Reformulation, Extension, and Application of the Formal Framework for P Systems

A Thesis Proposal by

Ren Tristan A. de la Cruz

Algorithms and Complexity Laboratory
Department of Computer Science
College of Engineering
University of the Philippines

for the degree of
Doctor of Philosophy in Computer Science

Thesis Proposal Panel Members:

Francis George C. Cabarle, Ph.D., *Adviser*

2020 December 10

Abstract

Membrane computing is a field of computer science that studies biologically-inspired parallel and distributed models of computations known as *P systems*. At the moment, there are hundreds of P systems variants with their own syntax and often informally defined semantics. *Formal framework* attempts to formally define a general syntax and procedural semantics for wide variety of P systems. This research proposal is about the reformulation, extension, and application of the said formal framework.

Table of Contents

1 Introduction

Membrane computing is a field of theoretical computer science that studies different models of computation known as *P systems*. The term '*P systems*' refers to a family of models of computation which are inspired by biological processes. P system models use abstractions of biological processes as computational operations. For example, different types of rules (operations) used by most P system variants are abstractions of processes like *chemical reaction* and *ion transport* that occur inside biological cells. Most P system variants use *object symbols* as the objects of computation. One can think of these object symbols as abstraction of physical *molecules* or *ions*. P systems store *multisets* of these object symbols inside regions enclosed by *membranes*. A P system has a collection of these membranes with multisets of objects symbols inside. The membranes can be 'connected' to each other to form a *membrane structure*.

Aside from studying P systems themselves, membrane computing also deals with applications of P systems. P systems have been used to solve 'hard' (NP-complete) problems. They have been used to model natural phenomena like biological oscillations, population dynamics, and sodium-potassium pump mechanism. Other P system applications include boolean circuit simulation, fault diagnosis models for electric locomotive, linguistic applications, etc.

There are tens, if not hundreds, of P system variants. Their syntax are well-defined but the semantics are often described in an informal manner. *Formal framework* is an attempt to formalize not only syntax but also the procedural semantics of a wide variety of P systems. There are currently three versions of the framework, one for P systems with static membrane structures, one for P systems with dynamic membrane structures, and another for static P systems with input-output. The formal frameworks can be used to analyze, compared, and extended P systems.

The rest of the document can be divided into three parts. The first part, Section 2, describes P systems and formal frameworks in a semi-formal manner. The second part, Section 3, gives the proposal itself with the research objectives and schedule. The third part, the Appendix, provides the full formal description of the formal frameworks.

2 Preliminaries

2.1 P Systems

The following sections describe the components that are common to most (if not all) P system variants.

2.1.1 Membrane Structure

A P system has set of membranes that can be connected to each other. These connected membranes form a *membrane structure*. Early P systems are described as *cell-like* since their membrane structures are inspired by nested membranes inside a cell. Figure 1 shows an example of a cell-like membrane structure.

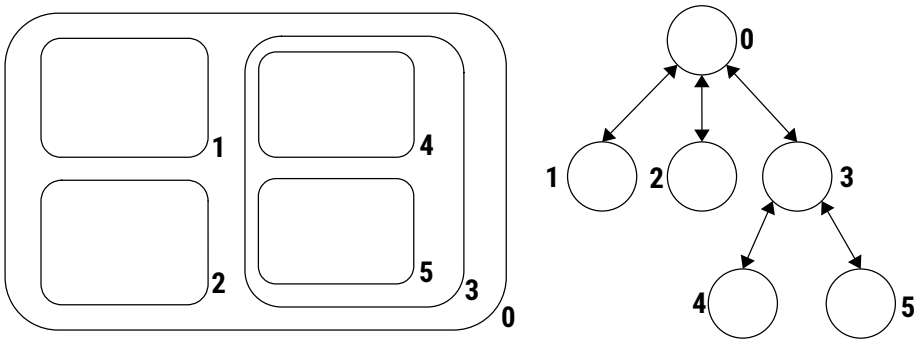

Figure 1: Cell-like Membrane Structure

In Figure 1 you can see two representations of a 6-membrane cell-like membrane structure. The left diagram shows the nesting of the membranes while right diagram shows the tree topology of the membranes. The outer most membrane, membrane 0, is the root node of the tree while membranes that do not contain internal membranes (membranes 1,2,4,5) are the leaf nodes of the tree.

Other P system variants use the more general graph membrane structure. P systems described as *tissue-like* or *neural-like* use graph membrane structures. In *tissue-like* P systems, the membranes are called *cells* and the connected cells form a '*tissue*'. In *neural-like* P systems, the membranes are called *neurons* and the connected neurons form a *neural network*. Figure 2 shows an example of a graph membrane structure with 7 membranes.

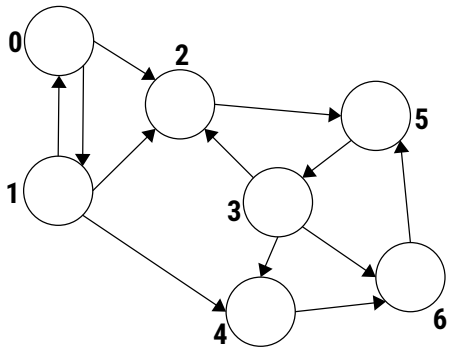

Figure 2: Graph Membrane Structure

2.1.2 Multisets of Object Symbols

As mentioned in Section 1, the objects of computation in P systems are abstract object symbols. Any P system has a fixed set of object symbols or an *alphabet*. We denote this alphabet of object symbols as V. A multiset over alphabet V is simply a set of object symbols from V in which multiple instances of the same object symbols are allowed. For example, given the alphabet $V = \{a, b, c, d\}$, the following are some multisets over V:

- $\{a, a, b\}$
- $\{a, a, b, c, c, d\}$
- $\{c, d, d\}$
- $\{a, c, c, c, d, d\}$
- $\{a, a, b, b, b, c, c, c, c, d, d, d, d, d\}$

Multisets can be written as strings. For example:

- $\{a, a, b\} = aab = a^2b$
- $\{a, a, b, c, c, d\} = aabccd = a^2bc^2d$
- $\{c, d, d\} = cdd = cd^2$
- $\{a, c, c, c, d, d\} = acccdd = ac^3d^2$
- $\{a, a, b, b, b, c, c, c, c, d, d, d, d, d\} = aabbbccccddddd = a^2b^3c^4d^5$

In a P system, regions enclosed by membranes store multisets of object symbols. Figure 3 shows two examples of membrane structures (cell-like and graph) with multisets of symbols in the regions.

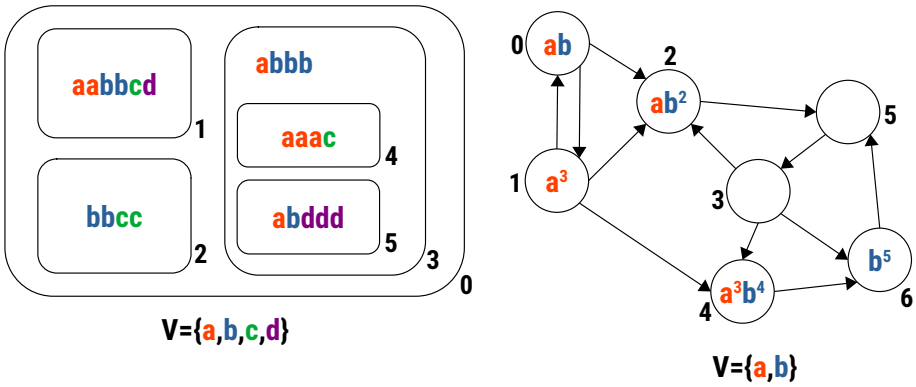

Figure 3: Multisets in Membrane Structures

The left diagram in Figure 3 shows the membrane structure from Figure 1 with multisets over alphabet $V = \{a, b, c, d\}$. Membrane 0 is empty (or contains the empty multiset). Membrane 1 contains the multiset *aabbcd*. Membrane 2 contains the multiset *bbcc*. Membrane 3 contains the multiset *abbb*, etc. The right digram in Figure 3 shows the membrane structure from Figure 2 with multisets over alphabet $V = \{a, b\}$. Cell 0 contains the multiset *ab*. Cell 1 contains the multiset a^3. Cell 2 contains the multiset ab^2. Cell 3 is empty, etc.

2.1.3 Rules of Different Types

A P system uses a set of rules to perform computation. Different P system variants use different types of rules. In general, a rule transforms the multisets inside the membranes and/or transforms the membrane structure itself. Static structure P systems only use rules that primarily transform the multisets inside the membranes and do not change the structure of the system. Dynamic structure P systems have rules that can add new membranes, delete existing membranes, and change the connections between membranes.

Rules for transforming multisets inside the membranes are based on *multiset rewriting rules*. A *multiset rewriting rule* is written as $M \rightarrow M'$ where M and M' are both multisets over the same alphabet. The rule rewrites the multiset M to the multiset M'.

For example: Alphabet is $V = \{a, b, c, d\}$

- `Rule 1`: $a^2b^2 \rightarrow cd$
  - Applying `Rule 1` once to multiset $a^4b^5c^3d^2$ will transform it to $a^2b^3c^4d^3$.
  - Applying `Rule 1` twice to multiset $a^4b^5c^3d^2$ will transform it to bc^5d^4.
- `Rule 2`: $ad \rightarrow ab^2c^3$
  - Applying `Rule 2` once to multiset $a^4b^5c^3d^2$ will transform it to $a^4b^7c^6d$.
  - Applying `Rule 2` twice to multiset $a^4b^5c^3d^2$ will transform it to $a^4b^9c^9$.

The first P system, known as *transition P system* [9], is a cell-like P system that uses modified multiset rewriting rules. Rules in transition P systems are multiset rewriting rules that use the membrane structures of the systems. A rule is associated with the region enclosed by a specific membrane. A rule will 'consume' a multiset of object symbols in its region then it can produce multisets of object symbols in the same (*here*) region, in the region outside (*out*) its membrane, and/or in the regions enclosed by membranes inside (*in*) the rule's own membrane. Figure 4 shows an example of a rule being applied.

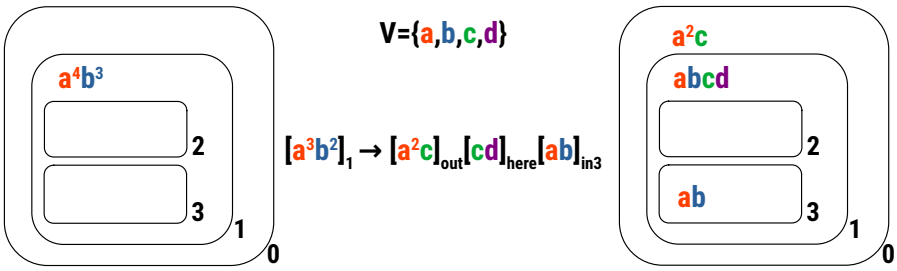

Figure 4: Application of Transition Rule $[a^3b^2]_1 \rightarrow [a^2c]_{out}[cd]_{here}[ab]_{in_3}$

The left diagram on Figure 4 shows the *configuration* of the system before rule application. Membrane 1 contains the multiset a^4b^3 while the other membranes are empty. The rule $[a^3b^2]_1 \rightarrow [a^2c]_{out}[cd]_{here}[ab]_{in_3}$ is associated with the region enclosed by membrane 1. The rule consumes a^3b^2 from membrane 1, produces a^2c outside membrane 1 (in membrane 0), and produces ab inside membrane 3. The right diagram on Figure 4 shows the configuration of the system after the rule is applied.

It can also be specified in a rule in a transition P system that the membrane associated with the rule be *dissolved* after the rule was applied. Figure 5 shows how a rule can *dissolve* a membrane. Rule $[a^2b^2]_1 \rightarrow [ab]_{in_2}[cd]_{in_3}\delta$ consumes multiset a^2b^2 from membrane 1, produces multiset ab in membrane 2, produces multiset cd in membrane 3, and *dissolves* membrane 1 (this is specified by δ at the end of the rule). The remaining multiset in membrane 1 will go to its parent membrane, membrane 0.

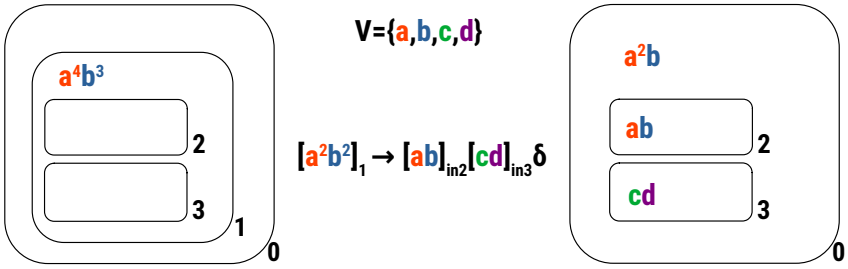

Figure 5: Application of Transition Rule $[a^2b^2]_1 \rightarrow [ab]_{in_2}[cd]_{in_3}\delta$

Tissue P systems' [6] rules are very similar to rules in transition P systems. The difference is that tissue P systems' membrane structures form graphs that do not have directionality or orientation unlike the tree membrane structures of transition P systems. When you are in the node of a tree in the membrane structure of a transition P system, the rule can send produced multisets *out* in the direction of the parent node, in the same node (*here*), or in the direction of the children nodes (*in*). In tissue P systems, a rule can send produced multisets in the same node (*here*) or to the adjacent nodes (*out*). Aside from this rule convention, rules in tissue P systems also have multiple different semantic meanings.

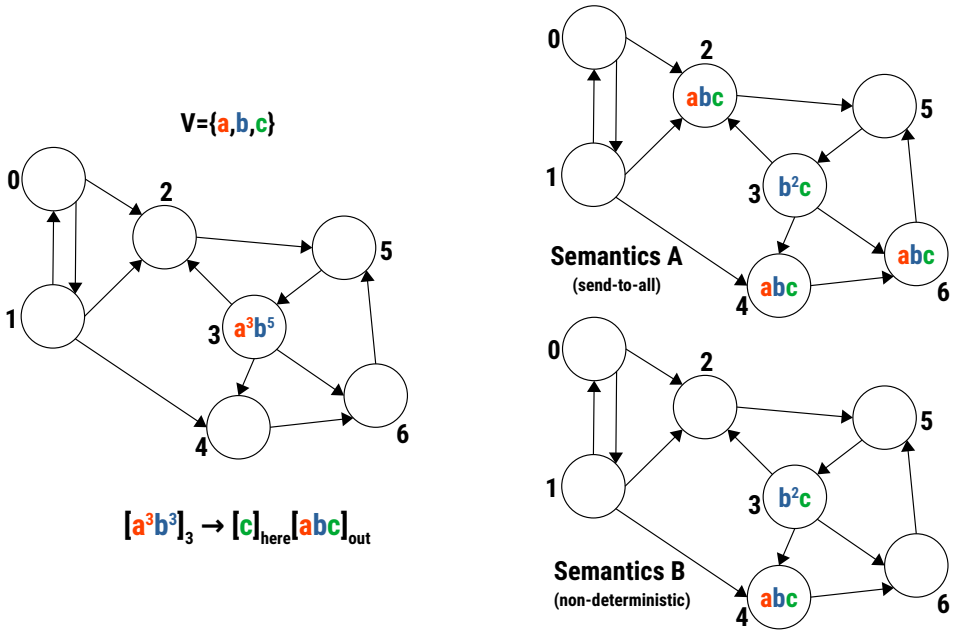

Figure 6: Application of Tissue P system Rule $[a^3b^3]_3 \rightarrow [c]_{here}[abc]_{out}$

Figure 6 shows two interpretations (Semantics A and B) for the rule $[a^3b^3]_3 \rightarrow [c]_{here}[abc]_{out}$. For semantics A, the rule consumes multiset a^3b^3 from cell 3, produces multiset c in cell 3, and sends multiset abc to all adjacent cells (cells 2,4,6). For semantics B, the rule consumes multiset a^3b^3 from cell 3, produces multiset c in cell 3, and *nondeterministically* sends multiset abc to one of the cells adjacent to cell 3 (either cell 2,4, or 6).

P systems *with active membranes* [7] introduced a type of rule that allows creation of new membranes. Figure 7 shows a rule of this type. The rule $[a]_y \rightarrow [b]_y[c]_y$ in Figure 7 has a very similar syntax to previous rules but its semantics is different. The rule consumes the object symbol a in membrane labeled y. The rule will then duplicate the membrane copying all the remaining object symbols to both membrane copies. Both membrane copies are also labeled y. In this P system variant, membrane labels are not membrane ids so multiple membranes can have the same label. In one of the copies, object symbol b is produce while in the other copy object symbol c is produced. Figure 7 shows the rule being applied twice in two steps.

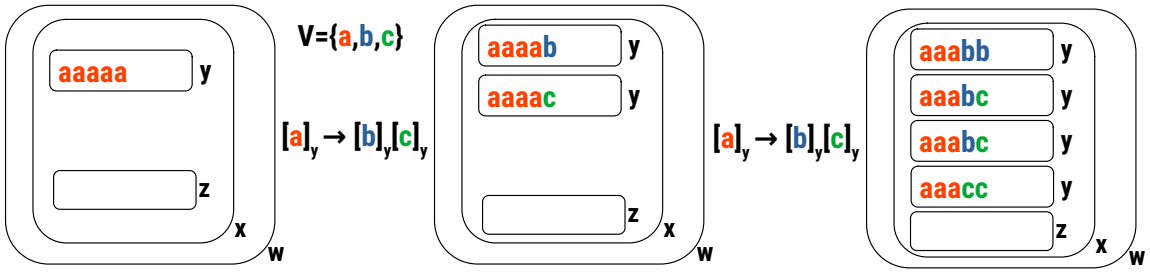

Figure 7: Application of the Active Membrane Rule $[a]_y \rightarrow [b]_y[c]_y$

The types of rules described above form a small set of samples of all possible types of rules. A large portion of the other types of rules that are not discussed are variations of the rules above. Some of them may have a slightly different semantics to the rules above.

2.1.4 Derivation Modes

P systems are parallel computing models. A P system can apply multiple rules at the same time. Different P systems can have different conventions that say which combinations of rules can be applied at the same time. This convention is known as the *derivation mode* of the system. A common derivation mode is called *maximally parallel* derivation mode. In maximally parallel mode, the system will apply a combination of rules such that you can no longer add instances of rules to this *maximal* combination of rules. For example, you have:

- `Rule 1:` $[ab]_1 \rightarrow [xy]_{here}$
- `Rule 2:` $[b^2]_1 \rightarrow [xyz]_{here}$

`Rule 1` consumes ab from membrane 1 while `Rule 2` consumes b^2 from membrane 1. If membrane 1 contains the multiset a^3b^6, one maximal combination of rules is 3×`Rule 1 + Rule 2`. This combination will consume the multiset a^3b^5. With the multiset b remaining no additional rules can be added to the combination making the combination maximal. Another maximal combination of rules is 3×`Rule 2`. This combination will consume the multiset b^6. With the multiset a^3 remaining no additional rules can be added. Another maximal combination is 2×`(Rule 1 + Rule 2)` consuming the multiset a^2b^6. In maximally parallel mode, only a maximal combination of rules can be applied. For example, the system can not apply the combination with one instance of `Rule 1`, if the system can still apply additional rules then it will add these rules to the combination. In this mode, the system *maximizes* the rules it can apply.

Minimally parallel mode is often used in *neural-like* P system variants. In the minimally parallel mode, at most one applicable rule is applied per membrane. The 'parallel' part is that multiple membranes can apply a rule at the same time. The 'minimal' part is that at most one rule can be applied per membrane.

There are other derivations modes. Some P system variants have the same rule types and membrane structure but only differ in the derivation modes they use.

2.2 Formal Framework

The *first formal framework (FF1)* [4] is an attempt to formally define procedural semantics for a large number of well-known variants of tissue P systems with static membrane structures. The *second formal framework (FF2)* [3] is a similar attempt for P systems with dynamic membrane structures. The *third formal framework (FF3)* [11] is an extension of FF1 with the purpose of expressing spiking neural P systems [5] in this extended framework.

The formal frameworks use and formalize notions/concepts common to most of their target P systems. The main notions available in all three versions of the framework are the following:

1. *configuration* of the system
2. *rules* of the system
3. *applicability* of a combination of rules
4. *allowed* combinations of rules
5. how to *apply* a combination of rules
6. *halting* condition for the system

2.2.1 Configuration

In FF1, the *configuration* of the system can be represented by a fixed-size vector of multisets $C = (u'_1, ..., u'_n)$, u'_i being the multiset in cell i. The configuration in FF1 does not contain information about the connections between cells. Instead, the information about the connection between cells is embedded in the rules themselves. For example, `Rule 1` is a rule that transfers objects from cell A to cell B therefore it requires a connection between cell A and cell B. Since FF1 deals only with static membrane structure, if there is no connection between cell A and cell B then `Rule 1` is useless in the system. In FF1, if a rule is in the system

and it requires certain connections between cells, then it is implied that those connections exist in the system. For a static system, you only need the vector of multisets (of the cells) in order to define the configuration of the system since the multisets are the only ones that can change. FF3 has the same configuration.

In FF2, the configuration $\mathcal{C}$ of the system is defined by the pair (L, ρ). $L = \{(i_1, l_1, w_1), ..., (i_n, l_n, w_n)\}$ is a list of labeled cells. The triple (i_j, l_j, w_j) is a cell with id i_j, label l_j, and multiset w_j. ρ is a set of cell id pairs that represent the connections between cells. L can change not only by changing the labels or contents of the cells in the list but also by adding/removing cells to/from the list. ρ can also change by adding or removing connections (pair of cell ids) between cells.

In some P system variants like P systems with active membranes [7], cells have additional properties that can change throughout the computation. For example, the cells in P systems with active membranes have a *charge/polarity* property that can either be + (positive), − (negative), or 0 (neutral). Such properties can be represented by adding extra symbols to the system's alphabet. Using this technique of adding extra symbols, FF1 and FF2's configurations can already represent most of these additional cell properties.

For the full technical details on how a configuration is defined in FF1 see Appendix A.4 and see Appendix B.1 for the details on how a configuration is defined in FF2. Configuration for FF3 is the same as configuration in FF1.

2.2.2 Interaction Rules

In all three FFs, the system's rules are referred to as *interaction rules*. In FF1, an *interaction rule* has the form $(X \to Y; P, Q)$. $X = (x_1, ..., x_n)$ is a vector of multisets where x_i represents the multiset that will be *consumed* in cell i if the rule is used. $Y = (y_1, ..., y_n)$ is a vector of multisets where y_j is the multiset that will be *produced* in cell j if the rule is used. $P = (p_1, ..., p_n)$ is a vector of sets of multisets where p_i contains different multisets (*permitting conditions*) that should be in cell i for the rule to be *eligible*. We will elaborate on the concept of *rule eligibility* in the Section 2.2.3. $Q = (q_1, ..., p_n)$ is a vector of sets of multisets where q_i contains different multisets (*forbidding condition*) that should not be in cell i for the rule to be eligible. The notation $X \to Y$ means the multisets in X are *rewritten* as the multisets in Y.

The interaction rules in FF3 (see Appendix C.1) has a very similar form to the interaction rule in FF1. The difference is that rules in FF3 use a vector of regular expressions $E = (E_1, ..., E_i, ..., E_n)$ as *permitting conditions* and there are not forbidding conditions. The semantics of the rules related to these regular expression is particular to spiking neural P system variants.

For FF1 and FF2, a form of multiset rewriting rule is sufficient. For FF2, changes in the membrane structure should also be capture by the rule. An interaction rule in FF2 has the form:

$$r = (Label, \rho, Perm, For, Rewrite, Label\text{-}Rename, Delete,$$

$$Delete\text{-}and\text{-}Move, Generate, Generate\text{-}and\text{-}Copy, Change\text{-}Relation)$$

Aside from multiset rewriting, a rule in P systems with dynamic membrane structures can also create new membranes, delete existing membranes, and change connections between membranes. This is the reason why a rule in FF2 has 11 components. The component *label* specifies the labels of the cells the rule is concerned with. The component ρ specifies the connection needed for the rule to be *eligible*. The components $Perm$ (permitting conditions), For (forbidding conditions), and $Rewrite$ (rewriting rule) are similar to the components of the interaction rule for FF1. They specify conditions for rewriting the multisets and the rewriting rule itself. The component $Label\text{-}Rename$ specifies the some cells new labels. The component $Delete$ specifies the cells (and contained multisets) to be deleted. The component $Delete\text{-}and\text{-}Move$ specifies the cells that will be deleted but whose contents will be moved to other cells. The component $Generate$ specifies the new cells (with new content) that will be created while the component $Generate\text{-}and\text{-}Copy$ specifies new cells that will be created whose content will be rewritten version of copies of multisets from existing cells. The component $Change\text{-}Relation$ specifies that changes to the connections between cells. Complete description for a FF2 interaction rule is in Appendix B.2.

2.2.3 Applicability of a Multiset of Rules

P systems are parallel models of computation. Multiple rules associated with different membranes can be applied in parallel. It is also possible for a single rule to be applied multiple times at the same time as shown

in Section 2.1.3. Taking the set of rules as the alphabet, the *combination* of rules in Section 2.1.3 actually refers to a multiset over the set of rules or simply a multiset of rules.

Informally, the notion of the *applicability* of a multiset of rules means that each rule in the multiset can be applied a specified number of times (defined by the multiset) at the same time. For example, applying the multiset ${r_1}^3{r_2}^5{r_5}^2$ means applying rule r_1 three times, rule r_2 five times, and rule r_5 two times. The multiset ${r_1}^3{r_2}^5{r_5}^2$ is applicable if applying all the rules will not cause issues like needing to consume more objects than the cells can provide. For example, if a rule r_1 consumes ab from cell 1 and cell 1 contains the multiset a^3b^{10}, multiset ${r_1}^4$ is not applicable since the multiset needs to consume a^4 from cell 1 but cell 1 only has a^3 to provide. The FFs formalized this notion of *applicability*.

The FFs starts with the notion of *eligibility* of a rule. In FF1, a rule $r : (X \to Y; P, Q)$ is eligible if three conditions hold: (1) multisets x_i in X are contained by the multisets u'_i in the cells, $x_i \subseteq u'_i$, (2) for each cell i, the set of permitting conditions p_i in P contains multisets that should be contained by multiset u'_i of cell i, $\forall p \in p_i,\ p \subseteq u'_i$, (3) for each cell i, the set of forbidding conditions q_i in Q contains multisets that should not be contained by multiset u'_i of cell i, $\forall q \in q_i,\ q \not\subseteq u'_i$. The first condition checks if there are enough objects in the cells for the rule to consume. The second condition checks for permitting conditions for each cell. The third conditions checks for forbidding conditions for for each cell.

FF2 has the same 3 conditions for rule eligibility. These 3 conditions can be check using FF2 rule's $Perm$, For, and $Rewrite$ components. Aside from these 3 conditions, for FF2 rule eligibility the necessary $labels$ and connections ρ should be available in the system's configuration $\mathcal{C} = (L, \rho)$.

The notion of rule eligibility can be extended to applicability of a multiset of rules by simply adding the condition that there are enough objects in the cells for the rules to consume (consumption condition). In FF2, two additional conditions are (1) the label renaming components of the rules should not conflict with each other (i.e. a conflict would be rule r_1 relabelling cell i with label l while r_2 relabelling cell i with label l'), (2) the $Change\text{-}Relation$ component of the rules should be commutative (i.e. rule r_1 applying its changes to the cell connections followed by rule r_2 applying its changes to the cell connections should result to the same final set of connections if the order of rule applications is reversed).

Full details for checking applicability of multiset of rules is in Appendix A.7 for FF1/FF3 and in Appendix B.4 for FF2.

2.2.4 Derivation Modes

If multiset of rules R is applicable, it is not necessarily the case that it is a candidate for the multiset that will be applied to the system. There is an additional consideration known as the *derivation mode*. The derivation mode of the system tells which of the applicable multisets of rules are candidate to be applied to the systems. Some of the common derivation modes are listed below:

- *Sequential:* This derivation mode requires the system to apply at most one rule once at a time hence the word *sequential*. This means the only candidate multisets for application are the multisets with one rule that is applied once.
- *Asynchronous:* This derivation mode is the most flexible by allowing any applicable multisets of rules to be a candidate.
- *Maximally parallel:* This derivation mode requires the candidate multiset to be *maximal* which means no additional rules can be added to the multiset without breaking the object consumption condition for applicability of the multiset of rules.
- *Minimally parallel:* This derivation mode requires the candidate multiset to have some minimal level of parallelism. For example, the set of rules is partitioned and each partition of rules is associated with a membrane. A candidate multiset of rules is minimally parallel if for each partition of rules at most one rule in the partition appears in the candidate multiset and is only applied once. In this example, at most one rule can be applied once per membrane.

The formal definitions of the derivation modes above are available in Appendix A.8. The appendix also contains additional derivation modes. These derivation modes are available to all FFs.

2.2.5 Halting Conditions

Halting condition specifies criteria to tell if the system is in the halt state. Three halting conditions are defined in FF1 (see Appendix A.10 for details).

- *Total Halting* is the most common halting condition. It states that the system halts if there is no applicable multiset of rules.
- *Adult Halting* says that the system is in the halt state if there are still applicable multisets of rules but applying any of these to the system's current configuration will simply result to the same configuration. The system is basically stuck applying those rules without changing its configuration.
- *Partial Halting* is where you consider the system in halt state if there are still applicable multisets of rule but none of the applicable multisets contain one rule from each of the partitions of rules.

2.2.6 Additional Notions in FF3

Spiking neural P systems and many of its variants are often used as transducers that accept input signals and produce output signals. The signals are known as *spike trains*. For this reason, the formal framework for spiking neural P systems formalized the notions of SN P system's input and output. FF3 (see Appendix C) defines the *Input* function and the *Output* function to formalized the input and output notions respectively.

The *Input* function's purpose is to produce a size n vector of multisets, n being the number of cells in the system. The *Input* function takes a time parameter which specifies the current computation step. The function will then return the input vector of multisets for that given time. At time t, if cell i is an input cell then the i^{th} element of the input vector is the multiset input for cell i at time t otherwise if cell i is not an input cell then the i^{th} element of the input vector will be empty (the empty multiset).

The *Output* function produces a size n vector of multisets that represent the output of the system. It takes time parameter t, the computation history $C(t) = C_0, ..., C_t$ up to time t, and the ids of the output cells c_{out} then it returns a size n vector of multisets that is similar to configuration C_t expect all the multisets which are not in the output cells (c_{out}) will be set as empty multisets.

2.2.7 Using the Formal Frameworks

In [10], both FF1 and FF2 were used to implement different features of P system variants. These features include *membrane thickness/polarity/labels*, *rule priorities*, and *membrane dissolution*. P system variants like *catalytic P systems*, *P systems with symport/antiport* [8], *P colonies*, and *probabilistic P systems* were also implemented as specific FF models. [10] shows that the formal framework can be used to model a wide variety of P systems with diverse features and semantics.

The FF can be used to compare different P system variants. By translating P system variants to their corresponding FF models, one can directly compare these FF models. The FF can be used a common language to analyze, compare and extended different P system variants.

3 Proposal

The research proposal has three aspects: (1) the reformulation of the formal framework, (2) the extension of the formal framework, and (3) the application of the formal framework. Figure 8 shows the three aspects of the proposal.

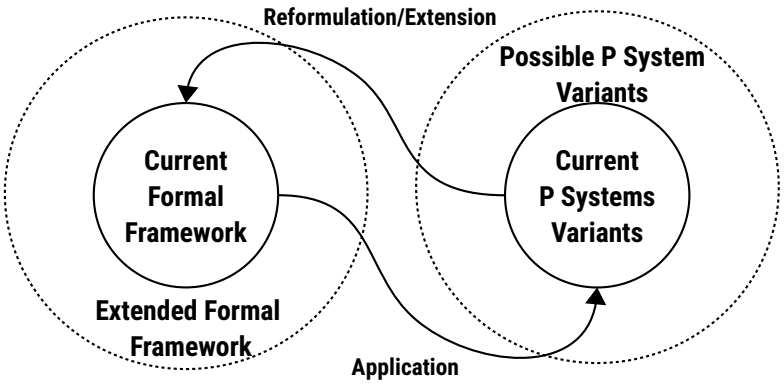

Figure 8: Research Directions between Formal Framework and P Systems

Reformulating the formal framework means changing the framework by changing the notions/concepts used or using different formalizations for these notions but not affecting the usefulness of the framework. Extending the framework means adding new notions and formalizations to extended the scope or usefulness of the framework. An extended framework can mean it can model more P system variants or that there are more notions in the framework that can provide more insights to the workings of existing 'supported' P system variants. Application of the framework means using the framework to analyze, compare, and/or extended existing P system variants.

3.1 Objectives

The following are the specific objectives of this proposal:

1. (Reformulation) Combine FF2 and FF3 into a single formal framework. This involves the addition of the *Input* and *Output* functions from FF3 to FF2. It also involves the use of regular multiset languages for *permitting* or *forbidding* conditions of the interaction rule. The purpose of this objective is to have a single formal framework (FF) that can be used of static or dynamic P systems.

2. (Reformulation) Reformulate the interaction rule in FF (from objective 1) in a *bottom-up* manner instead of the *top-down* approach of the FF. The rule in the FF (or specifically FF2) contains 11 components because it is trying to be the most general and unrestricted version of a rule such that the rule types from the P system variants are simply restricted versions of the more general FF interaction rule. We call this approach of finding the most general and unrestricted form of the rule as *top-down*. A rule can instead be defined as a 'combination' of simpler 'elementary' rules. We start from the *bottom* with this 'elementary' rules and use them to define a general rule which is a combination of these 'elementary' rules.

3. (Application) Perform a comprehensive survey of the different P system variants and use the FF to create the equivalent FF models of the different P system variants.

4. (Extension) While doing the comprehensive survey of P system variants, if there are variants that are difficult or impossible to create an FF model for, formalize the features of these variants and use them to extended the FF. Examples for such P system variants with features that are not (directly) represented are available in [1] and [2].

5. (Application) Create a simulator for FF models. Combining this simulator with the FF models of the P systems from the survey (objective 4) will result in a fairly general simulator than can simulate a wide variety of P systems.

3.2 Schedule

Figure 9 shows the research timeline. It shows the time allocated for each of the five objects and the additional task of writing the manuscript.

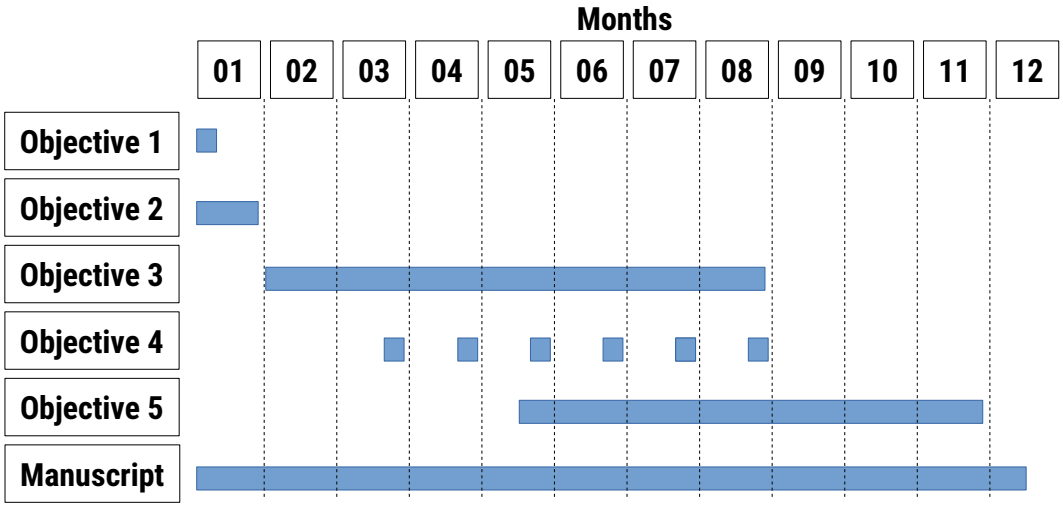

Figure 9: Research Schedule

References

[1] Artiom Alhazov, Sergiu Ivanov, and Yurii Rogozhin. Polymorphic p systems. In Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, pages 81–94, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[2] Fernando Arroyo, Angel Baranda, Juan Castellanos, and Gheorghe Paun. Membrane computing: The power of (rule) creation. *Journal of Universal Computer Science*, 8:369–381, 01 2002.

[3] Rudolf Freund, Ignacio Pérez-Hurtado, Agustín Riscos-Núñez, and Sergey Verlan. A formalization of membrane systems with dynamically evolving structures. *International Journal of Computer Mathematics*, 90(4):801–815, 2013.

[4] Rudolf Freund and Sergey Verlan. A formal framework for static (tissue) p systems. In George Eleftherakis, Petros Kefalas, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, pages 271–284, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[5] Mihai Ionescu, Gheorghe Păun, and Takashi Yokomori. Spiking neural p systems. *Fundam. Inf.*, 71(2,3):279–308, February 2006.

[6] Carlos Martín-Vide, Gheorghe Păun, Juan Pazos, and Alfonso Rodríguez-Patón. Tissue p systems. *Theoretical Computer Science*, 296(2):295 – 326, 2003. Machines, Computations and Universality.

[7] Andrei Păun. On P Systems with Active Membranes. In I. Antoniou, C. S. Calude, and M. J. Dinneen, editors, *Unconventional Models of Computation, UMC'2K*, pages 187–201, London, 2001. Springer London.

[8] Andrei Pâun and Gheorghe Pâun. The power of communication: P systems with symport/antiport. *New Generation Computing*, 20(3):295–305, Sep 2002.

[9] Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108 – 143, 2000.

[10] Sergey Verlan. Using the formal framework for p systems. In Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, pages 56–79, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[11] Sergey Verlan, Rudolf Freund, Artiom Alhazov, and Linqiang Pan. A formal framework for spiking neural p systems. In Gheorghe Păun, editor, *Proceedings of the 20th International Conference on Membrane Computing*. Bibliostar, 2019.

Appendices

A First Formal Framework (FF1)

A.1 Multisets

1. $V = \{a_1, ..., a_k\}$ - alphabet.
2. $M : V \to \mathbb{N}$ - multiset over V.
3. $M(a)$ - number of occurrence of symbol a in multiset M.
4. $|M| = \sum_{a \in V} M(a)$ - size of multiset M.
5. $a_1^{M(a_1)}...a_k^{M(a_k)}$ - string notation for multiset M.
6. $supp(M) = \{a \in V | M(a) \geq 1\}$ - *support* of multiset M.
7. $\langle V, \mathbb{N} \rangle$ - set of *finite* multiset over V.
8. $M : V \to \mathbb{N} \cup \{\infty\}$ - a multiset over V that allows infinite symbol occurrence.
9. $\langle V, \mathbb{N}_\infty \rangle$ - set of *possibly non-finite* multisets over V.
10. $X \leq Y$ or $X \subseteq Y$ - multiset *inclusion* relation or *submultiset* relation.
    - X, Y are multisets over V.
    - $X(a) \leq Y(a)$, $\forall a \in V$
11. $Z = X + Y$ or $Z = X \cup Y$ - multiset *union/addition*.
    - X, Y are multisets over V.
    - Z is the 'sum' or 'union' of multisets X, Y.
    - $Z(a) = X(a) + Y(a)$, $\forall a \in V$
12. $Z = X - Y$ - multiset *difference*.
    - X, Y are multiset over V.
    - $X \geq Y$.
    - $Z(a) = X(a) - Y(a)$, $\forall a \in V$
13. $X \leq Y$ - vector of multisets *inclusion* relation.
    - $X = (x_1, ..., x_m)$. X is a vector of multisets. x_i is a multiset over V.
    - $Y = (y_1, ..., y_m)$. Y is a vector of multisets. y_i is a multiset over V.
    - $x_i \leq y_i$, $\forall i, 1 \leq i \leq m$
14. $Z = X + Y$ or $Z = X \cup Y$ - vector of multisets *addition*.
    - $X = (x_1, ..., x_m)$. X is a vector of multisets. x_i is a multiset over V.
    - $Y = (y_1, ..., y_m)$. Y is a vector of multisets. y_i is a multiset over V.
    - $Z = (z_1, ..., z_m)$. Z is a vector of multisets. z_i is a multiset over V.
    - $z_i = x_i + y_i$, $\forall i, 1 \leq i \leq m$
15. $Z = X - Y$ - vector of multiset *difference*.
    - $X = (x_1, ..., x_m)$. X is a vector of multisets. x_i is a multiset over V.
    - $Y = (y_1, ..., y_m)$. Y is a vector of multisets. y_i is a multiset over V.
    - $Z = (z_1, ..., z_m)$. Z is a vector of multisets. z_i is a multiset over V.
    - $X \geq Y$.
    - $z_i = x_i - y_i$, $\forall i, 1 \leq i \leq m$

A.2 Network of Cells

1. $\Pi = (n, V, w, Inf, R)$ is the network of cells.
2. n is the number of cells.
3. V is the finite alphabet.
4. $w = (w_1, ..., w_n)$ is a vector of finite multisets over V.
   - $w_i \in \langle V, \mathbb{N} \rangle$, $\forall i$, $1 \leq i \leq n$
   - w_i is the multiset associated with cell i.
5. $Inf = (Inf_1, ..., Inf_n)$ - vector of subsets of V.
   - $Inf_i \subseteq V$.
   - Inf_i is the set of symbols occurring infinitely often in cell i.
6. R is finite set of *interactive rules*.

A.3 Interactive Rule

1. $(X \to Y; P, Q)$ is the form of an interactive rule.
2. $X = (x_1, ..., x_n)$ is a vector of multisets *consumed* by the rule.
   - x_i is a multiset over V, $\forall i$, $1 \leq i \leq n$.
3. $Y = (y_1, ..., y_n)$ is a vector of multisets *produced* by the rule.
   - y_i is a multiset over V, $\forall i$, $1 \leq i \leq n$.
4. $P = (p_1, ..., p_n)$ is a vector of sets of multisets.
   - p_i is a set of multiset over V, $\forall i$, $1 \leq i \leq n$.
   - p_i contains all the required ('*permitting*') multisets for cell i.
5. $Q = (q_1, ..., q_n)$ is a vector of sets of multisets.
   - q_i is a finite set of multiset over V, $\forall i$, $1 \leq i \leq n$.
   - q_i contains all the forbidden ('*forbidding*') multisets for cell i.
6. $X \to Y$ is the rewriting rule that rewrites of symbols in multisets x_i (in X) to symbols in multisets y_j (in Y).

A.4 Configurations

1. $C = (u'_1, ..., u'_n)$ is a vector of multisets known as a *configuration* of Π.
   - $u'_i \in \langle V, \mathbb{N}_\infty \rangle$, $\forall i$, $1 \leq i \leq n$.
2. $Inf^\infty = (Inf_1^\infty, ..., Inf_n^\infty)$ (NOTE)
   - Vector of infinite multisets.
   - $Inf_i^\infty = b_1^\infty \cdots b_k^\infty$, $b_i \in Inf_i$, $\forall i$, $1 \leq i \leq k = |Inf_i|$
3. $C^f = (u_1, ..., u_n)$
   - Finite part of the configuration C.
   - $u'_i = u_i + Inf_i^\infty$, $u_i \cap Inf_i = \emptyset$, $\forall i$, $1 \leq i \leq n$
   - $u'_i = u_i + Inf_i^\infty$, $supp(u_i) \cap Inf_i = \emptyset$, $\forall i$, $1 \leq i \leq n$ (NOTE)
   - $C_0^f = w = (w_1, ..., w_n)$ - finite parts of the initial configuration C.
   - $C_0 = w + Inf^\infty$ - full initial configuration of Π.

A.5 Eligibility of an Rule and Applicability of a Multiset of Rules

1. $eligibe(r, C) \Leftrightarrow \forall i,\ (1 \leq i \leq n),\ (x_i \subseteq u_i), (\forall p \in p_i, p \subseteq u_i), (\forall q \in q_i, q \not\subseteq u_i)$
   - Eligibility of rule r with respect to configuration C.
   - $C^f = (u_1, ..., u_n)$ - vector of finite multisets (u_i). Multisets in the cells.
   - $X = (x_1, ..., x_n)$ - vector of multisets (x_i). Multisets to be 'consumed'.
   - $P = (p_1, ..., p_n)$ - vector of multisets (p_i). Sets of required multisets for all cells.
   - $Q = (q_1, ..., q_n)$ - vector of multisets (q_i). Sets of required multisets for all cells.
2. $\exists j, x_j \cap (V - Inf_j) \neq \emptyset$
   - There is at least one multiset x_j in X such that least one symbol appearing in x_j has finite multiplicity.
3. $Eligible(\Pi, C) = \{r \in R | eligible(r, C)\}$
   - Set of all eligible rules with respect to configuration C.

A.6 Marking Algorithm

- $r_i : (X_i \to Y_i; P_i, Q_i)$ - interaction rule.
- $X_i = (x_{i,1}, ..., x_{i,n})$ - vector of multisets.
- $X'_i = (x'_{i,1}, ..., x'_{i,n})$ - vector of multisets.
- $x_{i,j} = x'_{i,j} + Inf_j^{\infty}$. $x'_{i,j} \cap Inf_j = \emptyset$
- $C^f = (v_1, ..., v_n)$ - vector of multisets. Finite part of the configuration.
- $R = \{r_1, ..., r_h\}$ - set of eligible rules r_i.
- $R' \in \langle R, \mathbb{N} \rangle$ - finite multiset of eligible rules.

1. $Mark_0(\Pi, C, R') = (\lambda, ..., \lambda)$. $i = 1$.
2. If $X'_i \leq (C^f - Mark_{i-1}(\Pi, C, R'))$
   - `TRUE`: $Mark_i(\Pi, C, R') = (C^f - Mark_{i-1}(\Pi, C, R')) - X'_i$
   - `FALSE`: return `false`;
3. If $i = k$
   - `TRUE`: return `true` and $Mark(\Pi, C, R') = Mark_k(\Pi, C, R')$
   - `FALSE`: $i = i + 1$. Go back to step 2.

A.7 Applicability of a Multiset of Rules

1. $appl(R', C) \Leftrightarrow$ marking algorithm returns `true` and $Mark(\Pi, C, R')$.
   - Applicability of multiset of rules R' with respect to configuration C.
2. $Appl(\Pi, C) = \{R' \in \langle R, \mathbb{N} \rangle \mid R = Eligible(\Pi, C), appl(R', C)\}$
   - Set of all applicable multisets of rules with respect to configuration C.
3. $Apply(\Pi, C, R') = C - Mark(\Pi, C, R') + \sum_{i=1}^{k} Y'_i$
   - A new configuration when rules in R' are applied.

A.8 Derivation Modes

1. $Appl(\Pi, C, \delta) \subseteq Appl(\Pi, C)$
   - Set of applicable multisets of rules in δ-mode.
2. $Appl(\Pi, C, asyn) = Appl(\Pi, C)$
   - $\delta = asyn$. *Asynchronous* mode.
3. $Appl(\Pi, C, sequ) = \{R' \in Appl(\Pi, C) \mid |R'| = 1\}$
   - $\delta = sequ$. *Sequential* mode.
   - Only one rule is applicable per step.
4. $Appl(\Pi, C, max) = \{R' \in Appl(\Pi, C) \mid \nexists R'' \in Appl(\Pi, C),\ R' \subsetneq R''\}$
   - $\delta = max$. *Maximally parallel* mode.
   - The multiset of rule R' is maximal if by adding any (multiplicity of) rules will make the multiset of rules no longer applicable.
5. $Appl(\Pi, C, min) = \{R' \in Appl(\Pi, C) \mid \ \nexists R'' \in Appl(\Pi, C),\ R' \subseteq R'', \exists j, (R'' - R') \cap R_j \neq \emptyset, R' \cap R_j = \emptyset\}$
   - $\delta = min$. *Minimally parallel* mode.
   - There should be at least one rule in R' from each partition $R_j, \forall i, 1 \leq i \leq h$.
   - $R = R_1 \cup R_2 \cup \cdots \cup R_h$. Rule set R is be partitioned.
   - $R' \subseteq R''$. R'' 'extends' R'.
6. $\delta \in \{asyn, sequ, max, min\}$
   - Basic derivation modes.
7. $Appl(\Pi, C, max_{rule}\delta) = \{R' \in Appl(\Pi, C, \delta) \mid \ \nexists R'' \in Appl(\Pi, C, \delta)\ |R''| > |R'|\}$
   - *Maximum rules* δ-mode.
   - Multiset of rules in δ-mode with maximum number of rule applications.
8. $Appl(\Pi, C, max_{set}\delta) = \{R' \in Appl(\Pi, C, \delta) \mid \ \nexists R'' \in Appl(\Pi, C, \delta)\ ||R''|| > ||R'||\}$
   - *Maximum set* δ-mode.
   - Multiset of rules in δ-mode with maximum set of rules.
9. $Appl(\Pi, C, all_{set}\delta) = \{R' \in Appl(\Pi, C, \delta) \mid \forall j, 1 \leq j \leq h, (R_j \cap \bigcup_{X \in Appl(\Pi, C)} X \neq \emptyset) \rightarrow (R_j \cap R' \neq \emptyset)\}$
   - *All set* δ-mode.
   - Multiset of rules in δ-mode in which all partitions of the rule set contributes to R'.

A.9 Transitions

1. $C \Rightarrow_{(\Pi,\Delta)} C' \Leftrightarrow \exists R' \in Appl(\Pi, C, \Delta),\ C' = Apply(\Pi, C, R')$
   - Transition in Δ-mode.
2. $C \Rightarrow^*_{(\Pi,\Delta)} C'$
   - Transitive closure and reflexive nature of the transition relation $\Rightarrow_{(\Pi,\Delta)}$
3. $accessible(C, \Pi, \Delta) \Leftrightarrow C_0 \Rightarrow_{(\Pi,\Delta)} C$

- Accessibility of configuration C in Δ-mode.
- C_0 is the initial configuration.

4. $Acc(\Pi, \Delta) = \{C \mid accessible(C, \Pi, \Delta)\}$
   - Set of all accessible configurations using Δ-mode derivation.
5. $Deterministic(\Pi, \Delta) \Leftrightarrow \forall C \in Acc(\Pi, \Delta), Appl(\Pi, C, \Delta)| \leq 1$
   - System Π is deterministic with respect to Δ-mode derivation.

A.10 Halting Conditions

1. $H(\Pi, \Delta) = \{C' \in Acc(\Pi, \Delta) \mid Appl(\Pi, C', \Delta) = \emptyset\}$
   - Set of *total halting* configurations.
   - Accessible configurations where there are no applicable multisets of rules.
2. $A(\Pi, \Delta) = \{C' \in Acc(\Pi, \Delta) \mid Appl(\Pi, C', \Delta) \neq \emptyset, \forall R' \in Appl(\Pi, C', \Delta), Apply(\Pi, C', R') = C'\}$
   - Set of *adult halting* configurations.
   - Accessible configurations where for all applicable rule R' applying R' to configuration C' result to C'.
3. $h(\Pi, \Delta) = \{C' \in Acc(\Pi, \Delta) \mid \nexists R' \in Appl(\Pi, C', \Delta), \forall i, 1 \leq i \leq h, R' \cap R_j \neq \emptyset\}$
   - Set of *partial halting* configurations.
   - Accessible configurations there are no multiset R' of applicable rules such that R' contains rules from all partions R_j.

B Second Formal Framework (FF2)

B.1 Configurations

1. $C = \{(i_1, w_1), ..., (i_n, w_n)\}$
   - A *basic configuration* C is a list of pairs (i_j, w_j). Pairs (i_j, w_j) are called *cells*.
   - $i_j \in \mathbb{N}$ is the *id* of cell j. The id is unique per cell. i.e. $j \neq k \rightarrow i_j \neq i_k$. $[\forall j, k, 1 \leq j, k, \leq n]$
   - $w_j \in O^*$ is the *content* of some cell j. $[\forall j, 1 \leq j \leq n]$
   - $size(C)$ denotes the size of basic configuration C.
   - $id(x)$ is a bijective function takes a cell (x) and returns the cell's id.
   - $id(j) = j$. For simplicity, the cell's position in the list is the same as its id.
2. $\mathbb{C} = \{C \mid size(C) > 0\}$
   - $\mathbb{C}$ is the set of all basic configurations with size greater than 0.
3. $\mathcal{C} = (L, \rho) = (\{(i_1, l_1, w_1), ..., (i_n, l_n, w_n)\}, \rho)$
   - $\mathcal{C}$ is a configuration (where C is a *basic configuration*).
   - $L = (i_1, l_1, w_1)...(i_n, l_n, w_n)$ is a list of *labeled cells*.
   - $l_j \in Lab$ is the *label* of cell j where Lab is a set of *labels*. $[\forall j, 1 \leq j \leq n]$
   - Two cells can have the same label but not the same *id*.
   - $\rho \subseteq \mathbb{N} \times \mathbb{N}$ is a relation the represents the connection between cells.

- $\mathcal{C}_m = L$ and $\mathcal{C}_\rho = \rho$.
- $\overline{\mathcal{C}}_m \in \mathbb{C}$ is the projection of $\mathcal{C}_m$ as basic (unlabeled) configuration.

4. $\mathfrak{C} = \{\mathcal{C} \mid \mathcal{C} = (L, \rho)\}$
   - $\mathfrak{C}$ is the set of all possible configurations.

B.2 Components of a Rule

1. $r = (Labels, \rho, Perm, For, Rewrite, Label\text{-}Rename, Delete, Delete\text{-}and\text{-}Move, Generate, Generate\text{-}and\text{-}Copy, Change\text{-}Relation)$
   - r is a rule.
2. $Labels(r) = (l_1, ..., l_k) \in Lab^k$
   - $Label(r)$ is a list of cell labels that introduces k *virtual cells*.
   - k labels for virtual cells.
   - $\mathbb{N}_k = \{1, ..., k\}$.
   - $\mathbb{C}_k \subseteq \mathbb{C}$ is the set of basic configurations, where the only allowed cell ids are in $\mathbb{N}_k$.
3. $\rho(r) \subseteq \mathbb{N}_k \times \mathbb{N}_k$
   - $\rho(r)$ is a relation on virtual cells (e.g. indicating parent relation).
4. $Perm(r) \subseteq \mathbb{C}_k$
   - $Perm(r)$ defines the permitting condition.
5. $For(r) \subseteq \mathbb{C}_k$
   - $For(r)$ defines the forbidding condition.
6. $Rewrite(r) = U \to V$
   - $U, V \in \mathbb{C}_k$
   - $Rewrite(r)$ is a general rewriting rule, rewriting a finite basic configuration U to another finite basic configuration V.
7. $Label\text{-}Rename(r) \in (\mathbb{N}_k \times Lab)^*$
   - $Label\text{-}Rename(r)$ allows us to specify new labels for the cells whose index (or id) appears in the list.
8. $Delete(r) \in \mathbb{N}_k^*$
   - $Delete(r)$ gives the indices of the virtual cells to be deleted.
9. $Delete\text{-}Move(r) \in (\mathbb{N}_k \times \mathbb{N}_k)^*$
   - $Delete\text{-}Move(r)$ is a list of pairs of indices, e.g. (i, j), that indicates that virtual cell i should be deleted and its content should be moved to virtual cell j.
10. $Generate(r) \in (\mathbb{N}' \times Lab \times O^*)^*$
    - $Generate(r)$ is a list of triples where each triple contains a *primed* index, a label, and a multiset i.e. (j', h, u).
    - This component introduces new cells (with *primed* index) to be created when the rule is applied.
11. $Generate\text{-}Copy(r) \in (\mathbb{N}' \times Lab \times \mathbb{N} \times (O^* \times O^*))^*$

- *Generate-Copy*(r) is a list of quadruples where each quadruple contains a *primed* index, a label, an index, and a rewriting rule i.e. $(j', h, i, u \to v)$.
- Each tuple $(j', h, i, u \to v)$ means that a new cell should be created, having a new id j' and the label h, by first copying every object inside cell i and then applying the rewriting rule to contents of the new cell.

12. *Change-Relation*

- *Change-Relation* is a *graph transducer* that updates the relation ρ.
- This transducer should be recursive and it can only add and remove edges.

B.3 Eligibility of an Instance Vector

1. $\boldsymbol{i} = (i_1, ..., i_n) \in \mathbb{N}^n$

- An *instance* i is a vector of indices.

2. $C\langle \boldsymbol{i} \rangle = (\boldsymbol{i}|_{j_1}, w_1)...(\boldsymbol{i}|_{j_k}, w_k) = (i_{j_1}, w_1)...(i_{j_k}, w_k)$

- $C\langle \boldsymbol{i} \rangle$ is the *instantiation* of basic configuration C by instance $\boldsymbol{i}$.
- $C = (j_1, w_1)...(j_n, w_k) \in \mathbb{C}$.
- The set of cell ids$\{j_1, ..., j_k\}$ may not be equal to $\{1, ...k\}$.
- $size(C) = k \leq |\boldsymbol{i}|$.
- Some cell ids (j_q) may be outside the range $[1, ..., k = size(C)]$ but they are always in the range $[1, ..., n]$.
- The id of cell q is j_q. $\boldsymbol{i}|_{j_q}$ denotes the j_q-th index in vector/instance $\boldsymbol{i}$ which is i_{j_q}.

3. $r\langle \boldsymbol{i} \rangle$

- $r\langle \boldsymbol{i} \rangle$ is the instantiation of rule r using instance $\boldsymbol{i}$.
- Replace all virtual ids k by $\boldsymbol{i}|_k$ in all components of rule r involving virtual ids (expect for parts *Labels*(r) and *Generate*(r)).

4. *Eligible*$(\boldsymbol{i}, r, \mathcal{C})$

- *Eligible*$(\boldsymbol{i}, r, \mathcal{C})$ is the *eligibility* of index vector $\boldsymbol{i}$ for instantiation for a rule r in a configuration $\mathcal{C}$.
- $\boldsymbol{i} = (i_1, ..., i_k) \in \mathbb{N}^k$.
- r is a rule.
- $\mathcal{C} = (L, \rho)$.
- *Eligible*$(\boldsymbol{i}, r, \mathcal{C})$ is true if the following conditions are true:
  - Indices i_j in index vector $\boldsymbol{i}$ are different from each other.
  - $\forall j, 1 \leq j \leq k, 1 \leq i_j \leq size(\mathcal{C}), lab(i_j) = l_j$ where $Labels(r) = (l_1, ..., l_k)$.
  - $(j, m) \in \rho(r) \Rightarrow (i_j, i_m) \in \mathcal{C}_\rho$
- $\mathcal{I}_\mathcal{C}(r) = \{\boldsymbol{i} \mid Eligible(\boldsymbol{i}, r, \mathcal{C})$ is true$\}$.

B.4 Applicability of a Multiset of Rules

1. $\mathit{Applicable}(R, \mathcal{C})$
   - $\mathit{Applicable}(R, \mathcal{C})$ is the set of *applicable* multisets of instantiated rules from R.
   - $R = \{r_1, .., r_n\}$ is a multiset of rules. $r_i \neq r_j \in R$ are not necessarily different rules.
   - $\mathcal{C} = (L, \rho)$ is a configuration.
   - $\mathcal{I}_{\mathcal{C}}(r_i) = \{v_{i,1}, ..., v_{i,k_i}\},\ \forall i, 1 \leq i \leq n$ where v_{i,j_i} is an instance vector.
   - $\{(r_1, v_{1,j_1}), ..., (r_n, v_{n,j_n})\}$ is a multiset of instantiated rules.
   - $\{(r_1, v_{1,j_1}), ..., (r_n, v_{n,j_n})\} \in \mathit{Applicable}(R, \mathcal{C})$ if following 5 conditions are true:
2. $\forall i, 1 \leq i \leq n, \forall P, P \in (\mathit{Perm}(r_i) \cup \mathit{DPerm}(r_i)),\ P\langle v_{i,j_i}\rangle \subseteq \overline{\mathcal{C}}_m$
   - $\mathit{DPerm}(r) = \{(i, u) \mid (j', h, i, u \to v) \in \mathit{Generate\text{-}and\text{-}Copy}(r)\}$
3. $\forall i, 1 \leq i \leq n, \forall Q, Q \in \mathit{For}(r_i), Q\langle v_{i,j_i}\rangle \not\subseteq \overline{\mathcal{C}}_m$
4. $\bigcup_{i=1}^{n} \mathit{Bound}(r\langle v_{i,j_i}\rangle) \subseteq \overline{\mathcal{C}}_m$
   - $\mathit{Bound}(r)$ is the left-hand side of the rewriting rule in the component $\mathit{Rewrite}(r)$.
   - $\mathit{Rewrite}(r) = U \to V$
5. $\forall i, j, k,\ (s, l_1) \in \mathit{Label\text{-}Rename}(r_i\langle v_{i,j_i}\rangle)$ & $(s, l_2) \in \mathit{Label\text{-}Rename}(r_k\langle v_{k,j_k}\rangle) \to l_1 = l_2$
6. Consecutive applications of $\mathit{Change\text{-}Relation}(r_i)$ and $\mathit{Change\text{-}Relation}(r_j)$ yields the same results regardless of order of applications for $1 \leq i, j \leq n$.
7. $\mathit{Applicable}(\Pi, \mathcal{C}) = \bigcup_{\mathit{Applicable}(R,\mathcal{C}) \neq \emptyset} \mathit{Applicable}(R, \mathcal{C})$

B.5 Applying a Multiset of Rules

1. $L_1 = \{(i_1, l_1, w'_1)...(i_n, l_n, w'_n)\}$
   - $$w'_j = w_j - \left(\bigcup_{(r_k, v_k) \in RI} U_k|_j \right) + \left(\bigcup_{(r_k, v_k) \in RI} V_k|_j \right)$$
   - $RI = \{(r_1, v_1)...(r_n, v_n)\}$
   - $\mathcal{C} = (\{(i_1, l_1, w_1)...(i_n, l_n, w_n)\}, \rho)$
   - $\mathit{Rewrite}(r_k\langle v_k\rangle) = U_k \to V_k$
   - $U_k, V_k \in \mathbb{C}_k$
2. $L_2 = \{(i_1, l'_1, w'_1)...(i_n, l'_n, w'_n)\}$
   - $$l'_j = \begin{cases} e_s, & \text{if } \exists (r_k, v_k) \in RI \text{ such that } (j, e_s) \in \mathit{Label\text{-}Rename}(r_k\langle v_k\rangle) \\ l_j, & \text{otherwise} \end{cases}$$
3. $L_c = L_c(r_1) \cdot \cdots \cdot L_c(r_n)$
   - $L_c(r_k) = \{(m_1, h_1, u_1) \cdots (m_t, h_t, u_t)\}$
   - $\mathit{Generate}(r_k) = \{(1', h_1, u_1) \cdots (t', h_t, u_t)\}$
4. $L'_c = L'_c(r_1) \cdot \cdots \cdot L'_c(r_n)$
   - $L'_c(r_k) = \{(m_{t+1}, e_1, w'_{n_1} - u_1 + v_1)...(m_{t+s}, e_s, w'_{n_s} - u_s + v_s)\}$

- *Generate-and-Copy*$(r_k) = \{(1', e_1, n_1, u_1 \to v_1)...(s', e_s, n_s, u_s \to v_s)\}$
- $(i_j, l'_j, w'_j) \in L_2$

5. $L_3 = L_2 \cdot L_c \cdot L'_c$
6. $L_4 = \{(i_1, l'_1, w''_1) \cdots (i_n, l'_n, w''_n)\}$
   - $$w''_j = w'_j + \left(\bigcup_{\text{last}(i_k)=i_j} w'_k \right)$$
   - $(i_j, l'_j, w'_j) \in L_3$
   - $\mathbf{p} = (p_1, ..., p_j, ..., p_n)$ - vector of "destination" cell ids for multisets from cells that may be deleted.
   - $p_j = p(i_j)$ - destination cell for multiset from cell i_j.
   - $$p(i_j) = \begin{cases} *, & \text{if } \exists (r_k, v_k) \in RI \text{ such that } i_j \in \mathit{Delete}(r_k\langle v_k \rangle) \\ e, & \text{if } \exists (r_k, v_k) \in RI \text{ such that } (i_j, e) \in \mathit{Delete\text{-}and\text{-}Move}(r_k\langle v_k \rangle) \\ i_j, & \text{otherwise} \end{cases}$$
   - *Delete*$(r) \in \mathbb{N}_k^*$ is the list of cell ids to be deleted.
   - *Delete-and-Move*$(r) \in (\mathbb{N}_k \times \mathbb{N}_k)^*$. $(i, j) \in$ *Delete-and-Move*(r) means to delete cell i and move its multiset to cell j.
   - If $p_k = p(i_k) = i_k$, then cell i_k will not be deleted.
   - If $p_k = p(i_k) \neq i_k$ (either $p(i_k) = *$ or $p(i_k) = e$), then cell i_k will be deleted.
   - If $p_k = i_k$, then there is a sequence $x_1, ..., x_{j-1}, ..., x_m$ of cell ids where:
     - $x_1 = p_k$
     - $x_j = p(x_{j-1})$ for $2 \leq j \leq m$
     - $x_m = z$

     There is a chain of "delete-and-move" and z is the last cell id to not be deleted or to be deleted and not moved.
   - $last(i_j) = z$
7. $L_5 = \{(i_1, l'_1, w''_1)...(i_{n_1}, l'_{n_1}, w''_{n_1})\}$ where $(i_j, l'_j, w''_j) \in L_4$ and $p_j = i_j$.
   - L_5 contains L_4 cells with the deleted cells removed.
8. *Apply*$(RI, \mathcal{C}) = (L_2, \mathcal{C}'_\rho)$
   - $\mathcal{C}'_\rho$ is the updated 'parent' relation after *CREATE-NODES*, *DELETE-NODES*, and *Change-Relation*$(r_k\langle v_k \rangle)$ have been computed for all $(r_k, v_k) \in RI$ on $\mathcal{C}_\rho$.

C Additional Notions in the Third Formal Framework (FF3)

1. $X \to Y; E$ is a form of the interaction rule.
   - $X = (x_1, ..., x_n)$ is a vector of multisets that the rule will consume. x_i is the multiset that will be consumed from cell i.
   - $Y = (y_1, ..., y_n)$ is a vector of multisets that the rule will produce. y_i is the multiset that will be produced at cell i.
   - $E = (E_1, ..., E_n)$ is a vector of regular expressions over the alphabet V of the system. E_i is a regular expression over V. It defines the *permitting condition* related to cell i. $L(E_i)$ is the language defined by regular expression E_i while $L^\circ(E_i)$ is the regular multiset language define by the regular expression E_i.

2. $\Pi = (n, V, w, c_i n, c_o u, Inf, R)$ is the network of cells. It has the same components as the network of cells in FF1 (see Appendix A.4) except for the additional c_{in} and c_{out} components specifying the input cells and the output cells respectively.

3. $Input : \mathbb{N} \times 2^V \to V^n$ (probably should be $Input : \mathbb{N} \times 2^{\mathbb{N}} \to \langle V, \mathbb{N} \rangle^n$) is an *input function*.

   - $Input(t, c) \in \langle V, \mathbb{N} \rangle^n$ is a size n vector of multisets that represents the input to the system which can be added to the configuration of the system. $t \in \mathbb{N}$ is the time (step) while $c \in 2^{\mathbb{N}}$ is the set of 'ids' of the input cells.

4. $\pi(S, X) : 2^{\mathbb{N}} \times (\langle V, \mathbb{N} \rangle)^n \to (\langle V, \mathbb{N} \rangle)^n$ (original: $2^{\mathbb{N}} \times \langle V, \mathbb{N} \rangle \to \langle V, \mathbb{N} \rangle$) is a *keeping function*.

   - π takes a set of cell ids $S \in 2^{\mathbb{N}}$ and a vector of multiset $X = (X_1, ..., X_i, ..., X_n)$ and zero-out all the multisets X_i if $i \notin S$.
   - $\pi(S, X) = (X'_1, ..., X'_i, ..., X'_n)$ where $X'_i = X_i$ if $i \in S$ otherwise $X'_i = 0$ if $i \notin S$.

5. $Output : \mathbb{N} \times NC \times (V^n)^* \times 2^V \times \Delta \to S$ is an *output function*.

   - The output function $Output$ takes the following as parameters: (1) the time $t \in \mathbb{N}$, (2) a system $\Pi \in NC$ where NC is the set of *network of cells*, (3) a configuration history $C(t) \in (\langle V, N \rangle^n)^*$ (originally $(V^n)^*$) where $C(t) = C_0, ..., C_t$, (4) a set of ids of the output cells $c_{out} \in 2^{\mathbb{N}}$ (originally 2^V), a derivation mode $\delta \in \Delta$.
   - The output function then returns $Output(t, \Pi, C(t), c_{out}, \delta)$ which is a size n vector of multisets that represent the output of the system at time t. The $Output$ function uses the keeping function π.
   - $Output(t, \Pi, C(t), c_{out}, \delta) = \pi(c_{out}, C_t)$ if $Applicable(\Pi, C_t, \delta) = \emptyset$ (if there are not more applicable multisets of rules). Otherwise, $Output(t, \Pi, C(t), c_{out}, \delta) = \emptyset$.