Christiansen Grammar for Some P Systems

Alfonso Ortega de la Puente, Rafael Núñez Hervás,
Marina de la Cruz Echeandía, Manuel Alfonseca

Escuela Politécnica Superior
Departamento de Ingeniería Informática
Universidad Autónoma de Madrid
Ctra. de Colmenar km 15, 28049 Madrid, Spain
E-mail: {alfonso.ortega, r.nunnez, marina.cruz, Manuel.Alfonseca}@uam.es

Summary. The main goal of this work is to formally describe P systems. This is a necessary step to subsequently apply Christiansen grammar evolution (an evolutionary tool developed by the authors) for automatic designing of P systems. Their complex structure suggests us two decisions: to restrict our study to a subset of P systems that ease the representation while keeping a suitable complexity and to select a powerful enough formal tool. Our work is restricted to a kind of P system that can simulate any logical function by means of delay symbols and two mobile catalysts. Like in general P systems, some components of these "logical" P systems depend on other components (for example, the number of axioms and regions and the set of possible indexes for the symbols in their rules depend on the membrane structure). So, a formal representation able to handle context dependent constructions is needed. Our work uses Christiansen grammars to describe P systems.

1 Introduction

1.1 P systems

P Systems [7] are a kind of parallel and distributed computation model inspired by the way in which cells process chemical compounds, energy and information. They can be briefly described as a collection of regions whose external membrane is called *skin*.

Each region in the model has four main components: a membrane that isolates it from the rest of the P system, several symbols that are contained by it, production rules that describe the way in which these symbols can change and a set of membranes also contained by it.

The structure of the symbols contained by a membrane is called a *multiset*. A *multiset* is a *set* in which *multiple copies* of each symbol are allowed. At each moment, every region applies all its applicable production rules as much as possible.

An order relationship, called *priority* ([8, 9]) can be used to solve the collisions between rules applicable at the same moment in the same region.

Symbols can move across the membranes. That is, after applying a rule in a region, some of its symbols can exit from the region or enter one of its directly included regions.

Each kind of P system has a way to communicate the result of its computations. In this paper, a given region among those in the P system, called the *output region*, is used for this purpose.

Formal definition

P systems can be formally defined as follows.

A P system of degree $m \geq 1$ is the $(2m + 4)$-fold:

$$\Pi = (O, K, \mu, \omega_0, \ldots, \omega_{m-1}, (R_0, \rho_0), \ldots, (R_{m-1}, \rho_{m-1}), i_o),$$

where:

- O is an alphabet. Its elements are called *objects*.
- $K \subseteq O$ is a distinguished subset of O, called the set of catalysts. These symbols remain unchanged when they appear in a rule.
- μ is a membrane structure with m membranes (usually labelled from 0 to $m - 1$),
- ω_i, $0 \leq i < m$, specifies the initial multiset of the i^{th} region.
- R_i, $0 \leq i < m$, is the finite set of evolution rules for the i^{th} region.
- ρ_i is a partial order relation over R_i (the *priority* relation). Each evolution rule joins two multisets of symbols. Subscripts $\{here,\ out,\ in_j\}$ are used to signal the motion of the symbols across the membranes; *here* means no movement, *out* specifies that the symbol exits its region, and in_j means that the symbol goes into the j^{th} region. This region must belong to the current membrane. Catalysts appear at most once in the left hand side of a rule and once in the right hand side. Catalysts can also be subscripted.
- i_o is the name of the output membrane. If $i_o = -1$, then the output is collected from the environment.

1.2 P systems to compute logical functions

Reference [1] describes several ways in which P systems can simulate logical circuits. They follow two steps: simulating a complete set of logical operands (such as *and*, *or*, *not*) and designing a way to combine logical functions. One of these approaches simulates the basic logical operands by means of P systems with binary alphabets ($\{0, 1\}$) extended by two mobile catalysts and some delay symbols. This is the approach used in this paper. Reference [1] contains a detailed description and examples of the way in which this kind of P systems simulates any logical function.

1.3 Christiansen grammars

Christiansen grammars are an adaptable extension of attribute grammars in which the first attribute associated to every symbol is a Christiansen grammar. Like attribute grammars, they have an ,universal computation power.

The first attribute of each non-terminal symbol contains the rules applicable to the corresponding symbol. Attributes are computed while the grammar is being used, so it is possible to change the grammar on the fly.

This paper follows the same formal notation for Christiansen grammars [10], which uses, respectively, the symbols $\downarrow$ and $\uparrow$ before the name of inherited and synthesized attributes.

2 Motivation

One of our main topics of interest is the formal specification of complex systems that makes it possible to apply formal tools to their design, or study some of their properties. Our research group has successfully applied this approach to other formal computational devices (L systems, cellular automata [5, 6, 2, 3]) and proposed a new evolutionary automatic programming algorithm (Christiansen grammar evolution or CGE [4]) as a powerful tool to design complex systems to solve specific tasks.

CGE wholly describes the candidate solutions, with Christiansen grammars, both syntactically and semantically, and hence, improves the performance of other approaches, because CGE reduces the search space by excluding non-promising individuals with syntactic or semantic errors.

P systems are abstract devices with a complex structure, because some of their components depends on others (for example: the number of axioms and regions and the set of proper suffixes to the right hand side of their evolution rules depends on the membrane structure μ). This dependence makes it difficult to use genetic techniques to search P systems because, in this circumstance, genetic operators usually produce a great number of incorrect individuals (both syntactically and semantically).

This paper describes a first step to formally tackle the design of P systems. The next step will be to apply Christiansen grammar evolution to the design of P systems.

As previously described, there are several ways of designing P systems to simulate logic circuits. Each of these possibilities imposes some restrictions on P systems, because it is not necessary to use the whole power of P systems to solve this problem.

This work is focused only on the formal description of these P systems. It neither proposes any new technique to simulate logical functions nor improves the complexity of the approaches described in [1]. Our final goal is to test CGE for designing of P systems to solve the same task (to simulate any logical function)

and to compare both solutions. The evaluation of the results will be used in order to propose a methodology to automatically design P systems.

3 Christiansen grammar for P systems that compute logical functions

3.1 Informal description

A Christiansen grammar for P systems that compute logical functions must solve the following situations:

- Every symbol of the alphabet (O), including catalysts, must be available to the set of rules of each region. So, the Christiansen grammar must have as many attributes as needed to take this information into account.
- μ contains both the degree and the structure in the regions of the P system. The rules in the Christiansen grammar describing each region records these data by means of the attributes *degree* and *daughters*. The latter is a list of the regions that directly belong to it.
- Once a region is added to the system, the Christiansen grammar must add new symbols to represent its axiom (w) and its rule set ($R_{number\_of\_region}$).
- An axiom is a *multiset* of symbols taken from O. Each evolution rule joins two *multisets*. There are some constructions dependent on the context that deal with the occurrence of catalysts:
  - A catalyst appears only once in a *multiset*. So, the semantic actions of the rules in the Christiansen grammar that specify that a catalyst can belong to a *multiset*, must remove this possibility once the catalyst is derived.
  - If a catalyst appears in the left hand side of a rule, it must also appear once in the right hand side. So, the actions of the rules that describe a production rule of a region must force the occurrence of a catalyst to the right hand side only when it belongs to the left hand side. As stated in the previous paragraph, our Christiansen grammar must forbid that a catalyst appears more than once in a *multiset*.
- This paper handles *multisets* as *strings*. The order of the symbols in the strings is not relevant: two strings that differ only in the order of their symbols are considered to be the same *multiset*.

3.2 Formal description

The Christiansen grammar proposed in this paper can be represented by the derivation rules formally described below.

1. $\mathit{\Pi}_B\{\downarrow g\} \rightarrow V\{\downarrow g, \uparrow g_V\}C\{\downarrow g_V, \uparrow g_C\}C'\{\downarrow g_C, \uparrow g_{C'}\}$
   $\mathrm{M}\{\downarrow g_{C'}, \uparrow g_{M,}\}W\{\downarrow g_M\}R\{\downarrow g_M\}o\{\downarrow g_M\},$

where:

- Π_B stands for the whole P system.
- V is the alphabet of the P system excluding the catalysts. It is convenient to this paper that catalysts are considered apart from the rest of the alphabet. So, V will stand, throughout this work, for the alphabet excluding the catalysts. V changes the initial Christiansen grammar to record its symbols. The hardest work deals with the delay symbols. The new grammar is synthesized by the attribute $\uparrow g_V$.
- C is the first catalyst and inherits from its left brother the *current* grammar ($\downarrow g_V$). When this catalyst is used, $\downarrow g_V$ must be modified. The result is assigned as the synthesized value of $\uparrow g_C$.
- C' is the second catalyst. $\uparrow g_{C'}$ plays the same role as $\uparrow g_C$ in the previous symbol.
- M stands for the membrane structure.
- W represents the axioms of the regions and inherits the changes from M.
- R represents the rule sets of the regions. It is important to notice that R inherits the *current* grammar from M rather than W because changes needed by each axiom are *local* to the axiom.
- o is the output region. It only needs to know the structure of M.

2. $V\{\downarrow g, \uparrow g_V \cup \{S_V\{\downarrow g\} \rightarrow 0|1\}\} \rightarrow 01S_d\{\downarrow g \uparrow g_V\}$.

As previously described, P systems that compute logical functions are binary. So

$$\{0, 1\} \subseteq V.$$

In addition, several symbols (x', x'', ...) are used to synchronize the propagation of data between different regions. They will be called *delay symbols* throughout this paper.

V contains 0, 1 and the delay symbols (represented by the non-terminal symbol S_d).

Other components of the P system will access the symbols that belong to V by means of the new non-terminal symbol S_V.

Delay symbols are strings that begin with x followed by any amount (including 0) of symbols "′". X_d represents each delay symbol and I_d its corresponding string of "′"s, which is stored in its second attribute. X_d adds to the *current grammar* a new rule whose left hand side is S_d, providing further access to the derived delay symbol.

3. $S_d\{\downarrow g, \uparrow g\} \rightarrow \lambda$,
   $S_d\{\downarrow g, \uparrow g_S\} \rightarrow X_d\{\downarrow g \uparrow g_x\}S_d\{\downarrow g_x \uparrow g_S\}$,
   $X_d\{\downarrow g, \uparrow g \cup \{S_V\{\downarrow g\} \rightarrow x.string\}\} \rightarrow xI_d\{\downarrow g \uparrow string\}$,
   $I_d\{\downarrow g, \uparrow \lambda\} \rightarrow \lambda$,
   $I_d\{\downarrow g, \uparrow' .string\} \rightarrow' I_d\{\downarrow g, \uparrow string\}$,
4. $C\{\downarrow g, \uparrow g \cup \{S_C\{\downarrow g, \uparrow g\} \rightarrow c\}\} \rightarrow c$,
   $C'\{\downarrow g, \uparrow g \cup \{S_{C'}\{\downarrow g, \uparrow g\} \rightarrow c'\}\} \rightarrow c'$,

where S_C and $S_{C'}$ are two new non-terminal symbols that represent respectively the catalysts c and c'. This P system uses at most two catalysts. Notice that unlike S_V, S_C needs two attributes, because controlling the number of occurrences of each catalyst is a task that depends on the context. This task is accomplished by means of the second attribute, and will be fully described further.

5. $M\{\downarrow g, \uparrow g_\mu\} \rightarrow (\mu\{\downarrow g, \uparrow g_\mu, \downarrow degree = 0, \uparrow degree_\mu, \uparrow daughters, \downarrow completed = yes\})$.

M is a symbol to ensure that membrane structure is always contained in the skin. μ, that appears in the right hand side of the following rule, is the membrane structure. Its last five attributes are used to keep the main changes needed by our Christiansen grammar. They will be described in detail in the following paragraphs. The resulting grammar is the value synthesized for attribute $\uparrow g_\mu$.

Notice that M does not modify the value synthesized by μ for its second attribute.

The membrane structure (μ) is a string of balanced parenthesis. μ is a main component of Π_B since it determines its structure and the number of axioms and the sets of rules needed. So the semantic actions of its rules make the biggest changes to the grammar.

The following additional rules are involved in these changes:

6. $w\{\downarrow g, \uparrow g\} \rightarrow \lambda$,
$w\{\downarrow g, \uparrow g\} \rightarrow S_V\{\downarrow g\}w\{\downarrow g\}$,
$w\{\downarrow g, \uparrow g_w\} \rightarrow S_C\{\downarrow g, \uparrow g_C =\downarrow g - \{w \rightarrow S_C w\}\}w\{\downarrow g_C, \uparrow g_w\}$,
$w\{\downarrow g, \uparrow g_w\} \rightarrow S_{C'}\{\downarrow g, \uparrow g_{C'} =\downarrow g - \{w \rightarrow S_{C'} w\}\}w\{\downarrow g_{C'}, \uparrow g_w\}$.

Each axiom is a string of symbols that belong to V (including λ). Catalysts are also allowed but they can appear only once. This condition is accomplished in the semantic actions of the third and fourth rule, that respectively remove $w \rightarrow S_C w$ and $w \rightarrow S_{C'} w$ from the grammar.

7. $W\{\downarrow g, \uparrow g\} \rightarrow \lambda$,
$R\{\downarrow g, \uparrow g\} \rightarrow \lambda$.

Each time a new region is derived, a new axiom and a new region are respectively added to W and R. These new axioms and regions are concatenated to the right hand sides of their rules. So they are initially set to λ.

8. $o\{\downarrow g, \uparrow g\} \rightarrow -1$.

Each time a new region is derived, a new possible output region is added to o. These new rules are added to the initial one, that represents the environment (-1).

9. Let j be $degree_{\mu 2} + 1^{(0)}$.

$\mu\{\downarrow g, \uparrow g_\mu, \downarrow degree, \uparrow degree_{\mu 2} + 1^{(0)}, \uparrow daughters_{\mu 1}. \uparrow degree_{\mu 2}, \downarrow completed\}$
$\rightarrow \mu^1\{\downarrow g, \uparrow g_{\mu 1}, \downarrow degree, \uparrow degree_{\mu 1}, \uparrow daughters_{\mu 1}, \underline{\downarrow \textbf{\textit{completed=no}}}\}$

$(\mu^2\{\downarrow g_{\mu 1}, \uparrow g_{\mu 2}, \downarrow degree_{\mu 1}, \uparrow degree_{\mu 2},$
$\uparrow daughters(\uparrow \_), \underline{\downarrow \textbf{\textit{completed=yes}}}\})$
$\{if \downarrow completed^{(5)}$ equals to no, then $\uparrow g_{\mu=} \uparrow g$
$else$
$\uparrow g_\mu =\downarrow g_{\mu 2} \cup$
$\{/*$These actions describe the region $\uparrow degree_{\mu 2}$ whose daughters are contained in $\uparrow daughters_{\mu 1}.degree_{\mu 2} * /$
$W\{\downarrow g\} \rightarrow right(W)^{(1)}.w\{\downarrow g, \uparrow \_\}^{(2)},$
$o\{\downarrow g_R\} \rightarrow j,$
$R\{\downarrow g, \uparrow g\}$
$\rightarrow right(R).R_j\{\downarrow g, \downarrow$ (production number)$0, \uparrow pn, \},$
$R_j\{\downarrow g, \downarrow pn, \uparrow pn\} \rightarrow \lambda,$
$R_j\{\downarrow g, \downarrow pn, \uparrow pn_2\} \rightarrow p_{j\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_j^2\{\downarrow g^{(3)}, \downarrow pn+1, \uparrow pn_2\}$
$\{$ Let $k =\downarrow pn$
and $^{(6)}\{r\} = \{pi_{j,k} \rightarrow \lambda, pi_{j,k} \rightarrow S_{CI',j,k}, pi_{j,k} \rightarrow S_{CI,j,k},$
$pi_{j,k} \rightarrow S_{CI,j,k}S_{CI',j,k}, pi_{j,k} \rightarrow S_{CI,j,k}S_{CI',j,k}\}$
$\downarrow g_p =\downarrow g \cup \{$
$p_{j,k}\{\downarrow g, \downarrow pn\} \rightarrow pd_{j,k}\{\downarrow g, \uparrow g_{pd}\} \rightarrow pi_{j,k}\{\downarrow g_{pd}, \uparrow g_{pi}\},$
$pd_{j,k}\{\downarrow g, \uparrow g\} \rightarrow S_v\{\downarrow g\},$
$pd_{j,k}\{\downarrow g, \uparrow g\} \rightarrow S_v\{\downarrow g\}pd_{j,k}\{\downarrow g, \uparrow g\},$
$pd_{j,k}\{\downarrow g, \uparrow g_{pd}\} \rightarrow S_C\{\downarrow g, \uparrow g_C =\downarrow g$
$- \{pd_{j,k} \rightarrow S_C pd_{j,k}, pd_{j,k} \rightarrow S_C\}\}$
$pd_{j,k}\{\downarrow g_C, \uparrow g_{pd1}\}$
$\{\uparrow g_{pd} =\uparrow g_{pd1} - \{r\} \cup \{pi_{j,k}\{\downarrow g, \uparrow g\} \rightarrow S_{CI,j,k}\{\downarrow g, \uparrow g\}.right(r)\}\}$
$pd_{j,k}\{\downarrow g, \uparrow g_{pd}\} \rightarrow S_{C'}\{\downarrow g, \uparrow g_{C'} =\downarrow g -$
$\{pd_{j,k} \rightarrow S_{C'} pd_{j,k}, pd_{j,k} \rightarrow S_{C'}\}\}$
$pd_{j,k}\{\downarrow g_{C'}, \uparrow g_{pd1}\}$
$\{\uparrow g_{pd} =\uparrow g_{pd1} - \{r\} \cup \{pi_{j,k}\{\downarrow g, \uparrow g\} \rightarrow S_{CI',j,k}\{\downarrow g \uparrow g\}.right(r)\}\}$
$pd_{j,k}\{\downarrow g, \uparrow g_{pd}\} \rightarrow S_C\{\downarrow g, \uparrow g_C =\downarrow g$
$- \{pd_{j,k} \rightarrow S_C pd_{j,k}, pd_{j,k} \rightarrow S_C\}\}$
$\{\uparrow g_{pd} =\uparrow g_C - \{r\} \cup \{pi_{j,k}\{\downarrow g, \uparrow g\} \rightarrow S_{CI,j,k}\{\downarrow g, \uparrow g\}.right(r)\}\}$
$pd_{j,k}\{\downarrow g, \uparrow g_{pd}\} \rightarrow S_{C'}\{\downarrow g, \uparrow g_{C'} =\downarrow g$
$- \{pd_{j,k} \rightarrow S_{C'} pd_{j,k}, pd_{j,k} \rightarrow S_{C'}\}\}$
$\{\uparrow g_{pd} =\uparrow g_{C'} - \{r\} \cup \{pi_{j,k}\{\downarrow g, \uparrow g\} \rightarrow S_{CI',j,k}\{\downarrow g, \uparrow g\}.right(r)\}\}$
$S_{CI,j,k}\{\downarrow g, \uparrow g\} \rightarrow c_t : t \in \{out, \lambda, in_r, , r \in \uparrow daughters_{\mu 1}.j\}$
$S_{CI',j,k}\{\downarrow g, \uparrow g\} \rightarrow c'_t : t \in \{out, \lambda, in_r, , r \in \uparrow daughters_{\mu 1}.j\}$
$pi_{j,k}\{\downarrow g, \uparrow g\} \rightarrow \lambda,$
$pi_{j,k}\{\downarrow g, \uparrow g_i\} \rightarrow s\{\downarrow g\}pi_{j,k}\{\downarrow g, \uparrow g_i\}$
$s \in S_V \cup \{s_d : s \in S_V \wedge d \in \{out\} \cup \{in.j : j \in^{(4)} \uparrow daughters_{\mu 1}.j\}\}$
$\}$
$\}$
$\}$

Most work is done when the regions are closed.

(a) The last five attributes of μ handles two properties of the membrane structure:

- The **degree** or number of regions in the P system: the second and third attributes are used for this purpose. $\downarrow$*degree* inherits the *current number of regions* that initially equals 0, and the first rule (the one that closes each region) synthesizes the updated value ($\uparrow degree_{\mu 2} + 1$), which is one more than the number of regions accumulated by μ^1 and μ^2. This same value ($\uparrow degree_{\mu 2} + 1$) is the number of the *current region* and is used to identify it.
- The **inclusion relationship between regions** is contained in the penultimate attribute (*daughters*), the list of the numbers of the regions directly included by the just closed region. Notice that no deeper level is taken into account, and notice too that to get the daughters of μ, it suffices to stick the number of the *current region–1* to the daughters of μ^1.

(b) New axioms and regions are added to the right hand sides of their corresponding rules (those whose left hand sides are respectively W and R).

(c) All the axioms can be described in the same way (w), but the rule set of each region depends on the structure of the region: the symbols to the right hand sides of the rules can have as index the number of any of the regions that they directly include. So, a different non-terminal symbol (R_{num_region}) and a different set of rules are used for each region.

(d) Not all the occurrences of μ must describe a region. One must do it only when a region is really completed. The last attribute of μ (*completed*) takes into account this circumstance. If the value of this attribute is *no* the grammar inherited by μ remains unchanged. The reader can verify that, even if it closes a new region by deriving the first rule, the first μ to the right hand side of the first rule cannot describe any region. Only the last μ really completes a region.

(e) Several conventions are used to simplify writing the preceding rules:

- j will be used rather than $degree_{\mu 2} + 1$ to represent the number of the *current region*.
- All the attributes not used in their rules will be called _. For example, rather than

$$\mu^2\{\downarrow g_{\mu 1}, \uparrow g_{\mu}, \downarrow degree_{\mu 1}, \uparrow degree_{\mu 2}, \uparrow daughters, \downarrow completed\}$$

we will write

$$\mu^2\{\downarrow g_{\mu 1}, \uparrow g_{\mu}, \downarrow degree_{\mu 1}, \uparrow degree_{\mu 2}, \uparrow \_, \downarrow completed\},$$

since the penultimate attribute of μ_2 is not used in the rule. $\uparrow$*daughters* is not really needed to compute the daughters of its father (the symbol μ at the left hand side). The daughters of μ are only the regions that it includes at the first level.

Additional notes

- $^{(0)}$Notice that the number of the region described by these semantic actions is $j = degree_{\mu 2}+1$, which is the same as the *degree* synthesized; however, the value synthesized for *daughters* is $daughters_{\mu 1}.degree_{\mu 2}$ (so $degree_{\mu 2} = j - 1$). It is very important to "delay" the "propagation upwards" because the degree must be increased, to identify and describe the just closed region, after computing its list of daughters.
- $^{(1)}right(S)$, where S is a non-terminal symbol, is a function that returns the right hand side of the rule (there is at most one) in the grammar whose left symbol is S, or λ if there is no such rule.
- $^{(2)}$The grammar synthesized by each axiom is not used, the key in this process is to keep the initial grammar independent of each axiom.
- $^{(3)}$ Changes made to the grammar by a production rule are local to the rule. So $p_{j\downarrow pn}$ and R^2_j have the same grammar $\downarrow g$ as the value of their first attribute.
- $^{(4)}$ "$j \in \uparrow daughters$" represents the set of symbols in the $\uparrow daughters$ string.
- $^{(5)}$ In these semantic actions, *completed* stands for the corresponding attribute of the symbol μ at the left hand side, although the same name is used in the right hand side to simplify the notation.
- $^{(6)}$ Notice that, by construction, the set $\{ pi_{j,k} \to \lambda,\ pi_{j,k} \to S_{CI',j,k},\ pi_{j,k} \to S_{CI,j,k},\ pi_{j,k} \to S_{CI,j,k}S_{CI',j,k},\ pi_{j,k} \to S_{CI,j,k}S_{CI',j,k} \}$ has only one element, and hence the expression $\{r\} = \{pi_{j,k} \to \lambda, pi_{j,k} \to S_{CI',j,k},\ pi_{j,k} \to S_{CI,j,k},\ pi_{j,k} \to S_{CI,j,k}S_{CI',j,k},\ pi_{j,k} \to S_{CI,j,k}S_{CI',j,k}\}$ is correct.

10. $\mu\{ \downarrow g, \uparrow g_\mu, \downarrow degree, \uparrow degree \uparrow daughters = \lambda, \downarrow completed\} \to \lambda$
    $\{$ if $\downarrow completed$ equals to *no* then $\uparrow g_{\mu=} \uparrow g$

else

Let be $j = degree$

$\uparrow g_\mu = \{$/*These actions describe the region $\uparrow$ *degree* that has no daughters */

$W\{ \downarrow g\} \to right(W).w\{ \downarrow g, \uparrow \_\}^{(5)}$,

$R\{ \downarrow g, \uparrow g\}$
$\to right(R).R_j\{ \downarrow g, \downarrow$ (production number) $0, \uparrow pn, \}$,

$R_j\{ \downarrow g, \downarrow pn, \uparrow pn\} \to \lambda$,

$R_j\{ \downarrow g, \downarrow pn, \uparrow pn_2\} \to p_{j\downarrow pn}\{ \downarrow g_p, \downarrow pn + 1\}R^2_j\{ \downarrow g, \downarrow pn + 1, \uparrow pn_2\}$
$\{Letk =\downarrow pnand\{r\} = \{pi_{j,k} \to \lambda, pi_{j,k} \to S_{CI',j,k}, pi_{j,k} \to S_{CI,j,k},$
$pi_{j,k} \to S_{CI,j,k}S_{CI',j,k}, pi_{j,k} \to S_{CI,j,k}S_{CI',j,k}\}$
$\downarrow g_p =\downarrow g \cup \{$
$p_{j,k}\{ \downarrow g, \downarrow pn\} \to pd_{j,k}\{ \downarrow g, \uparrow g_{pd}\} \to pi_{j,k}\{ \downarrow g_{pd}, \uparrow g_{pi}\}$,
$pd_{j,k}\{ \downarrow g, \uparrow g\} \to S_v\{ \downarrow g\}$,
$pd_{j,k}\{ \downarrow g, \uparrow g\} \to S_v\{ \downarrow g\}pd_{j,k}\{ \downarrow g, \uparrow g\}$,
$pd_{j,k}\{ \downarrow g, \uparrow g_{pd}\} \to S_C\{ \downarrow g, \uparrow g_C =\downarrow g$
$- \{pd_{j,k} \to S_C pd_{j,k}, pd_{j,k} \to S_C\}\}$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad pd_{j,k}\{\downarrow g_C, \uparrow g_{pd1}\} \\
\{\uparrow g_{pd} = \uparrow g_{pd1} - \{r\} \cup \{pi_{j,k}\{\downarrow g, \uparrow g\} \rightarrow S_{CI,j,k}\{\downarrow g, \uparrow g\}.right(r)\}\} \\
\quad pd_{j,k}\{\downarrow g, \uparrow g_{pd}\} \rightarrow S_C\text{'}\{\downarrow g, \uparrow g_C\text{'} = \downarrow g - \\
\qquad\qquad\qquad\qquad\qquad\qquad \{pd_{j,k} \quad \rightarrow S_C\text{'}, pd_{j,k}, pd_{j,k} \quad \rightarrow S_C\text{'},\}\} \\
\qquad\qquad\qquad\qquad\qquad\qquad pd_{j,k}\{\downarrow g_C\text{'}, \uparrow g_{pd1}\} \\
\{\uparrow g_{pd} = \uparrow g_{pd1} - \{r\} \cup \{pi_{j,k}\{\downarrow g, \uparrow g\} \rightarrow S_{CI}\text{'}_{,j,k}\{\downarrow g, \uparrow g\}.right(r)\}\} \\
\quad pd_{j,k}\{\downarrow g, \uparrow g_{pd}\} \rightarrow S_C\{\downarrow g, \uparrow g_C = \downarrow g \\
\qquad\qquad\qquad\qquad\qquad\qquad - \{pd_{j,k} \quad \rightarrow S_C pd_{j,k}, pd_{j,k} \quad \rightarrow S_C\}\} \\
\{\uparrow g_{pd} = \uparrow g_C - \{r\} \cup \{pi_{j,k}\{\downarrow g, \uparrow g\} \rightarrow S_{CI,j,k}\{\downarrow g, \uparrow g\}.right(r)\}\} \\
\quad pd_{j,k}\{\downarrow g, \uparrow g_{pd}\} \rightarrow S_C\text{'}\{\downarrow g, \uparrow g_C\text{'} = \downarrow g \\
\qquad\qquad\qquad\qquad\qquad\qquad - \{pd_{j,k} \quad \rightarrow S_C\text{'}, pd_{j,k}, pd_{j,k} \quad \rightarrow S_C\text{'},\}\} \\
\{\uparrow g_{pd} = \uparrow g_C\text{'} - \{r\} \cup \{pi_{j,k}\{\downarrow g, \uparrow g\} \rightarrow S_{CI}\text{'}_{,j,k}\{\downarrow g, \uparrow g\}.right(r)\}\} \\
\quad S_{CI,j,k}\{\downarrow g, \uparrow g\} \rightarrow c_t : t \in \{out, \lambda\} \\
\quad S_{CI}\text{'}_{,j,k}\{\downarrow g, \uparrow g\} \rightarrow c'_t : t \in \{out, \lambda\} \\
\quad pi_{j,k}\{\downarrow g, \uparrow g\} \rightarrow \lambda, \\
\quad pi_{j,k}\{\downarrow g, \uparrow g_i\} \rightarrow s\{\downarrow g\} pi_{j,k}\{\downarrow g, \uparrow g_i\} \\
\quad , , s \in S_V \cup \{s_d : s \in S_V \wedge d \in \{out\}\} \\
\quad o\{\downarrow g_R\} \rightarrow j \\
\quad \} \\
\} \\
\}
\end{array}
$$

Notice that the semantic actions of this rule are very similar to the previous one. This rule describes an empty region ($\uparrow daughters = \lambda$), so its symbols can only remain in the region (without any index) or exit from it (by using *out* as the index).

3.3 Example

Figure 1 shows a derivation tree for the P system

$\Pi_B = (V = \{0, 1, x'\}, C = \{c, c'\}, \mu = ((()_0()_1)_2)_3, \omega_0 = \lambda, \omega_1 = 101, \omega_2 = 0, \omega_3 = 11,$

$$
\begin{array}{c}
(R_0 = \emptyset, \rho_0 = \emptyset), \\
(R_1 = \{ \\
p_{10} = 1x' \rightarrow \lambda, \\
p_{11} = 1 \rightarrow \lambda, \\
p_{12} = 1c \rightarrow c_{out}0 \\
\} \\
, \rho_1 = \{p_{10} \geq p_{11} \geq p_{12}\}), \\
(R_2 = \emptyset, \rho_2 = \emptyset), \\
(R_3 = \emptyset, \rho_3 = \emptyset), \\
i_o = 2).
\end{array}
$$

Some information will not be explicitly represented:

- The correspondence between axioms and regions will be deduced from their positions: so, the j^{th} axiom will belong to the region R_{j-1} (notice that the index of the regions is zero based).
- The order relationship for the rules of a region is determined by their appearance order.

The derivation process will be briefly described. The tree in Figure 1 shows the derivation tree of this P system. Different colors are used to remark some circumstances that help to understand how our Christiansen grammar works:

- Color gray highlights the information that is known from the beginning of the process (the structures of Π_B, and the axioms). Hence, these derivation rules belong to the initial grammar.
- Color blue indicates the information that depends on the derived alphabet (specially the subset of delay symbols). The rules that describe non-terminal symbols S_V, S_d, S_C and $S_{C'}$ are added to the grammar after deriving the symbols V, C and C'.
- M (the membrane structure) is the main component of the P system because it determines the number of regions (the degree) and their structure. So, it is mandatory to finish the derivation of M before adding to the grammar the derivation rules that describe the number of axioms and regions and the basic structure of each region as a set of production rules. Color red designates this dependence.
- Each region records the number of its rules. Each rule is described by three non-terminal symbols (one for its left hand side, one for its right hand side and another one for the complete rule) that are identified by two indices j, k (corresponding to the k^{th} rule of the j^{th} region). So, it is impossible to add the derivation rules for these three non-terminals after deriving the region itself. Color green highlights this situation.
- Finally, the structure to the right hand side of the rules depends on the left hand side because of the catalysts. So, the description of the right hand sides in the grammar can be modified after deriving the left hand sides. This dependence is emphasized with color orange.

We will highlight the most significant steps in the derivation of this example:

- (I) After specifying the complete alphabet.
- (II) After processing the membrane structure.
- (III) After describing R_1 (except $pi_{1,2}$ the right hand side of its last rule $p_{1,2}$).
- (IV) After finishing the P system.

The first rule applied is

1. $\Pi_B\{\underline{\downarrow g}\} \rightarrow V\{\underline{\downarrow g}, \uparrow g_V\}C\{\downarrow g_V, \uparrow g_C\}C'\{\downarrow g_C, \uparrow g_{C'}\}$
   $\mathrm{M}\{\downarrow g_{C'}, \uparrow g_M\}\overline{W}\{\downarrow g_M\}R\{\downarrow g_M\}o\{\downarrow g_M\}$.

   V inherits its first attribute from Π_B.
   When the rule

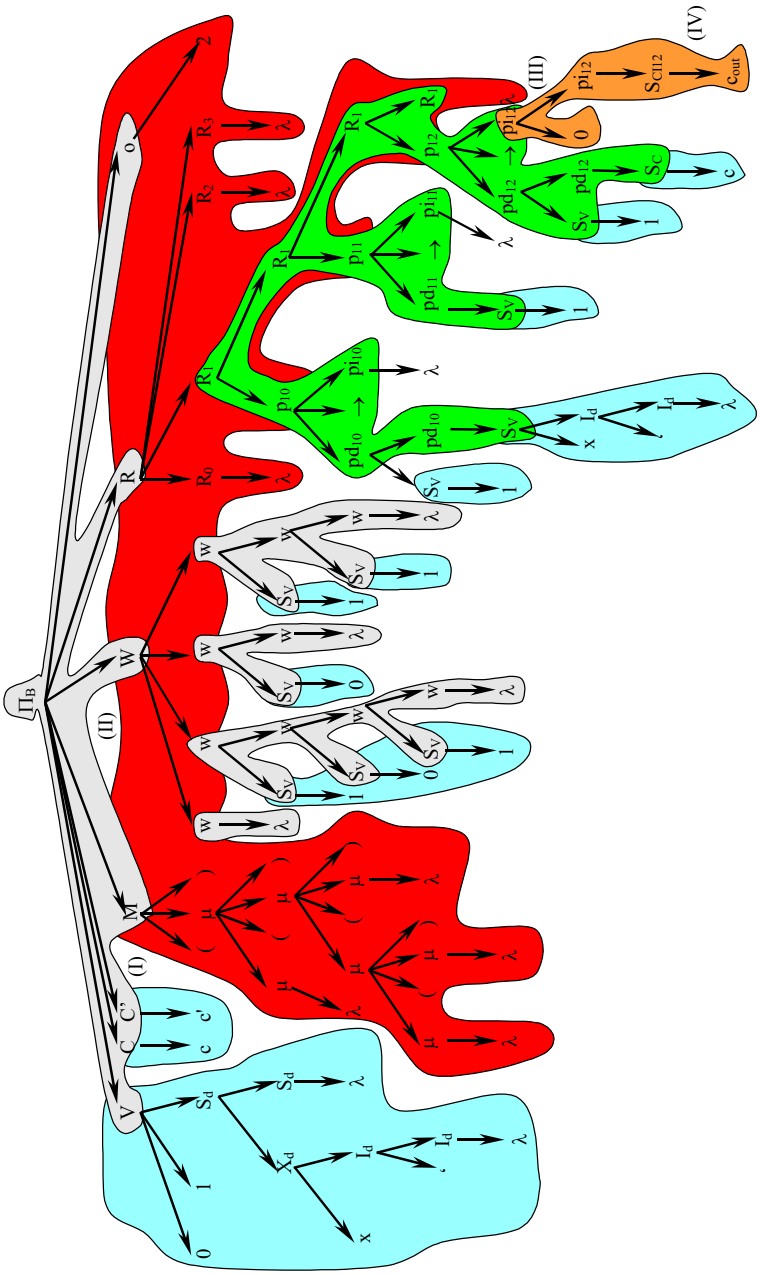


Fig. 1. Derivation tree of the example.

2. $V\{\underline{\downarrow g}, \uparrow g_V \cup \{S_V\{\downarrow g\} \rightarrow 0|1\}\} \rightarrow 01S_d\{\underline{\downarrow g} \uparrow g_V\}$

is applied, S_d inherits the value of its first attribute from V. S_d must synthesize its second attribute before V can add the rules $S_V\{\downarrow g\} \rightarrow 0|1$ to it.

Figure 2 graphically shows the evaluation of $\uparrow g_V$ and its modification by the semantic actions of the previous rule. Red arrows are used to highlight synthesis, while blue arrows represent inheritance. The modifications to the *current grammar* are shown between brackets.

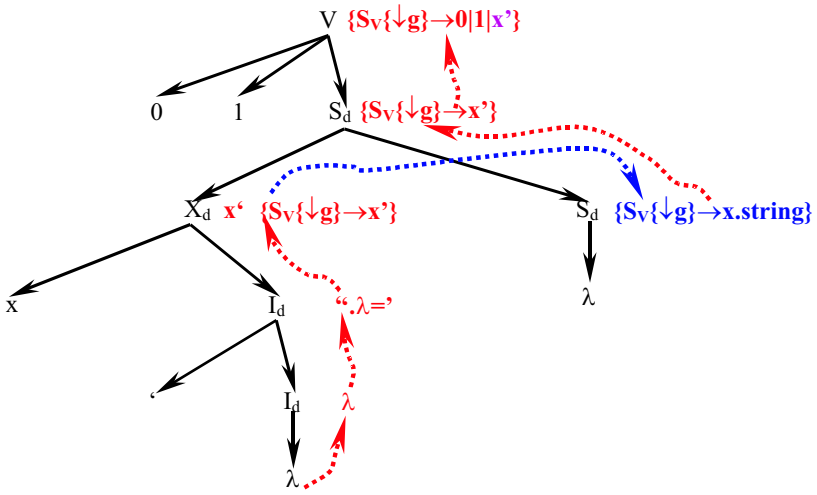


Fig. 2. Derivation tree after applying the rule number 2.

At this moment the rule

1. $\Pi_B\{\underline{\downarrow g}\} \rightarrow V\{\underline{\downarrow g}, \underline{\uparrow g_V}\}C\{\underline{\downarrow g_V}, \uparrow g_C\}C'\{\downarrow g_C, \uparrow g_{C'}\}$
   $\mathrm{M}\{\downarrow g_{C'}, \uparrow g_M\}W\{\downarrow g_M\}R\{\downarrow g_\mu\}o\{\downarrow g_\mu\}$

can be revisited, because symbol V has finished the evaluation of its attributes and C can inherit the value of its first attribute from V. Figure 3 graphically shows the computation of the attributes while applying the rules

4. $C\{\downarrow g, \uparrow g \cup \{S_C\{\downarrow g, \uparrow g\} \rightarrow c\}\} \rightarrow c$
   $C'\{\downarrow g, \uparrow g \cup \{S_{C'}\{\downarrow g, \uparrow g\} \rightarrow c'\}\} \rightarrow c'.$

After finishing the evaluation of the attributes of C', the remaining substring to the right hand side of the first rule can be processed.

1. $\Pi_B\{\underline{\downarrow g}, \uparrow g_R\} \rightarrow V\{\underline{\downarrow g}, \underline{\uparrow g_V}\}C\{\underline{\downarrow g_V}, \underline{\uparrow g_C}\}C'\{\underline{\downarrow g_{C'}}, \underline{\uparrow g_{C'}}\}$ (0)
   $\mathrm{M}\{\underline{\downarrow g_{C'}}, \uparrow g_M\}W\{\downarrow g_M\}R\{\downarrow g_M\}o\{\downarrow g_M\}.$

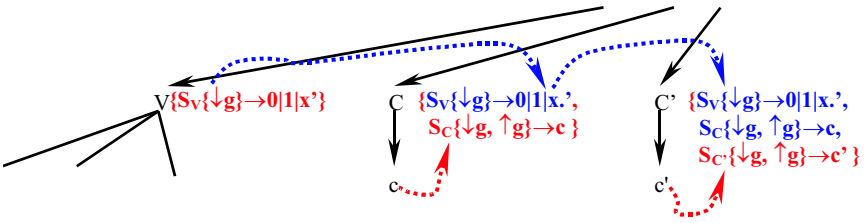


Fig. 3. Derivation tree after applying the rule number 4.

The non-terminal M inherits the value of its first attribute from C'.
The only rule applicable to M is

5. $M\{\downarrow g = \downarrow g_{C'}, \uparrow g_M = \uparrow g_\mu\} \rightarrow (\mu\{\downarrow g, \uparrow g_\mu, \underline{\downarrow degree = 0}, \uparrow degree_\mu, \uparrow daughters,$
$\underline{\downarrow completed = yes}\})$.

Notice that M only inherits its first attribute. M will synthesize its last attribute after processing μ.

The next step will determine the structure of the P system and make the major changes to the grammar. Figure 4 graphically shows the result of the process.

Once μ is fully derived, we can continue with M. Its second attribute does not modify the second one for the symbol μ to the right hand side.

$M\{\underline{\downarrow g}, \underline{\uparrow g_\mu = (d)}\} \rightarrow (\mu\{\underline{\downarrow g}, \underline{\uparrow g_\mu = (d)}, \underline{\downarrow 0}, \underline{\uparrow degree_{\mu 2} + 1 = 3},$
$\underline{\uparrow daughters_{\mu 1}.degree_{\mu 2} = 2}, \underline{\downarrow completed = yes}\})$.

Now it is possible to process W (the axioms of the regions of Π_B), which is the next symbol in expression (0), that inherits (d) from M has its initial grammar. Expression (0′) updates the attributes of μ and W. Notice that R and o can also evaluate their initial grammars to (d).

(0') $\Pi_B\{\downarrow g, \uparrow g_R\} \rightarrow$ V$\{\underline{\downarrow g}, \underline{\uparrow g_V}\}$ C$\{\underline{\downarrow g_V}, \underline{\uparrow g_C}\}$ C'$\{\underline{\downarrow g_C}, \underline{\uparrow \boldsymbol{g}_{C'}}\}M\{\underline{\downarrow g_{C'}}, \underline{\uparrow g_\mu = (d)}\}$
W$\{\underline{\uparrow \boldsymbol{g}_\mu = (d)}\}$ R$\{\underline{\uparrow \boldsymbol{g}_\mu = (d)}, \uparrow g_R\}o\{\underline{\uparrow g_\mu = (d)}\}$

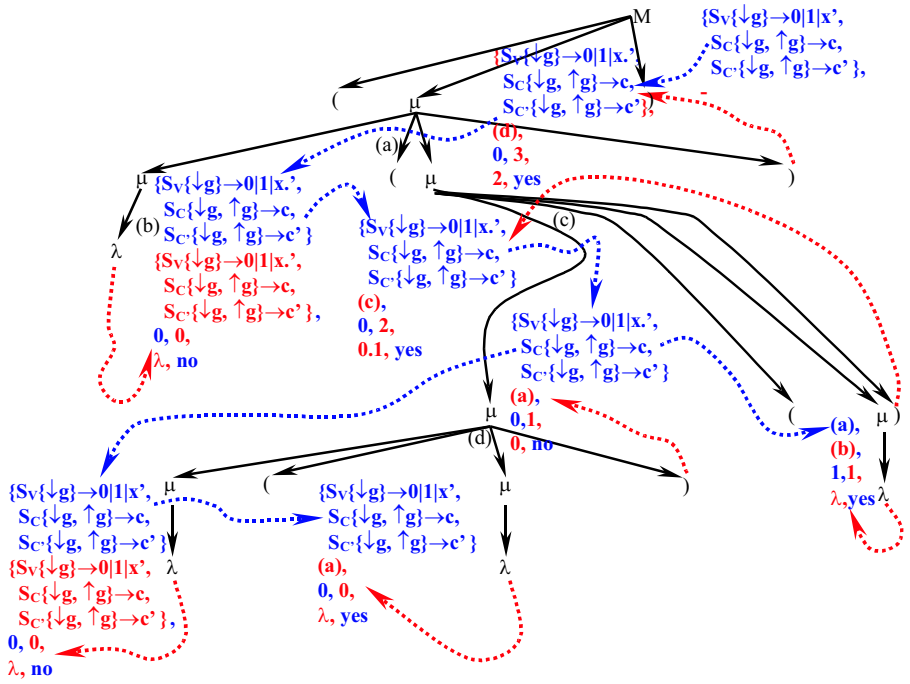
Grammar (d) has only one rule applicable to W

7. $W\{\underline{\downarrow \boldsymbol{g} = (d)}\} \rightarrow \{\underline{\downarrow \boldsymbol{g} = (d)}, \uparrow_- \}w\{\underline{\downarrow \boldsymbol{g} = (d)}, \uparrow_- \}$
w$\{\underline{\downarrow \boldsymbol{g} = (d)}, \uparrow_- \}w\{\underline{\downarrow \boldsymbol{g} = (d)}, \uparrow_- \}$

No change is made to the initial grammar. The reader can verify in the initial derivation tree that each axiom is derived by means of pure context free mechanisms.

At this point, we process symbol R.
Grammar (d) has only a rule applicable to R.

$R\{\underline{\downarrow \boldsymbol{g} = (d)}\} \rightarrow R_0\{\underline{\downarrow \boldsymbol{g} = (d)}, \downarrow 0, \uparrow pn\}$



(c)={ $S_V\{\downarrow g\}\to 0|1|x'$, $S_C\{\downarrow g, \uparrow g\}\to c$, $S_{C'}\{\downarrow g, \uparrow g\}\to c'$,
$W\{\downarrow g\}\to w\{\downarrow g, \uparrow\_\}w\{\downarrow g, \uparrow\_\}$, $o\{\downarrow g_R\}\to -1|0|1|2$
$R\{\downarrow g,\uparrow g\}\to R_0\{\downarrow g, \uparrow\_, \downarrow 0, \uparrow pn\}\ R_1\{\downarrow g, \uparrow\_, \downarrow 0, \uparrow pn\}\ R_2\{\downarrow g, \uparrow\_, \downarrow 0, \uparrow pn\}$,
$R_0\{\downarrow g, \uparrow g, \downarrow pn, \uparrow pn\}\to\lambda$,
$R_0\{\downarrow g, \uparrow\_, \downarrow pn, \uparrow pn_2\}\to p_{0\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_0^2\{\downarrow g, \uparrow g_R, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$
$R_1\{\downarrow g, \uparrow g, \downarrow pn, \uparrow pn\}\to\lambda$,
$R_1\{\downarrow g, \uparrow\_, \downarrow pn, \uparrow pn_2\}\to p_{1\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_1^2\{\downarrow g, \uparrow g_R, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$
$R_2\{\downarrow g, \uparrow g, \downarrow pn, \uparrow pn\}\to\lambda$,
$R_2\{\downarrow g, \uparrow\_, \downarrow pn, \uparrow pn_2\}\to p_{2\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_2^2\{\downarrow g, \uparrow g_R, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$
}

(d)= { $S_V\{\downarrow g\}\to 0|1|x'$, $S_C\{\downarrow g, \uparrow g\}\to c$, $S_{C'}\{\downarrow g, \uparrow g\}\to c'$,
$W\{\downarrow g\}\to w\{\downarrow g, \uparrow\_\}w\{\downarrow g, \uparrow\_\}\ w\{\downarrow g, \uparrow\_\}\ w\{\downarrow g, \uparrow\_\}$, $o\{\downarrow g_R\}\to -1|0|1|2|3$
$R\{\downarrow g\}\to R_0\{\downarrow g, \downarrow 0, \uparrow pn\}R_1\{\downarrow g, \downarrow 0, \uparrow pn\}$
$R_2\{\downarrow g, \downarrow 0, \uparrow pn\}R_3\{\downarrow g, \downarrow 0, \uparrow pn\}$,
$R_0\{\downarrow g, \downarrow pn, \uparrow pn\}\to\lambda$,
$R_0\{\downarrow g, \downarrow pn, \uparrow pn_2\}\to p_{0\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_0^2\{\downarrow g, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$
$R_1\{\downarrow g, \downarrow pn, \uparrow pn\}\to\lambda$,
$R_1\{\downarrow g, \downarrow pn, \uparrow pn_2\}\to p_{1\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_1^2\{\downarrow g, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$
$R_2\{\downarrow g, \downarrow pn, \uparrow pn\}\to\lambda$,
$R_2\{\downarrow g, \downarrow pn, \uparrow pn_2\}\to p_{2\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_2^2\{\downarrow g, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$
$R_3\{\downarrow g, \downarrow pn, \uparrow pn\}\to\lambda$,
$R_3\{\downarrow g, \downarrow pn, \uparrow pn_2\}\to p_{3\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_3^2\{\downarrow g, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$
}

Fig. 4. Derivation tree after processing μ.

$R_1\{\downarrow \boldsymbol{g=(d)}, \downarrow 0, \uparrow pn\ \}$
$R_2\{\downarrow \boldsymbol{g=(d)}, \downarrow 0, \uparrow pn\ \}$
$R_3\{\downarrow \boldsymbol{g=(d)}, \downarrow 0, \uparrow pn\ \}$

It is worth noticing that each region does not depend on the other regions. They inherit the initial grammar from R and use 0 as the initial number of production rules.

The last eight rules in grammar (d)describe R_0, R_1, R_2 and R_3 as sets of 0 or more production rules. Each production rule has two indices that identify it (j and k). So $p_{j,k}$ is the k^{th} rule for the j^{th} region.

Grammar (d), however, does not describe each production rule. The semantic actions of the following rules modify and complete (d):

$R_0\{\downarrow g, \downarrow pn, \uparrow pn_2\} \rightarrow p_{0\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_0^2\{\downarrow g, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$
$R_1\{\downarrow g, \downarrow pn, \uparrow pn_2\} \rightarrow p_{1\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_1^2\{\downarrow g, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$
$R_2\{\downarrow g, \downarrow pn, \uparrow pn_2\} \rightarrow p_{2\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_2^2\{\downarrow g, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$
$R_3\{\downarrow g, \downarrow pn, \uparrow pn_2\} \rightarrow p_{3\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_3^2\{\downarrow g, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$

The following paragraphs explain this process.

Three of the four regions (R_0, R_2 and R_3) are empty. The subtree in figure 5 corresponds to region R_1.

Rule $R_1\{\downarrow g, \downarrow pn, \uparrow pn_2\} \rightarrow p_{1\downarrow pn}\{\downarrow g_p, \downarrow pn+1\}R_1^2\{\downarrow g, \downarrow pn+1, \uparrow pn_2\}\{\cdots\}$ is initially applied to symbol $R_1\{\downarrow g = (d), \downarrow 0, \uparrow pn\ \}$. Expression (g) shows this situation.

(g)

$R_1\{\downarrow g=(d), \downarrow 0, \uparrow pn_2\} \rightarrow$
$p_{10}\{\downarrow \boldsymbol{g_p=(e)}, \downarrow \boldsymbol{pn+1=0+1=1}\}$
$R_1^2\{\downarrow \boldsymbol{g=(d)}, \downarrow pn+1=0+1=1, \uparrow pn_2\}$
{ Let $\{r\} = \{pi_{1,0} \rightarrow \lambda, pi_{1,0} \rightarrow S_{CI',1,0}, pi_{1,0} \rightarrow S_{CI,1,0},$
$pi_{1,0} \rightarrow S_{CI,1,0}S_{CI',1,0}, pi_{1,0} \rightarrow S_{CI,1,0}S_{CI',1,0}\ \}$
$\downarrow \boldsymbol{g}_p \boldsymbol{=(e)=(d)} \cup \{$
$\boldsymbol{p}_{1,0}\{\downarrow \boldsymbol{g, \downarrow pn}\} \rightarrow \boldsymbol{pd}_{1,0}\{\downarrow \boldsymbol{g, \uparrow g_{pd}}\} \rightarrow \boldsymbol{pi}_{1,0}\{\downarrow \boldsymbol{g_{pd}, \uparrow g_{pi}}\},$
$\boldsymbol{pd}_{1,0}\{\downarrow \boldsymbol{g, \uparrow g}\} \rightarrow \boldsymbol{S}_v\{\downarrow \boldsymbol{g}\},$
$\boldsymbol{pd}_{1,0}\{\downarrow \boldsymbol{g, \uparrow g}\} \rightarrow \boldsymbol{S}_v\{\downarrow \boldsymbol{g}\}\boldsymbol{pd}_{1,0}\{\downarrow \boldsymbol{g, \uparrow g}\},$
$\boldsymbol{pd}_{1,0}\{\downarrow \boldsymbol{g, \uparrow g_{pd}}\} \rightarrow \boldsymbol{S}_C\{\downarrow \boldsymbol{g, \uparrow g}_C \boldsymbol{=\downarrow g}$
$\boldsymbol{-\{pd}_{1,0} \boldsymbol{\rightarrow S}_C \boldsymbol{pd}_{1,0}\boldsymbol{, pd}_{1,0} \boldsymbol{\rightarrow S}_C\}\}$
$\boldsymbol{pd}_{1,0}\{\downarrow \boldsymbol{g}_C, \uparrow \boldsymbol{g}_{pd1}\}$
$\{\uparrow \boldsymbol{g}_{pd} \boldsymbol{=\uparrow g}_{pd1} \boldsymbol{-\{r\} \cup \{pi}_{1,0}\{\downarrow \boldsymbol{g, \uparrow g}\} \rightarrow \boldsymbol{S}_{CI,1,0}\{\downarrow \boldsymbol{g \uparrow g}\}\boldsymbol{.right(r)}\}\}$
$\boldsymbol{pd}_{1,0}\{\downarrow \boldsymbol{g, \uparrow g_{pd}}\} \rightarrow \boldsymbol{S}_C\{\downarrow \boldsymbol{g, \uparrow g}_{C'} \boldsymbol{=\downarrow g-}$
$\{\boldsymbol{pd}_{1,0} \boldsymbol{\rightarrow S}_{C'} \boldsymbol{pd}_{1,0}\boldsymbol{, pd}_{1,0} \boldsymbol{\rightarrow S}_{C'}\}\}$
$\boldsymbol{pd}_{1,0}\{\downarrow \boldsymbol{g}_{C'}, \uparrow \boldsymbol{g}_{pd1}\}$
$\{\uparrow \boldsymbol{g}_{pd} \boldsymbol{=\uparrow g}_{pd1} \boldsymbol{-\{r\} \cup \{pi}_{1,0}\{\downarrow \boldsymbol{g, \uparrow g}\} \rightarrow \boldsymbol{S}_{CI',1,0}\{\downarrow \boldsymbol{g \uparrow g}\}\boldsymbol{.right(r)}\}\}$
$\boldsymbol{pd}_{1,0}\{\downarrow \boldsymbol{g, \uparrow g_{pd}}\} \rightarrow \boldsymbol{S}_C\{\downarrow \boldsymbol{g, \uparrow g}_C \boldsymbol{=\downarrow g}$
$\boldsymbol{-\{pd}_{1,0} \boldsymbol{\rightarrow S}_C \boldsymbol{pd}_{1,0}\boldsymbol{, pd}_{1,0} \boldsymbol{\rightarrow S}_C\}\}$
$\{\uparrow \boldsymbol{g}_{pd} \boldsymbol{=\uparrow g}_C \boldsymbol{-\{r\} \cup \{pi}_{1,0}\{\downarrow \boldsymbol{g, \uparrow g}\} \rightarrow \boldsymbol{S}_{CI,1,0}\{\downarrow \boldsymbol{g \uparrow g}\}\boldsymbol{.right(r)}\}\}$

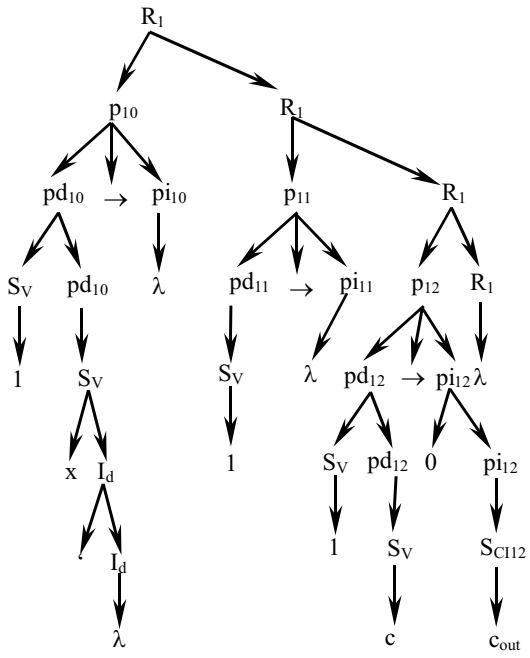


Fig. 5. Subtree of region R_1.

$\underline{\boldsymbol{pd}_{1,0}\ \{\downarrow g,\ \uparrow g_{pd}\} \rightarrow\ \boldsymbol{S}_{C'}\{\downarrow g, \uparrow g_{C'} = \downarrow g}$
$\underline{\text{-}\{\ \boldsymbol{pd}_{1,0} \rightarrow \boldsymbol{S}_{C'}\boldsymbol{pd}_{1,0},\ \boldsymbol{pd}_{1,0} \rightarrow \boldsymbol{S}_{C'}\}\}}$
$\underline{\{\uparrow g_{pd} = \uparrow g_{C'}\ \text{-}\{r\} \cup \{\boldsymbol{pi}_{1,0}\{\downarrow g, \uparrow g\} \rightarrow \boldsymbol{S}_{CI',1,0}\{\downarrow g \uparrow g\}.right(r)\}}$
$\underline{\}}$
$\underline{\boldsymbol{S}_{CI,1,0}\ \{\downarrow g,\ \uparrow g\} \rightarrow c_t : t \in \{out,\ \lambda\ \}}$
$\underline{\boldsymbol{S}_{CI',1,0}\ \{\downarrow g,\ \uparrow g\} \rightarrow c'_t : t \in \{out,\ \lambda\ \}}$
$\underline{\boldsymbol{pi}_{1,0}\ \{\downarrow g,\ \uparrow g\} \rightarrow\ \ \lambda,}$
$\underline{\boldsymbol{pi}_{1,0}\ \{\downarrow g,\ \uparrow g_i\ \} \rightarrow s\{\downarrow g\}\ \boldsymbol{pi}_{1,0}\ \{\downarrow g,\ \uparrow g_i\ \}}$
$\underline{,\ s \in \boldsymbol{S}_V \cup \{s_d :\ s \in \boldsymbol{S}_V \wedge\ d \in \{out\}\}}$
$\underline{\}}$
$\}$

Notice that these semantic actions have two main purposes: to compute the number of production rules of the region to be used as the second index of the corresponding non terminal symbols (p_{jk}) and to complete the grammar with the definition of $pd_{j,k}$ and $pi_{j,k}$ that represent respectively the left and right hand sides of the production rules.

Figure 6 shows the derivation tree before modifying the grammar to force the catalyst to appear in the right hand side of $p_{1,2}$.

Figure 7 shows the grammars (i) and (h).

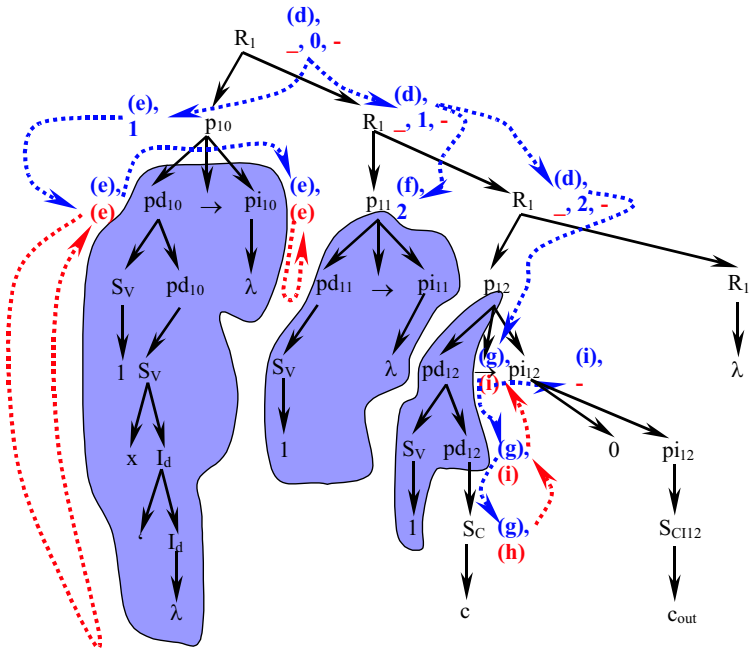


Fig. 6. Derivation tree before processing $\mathrm{pi}_{1,2}$.

Notice that (i), that becomes the initial grammar for $pi_{1,2}$, has replaced

$$pi_{1,2}\{\downarrow g, \uparrow g\} \rightarrow \lambda$$

by

$$pi_{1,2}\{\downarrow g, \uparrow g_i\} \rightarrow S_{CI,1,2}\{\downarrow g, \uparrow g\}.$$

Given that $pi_{1,2}\{\downarrow g, \uparrow g\} \rightarrow \lambda$ was the only way to finish the process of the non-terminal $pi_{1,2}$ (its other rules are recursive), (i) ensures that $S_{CI,1,2}$ appears exactly once in the last derivation associated to $pi_{1,2}$, and hence (after finishing $pi_{1,2}$ the complete process of rule $p_{1,2}$ also finishes) it is unnecessary to modify the grammar to force that non-terminal $S_{CI,1,2}$does not appear again.

The last rule applied to derive the output region $(o \rightarrow 2)$ is one of the possibilities included by (i) and finishes the derivation of the example.

4 Conclusions and future research

This paper uses Christiansen grammars to formally describe an interesting kind of P systems, those that simulate logical functions by means of delay symbols and two mobile catalysts. In this way, our work suggests a formal description of P

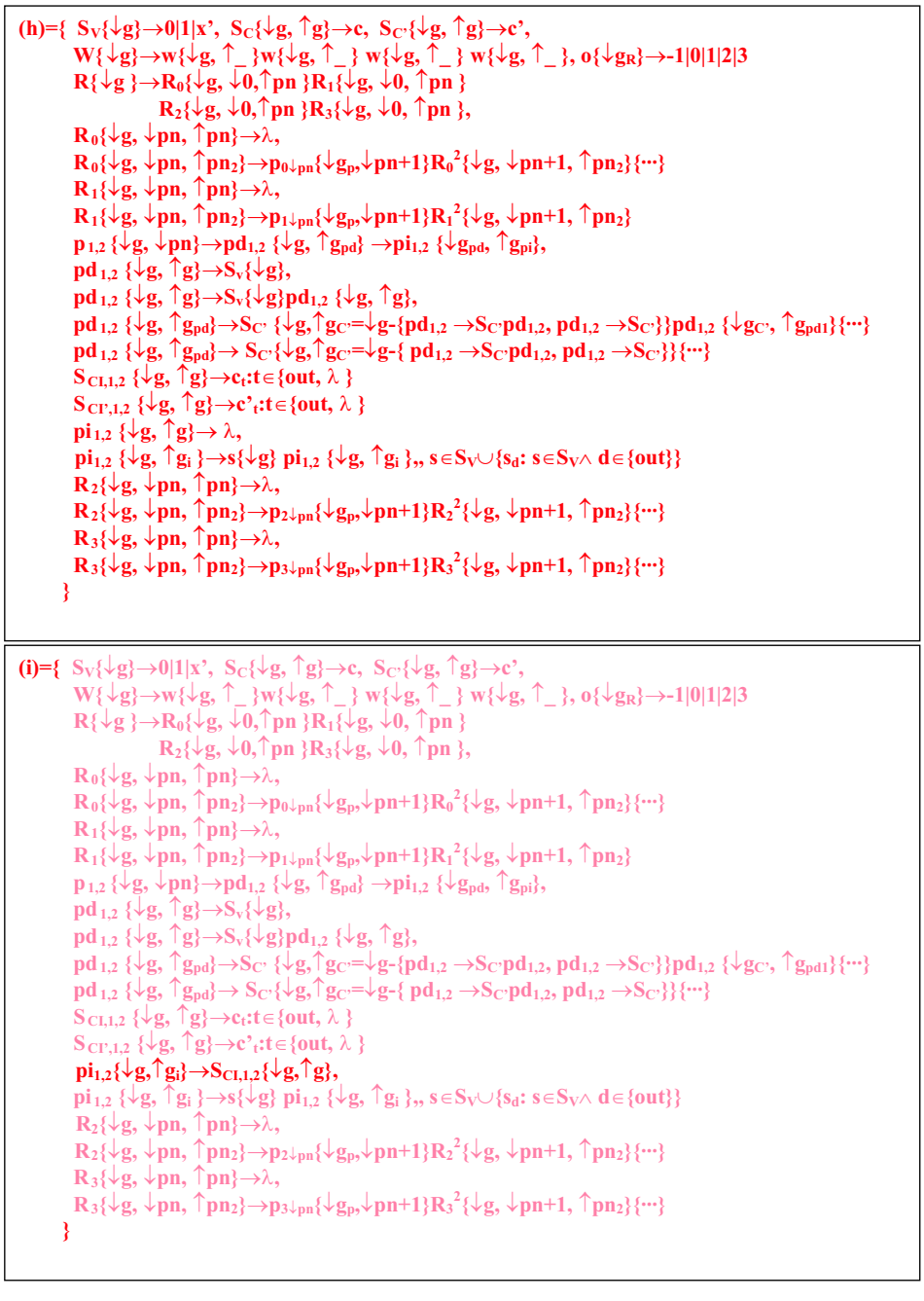


Fig. 7. Grammars of Figure 6.

systems constructions. In fact, it seems possible to use the techniques proposed in this work to translate the hierarchy of the components of any P system into Christiansen grammar mechanisms.

Our future work will extend this paper in the following lines:

- This proposal will be used in the automatic design of P systems that simulate logical functions by means of CGE (Christiansen grammar evolution).
- Christiansen grammars (and CGE) will be applied to other types of P systems, even to P systems without restrictions.

References

1. R. Ceterchi, D. Sburlan: Simulating Boolean circuits with P systems. *Membrane Computing, International Workshop, WMC 2003. Tarragona, Spain, July, 17-22, 2003, Revised Papers*, LNCS 2933, Springer-Verlag, Berlin, 2004, 104–122.
2. A.L.A. Dalhoum, A. Ortega, M. Alfonseca: Cellular automata equivalent to PD0L systems. *Proc. of International Arab Conference on Information Technology* (ACIT 2003), December 20-23, 2003, Alexandria, Egypt, 819–825.
3. A.L.A. Dalhoum, A. Ortega, M. Alfonseca: Cellular automata equivalent to D0L systems. *3rd WSEAS International Conference on Systems Theory and Scientific Computation, Special Session on Cellular Automata and Applications*, November 15-17, 2003, Rodas, Greece.
4. A. Ortega, M. Cruz Echeandía, M. Alfonseca: Christiansen grammar evolution: Grammatical evolution with semantic. Submitted, 2004.
5. A. Ortega, A.L.A. Dalhoum, M. Alfonseca: Grammatical evolution to design fractal curves with a given dimension. *IBM Jr. of Res. and Dev.* (SCI JCR 2.560), 47, 4 (2003), 483–493.
6. A. Ortega, A.L.A. Dalhoum, M. Alfonseca: Cellular automata equivalent to DIL Systems. *Proc. of 5th Middle East Symposium on Simulation and Modelling* (MESM 2003), Eurosim, January 5-7, 2004, Sharjah, United Arabic Emirates, 120–124.
7. Gh. Păun: Computing with membranes. An introduction. *Bulletin of the EATCS*, 68 (February 1999), 139–152.
8. Gh. Păun: Computing with membranes. *Journal of Computer and Systems Sciences*, 61, 1 (2000), 108–143.
9. Gh. Păun: Computing with membranes: Attacking NP-complete problems. *Unconventional models of Computation, UMC'2K*, Bruxelles, 2000, 94–115.
10. J.N. Shutt: *Recursive Adaptable Grammars.* A thesis submitted to the Faculty of the Worcester Polytechnic Institute in partial fulfilment of the requirements for the degree of Master of Science in Computer Science. August 10, 1993 (enmended December 16, 2003).