus E) \neq \emptyset$; i.e., $a \notin E$ and/or $b \notin E$. Moreover ϱ represents the semantics of the rule;
- $h \subseteq \{1,2,\ldots n\}$ represents subset of cells as *output cells.*

Similarly to *GCPS*, *FtCPS* model handles n cells numbered from 1 to n, which contain multisets of symbols over O; initially cell i contains the multiset w_i. The cells interact with each other by means of the rules in R having the form $(a,i)(b,j) \xrightarrow{\varrho} (a,k)(b,l)$, with $a,b \in O$ and $0 \leq i,j,k,l \leq n$. This formalization of an interaction rule specifies a condition, namely, symbols a and b must be in membranes i and j, respectively, and the effect of performing the rule is that a moves to membrane k and in parallel b moves to membrane l. Moreover the application of the rule is controlled by its semantics denoted by ϱ, which depends on the number of the symbols on which the rule is to be applied.The following types of semantics are allowed:

A rule $(a,i)(b,j) \xrightarrow{\varrho} (a,k)(b,l)$ with

- $\varrho = (p,q)$ (*precise semantics*) affects p instance of a in cell i and q instances of b in cell j, and move all of them in one step to cell k and cell l respectively;
- $\varrho = (*,*)$ (or *star-star semantics*) affects *all* instances of a in i and *all* instances of b in j;
- $\varrho = (X, n-X)$ (*complementer semantics*) affects X instances of a in i and n-X instances of b in j, where X marks the current number of occurrences of symbol a in cell i; for instance if there are 3 symbols of a in cell i and $n = 5$, therefore to be performable, this semantics requires $5 - 3 = 2$ symbols of b in cell j;
- if we do not specify precisely, the rule operates with default maximal parallel semantics.

In terms of *FtCP* system, the followings can be remarked:

- *GCPS* allows one cell as output cell only. This restriction would make the model inapplicable for investigation of control flow structures, because such simple patterns cannot be described in this framework, which contains more than one output jobs (simple split pattern has this property for instance).
- As a remark, Verlan et al. notice in [21] that variants of semantic manners can be considered and several options are enumerated (one to all semantics, star-star semantics- arbitary values etc.). Our work is inspired these remarks, nevertheless, one but essential difference in our model is that we allow to define different semantics for each communication rule, while the *GCPS* model one execution semantics manner is defined *globally for all* interaction rules.

Start of a node execution in terms of *FtCPS* is the appearance of a specific symbol called *control symbol*(symbol s in follows) in the cell that represents the node. Simulating that the node is finished is done by moving symbol s from the cell to an other cell. Hence workflow enactment can be considered as applying the communication rules on the whole system.

In the implementation in Section 8 we use different notation for denoting some cells in the definitions of the pattern implementations, but these modifications only serve the better understanding and have no any consequences in aspect of the semantics of the system. We call those cells as *entry point* of the system and sublabel them with *pre*, which contains symbol s in the initial configuration, and those one as *exit point* with sublabel *post*, which are in the output set.

7.1. Complex workflow compositions

In order to use *FtCPS* as a formal model for workflow languages, the model must be able to satisfy the following criteria:

- All of the relevant control flow patterns must be expressed by an *FtCP* system.
- The system must provide a solution for creating or generating complex control flow strucures based on the patterns.

The pattern implementations are introduced together with the functioning of certain patterns in Section 8.

Taking into consideration that the execution of the pattern starts by moving a symbol s to one of its *entry points*, and the interpretation finishes if no communication rules can be applied, more complex control flow structures can be created by interconnecting these elementary patterns. It can be done by contstructing the union of the structures, resolving the conflicting labels and introducing additional communication rule(s) that perform movement of symbol s from an *exit point* of the former to an *entry point* to the latter structure. Formally, assuming $\Pi_1 = (O_1, E_1, w_1^1, \ldots, w_n^1, R_1, h_1)$ and $\Pi_2 = (O_2, E_2, w_1^2 \ldots, w_m^2, R_2, h_2)$ two *FtCP* system, moreover let $i \in \Pi_1$ and $j \in \Pi_2$ be an *exit point* and *entry point* respectively. Then,

$$\Pi_{union} = (O_1 \cup O_2, E_1 \cup E_2, w_{1_1}, \ldots w_{1_n}, w_{2_1}, \ldots, w_{2_j} \backslash s, \ldots, w_{2_m}, R_{union}, \{h_1 \backslash i\} \cup h_2)$$

where:

$$R_{union} = R_1 \cup R_2 \cup \{(s, 1_i)(\#, 2_j) \xrightarrow{(*,1)} (s, 2_j)(\#, 2_j)\} \tag{1}$$

As it can be seen the union structure contains the alphabet, the cells, the communication rules defined in both component structures. As cell j is an *entry point* in Π_2, it contains a *control* symbol initially. Hence, to achieve that Π_2 starts right after at least one *control* symbol arrives to cell i, w_j^2 must not contain symbol s initially. In contrast R_{union} is extended by an additional communication rule that implements the transfer of one or more occurrences of symbol s from w_i^1 to w_j^2, hence once at least one symbol s arrives to i in Π_1 in the next timestep it is moved to j in Π_2 and enables the interpretation of rules involving cell j in Π_2. Finally regarding the union structure cell i is not an *exit point* anymore, it is withdrawn from the set of the output cells.

8. Defining workflow patterns in the model

In this paper we define the minimal set of workflow patterns in *FtCPS*. After a comprehensive investigation we realized that all of the patterns can be modeled by a *FtCPS*, (which fact shows the expressibility and flexibility of the modelling environment) however, in some other cases the constructed structure utilizes the specific properties of *FtCPS*, such as allowing specifing ϱ semantics on a rule.

For completeness, at first we define the two simplest patterns in *FtCPS*, one that defines one node only, and another, which implements two cells connected.

8.1. Node

A *Node* pattern means only one activity to be interpreted without any dependencies or conditions. It can be defined as a *FtCPS* in a relatively easy way, there are three cells, one is to represent the node itself, and two others for its *pre* and *post* state. Moreover cell *cont* represents a container node in the system to where the unnecessary symbols can be moved. Formally:

$$\Pi_{\text{node}} = (\{s, \#\}, \{\}, \{w_{1_{pre}}, w_1, w_{1_{post}}, cont\}, R, 1_{post})$$

where:

$$\begin{aligned} w_{1_{pre}} &= s \\ w_1 &= \# \\ w_{1_{post}} &= \# \\ R = \{&(s, 1_{pre})(\#, 1) \to (s, 1)(\#, cont), \qquad (2) \\ &(s, 1)(\#, 1_{post}) \to (s, 1_{post})(\#, cont)\} \qquad (3) \end{aligned}$$

Analysis: As it can be seen cell 1_{pre} contains a symbol s denoting that the node can be executed, all the others have a symbol # guaranteeing their *liveness*. The computation starts with the application of rule 2 that moves one instance of symbol s into the *cell 1* and in parallel moved symbol # to the container. As 1_{pre} is empty, it "dies" meaning that no rules can be applied on it in the future. Then rule 3 is applied moving symbol s to from cell 1 to cell 1_{post} and in parallel with symbol # which is moved from cell 1_{post} to *cont*. The symbol s appears in 1_{post} meaning that cell 1 has been executed and, since there are no more rules to be applied, the computation halts.

8.2. Sequence

Sequence construction implements directed connection of two nodes, i.e. node B is connected to node A means that node B must be executed after node A has been finished. This behaviour is implemented in the next *FtCPS*.

$$\Pi_{\text{Sequence}} = (\{s, \#\}, \{\}, \{w_{1_{pre}}, w_1, w_2, w_{2_{post}}, cont\}, R, 2_{post})$$

where:

$$
\begin{aligned}
w_{1_{pre}} &= s \\
w_1 = w_2 &= w_{2_{post}} = \# \\
R = \{&(s, 1_{pre})(\#, 1) \rightarrow (s, 1)(\#, cont), && (4)\\
&(s, 1)(\#, 2) \rightarrow (s, 2)(\#, cont), && (5)\\
&(s, 2)(\#, 2_{post}) \rightarrow (s, 2_{post})(\#, cont)\} && (6)
\end{aligned}
$$

Analysis: The system consists of 5 cells: 1_{pre} and 2_{post} are the entry and exit cells of the system respectively; *cell 1* is the node to be executed at first, while *cell 2* will be executed after *cell 1*; moreover *cont* represents the container cell. According to the initial configuration, rule 4 is applied moving symbol s from 1_{pre} to *cell 1*, while symbol # is being moved from *cell 1* to the container. Next rule 5 is applied thus, symbol s is moved to *cell 2* implementing the essential meaning of the pattern. Finally 6 is applied that moves symbol s to the exit cell (2_{post}) of the system.

As no rule can be applied anymore, the computation halts.

8.3. Parallel Split

This construction represents splitting the interpretation chain to more branches that are executed concurrently. For instance, there are three nodes (A,B,C), node A is connected to both B and C, and B and C are not connected to each other. When node A is interpreted, next node B and node C are allowed to be executed in parallel. It is illustrated in Figure 3.

$$\Pi_{\text{ParSplit}} = (\{s, \#\}, \{\}, \{w_{1_{pre}}, w_1, w_2, w_3, w_{2_{post}}, w_{3_{post}}, w_{cont}\}, R, \{2_{post}, 3_{post}\})$$

where:

$$
\begin{aligned}
w_{1_{pre}} &= s\# \\
w_1 &= \#\# && (7)\\
w_{cont} &= s && (8)\\
w_2 = w_3 &= w_{2_{post}} = w_{3_{post}} = \{\#\} \\
R = \{&(s, 1_{pre})(\#, 1_{pre}) \rightarrow (s, 1)(\#, cont), && (9)\\
&(s, 1)(\#, 1) \rightarrow (s, cont)(\#, cont), && (10)\\
&(s, cont)(s, cont) \rightarrow (s, 2)(s, 3), && (11)\\
&(s, 2)(\#, 2) \rightarrow (s, 2_{post})(\#, cont), && (12)\\
&(s, 3)(\#, 3) \rightarrow (s, 3_{post})(\#, cont)\} && (13)
\end{aligned}
$$

Analysis: According to the signature of the pattern to be implemented, the Π_{ParSplit} consists of one entry cell (labelled as 1_{pre}) and two exit cells (denoted by 2_{post} and 3_{post}). In addition three cells (denoted by 1, 2 and 3) represents the nodes in the

pattern, and an additional cell called *cont* is the container. Initially 1_{pre} and *cont* contain one instance of symbol s, all the others have # only inside. The computations starts by applying rule 9, that simulates that cell 1 is being executed. In the next step rule 10 moves symbol s and # from cell 1 to the *container*. Hence it stores 2 instances of symbol s, enabling 11 to be applied, which moves these symbols to cell 2 and cell 3, representing the starting of the subsequent parallel nodes. Then rule 12 and 13 can be applied meaning that all these nodes are executed, therefore the pattern is interpreted, finally as no other rules to be applied, the computation halts.

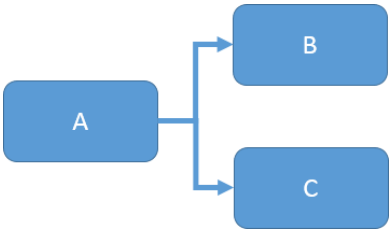

Figure 3. Parallel Split pattern.

8.4. Exclusive Choice

The following pattern (illustrated in Figure 4) is equivalent to XOR in *Bool Algebra*, if node B and C follows node A, and one, and only one subsequent node must be enabled for execution after A is finished. This pattern can be implemented as an *FtCPS* in a straight-forward way.

$$\Pi_{\text{ExclCho}} = (\{s, \#\}, \{\}, \{w_{1_{pre}}, w_1, w_2, w_3, w_{2_{post}}, w_{3_{post}}, cont\}, R, \{2_{post}, 3_{post}\})$$

where:

$$\begin{aligned} w_{1_{pre}} &= s \\ w_1 = w_2 &= w_{2_{post}} = w_{3_{post}} = \# \\ R = \{&(s, 1_{pre})(\#, 1) \to (s, 1)(\#, cont), && (14)\\ &(s, 1)(\#, 2) \to (s, 2)(\#, cont), && (15)\\ &(s, 2)(\#, 2_{post}) \to (s, 2_{post})(\#, cont), && (16)\\ &(s, 1)(\#, 3) \to (s, 3)(\#, cont), && (17)\\ &(s, 3)(\#, 3_{post}) \to (s, 3_{post})(\#, cont)\} && (18) \end{aligned}$$

Analysis: According to the signature of the pattern, one cell called 1_{pre} represents its entry point, while two others labeled as 2_{post} and 2_{post} are the exit points of the system. Moreover there is a *container* cell as always, and three other cells given in this case representing the nodes in the pattern. With the given initial configuration rule 14 is applied moving symbol s to *cell 1*. Then either rule 15 or 17 can be applied,

chosen non-deterministically. As it can be seen, this non-deterministic choice meets the required behaviour meaning that precisely one of them will be "activated" for execution. Then, depending on which rule has been chosen, a symbol s appears either in cell 2 or cell 3 (by application of rule 16 or 18). Since no other rules can be performed after this step, the computation stops.

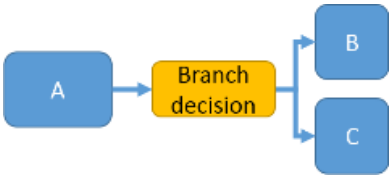

Figure 4. Exclusive Choice pattern.

8.5. Generalized AND-Join

This pattern has already been introduced in Section 6 and illustrated in Figure 2 thus, here we just give the proper *FtCPS* construct that fits to the required behaviour.

$$\Pi_{\text{GenJoin}} = (\{a, \#\}, \{\}, \{w_{1_{pre}}..w_{n_{pre}}, w_1, \ldots w_{n+1}, w_{(n+1)_{post}}, w_{cont}\}, R, (n+1)_{post})$$

where:

$$w_{(n+1)_{post}} = \#$$
$$w_{cont} = \#$$

and $\forall j \in \{1, \ldots, n\}$:

$$w_j = \#$$
$$w_{j_{pre}} = s$$

moreover $\forall i \in \{1, \ldots, n\}$:

$$R = \{(s, i_{pre})(\#, i) \rightarrow (s, i)(\#, cont), \tag{19}$$
$$(s, i)(\#, i) \rightarrow (s, cont)(\#, cont), \tag{20}$$
$$(s, i)(\#, i) \rightarrow (s, i)(\#, i), \tag{21}$$
$$(s, cont)(s, cont) \xrightarrow{(n-1,1)} (s, cont)(s, n+1), \tag{22}$$
$$(s, n+1)(\#, (n+1)_{post}) \rightarrow (s, (n+1)_{post})(\#, cont)\} \tag{23}$$

Analysis: For simulating the behavior of a Generalized AND-join pattern with n parallel branches, $2n + 3$ cells are needed, n of them represent the different branches (numbered 1 to n); another n cells labeled by $1_{pre} \ldots n_{pre}$ are entry points (one for

each parallel branch); moreover there is a *container* cell similarly to the previous pattern implementations; in addition a cell denoted by $n+1$ is the subsequent node to be executed when all the branches have been finished having an exit point associated with $(n+1)_{post}$ label.

Initially all *entry points* contain one instance of symbol s, the *container* has one instance of symbol #; cell $n+1$ and $(n+1)_{post}$ contain symbol # in one instance each. The computation starts by applying rule 19 for each cell $i \in \{1 \ldots, n\}$, resulting that one instance of symbol s appears in each cell $i \in 1 \ldots n$. Then for each cell $i \in 1 \ldots n$, both rules 20 and 21 are applicable (the left hand side of them are satisfied), rule 20 moves symbol s to the *container*, while rule 21 is just occupies the symbols, but does nothing with them. Allowing non-deterministic choice in this step enables the system to simulate that the parallel nodes can finish in different steps. Hence k instances of symbol s appear in cell *cont* and rule 20 and 21 are applied in the further steps as long as every symbol s from every branch move to the *container*. Once it happens, rule 22 is applied – as it can be seen, it requires n instances of symbol s in the container, and it performs moving of one instance to cell (n+1) representing that the subsequent node can be interpreted. Finally rule 23 is applied that moves symbol s to the cell $(n+1)_{post}$, which represents the exit point of the system, then the computation halts.

8.6. Multiple Merge

The Multiple Merge pattern operates on the same structure: it deals with n parallel branches, but instead of waiting for all branches to be finished, the subsequent branch will be started if any of the parallel branches finishes. In this sense it does not differ from the structure constructed by copying subsequent branch right after the parallel branches without merging anything. In contrast this pattern enables the subsequent node *each time* when the concurrent branches finish at a time even if more than one branch finishes at the same time (it is illustrated in Figure 5).

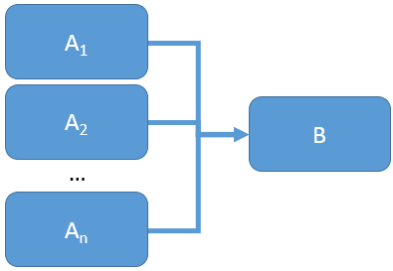

Figure 5. Multiple Merge pattern.

The following *FtCPS* provides the function of the pattern by guaranteeing that one and only one symbol is moved to the subsequent node at a time.

$$\Pi_{\text{MultMerge}} = \\ = (\{s, \#\}, \{\}, \{w_{1_{pre}} \dots w_{n_{pre}}, w_1, \dots w_{n+1}, w_{(n+1)_{post}}, w_{cont}\}, R, (n+1)_{post})$$

where:

$$w_{cont} = s\# \\ w_{n+1} = w_{(n+1)_{post}} = \#$$

and $\forall j \in \{1, \dots n\}$:

$$w_j = \# \\ w_{j_{pre}} = s$$

moreover $\forall i \in \{1, \dots, n\}$:

$$R = \{(s, i_{pre})(\#, i) \to (s, i)(\#, cont), \quad (24)$$
$$(s, i)(\#, i) \to (s, cont)(\#, cont), \quad (25)$$
$$(s, i)(\#, i) \to (s, i)(\#, i), \quad (26)$$
$$(s, cont)(s, cont) \xrightarrow{(*-1,1)} (s, cont)(s, n+1), \quad (27)$$
$$(s, n+1)(\#, (n+1)_{post}) \to (s, (n+1)_{post})(\#, cont)\} \quad (28)$$

Analysis: As it can be recognized, the implementation of this pattern in *FtCPS* operates on exactly the same structure as used in case of the Generalized-AND-Join pattern in the previous subsection, except for two aspects. First, minor difference is that the *container* cell contains a symbol s initially; second, major difference is in the application manner of rule 27. It will be applied if at least one symbol s arrives from one of the concurrent brances, and it performs moves to the subsequent cell labeled $n+1$, instead of waiting for the appearance of all the n instances in the *container*. Moreover, if more instances of symbol s arrive to the *container* in one step, rule 27 will be applied only once. Hence, the function of the pattern is realized.

8.7. Acyclic Synchronizing Merge

A node gets into different states during its execution on a remote infrastructure. These states can be classified into two major groups, *run-time states* are temporary ones which are mainly useful for debugging or tracking the execution in a fine grain level, while *end states* are those ones which will not be changed anymore, such as *error* or *finished*. This pattern provides merging methods as well as the previous patterns, but it is supported with attention to end states of the parallel nodes. It means that a subsequent branch will be enabled if *all* of the previous parallel branches are in end state, even if some of them have been failed. This behaviour is shown in Figure 6.

The next *FtCPS* that achieves the same behaviour contains special symbols denoting the state of the node: symbol $\uparrow$ means that the node is finished successfully,

while symbol ↓ means that the node is failed. We place n instances from both of them in the *container* cell. Then the following *FtCPS* simulates Acyclic Synchronizing Merge pattern.

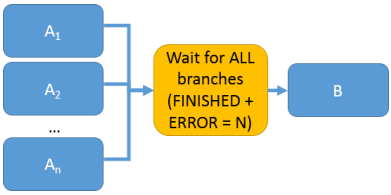

Figure 6. Acyclic Synchronizing Merge pattern.

$$\Pi_{\text{SyncMerge}} =$$
$$= (\{\uparrow, \downarrow, \#\}, \{\}, \{w_{1_{pre}}, \dots w_{n_{pre}}, w_1, \dots w_{n+1}, w_{(n+1)_{post}}, cont\}, R, (n+1)_{post}) \quad (29)$$

where:

$$w_{n+1} = \#; w_{cont} = \uparrow^n \downarrow^n$$

and $\forall i \in \{1, \dots, n\}$:

$$R = \{(s, i_{pre})(\#, i) \to (s, i)(\#, cont), \quad (30)$$
$$(s, i)(\uparrow, cont) \to (s, i)(\uparrow, i), \quad (31)$$
$$(s, i)(\downarrow, cont) \to (s, i)(\downarrow, i), \quad (32)$$
$$(\uparrow, i)(s, i) \to (\uparrow, cont)(s, cont), \quad (33)$$
$$(\downarrow, i)(s, i) \to (\downarrow, cont)(s, cont), \quad (34)$$
$$(s, cont)(s, cont) \xrightarrow{(n-1,1)} (s, cont)(s, n+1), \quad (35)$$
$$(s, n+1)(\#, n+1) \to (s, (n+1)_{post})(\#, cont)\} \quad (36)$$

Analysis: The system consists of $2n + 3$ cells, n stand for the concurrent branches (labeled by $1 \dots n$), n other, labeled by $1_{pre}, \dots, n_{pre}$ represent the entry points of the system, moreover cell *cont* is the container, cell $(n + 1)$ is the subsequent cell to be executed, and the last one is labeled as $(n + 1)_{post}$ is the exit point of the system.

The computation starts with the application of rules 30 (in parallel on each cell $i \in \{1 \dots, n\}$), moving one instance of symbol s to each cell $1 \dots n$. Then nondeterministic choice and application of rule 31 and 32 determines that the certain nodes are finished successfully (application of rule 31, that moves symbol ↑ to the cell) or failed (application of rule 32, that moves symbol ↓ to the cell). Next rules 33 and rules 34 are applied in parallel on all of the n cells, moving n instance of symbol s to the *container*. Rule 35 is applied only if all of the n symbols s are in the *container* cell. By performing it one instance of symbol s is moved to the subsequent (n+1) cell. At last, application of rule 36 moves this symbol to the exit point, presenting that the execution of this pattern is finished.

8.8. Blocking Partial Join

The Blocking Partial Join pattern is for converging multiple branches, similarly to the Generalized AND-Join, but while the latter construct waits for all branches to be finished, in case of Blocking Partial Join pattern the subsequent branch will be executed for each number of k of the branches finished. After enabling the subsequent node for k branches, the join construct resets and waits for collecting the next k branches; during this process the incoming branches are *blocked*. Figure 7 shows an example construct.

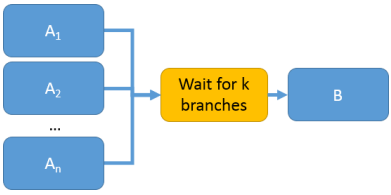

Figure 7. Blocking Partial Join pattern.

The next *FtCPS* is behaviourally equivalent to Blocking Partial Join pattern.

$$\Pi_{\text{PartJoin}} = (O, \{\gamma\}, \{w_0, \dots w_{n+1}, w_{n+1_{post}}, w_{cont}, w_{counter}\}, R, (n+1)_{post})$$

where:

$$\begin{aligned} O &= \{s, b, \#, \delta\} \\ w_{counter} &= b^k \delta^n \\ w_{n+1} &= w_{n+2} = w_{cont} = \{\#\} \end{aligned}$$

and $\forall i \in \{1, \dots, n\}$:

$$\begin{aligned} w_{i_{pre}} &= s \\ w_i &= \# \\ R = \{&(s, i_{pre})(\#, i) \to (s, i)(\#, cont), && (37)\\ &(s, i)(\#, n+1) \to (s, n+1)(\#, (n+1)_{post}), && (38)\\ &(b, counter)(\#, (n+1)_{post}) \to (b, n+1)(\#, n+1), && (39)\\ &(s, n+1)(s, n+1) \xrightarrow{(k-1,1)} (s, cont)(s, (n+1)_{post}), && (40)\\ &(b, n+1)(\delta, counter) \xrightarrow{(k,1)} (b, counter)(\delta, cont), && (41)\\ &(s, n+1)(\delta, cont) \xrightarrow{(n \div k, \lfloor n/k \rfloor)} (s, cont)(\delta, counter), && (42)\\ &(s, cont)(s, cont) \xrightarrow{(n-1,1)} (s, cont)(s, (n+1)_{post})\} && (43) \end{aligned}$$

Analysis: Initially we place symbol s in the *entry points* of the structure denoted by index $1_{pre}...n_{pre}$; the *counter* cell contains k number of symbol b. The computation starts by applying rules 37 on each cell i_{pre} for all $i \in \{1 \dots, n\}$ in parallel. Thus, the control symbol (s) appears in each cell i. In the next step one of rules 38 selected non-deterministically, and then rule 39 moves # symbol back to cell $n+1$ allowing new application of one of the rules 38 in the next step. The computation iterates these two rule applications in k times, that result k occurrences of symbol s and k occurrences of symbol b in cell $n+1$. Next, rule 39 is applied once and $k-1$ instances of symbol s move to the *container* while one occurrence of symbol s moves to the *exit point* of the structure. After this rule 38 moves the next s symbol to cell $n+1$ and in parallel all occurrences of symbol b are moved back to the counter, while one instance of δ moves to the *container*. This loop does $\lfloor n/k \rfloor$ iterations, moving $\lfloor n/k \rfloor$ occurrences of δ to the *counter* cell. At this step there are less than k control symbols are available in cell i for all $i \in \{1 \dots, n\}$. Therefore rule 42 moves the remaining $n \div k$ symbols of s to the *container* cell, which enables the rule 43 to be applied, which moves the last symbol s to the exit point of the system. As no other rules can be applied, the computation halts.

8.9. Interleaved Routing

As it is illustrated in Figure 8, this pattern specifies multiple excursion applicable on n parallel nodes connected to one another, guaranteeing that only one interaction takes effect at the same time. The following structure simulates this behaviour as an *FtCPS*.

$$\Pi_{\text{InterRout}} = (O, \{\}, W, R, \{post\})$$

where:

$$\begin{aligned} O &= \{s, \delta\} \\ W &= \{w_{1_{pre}}, \dots, w_{n_{pre}}, w_1, \dots w_n, w_{free}, w_{occ}, w_{res}, w_{post}\} \\ w_{free} &= \delta \qquad (44) \\ w_{post} &= \# \qquad (45) \end{aligned}$$

and $\forall i \in \{1, \dots, n\}$: $w_i = \#; w_{i_{pre}} = s$

$$\begin{aligned} R = \{&(s, i_{pre})(\#, i) \to (s, i)(\#, cont), \qquad (46) \\ &(s, i)(\delta, free) \to (s, res)(\delta, occ), \qquad (47) \\ &(s, res)(\delta, occ) \to (s, post)(\delta, free)\} \qquad (48) \end{aligned}$$

Analysis: This pattern can be implemented as an *FtCPS* in a relatively straightforward way. Suppose that the computation started and hence rules 46 have already been applied therefore, one occurrence of *control* symbol is in each cell i for all $i \in \{1, \dots, n\}$. However, only one symbol of δ is placed in cell $free$. Therefore, only one rule will be chosen non-deterministically among rules 47 which, besides moving

symbol s to cell res puts δ to cell occ preventing the others from being applied. Then rule 48 can be applied that moves symbol s to cell $post$ and symbol δ to cell $free$ enabling the application of one of the other rules 47 at the next step. This process continues until at least one symbol s is in any of cells $i \in \{1, \ldots, n\}$, then the computation finishes.

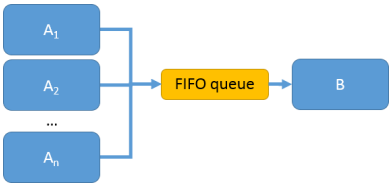

Figure 8. Interleaved Routing pattern.

8.10. Cancel Region

The pattern cancels a complete region of nodes due to runtime environmental events or user interventions (Fig. 9).

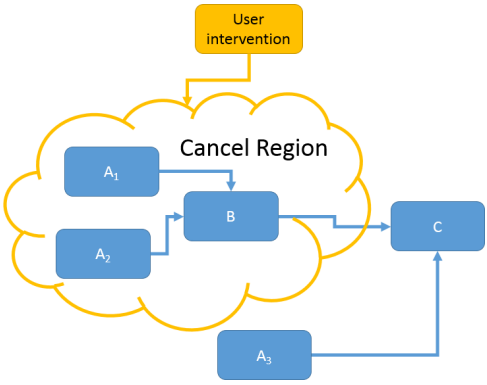

Figure 9. Cancel Region Example.

We construct the following *FtCPS* to implement this pattern.

$$\Pi_{\text{CancelReg}} = (\{s, \epsilon, \gamma\}, \{\}, \{w_{1_{pre}}, w_1 \ldots, w_n, w_{n_{post}}, w_{cont}\}, R, \{n_{post}\})$$

where:

$$w_{1_{pre}} = s; \forall i \in \{1, \ldots, n\} : w_i = w_{n_{post}} = \#; w_{cont} = \epsilon|\gamma$$

and $\forall i \in \{1, \ldots, n\}$:

$$R = \{(1_{pre}, s)(\#, 1_{pre}) \rightarrow (1_{pre}, 1)(\#, cont), \tag{49}$$

$$(a,i)(\epsilon,cont) \to (a,i+1)(\epsilon,cont), \tag{50}$$
$$(a,n)(\epsilon,cont) \to (a,n_{post})(\epsilon,cont)\} \tag{51}$$

moreover $\forall i \in [k \dots l-1]$:

$$(a,i)(\gamma,cont) \to (a,cont)(\gamma,n_{post})\} \tag{52}$$

Analysis: The alphabet, besides containing the symbol s, has special symbols as well: ϵ and γ mark whether the region is enabled to be interpreted (ϵ is in cell *cont*) or must be cancelled (γ appears in cell *cont*). Note that ϵ and γ cannot be in cell c at the same time. Without loss of generality we can assume that the cancel region contains n cells connected in a chain; moreover let cell k identify the entry point of the region and cell l mark its exit point. The entry point of the system contains symbol s only. The computation is done as follows.

By performing rule 49 the interpretation enters to the first cell. The computation is being processed by applying the rules of 50 one by one, as long as time step k. At that time there are two options ahead depending on the content of the *container* cell: if it contains symbol ϵ, then everything goes as before (by applying the next rule of 50 and moving symbol s to the subsequent cell) and – if no symbol γ appeared in the *container* cell during the execution, symbol s arrives to the exit point of the system. Otherwise, if the *container* contains an instance of symbol γ, then rule 52 are applied. In this case the symbol s disappears from the cell-chain of execution and goes to the *container*, while γ goes to the output cell marking that the computation has been canceled.

8.11. Multiple Instances with a priori Runtime Knowledge

This structure is the same as Parallel Split, but, as the number of sub-branches are generated according to run-time events (e.g. number of data generated previously), at design time we do not know how many sub-branches will be generated (Fig. 10).

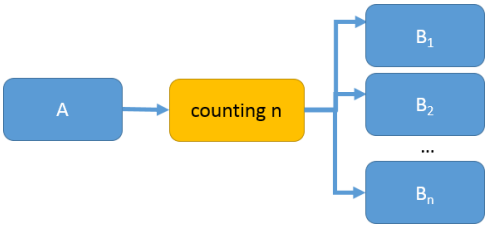

Figure 10. Multiple Instances with apriori Runtime Knowledge pattern.

Analysis: The realization is based on the low-level parallelization possibility of *FtCPS*, namely the possibility to apply a rule as many times concurrently as many instances of

symbols are in the certain cells which a rule refers to. Hence, utilizing this concurrent nature of *FtCPS*, multiple instance pattern can be formulated in *FtCPS* as follows:

$$\Pi_{\mathrm{MI}} = (\{s, \#, \}, \{s\}, w_{1_{pre}}, w_1, w_{1_{post}}, counter, cont, R, 1_{post})$$

where:

$$w_{1_{pre}} = \#'; w_1 = \#''; w_{1_{post}} = \#; counter = b^k; cont = \#''\}$$

and $\forall i \in \{1, \ldots, n\}$:

$$R = \{(s,0)(b, counter) \to (s, 1_{pre})(b, 1_{pre}), \tag{53}$$

$$(s, 1_{pre})(b, 1_{pre}) \to (s, 1)(b, cont), \tag{54}$$

$$(s, 1)(\#, 1) \to (s, 1_{post})(\#, cont)\} \tag{55}$$

The key point of this pattern is to solve that simulating the execution of a node in a number, which is unknown at the design time, in other words the number of the instances must not be hard-coded into the rules. According to the former definition of executing a node in terms of *FtCP* system, execution in many instances means appearing as many instances of symbol s in the given cell as many instances of node should be executed in parallel.

Utilizing the default maximal parallel semantics, it can be achieved easily.

The system consists of five cells, the entry point of the system is labeled as 1_{pre}; the node to be interpreted in many instances is labeled as 1; the exit point is 1_{post}; *counter* cell initially contains a specific symbol b in as many instances as many parallel interpretation of cell i must be achieved, finally cell *cont* is the container cell.

The implementation assumes that symbol s, which appearance in a cell represents its execution, is available in the environment in infinite instances. Symbol $\#, \#'$ and $\#''$ stand for control the computation by counting the timesteps done.

The computation starts by applying rule 53, which moves k instances of s to cell 1_{pre}. As step 2 rule 54 is applied in k instances and performs moving k instances of symbol s and b to cell 1 and to the container cell respectively. Finally k instances of symbol s will be moved to the *exit* point of the system represented by cell 1_{port} by applying rule 55.

9. Conclusion

In this paper we introduced a new variant of the *Generalized Communicating P Systems* called *Fine-tuned Communication P System* where independent application strategies can be defined for each communication rule. Moreover we shown that *FtCPS* can model the commonly used control flow patterns offering a powerful alternative in investigating workflow management systems with mathematical and formal methods. In details, we have identified the smallest set of the patterns from which all the others can be derived. Then we constructed and described in detail a semantically

equivalent *FtCPS* for each of them. In addition, we have described a method to create more complex control flow compositions in *FtCPS* using the patterns as building blocks. As future work we plan to adapt techniques into *FtCPS* which have already been defined in other fields of *Membrane Systems* to support further investigation of dynamic aspects.

Acknowledgements

The author would like to thank to Erzsebet Csuhaj-Varju and Zsolt Nemeth for the contribution and for all of the suggestions. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 608886 (CloudSME).

References

[1] Csuhaj-Varjú E., Vaszil G.: Generalized Communicating P Automata. In: *Automata, Universality, Computation*, pp. 219–236, Springer, 2015.

[2] Csuhaj-Varjú E., Vaszil G., Verlan S.: On generalized communicating P systems with one symbol. In: *Membrane Computing*, pp. 160–174, Springer, 2011.

[3] Csuhaj-Varjú E., Verlan S.: Power and size of generalized communicating P systems with minimal interaction rules. In: *Procedings of the 10th Workshop on Membrane Computing, WMC10, Curtea de Arges (Romania)*, pp. 547–551, Citeseer, 2009.

[4] Csuhaj-Varjú E., Verlan S.: On generalized communicating P systems with minimal interaction rules. *Theoretical Computer Science*, vol. 412(1), pp. 124–135, 2011.

[5] David R., Alla H.: Petri nets for modeling of dynamic systems: A survey. *Automatica*, vol. 30(2), pp. 175–202, 1994.

[6] Hoare C.A.R.: Communicating sequential processes. *Communications of the ACM*, vol. 21(8), pp. 666–677, 1978.

[7] Kleijn J., Koutny M.: A Petri net model for membrane systems with dynamic structure. *Natural Computing*, vol. 8(4), pp. 781–796, 2009.

[8] Ling S., Schmidt H.: Time Petri nets for workflow modelling and analysis. In: *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 4, pp. 3039–3044, 2000.

[9] Liu D., Wang J., Chan S.C., Sun J., Zhang L.: Modeling workflow processes with colored Petri nets. *Computers in Industry*, vol. 49(3), pp. 267–281, 2002.

[10] Martın-Vide C., Păun Gh., Pazos J., Rodrıguez-Patón A.: Tissue P systems. *Theoretical Computer Science*, vol. 296(2), pp. 295–326, 2003.

[11] Milner R.: *Lectures on a calculus for communicating systems*. Springer, 1985.

[12] Milner R.: *Communicating and mobile systems: the pi calculus*. Cambridge University Press, 1999.

[13] Păun A.: On P systems with active membranes. In: *Unconventional Models of Computation, UMC2K*, pp. 187–201, Springer, 2001.

[14] Păun Gh., Rozenberg G., Salomaa A.: *The Oxford handbook of membrane computing*. Oxford University Press, Inc., 2010.

[15] Qi Z., You J., Mao H.: P systems and Petri nets. In: *Membrane Computing*, pp. 286–303, Springer, 2004.

[16] Russell N., Ter Hofstede A.H., Edmond D., Van Der Aalst W.M.: Workflow data patterns. Tech. rep., QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.

[17] Russell N., Ter Hofstede A.H., Mulyar N.: Workflow controlflow patterns: A revised view. Tech. rep., BPM Center Report BPM-06-22, BPMcenter.org, 2006.

[18] Spicher A., Verlan S.: Generalized Communicating P Systems Working in Fair Sequential Model. *arXiv preprint arXiv:1108.3432*, 2011.

[19] Van Der Aalst W.M.: The application of Petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, vol. 8(01), pp. 21–66, 1998.

[20] Van Der Aalst W.M.: Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype. *BPTrends*, vol. 3(5), pp. 1–11, 2005.

[21] Verlan S., Bernardini F., Gheorghe M., Margenstern M.: Generalized communicating P systems. *Theoretical Computer Science*, vol. 404(1), pp. 170–184, 2008.

[22] Verma R., Ahmed T., Srivastava A.: Expressing Workflow and Workflow Enactment using P Systems. In: *The 15th International Conference of Membrane Computing, proceedings*, pp. 357–371, 2014.

Affiliations

Akos Balasko
Hungarian Academy of Sciences, 1111 Budapest, Kende str 13-17, Hungary,
balasko@sztaki.hu

Received: 1.12.2014
Revised: 18.05.2015
Accepted: 19.05.2015