Multiset Constraints and P Systems*

Agostino Dovier[1], Carla Piazza[2], and Gianfranco Rossi[3]

[1] Dip. di Informatica, Univ. di Verona. Strada Le Grazie 15, 37134 Verona (Italy). dovier@sci.univr.it

[2] Dip. di Matematica e Informatica, Univ. di Udine. Via Le Scienze 206, 33100 Udine (Italy). piazza@dimi.uniud.it

[3] Dip. di Matematica, Univ. di Parma. Via M. D'Azeglio 85/A, 43100 Parma (Italy). gianfr@prmat.math.unipr.it

Abstract. Multisets are the fundamental data structure of P systems. In this paper we relate P systems with the language and theory for multisets presented in [9]. This allows us, on the one hand, to define and implement P systems using multiset constraints in a constraint logic programming framework, and, on the other hand, to define and implement constraint solving procedures used to test multiset constraint satisfiability in terms of P systems with active membranes. While the former can be exploited to provide a precise formulation of a P system, as well as a working implementation of it, based on a first-order theory, the latter provides a way to obtain a P system for a given problem (in particular, NP problems) starting from a rather natural encoding of its solution in terms of multiset constraints.

1 Introduction

The notion of *multiset* has been recently re-discovered, analyzed, and employed in various areas of Logic and Computer Science. From the point of view of Computer Science, a multiset is nothing but a data structure that can contain elements; differently from sets, the number of occurrences of each element is taken into account and differently from lists, the order of the elements does not matter. This peculiarity makes them suitable for modeling phenomena where resources are generated or consumed. For instance, multisets are used to model linear logic proofs, where tokens are consumed to deduce new ones [24]. Multisets are the fundamental data structure of a number of computational frameworks, such as the Gamma coordination language [3], the Chemical Abstract Machine [4], and P systems modeling *membrane computing* [21]. In all these three approaches, which have been developed with different aims, multisets are viewed as entities containing data (possibly, other multisets), and programs are, basically, collections of *multiset rewriting rules* that can be applied with an high degree of parallelism.

* Partially supported by MURST project *Certificazione automatica di programmi mediante interpretazione astratta.*

Efforts have also recently been put forward for introducing multisets as first-class citizens in programming languages. Their high level of abstraction makes it natural to introduce multisets into *declarative programming paradigms* using constraint (logic) programming languages (e.g., [2, 9]). In order to allow the user to write declarative executable code, the interpreter of the language should be able to find the solutions of (i.e., solving) the formulae (i.e., the constraints) written by the user to describe the problem at hand. The correctness of such a solver must be proved on the basis of the properties that characterize multisets. Thus, it is important to develop a simple (first-order) theory of multisets that forces the desired behavior of the multiset data structure. If the solver works correctly on a minimal theory, then it will work correctly in any consistent extension of this theory. In [9] a minimal theory (called *Bag*) of multisets is developed parametrically together with minimal theories for lists, compact lists, and sets, and a solution to the multiset unification problem is provided (see also [8]). In [10] the results are extended to more complex constraints and the constraint logic programming language $CLP(\mathcal{BAG})$, able to deal with this kind of constraints, is presented.

This paper aims at establishing a connection between P systems and the constraint language of [9] based on multiset processing as the common denominator of the two proposals. The approach adopted is, as usual, to show how to define one formalism in terms of the other one and *vice versa*. Precisely, first we show how to define P systems using the language and the theory for multisets presented in [9]. This provides a precise formulation of P systems inside a first-order theory. Moreover, it allows us to define a translation from P systems into $CLP(\mathcal{BAG})$ programs that can be used as a simple, executable—though sequential—implementation of P systems.

In defining this mapping we will exploit the flexible multiset processing facilities offered by our multiset constraint solver, in particular *multiset unification*. This kind of "semantic" unification can be used not only as a means to compare two multisets, but also as an easy way to arbitrarily select a subset of elements satisfying some property from a given multiset. The possibility offered by our constraint language to deal with also partially specified multisets turns out to be essential to allow a concise definition of P systems inside our logic framework.

On the opposite side, we show how to implement part of the constraint solver of $CLP(\mathcal{BAG})$—namely, the fundamental part dealing with equality constraints (hence, multiset unification)—using P systems. The extension of this implementation to the whole constraint solver of $CLP(\mathcal{BAG})$ is trivial. Thus, once we have a description of the problem expressed as a multiset constraint over $\mathcal{L}_{Bag}$ we also have immediately a corresponding implementation as a P system. This constitutes an interesting viable alternative to obtain P systems algorithms for all those problems (typically, NP-complete problems) which admit a rather "natural" formulation based on multiset constraints.

The paper is organized as follows. First we review the above-mentioned multiset language, the axiomatic theory *Bag*, the constraints allowed in that language, and the constraint logic programming language $CLP(\mathcal{BAG})$ (Section 2). Then, in Section 3 we show how to use the multiset constraint language to define P systems and in Section 4 we use the $CLP(\mathcal{BAG})$ language to provide a simple implementation of P systems. In Section 5 we show, instead, how both the standard unification and the multiset unification algorithms can be encoded using P systems. Finally, some conclusions are drawn in Section 6.

2 Multiset Theory, Constraints, and Language

2.1 The theory *Bag*

The first-order language $\mathcal{L}_{Bag}$ we consider here is based on a signature $\Sigma = \langle \mathcal{F}_{Bag}, \Pi_{Bag} \rangle$ and a denumerable set $\mathcal{V}$ of logical variables. $\mathcal{F}_{Bag}$ is a set of constant and function symbols containing the constant symbol `nil`, the binary multiset constructor $\{\!\!\{ \cdot \,|\, \cdot \}\!\!\}$, plus possibly other (free) constant and function symbols. $\Pi_{Bag} = \{\text{“}=\text{”}, \text{“}\in\text{”}\}$ is the set of predicate symbols.

The distinguishing properties of multisets can be precisely stated by the first-order axiomatic theory *Bag*, defined parametrically in [9]. *Bag* is a *hybrid theory*: the objects it deals with are built out of interpreted as well as uninterpreted symbols. In particular, multisets may contain uninterpreted Herbrand terms as well as other multisets. A multiset can be built starting from any ground uninterpreted Herbrand term—called the *kernel* of the data structure—and then adding to this term the other elements that compose the multiset. We refer to this kind of data structures as *colored* hybrid multisets.

We use simple syntactic conventions and notations for terms denoting multisets. In particular, the multiset $\{\!\!\{ s_1 \,|\, \{\!\!\{ s_2 \,|\, \cdots \{\!\!\{ s_n \,|\, t \}\!\!\} \cdots \}\!\!\} \}\!\!\}$ will be denoted by $\{\!\!\{ s_1, \ldots, s_n \,|\, t \}\!\!\}$ or simply by $\{\!\!\{ s_1, \ldots, s_n \}\!\!\}$ when t is `nil` (i.e., the empty multiset). We will use capital letters for variables, small letters for constant and function symbols.

Example 1 ($\mathcal{L}_{Bag}$ terms).

1. $\{\!\!\{ a, b, a \}\!\!\}$ (i.e., $\{\!\!\{ a \,|\, \{\!\!\{ b \,|\, \{\!\!\{ a \,|\, \texttt{nil} \}\!\!\} \}\!\!\} \}\!\!\}$) is a term in $\mathcal{L}_{Bag}$ which, intuitively, denotes the multiset containing two occurrences of the element a and one occurrence of the element b;
2. $\{\!\!\{ a \,|\, X \}\!\!\}$ is a term in $\mathcal{L}_{Bag}$ which, intuitively, denotes any multiset containing at least one occurrence of the element a (in other words, it denotes a *partially specified multiset*);
3. $\{\!\!\{ \{\!\!\{ a \}\!\!\}, \{\!\!\{ b \}\!\!\} \}\!\!\}$ is a $\mathcal{L}_{Bag}$ term which, intuitively, denotes a multiset containing two nested singleton multisets.
4. $\{\!\!\{ a, b \,|\, c \}\!\!\}$, where a, b, c are terms built from constant symbols is a multiset containing (one occurrence of) a and b and colored by the kernel c. We also say that $\mathsf{ker}(\{\!\!\{ a, b \,|\, c \}\!\!\}) = c$.

The axiomatic theory *Bag* is shown in Fig. 1. For a detailed discussion, see [9].

(K)	$\forall x\, \bar{y}$	$(x \notin f(y_1, \ldots, y_n))$	$f \in \mathcal{F}_{Bag}, f \not\equiv \{\!\!\{ \cdot \mid \cdot \}\!\!\}$
(W)	$\forall x\, y\, v$	$(x \in \{\!\!\{ y \mid v \}\!\!\} \leftrightarrow x \in v \vee x = y)$	
(F_1')	$\forall \bar{x}\bar{y}$	$\begin{pmatrix} f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \\ \rightarrow x_1 = y_1 \wedge \cdots \wedge x_n = y_n \end{pmatrix}$	$f \in \mathcal{F}_{Bag}, f \not\equiv \{\!\!\{ \cdot \mid \cdot \}\!\!\}$
(F_2)	$\forall \bar{x}\bar{y}$	$(f(x_1, \ldots, x_m) \neq g(y_1, \ldots, y_m))$	$f, g \in \mathcal{F}_{Bag}, f \not\equiv g$
(F_3)	$\forall x$	$(x \neq t[x])$	
	where $t[x]$ denotes a term, having x as proper subterm		
(F_3^m)	$\forall \bar{x}\bar{y}z$	$\begin{pmatrix} \{\!\!\{ x_1, \ldots, x_m \mid z \}\!\!\} = \{\!\!\{ y_1, \ldots, y_n \mid z \}\!\!\} \\ \rightarrow \{\!\!\{ x_1, \ldots, x_m \}\!\!\} = \{\!\!\{ y_1, \ldots, y_n \}\!\!\} \end{pmatrix}$	$m, n > 0$
(E_k^m)	$\forall \bar{y}\bar{v}$	$\begin{pmatrix} \{\!\!\{ y_1 \mid v_1 \}\!\!\} = \{\!\!\{ y_2 \mid v_2 \}\!\!\} \leftrightarrow \\ (y_1 = y_2 \wedge v_1 = v_2) \vee \\ \exists z\, (v_1 = \{\!\!\{ y_2 \mid z \}\!\!\} \wedge v_2 = \{\!\!\{ y_1 \mid z \}\!\!\}) \end{pmatrix}$	

Fig. 1. The theory *Bag*

We focus here only on the meaning of axiom (E_k^m). The behavior of the multiset constructor symbol $\{\!\!\{ \cdot \mid \cdot \}\!\!\}$ is regulated by the following *equational axiom*

$$\boxed{(E_p^m) \qquad \forall xyz\, \{\!\!\{ x, y \mid z \}\!\!\} = \{\!\!\{ y, x \mid z \}\!\!\}}$$

which, intuitively, states that the order of elements in a multiset is immaterial (*permutativity property*). This axiom forces syntactically different terms to possibly represent the same multiset. However, axiom (E_p^m) lacks in a criterion for establishing disequalities between multisets, even if it is considered together with $KWF_1'F_2F_3$. Therefore, axiom (E_k^m) is introduced in [9] to model (E_p^m) and characterizing the *multiset extensionality* property: *Two (hybrid) multisets are equal if and only if they have the same number of occurrences of each element, regardless of their order.*

2.2 The constraints

A *multiset constraint* is a conjunction of atomic formulae or negation of atomic formulae in $\mathcal{L}_{Bag}$, that is, literals of the form $t_1 = t_2, t_1 \neq t_2, t_1 \in t_2, t_1 \notin t_2$, where t_i is a first-order term in $\mathcal{L}_{Bag}$.

Example 2 ($\mathcal{L}_{Bag}$ constraints).

1. $X \in \{\!\!\{ a, b \mid Y \}\!\!\} \wedge X \notin Y \wedge X \neq a$;
2. $\{\!\!\{ \{\!\!\{ h, h, o \}\!\!\}, \{\!\!\{ o, o \}\!\!\} \}\!\!\} = \{\!\!\{ W, O \}\!\!\} \wedge Y_1 \in W \wedge Y_2 \in W \wedge Y_1 \neq Y_2$ (remember that capital letters denote variables, while small letters represent constant symbols);
3. $\{\!\!\{ e, e, e, e, e \}\!\!\} = \{\!\!\{ X \mid R \}\!\!\} \wedge X \notin \{\!\!\{ e \mid S \}\!\!\}$.

Let E_{Bag} be the equational theory consisting of the equational axiom (E_p^m). The *standard model* $\mathcal{BAG}$ for the theory *Bag* is defined in the following way:

- Let $T(\mathcal{F}_{Bag})$ be the set of first-order ground terms built from symbols in $\mathcal{F}_{Bag}$ (i.e., the ordinary Herbrand Universe).

- The *domain* of the model is the quotient $T(\mathcal{F}_{Bag})/\equiv_{E_{Bag}}$ of $T(\mathcal{F}_{Bag})$ over the smallest congruence relation $\equiv_{E_{Bag}}$ induced by the equational theory E_{Bag} on $T(\mathcal{F}_{Bag})$.
- The *interpretation* of a term t is its equivalence class $[\,t\,]$.
- $=$ is interpreted as the identity on the domain $T(\mathcal{F}_{Bag})/\equiv_{E_{Bag}}$.
- The interpretation of membership is the following: $[\,t\,] \in [\,s\,]$ is `true` if and only if there is a term in $[\,s\,]$ of the form $\{\!\{\, t \mid r \,\}\!\}$ for some term r.

This model is a very important one. As a matter of fact in [10] we prove that if C is a constraint, its satisfiability (unsatisfiability) in $\mathcal{BAG}$ implies that it will be satisfiable (resp., unsatisfiable) in all the models of *Bag* (*correspondence property*).

The notion of satisfiability of a formula φ in a model $\mathcal{A}$ is based on the notion of valuation. A *valuation* σ is a function from a subset of $\mathcal{V}$ to the domain A of $\mathcal{A}$, extended to terms and formulas as usual. σ is said to be a *successful valuation* of a formula φ if $\sigma(\varphi) = $ `true`. Thanks to the correspondence property here we can safely restrict ourselves to consider the satisfiability in the model $\mathcal{BAG}$.

Example 3 (Valuations).

1. Consider the constraints of Example 2. The first two are satisfiable. In particular, any successful valuation for constraint 1 needs to map X to b (actually to the equivalence class $[\,b\,]$ containing b. We admit this slight abuse of notation), while any successful valuation for constraint 2 needs to map W to $\{\!\{\, h, h, o \,\}\!\}$ (water) and O to $\{\!\{\, o, o \,\}\!\}$ (oxygen). The third constraint, instead, is unsatisfiable.
2. Consider the constraint $X = \{\!\{\, Y, Z \,\}\!\} \wedge a \neq Y \wedge \mathtt{nil} \notin Y \wedge b \neq Z$. A successful valuation for it is $X \mapsto \{\!\{\, a, b \,\}\!\}, Y \mapsto b, Z \mapsto a$.

The technique used in [10] to check satisfiability of multiset constraints is based on a constraint solving procedure which is able to reduce non-deterministically any given constraint C to a simplified special form—called the solved form—if and only if C is satisfiable (otherwise, C is reduced to `false`).

A constraint C is in *solved form* if all its literals are in one of the following forms:

- $X = t$ and X does not occur neither in t nor elsewhere in C
- $X \neq t$ and X does not occur in t
- $t \notin X$ and X does not occur in t.

For instance, the constraint of Example 3.2 is in solved form. If C is a $\mathcal{L}_{Bag}$ constraint in solved form, then C is satisfiable in $\mathcal{BAG}$. A solved form constraint therefore can be seen as a finite representation of the possible infinite set of its solutions. From a solved form constraint C it is possible to assign terms to the variables occurring in it in order to obtain a solution. Moreover, all the solutions of C can be obtained in this way.

Solved form constraints can be obtained via suitable rewriting procedures. In Section 5 we will show how to implement some of them into P systems.

Example 4 (3-SAT). Consider the instance of 3-SAT:

$$(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_2 \vee X_3) \wedge (X_1 \vee X_2 \vee \neg X_3)$$

It can be translated into the following $\mathcal{L}_{Bag}$ constraint:

$$\begin{array}{l} \{\!\{ \{\!\{ X_1, Y_1 \}\!\}, \{\!\{ X_2, Y_2 \}\!\}, \{\!\{ X_3, Y_3 \}\!\}, \\ \quad \{\!\{ X_1, Y_2, X_3 \}\!\}, \{\!\{ Y_1, Y_2, X_3 \}\!\}, \{\!\{ X_1, X_2, Y_3 \}\!\} \}\!\} \\ = \{\!\{ \{\!\{ 0, 1 \}\!\}, \{\!\{ 0, 1 \}\!\}, \{\!\{ 0, 1 \}\!\}, \ \{\!\{ 1 \,|\, R_1 \}\!\}, \{\!\{ 1 \,|\, R_2 \}\!\}, \{\!\{ 1 \,|\, R_3 \}\!\} \}\!\} \end{array}$$

where the variables Y_i take the place of $\neg X_i$. Each successful valuation for this constraint is also a solution for the given 3-SAT problem and, vice versa, from each solution for the SAT problem it is immediate to assign values to the variables Y_i and R_i in order to obtain a successful valuation for the constraint. One such solution is, for instance:

$$X_1 = 1 \wedge X_2 = 1 \wedge X_3 = 1$$

(and $Y_1 = 0, Y_2 = 0, Y_3 = 0, R_1 = \{\!\{ 0, 1 \}\!\}, R_2 = \{\!\{ 0, 0 \}\!\}, R_3 = \{\!\{ 0, 1 \}\!\}$).

2.3 The language $CLP(\mathcal{BAG})$

The availability of a constraint solver which is able to prove satisfiability of multiset constraints in a selected structure makes it immediate to define a CLP language dealing with multiset constraints of the kind we have considered so far. This language is obtained as an instance of the general CLP scheme [16, 17] by instantiating it over the language, the theory, and the structure of multisets presented in the previous sections.

The resulting language—called $CLP(\mathcal{BAG})$—can be viewed as a natural extension of traditional logic programming languages, e.g., Prolog. The syntactic form of a $CLP(\mathcal{BAG})$ program, as well as its operational, algebraic, and logical semantics are those of the general CLP scheme [16, 17] suitably instantiated on the specific constraint domain of multisets. This is very similar to what is described in greater detail in [12] for the case of sets. A working prototype of the CLP language which embeds both sets and multisets is available at [23].

A $CLP(\mathcal{BAG})$ *program* P is a finite set of rules of the form

$$A : -c, \bar{B}.$$

where A is an atomic formula (the *head* of the rule), c is a $\mathcal{L}_{Bag}$-constraint and $\bar{B}$ is a (possibly empty) conjunction of atomic formulae. A *goal* is a rule with empty head. As usual, we assume that all the variables in a clause are universally quantified in front of the clause itself.

Example 5 ($CLP(\mathcal{BAG})$ program). The following is a $CLP(\mathcal{BAG})$ program that defines a predicate `once(X,M)` which is true if `X` occurs exactly once in the multiset `M`:

$$\begin{array}{l} \mathtt{once}(\mathtt{X}, \{\!\{ \mathtt{X} \,|\, \mathtt{R} \}\!\}) : - \\ \quad \mathtt{X} \notin \mathtt{R}. \end{array}$$

A sample successful goal for this program is $: -\ \mathtt{once}(\mathtt{b}, \{\!\{ \mathtt{a}, \mathtt{b}, \mathtt{a} \}\!\})$.

Note that clause selection in a $CLP(\mathcal{BAG})$ program is performed using multiset unification, that is two multisets are compared disregarding the order of their elements. Thus, for instance, unifying $\{\!\!\{\, \mathtt{X} \,|\, \mathtt{R} \,\}\!\!\}$ and $\{\!\!\{\, \mathtt{a}, \mathtt{b}, \mathtt{a} \,\}\!\!\}$ will non-deterministically bind X to all possible elements of the second multiset and R to the corresponding multisets composed of the remaining elements (that is, $\mathtt{X} = \mathtt{a} \wedge \mathtt{R} = \{\!\!\{\, \mathtt{b}, \mathtt{a} \,\}\!\!\}$, $\mathtt{X} = \mathtt{b} \wedge \mathtt{R} = \{\!\!\{\, \mathtt{a}, \mathtt{a} \,\}\!\!\}$, $\mathtt{X} = \mathtt{a} \wedge \mathtt{R} = \{\!\!\{\, \mathtt{a}, \mathtt{b} \,\}\!\!\}$).

$CLP(\mathcal{BAG})$ defines a simple, yet very expressive, framework that allows us to program using multisets in a very flexible way.

3 Defining P Systems in $\mathcal{L}_{Bag}$

A P system is a computational model based on transitions governed by multiset rewriting rules. P systems have been defined to mimic the evolution of biological systems (see, e.g., [21]). Their definition formalizes the intuitive idea that biological reactions can be seen as computations. In this section we show how P systems can be naturally defined using the first-order language $\mathcal{L}_{Bag}$ and the theory Bag.

3.1 Membrane Structures

The language MS of membrane structures is defined in [21] to be the set of strings over the alphabet $\{$"[", "]"$\}$ generated by the *infinite* set of rules:

- $[\,] \in MS$
- for $n \geq 1$, if $s_1, \ldots, s_n \in MS$, then $[s_1, \ldots, s_n] \in MS$.

A *membrane* is characterized by a pair of open and closed parentheses $[,]$. The *degree* of a membrane structure is the number of membranes in it. An equivalence relation $\sim$ is defined to govern membrane equivalence and an *ad-hoc* notion of *depth* is given. The outer membrane is called the *skin*.

Moving to $\mathcal{L}_{Bag}$, this is equivalent to say that MS is the set $T(\Sigma)$ of first-order ground terms over the signature $\Sigma = \{\mathtt{nil}, \{\!\!\{\, \cdot \,|\, \cdot \,\}\!\!\}\}$. Such set is generated by the *two* rules:

- $\mathtt{nil} \in MS$
- if $s, t \in MS$, then $\{\!\!\{\, s \,|\, t \,\}\!\!\} \in MS$.

Membranes are therefore identified with multisets. $\sim$ is exactly the equivalence relation $\equiv_{E_{Bag}}$ induced on $T(\Sigma)$ by the (E_p^m) permutativity axiom (c.f. Section 2).

To compute the *degree* of a term s it is sufficient to count the number of occurrences of the constant symbol $\mathtt{nil}$ in s. This assertion is justified by the following observations:

- if $\mathtt{nil}$ occurs as *element* of a multiset (i.e., as first argument of a term of the form $\{\!\!\{\, \_ \,|\, \_ \,\}\!\!\}$), such as in $\{\!\!\{\, s_1, \ldots, s_n, \mathtt{nil} \,|\, r \,\}\!\!\}$ then it must be counted as a pair of open and immediately closed brackets.

– Otherwise, if nil occurs as the tail of a multiset (i.e., as the second argument of a term of the form $\{\!\{\,\_\,|\,\_\,\}\!\}$), such as in $\{\!\{\, s_1, \ldots, s_n \,|\, \mathtt{nil} \,\}\!\}$ then it exactly identifies one non-empty multiset. We recall that the multiset $\{\!\{\, s_1, \ldots, s_n \,\}\!\}$ is represented by the term $\{\!\{\, s_1 | \{\!\{\, s_2 | \cdots \{\!\{\, s_n \,|\, \mathtt{nil} \,\}\!\} \cdots \}\!\}\,\}\!\}$.

The notion of *depth* is the well-established notion of *rank* of a (multi)set:

$$rank(s) = \begin{cases} 0 & \text{if } s = \mathtt{nil} \\ 1 + \max\{rank(t) : t \in s\} & \text{otherwise} \end{cases}$$

A compact way to state that a multiset $\{\!\{\, s_1, \ldots, s_m \,\}\!\}$ is included in a multiset $\{\!\{\, t_1, \ldots, t_n \,\}\!\}$ is that of imposing the constraint (R is a logic variable):

$$\{\!\{\, t_1, \ldots, t_n \,\}\!\} = \{\!\{\, s_1, \ldots, s_m \,|\, R \,\}\!\}\,.$$

3.2 Super-Cells

A *super-cell* is a membrane structure s in which each membrane (i.e., each multiset) can contain elements coming from a (possibly denumerable) universe U of other objects (i.e., non-membrane objects) [21]. Each membrane is uniquely identified by a natural number (its *label*).

An *active super-cell* is a super-cell which may contain more than one occurrence of membranes with the same label. Active super-cells are introduced in [22] to model the notion of membrane division.

To naturally represent labeling of membranes in $\mathcal{L}_{Bag}$ we will adopt the notion of *kernel*.[1] We fix the signature Σ to be defined as

$$\Sigma = \{\{\!\{\,\cdot\,|\,\cdot\,\}\!\}\} \cup \mathbb{N}_{\mathtt{nil}} \cup U$$

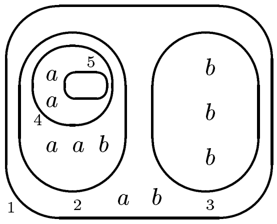
where $\mathbb{N}_{\mathtt{nil}} = \{\mathtt{nil} = \mathtt{nil}_1, \mathtt{nil}_2, \mathtt{nil}_3, \ldots\}$ is the denumerable set of constant symbols (to be used as multiset kernels), and U is a set of constant symbols (the *objects*) disjoint from $\mathbb{N}_{\mathtt{nil}}$.

An *active super-cell* is a ground Σ-term such that for all subterms of the form $\{\!\{\, s_1, \ldots, s_m \,|\, k \,\}\!\}$, where $m > 0$ and k is not of the form $\{\!\{\,\_\,|\,\_\,\}\!\}$, it holds that $k \in \mathbb{N}_{\mathtt{nil}}$ (in other words, symbols from U cannot be used as kernels). A *super-cell* is an active super-cell such that each $\mathtt{nil}_i$ occurs at most once.

Figure 2 shows a super-cell of degree 5 and the corresponding representation as an $\mathcal{L}_{Bag}$ term.

As for membrane structures, *equivalence* between super-cells is multiset equivalence $\equiv_{E_{Bag}}$. The notion of degree, instead, must be refined: the *degree* of an active super-cell s is the number of occurrences of constant symbols of $\mathbb{N}_{\mathtt{nil}}$ in it. Hereafter, we usually refer to super-cells where there is exactly one occurrence of each constant $\mathtt{nil}_1, \ldots, \mathtt{nil}_{degree(s)}$.

[1] As explained in Section 2, if there is at least one constant symbol $c \in \Sigma$, then it is possible to write a term t of the form $\{\!\{\, s_1, \ldots, s_n \,|\, c \,\}\!\}$. t identifies a multiset with the n elements $s_1, \ldots, s_n$ (not necessarily all distinct) and *colored* (labeled) by the constant c. We also say that c is the kernel of t, briefly $\mathsf{ker}(t) = c$.



$$
\begin{array}{ll}
\{\!\!\{\ a, & \\
\quad b, & \\
\quad \{\!\!\{ a, a, b, \{\!\!\{ a, a, \mathtt{nil}_5 \mid \mathtt{nil}_4 \}\!\!\} \mid \mathtt{nil}_2 \}\!\!\}, & \\
\quad \{\!\!\{ b, b, b \mid \mathtt{nil}_3 \}\!\!\} & \mid \mathtt{nil}_1 \}\!\!\}
\end{array}
$$

Fig. 2. A super-cell of degree 5 and the corresponding term

3.3 Transition *P* Systems

We give here the notion of P system using the language $\mathcal{L}_{Bag}$. A *transition P system of degree n* can be defined as a tuple

$$T_n = \langle U, \mu, (R_1, \rho_1), \ldots, (R_n, \rho_n), i_{\mathcal{O}} \rangle$$

where U is a universe of objects, μ is a super-cell of degree n, $i_{\mathcal{O}}$ is the output membrane, and, for all $i = 1, \ldots, n$, R_i is a finite set of evolution rules (to be described below) and ρ_i is a partial order relation over R_i.

A *computation* of T_n is a sequence of super-cells $\mu = \mu_0, \mu_1, \ldots, \mu_m$ where μ_{j+1} is obtained from μ_j by applying one or more evolution rules.

Observe that the n membranes occurring in μ are uniquely identified by their integer label i. Referring to the multiset representation, this means that the multisets in μ are distinguished each other by their kernels, $\mathtt{nil}_i, \ldots, \mathtt{nil}_n$. Hereafter, we will denote these multisets by $m_1, \ldots, m_n$.

An *evolution rule* is a pair of the form $u \to v$ where u is a string of elements of U and v is either of the form v' or $v'\delta$ where v' is a string over

$$(V \times \{\mathsf{here}, \mathsf{out}\}) \cup (V \times \{\mathsf{in}_1, \ldots, \mathsf{in}_n\}).$$

and V is an alphabet. A rule in R_i applies to the multiset m_i. Applying a rule causes some effects to occur either on m_i or on the multiset possibly containing m_i according to the form of the right-hand side of the rule. More precisely, the semantics of evolution rules, expressed in terms of the multiset representation introduced so far, can be described as follows. Let us consider first rules of the form $u \to v'$, that is without δ. Consider a rule in R_i with the following general form:

$$u_1 \ldots u_h \mapsto (v_1, \mathsf{here}) \cdots (v_{k_1}, \mathsf{here})(w_1, \mathsf{out}) \cdots (w_{k_2}, \mathsf{out})(z_1, \mathsf{in}_{x_1}) \cdots (z_{k_3}, \mathsf{in}_{x_{k_3}})$$

The rule can be applied to the multiset m_i if m_i contains (in any order) all the objects $u_1, \ldots, u_h$. In the language of multiset constraints, this fact can be expressed by requiring that the constraint

$$m_i = \{\!\!\{ u_1, \ldots, u_h \mid M \}\!\!\} \land \mathsf{ker}(M) = \mathtt{nil}_i$$

holds, for some multiset M (assume ker is the function returning the kernel of a ground multiset). Applying this rule yields the following effects:

a. **local effect:**
replace m_i in μ by $\{\!\!\{ v_1, \ldots, v_{k_1} \mid M \}\!\!\}$ (if $k_1 = 0$ replace m_i by M).
b. **exit effect:**
if $m_i \in m_j$ then replace m_j in μ by $\{\!\!\{ w_1, \ldots, w_{k_2} \mid m_j \}\!\!\}$; if m_i is the skin then simply remove the elements $w_1, \ldots, w_{k_2}$ from it.
c. **adjacency effect:**
for $j = 1, \ldots, k_3$, if $m_i \in m_{x_j}$ or $m_{x_j} \in m_i$ (i.e. the two membranes are adjacent), then add z_j to m_{x_j}. If for some $j = 1, \ldots, k_3$ the adjacency condition is not fulfilled, the application of the rule is <u>not</u> allowed.

Consider now the scheme of a δ rule (*dissolving* rule):

$$u_1 \ldots u_h \mapsto (v_1, \mathsf{here}) \cdots (v_{k_1}, \mathsf{here})(w_1, \mathsf{out}) \cdots (w_{k_2}, \mathsf{out})(z_1, \mathsf{in}_{x_1}) \cdots (z_{k_3}, \mathsf{in}_{x_{k_3}})\delta$$

After having obtained the local, exit, and adjacency effects as in the rule of the first kind, the δ causes the further effect of destroying m_i and carrying out its elements (unless m_i is the *skin* that cannot be destroyed). Thus, if $m_j = \{\!\!\{ m_i \mid m'_j \}\!\!\}$ (i.e. $m_i \in m_j$) and $m_i = \{\!\!\{ c_1, \ldots, c_\ell \mid \mathtt{nil}_i \}\!\!\}$ then replace m_j by $\{\!\!\{ c_1, \ldots, c_\ell \mid m'_j \}\!\!\}$.

3.4 *P* Systems with Active Membranes

In [22] new rules are allowed to occur in P systems. In particular we are interested here in the possibility of creating a *copy* of a membrane. Thus we consider the following simplified version of the *division rule* [22]

$$u_1 \ldots u_k \to ((v_1 \ldots v_h)(w_1 \ldots w_j))$$

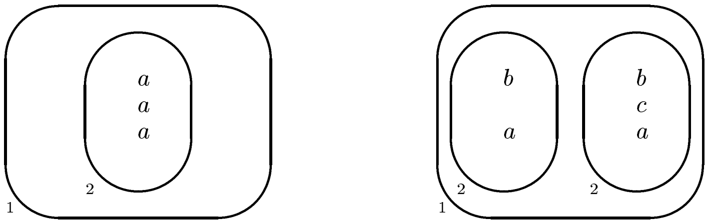
The semantics of this rule, expressed in terms of the multiset representation of super-cells introduced in this section, can be described as follows. The rule can be applied—like other rules—to the multiset m_i if m_i contains $u_1, \ldots, u_k$, that is if the constraint

$$m_i = \{\!\!\{ u_1, \ldots, u_h \mid M \}\!\!\} \wedge \mathsf{ker}(M) = \mathtt{nil}_i$$

holds, for some multiset M. Applying this rule yields the following effects:

- remove elements $u_1, \ldots, u_k$ from m_i; since $m_i = \{\!\!\{ u_1, \ldots, u_k \mid M \}\!\!\}$, M represents what remains of the multiset m_i;
- make two copies of M (both having the same kernel $\mathtt{nil}_i$) and add all the elements $v_1, \ldots, v_h$ to the first copy and $w_1, \ldots, w_j$ to the second copy; that is, in terms of our multiset representation, replace m_i in μ with the two new multisets
$$\{\!\!\{ v_1, \ldots, v_h \mid M \}\!\!\} \qquad \{\!\!\{ w_1, \ldots, w_j \mid M \}\!\!\}.$$

Fig. 3 shows an example of the effect of the application of the membrane division rule. Observe that this rule can be applied to more than one membrane, possibly simultaneously.



$$aa \to (b)\ (bc)$$

Fig. 3. Membrane division

4 Implementing P Systems in $CLP(\mathcal{BAG})$

We show that a transition P system can be written in an easy way as a program in a CLP language which supports a decision procedure for multiset constraints. In particular, we show how to translate a P system into a $CLP(\mathcal{BAG})$ program (see Section 2.3).

The main predicate governing the application of the evolution rules is:

```
p_interpreter(Membrane, Membrane) :-
    not rule(Membrane, _).
p_interpreter(MembraneIn, MembraneOut) :-
    rule(MembraneIn, MembraneInt),
    p_interpreter(MembraneInt, MembraneOut).
```

Note that the multiset transformation process terminates as soon as no rule can be applied successfully to the input multiset (first argument of `p_interpreter`). Each evolution rule of each set R_i of the given P system can be (automatically) translated into a single $CLP(\mathcal{BAG})$ clause, according to the multiset-based representation of membranes and of computations described in Section 3.3. As an example, consider the rule:

$$u_1 \dots u_h \mapsto (v_1, \mathsf{here}) \cdots (v_{k_1}, \mathsf{here})(w_1, \mathsf{out}) \cdots (w_{k_2}, \mathsf{out}).$$

Assume that m_i is not the outer membrane (in that case the clause must be slightly changed):

$$
\begin{array}{l}
\mathtt{rule(MembrIn, MembrOut) :-} \\
\quad \mathtt{choose(MembrIn, nil_i, M_i, M_j),} \\
\quad \mathtt{M_i} = \{\!\!\{ \mathtt{u_1, \dots, u_h} \mid \mathtt{M'_i} \}\!\!\}, \\
\quad \mathtt{replace(MembrIn, M_i, } \{\!\!\{ \mathtt{v_1, \dots, v_{k_1}} \mid \mathtt{M'_i} \}\!\!\}, \mathtt{M'_j),} \\
\quad \mathtt{replace(MembrIn, M_j, } \{\!\!\{ \mathtt{w_1, \dots, w_{k_2}} \mid \mathtt{M'_j} \}\!\!\}, \mathtt{MembrOut).}
\end{array}
$$

$\mathtt{M_j}$ is the membrane such that $\mathtt{M_i} \in \mathtt{M_j}$. Multiset unification in the second sub-goal is used to test applicability of this rule to $\mathtt{M_i}$ (find whether $\mathtt{u_1, \dots, u_h}$ belong to

$\mathtt{M_i}$). $\mathtt{v_1}, \ldots, \mathtt{v_{k_1}}$ are the here elements. The last sub-goal updates the membrane containing $\mathtt{M_i}$ using the out elements $\mathtt{w_1}, \ldots, \mathtt{w_{k_2}}$.

The auxiliary predicates `choose` and `replace` can be simply programmed in $CLP(\mathcal{BAG})$ as follows.

$\mathtt{choose(Membrane, Kernel, M_i, M_j)}$ chooses a multiset $\mathtt{M_i}$ in `Membrane` with kernel `Kernel` and the multiset $\mathtt{M_j}$ that contains it.

$$\begin{array}{l}
\mathtt{choose(\{\!\![\, M_i \,|\, R \,]\!\!\}, Kernel, M_i, \{\!\![\, M_i \,|\, R \,]\!\!\}) :-} \\
\quad \mathtt{kernel(M_i, Kernel).} \\
\mathtt{choose(\{\!\![\, M \,|\, R \,]\!\!\}, Kernel, M_i, M_j) :-} \\
\quad \mathtt{choose(M, Kernel, M_i, M_j).}
\end{array}$$

The predicate $\mathsf{replace}(A, B, C, D)$ finds one occurrence of B in A and replaces it with C obtaining D.

$$\begin{array}{l}
\mathtt{replace(A, A, C, C).} \\
\mathtt{replace(\{\!\![\, B \,|\, R \,]\!\!\}, B, C, \{\!\![\, C \,|\, R \,]\!\!\}).} \\
\mathtt{replace(\{\!\![\, M \,|\, R \,]\!\!\}, B, C, \{\!\![\, M' \,|\, R \,]\!\!\}) :-} \\
\quad \mathtt{M} \neq \mathtt{B,} \\
\quad \mathtt{replace(M, B, C, M').}
\end{array}$$

The predicate `kernel(M,K)` stating that the kernel of the multiset `M` is `K` can be programmed by:

$$\begin{array}{l}
\mathtt{kernel(nil_1, nil_1).} \\
\quad \vdots \quad \vdots \quad \vdots \\
\mathtt{kernel(nil_n, nil_n).} \\
\mathtt{kernel(\{\!\![\, A \,|\, B \,]\!\!\}, K) :-} \\
\quad \mathtt{kernel(B, K).}
\end{array}$$

To implement the division rule it is convenient to introduce also a predicate $\mathsf{replace2}(A, B, C1, C2, D)$ whose structure is the same as that of replace but it replaces B in A with both $C1$ and $C2$.

We give here an example running on the interpreter of $CLP(\mathcal{BAG})$ [23] of a P system computing the function n^2.

Example 6. Consider the P system such that $U = \{a, b\}$,

$$\mu = \{\!\![\, \{\!\![\, a, \underbrace{b, \ldots, b}_{n} \,|\, \mathtt{nil}_2 \,]\!\!\} \,|\, \mathtt{nil}_1 \,]\!\!\}$$

The set of rules R_1 is empty, while the rules in R_2 are:

$$\begin{array}{ll}
(1) & abb \mapsto ((bb)(ab)) \\
(2) & ab \mapsto b \\
(3) & bb \mapsto b(b, \mathsf{out})(b, \mathsf{out}) \\
(4) & b \mapsto (b, \mathsf{out})\delta
\end{array}$$

ordered as (1) > (2) > (3) > (4). (As common in P systems, we assume that $u \mapsto (v_1, \mathsf{here})$ can be abbreviated as $u \mapsto v_1$). At the end of each computation, n^2 occurrences of the object b are present in the outer membrane. All other membranes in it have disappeared. Basically, we have used the well-known fact that $n^2 = \sum_{i=1}^{n}(2i-1)$. We first produce membranes containing $b^n, b^{n-1}, \ldots, b^1$. Then we move out the tokens, in accordance with the mathematical law. The following $CLP(\mathcal{BAG})$ clauses can be used to simulate these rules:

$$
\begin{array}{l}
\%\%\%(1)\\
\mathtt{rule}(\mathtt{MembrIn}, \mathtt{MembrOut}) :-\\
\quad \mathtt{choose}(\mathtt{MembrIn}, \mathtt{nil}_2, \mathtt{M_i}, \mathtt{M_j}),\\
\quad \mathtt{M_i} = \{\!\!\{ \mathtt{a}, \mathtt{b}, \mathtt{b} \mid \mathtt{M'_i} \}\!\!\},\\
\quad \mathtt{replace2}(\mathtt{MembrIn}, \mathtt{M_i}, \{\!\!\{ \mathtt{b}, \mathtt{b} \mid \mathtt{M'_i} \}\!\!\}, \{\!\!\{ \mathtt{a}, \mathtt{b} \mid \mathtt{M'_i} \}\!\!\}, \mathtt{MembrOut}).\\
\%\%\%(2)\\
\mathtt{rule}(\mathtt{MembrIn}, \mathtt{MembrOut}) :-\\
\quad \mathtt{choose}(\mathtt{MembrIn}, \mathtt{nil}_2, \mathtt{M_i}, \mathtt{M_j}),\\
\quad \mathtt{M_i} = \{\!\!\{ \mathtt{a}, \mathtt{b} \mid \mathtt{M'_i} \}\!\!\},\\
\quad \mathtt{replace}(\mathtt{MembrIn}, \mathtt{M_i}, \{\!\!\{ \mathtt{b} \mid \mathtt{M'_i} \}\!\!\}, \mathtt{MembrOut}).\\
\%\%\%(3)\\
\mathtt{rule}(\mathtt{MembrIn}, \mathtt{MembrOut}) :-\\
\quad \mathtt{choose}(\mathtt{MembrIn}, \mathtt{nil}_2, \mathtt{M_i}, \mathtt{M_j}),\\
\quad \mathtt{M_i} = \{\!\!\{ \mathtt{b}, \mathtt{b} \mid \mathtt{M'_i} \}\!\!\},\\
\quad \mathtt{replace}(\mathtt{MembrIn}, \mathtt{M_i}, \{\!\!\{ \mathtt{b} \mid \mathtt{M'_i} \}\!\!\}, \mathtt{MembrInt}),\\
\quad \mathtt{replace}(\mathtt{MembrIn}, \mathtt{M_j}, \{\!\!\{ \mathtt{b}, \mathtt{b} \mid \mathtt{MembrInt} \}\!\!\}, \mathtt{MembrOut}).\\
\%\%\%(4)\\
\mathtt{rule}(\mathtt{MembrIn}, \mathtt{MembrOut}) :-\\
\quad \mathtt{choose}(\mathtt{MembrIn}, \mathtt{nil}_2, \mathtt{M_i}, \mathtt{M_j}),\\
\quad \mathtt{M_i} = \{\!\!\{ \mathtt{b} \mid \mathtt{M'_i} \}\!\!\},\\
\quad \mathtt{M_j} = \{\!\!\{ \mathtt{M_i} \mid \mathtt{MembrInt} \}\!\!\}, \qquad \delta \text{ removes } M_i \text{ from } M_j\\
\quad \mathtt{replace}(\mathtt{MembrIn}, \mathtt{M_j}, \{\!\!\{ \mathtt{b} \mid \mathtt{MembrInt} \}\!\!\}, \mathtt{MembrOut}).
\end{array}
$$

The $CLP(\mathcal{BAG})$ program that simulates the given P system, therefore, is composed by the clauses for predicates `p_interpreter` and `rule` along with those for the auxiliary predicates `choose`, `replace`, and `replace2`. A goal for this program can be:

$$:-\mathtt{p\_interpreter}(\{\!\!\{ \{\!\!\{ \mathtt{a}, \mathtt{b}, \mathtt{b}, \mathtt{b} \mid \mathtt{nil}_2 \}\!\!\} \mid \mathtt{nil}_1 \}\!\!\}, \mathtt{MembrOut}),$$

from which we get the computed answer

$$\{\!\!\{ \mathtt{b}, \mathtt{b}, \mathtt{b}, \mathtt{b}, \mathtt{b}, \mathtt{b}, \mathtt{b}, \mathtt{b}, \mathtt{b} \mid \mathtt{nil}_1 \}\!\!\}$$

Note that rule ordering is obtained by exploiting the corresponding $CLP(\mathcal{BAG})$ clause ordering. This solution works correctly for the first successful computation; however, if other alternative computations are attempted through backtracking, clause ordering is no longer sufficient to guarantee rule ordering. To force the desired ordering is always preserved one could add a unique identifier to each rule and slightly modify the definition of predicate `p_interpreter` so as to apply rules according to a list of rule identifiers which explicitly states the rule ordering for each membrane.

5 Implementing Constraint Solvers by P Systems

Given a first-order language $\mathcal{L}$, an interpretation structure $\mathcal{A}$, and the corresponding theory, we show how to implement the algorithm for deciding satisfiability in $\mathcal{A}$ of a $\mathcal{L}$-constraint C using a transition P system derived from C. Basically, the condition for this translation to be fully automatized is that the constraint solving algorithm is given via rewriting rules (a condition often fulfilled in constraint decision procedures). In this section we focus our attention on the solution of equality constraints, that is on the unification problem. We show how to implement both standard and extended unification (namely, multiset unification) algorithms using P systems.

5.1 Standard Unification

We start from the classical first-order unification problem—c.f., e.g., [15, 20, 1]. Let Σ be a first-order signature and C, $s_1 = t_1 \wedge \cdots \wedge s_n = t_n$, be a constraint composed by conjunctions of equations (i.e., an *equation system*). The unification problem, in its decision version, consists in checking if there is a *substitution* θ (a function from the variables occurring in C to the set of first-order terms) such that for all $i = 1, \ldots, n$, $\theta(s_i)$ and $\theta(t_i)$ are syntactically equal. A survey of the problem with the main results can be found in [1].

Let $T(C)$ be the transition P system

$$\langle U, \mu, (R_1, \rho_1), (R_2, \rho_2), i_{\mathcal{O}} \rangle$$

where:

- the universe U is the *set of all ground equations*, $\{s = t : s, t \in T(\Sigma)\}$
- $\mu = \{\!\!\{\, m_= \,|\, \mathtt{nil}_1 \,\}\!\!\}$
- $m_= = \{\!\!\{\, s_1 = t_1, \ldots, s_n = t_n \,|\, \mathtt{nil}_2 \,\}\!\!\}$
- $i_{\mathcal{O}} = 1$ (i.e. the output membrane is the skin)
- R_1 and R_2 are the sets of evolution rules for multisets with kernels $\mathtt{nil}_1$ and $\mathtt{nil}_2$, respectively, that implement the unification test.

Evolution rules for R_2 are obtained from the non-deterministic unification algorithm Unify [15, 20] and are shown in Figure 4. Actually, each rule is a meta-rule consisting of all its infinite possible instances. However, only a *finite* number of instances depending on C is needed, as explained below.

R_1, instead, is assumed to be empty in this case.

Classical results on Herbrand's non-deterministic algorithm ensure termination of the computation of the transition P system independently of the ordering chosen when rules are applied. Moreover, each non-deterministic execution

(r_1)	$X = X$	$\mapsto \varepsilon$ *(remove tautologies)*
(r_2)	$t = X$ t is not a variable	$\mapsto X = t$ *(i.e.,* $(X = t, \mathsf{here})$*)*
(r_3)	$X = t \;\; X = s$ X does not occur in t	$\mapsto X = t \;\; t = s$
(r_4)	$X = t \;\; Y = X$ X does not occur in t	$\mapsto X = t \;\; Y = t$
(r_5)	$X = t$ $X \not\equiv t$ and X occurs in t	$\mapsto (\mathtt{false}, \mathsf{out})$
(r_6)	$f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)$ $f \not\equiv g$	$\mapsto (\mathtt{false}, \mathsf{out})$
(r_7)	$f(s_1, \ldots, s_m) = f(t_1, \ldots, t_m)$	$\mapsto s_1 = t_1 \;\; \cdots \;\; s_m = t_m$

Fig. 4. Unification as evolution rules (R_2)

is equivalent (equivalence of m.g.u.'s modulo variants) [1]. Thus, no ordering is needed between rewriting rules to ensure termination, and all the possible non-deterministic computations of the P system can be considered equivalent.

By analyzing the computations of the P system it is easy to prove that the dimension of the terms involved can be bounded by the size of the initial constraint C used to define $T(C)$. More precisely, let *size* and *height* be defined as follows:

$$\mathit{size}(t) = \begin{cases} 0 & \text{if } t \text{ is a variable} \\ 1 + \sum_{i=1}^{n} \mathit{size}(t_i) & \text{if } t = f(t_1, \ldots, t_n) \\ \sum_{i=1}^{m} \mathit{size}(s_i) + \mathit{size}(t_i) & \text{if } t = s_1 = t_1 \wedge \cdots \wedge s_m = t_m \end{cases}$$

$$\mathit{height}(t) = \begin{cases} 0 & \text{if } t \text{ is a variable} \\ 1 + \max_{i=1}^{n} \mathit{height}(t_i) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Proposition 1. *Let $s = t$ be an equation occurring at some point during the computation of* $\mathsf{Unify}(C)$*. Then $\mathit{height}(s) \leq \mathit{size}(C)$ and $\mathit{height}(t) \leq \mathit{size}(C)$.*

Proof. Immediate, by classical results. □

Thus, the number of rules in R_2 can be chosen to be finite. Let S be the set of terms built using the function, constant, and variable symbols occurring in C of height bounded by $\mathit{size}(C)$. R_2 is chosen as the set of instances of the meta-rules of Fig. 4 over the set S.

Proposition 2. *Let $\mu = \mu_0, \ldots, \mu_m$ be a terminating computation of the P system $T(C)$. C does not admit solutions if and only if $\mathtt{false} \in m_1 \in \mu_m$.*

Proof. The evolution of m_2 mimics the execution of standard unification algorithm Unify (that is terminating and correct). A successful termination does not produce `false`. Otherwise, `false` is stored in the membrane m_1 and never removed. □

Observe that the P system obtained in this way, although generated by an instance C, is able to deal with a wide family of equation systems as inputs. Precisely, all input systems with variables, function and constant symbols from those of C, and with initial size bounded by that of C.

5.2 Multiset Unification

When one or more function symbols from Σ have some semantic properties, the unification problem becomes more complex. For instance, in the case of multiset unification (as it happens in Bag) we use a binary function symbol whose semantics is governed by axiom (E_p^m); in this case the unification problem turns out to be NP-complete [9]. We show here how to map the multiset unification algorithm of [9] to a P system with active membranes (using membrane division). This allows one, for instance, to obtain an alternative (polynomial-time) implementation of SAT using P systems, since SAT can be reduced to multiset unification (cf. Example 4).

Let $\Sigma = \{\!\{\mathtt{nil}, \{\!\{\cdot \mid \cdot\}\!\}, \dots\}\!\}$ be the signature of $\mathcal{L}_{Bag}$. The transition system $T(C)$ is the same as in Section 5.1, save for the rewriting rules R_1 and R_2. They are chosen in order to simulate the multiset unification algorithm of [9] and are reported in Fig. 5 and Fig. 6. Moreover, some ordering on the rules of R_2 should be imposed to ensure termination in all possible cases as described in [9].

(r_1)	$X = X$	$\mapsto \varepsilon$ *(remove tautologies)*
(r_2)	$t = X$ t is not a variable	$\mapsto X = t$ *(i.e., $(X = t, \mathsf{here})$)*
(r_3)	$X = t\ \ X = s$ X does not occur in t	$\mapsto X = t\ \ t = s$
(r_4)	$X = t\ \ Y = X$ X does not occur in t	$\mapsto X = t\ \ Y = t$
(r_5)	$X = t$ $X \not\equiv t$ and X occurs in t	$\mapsto \delta$
(r_6)	$f(s_1, \dots, s_m) = g(t_1, \dots, t_n)$ $f \not\equiv g$	$\mapsto \delta$
(r_7)	$f(s_1, \dots, s_m) = f(t_1, \dots, t_m)$ $f \not\equiv \{\!\{\cdot \mid \cdot\}\!\}$	$\mapsto s_1 = t_1 \ \cdots \ s_m = t_m$
(r_8)	$\{\!\{t_1, \dots, t_m \mid N\}\!\} = \{\!\{s_1, \dots, s_n \mid N\}\!\}$	$\mapsto \{\!\{t_1, \dots, t_m\}\!\} = \{\!\{s_1, \dots, s_n\}\!\}$
(r_9)	$\{\!\{t \mid s\}\!\} = \{\!\{t' \mid s'\}\!\}$	$\mapsto$ $((t = t'\ \ s = s')$ $(s = \{\!\{t' \mid N\}\!\}\ \ \{\!\{t \mid N\}\!\} = s'))$

Fig. 5. Multiset Unification as evolution rules (R_2)

Consider the rules for membranes with label 2 (i.e., multisets with kernel $\mathtt{nil}_2$). Rules r_1–r_4 are the same as for the standard unification case. Rule r_7 is now restricted to non-multiset terms. Rules r_8 and r_9 treat the case of equalities

(r_1)	$s = t\,\}$	$\mapsto \varepsilon$ *(remove equations)*

Fig. 6. Evolution rules for the skin (R_1)

between two multisets. These two rules introduce a slight notational ambiguity. As a matter of fact, the multisets occurring in those rules are not membranes but multiset terms in the first-order language in which the equations are written. This kind of confusion should be avoided by using different function symbols for building multiset terms in equations (e.g., $[\![\;]\!]$ and $[\![\cdot\,|\,\cdot]\!]$). Rule r_8 deals with the special case of two non-ground multiset terms with the same tail variable. Rule r_9, instead, deals with the general case. It requires the ability to express a (don't know) non-deterministic choice between two possible computations, corresponding to the two alternatives in the right-hand part of axiom (E_k^m). It is implemented by membrane division.

Rules r_5 and r_6 dealing with failures cause now the deletion of the membrane with label 2 in which the condition is fulfilled, putting all its remaining elements (if any) in the outer membrane. Therefore, equalities can be present in membrane 1 as effect of deletion of membranes with label 2. Rule r_1 in R_1 removes all of them. At the end of the execution, membrane 1 is either empty or it contains some membranes with label 2, each one corresponding to a successful non-deterministic branch of the computation.

Proposition 3. *Let $\mu = \mu_0, \ldots, \mu_m$ be a terminating computation of the P system $T(C)$. C is unsatisfiable if and only if m_1 is empty.*

Proof. Immediate from the fact that we mimic the algorithm in [9]. □

As an example consider the equality constraint C that models the instance of 3-SAT of Example 4. The P system $T(C)$ which tests the satisfiability of this constraint can be seen as (yet another) implementation of an algorithm to solve the 3-SAT problem using P systems. Computations of $T(C)$ run in non-deterministic polynomial time. If membrane division is implemented in an effective way, this yields a polynomial time method for solving 3-SAT (see also [22]).

The constraint solver of $CLP(\mathcal{BAG})$ (see [10]) deals not only with equality constraints, but also with disequality, membership, and not-membership constraints. The same technique used for implementing unification as evolution rules can be applied almost unaltered to implement also the rewriting procedures for the other kinds of constraints. Thus the whole constraint solver of $CLP(\mathcal{BAG})$ can be implemented as a P system.

Therefore, any problem which can be easily expressed as a multiset constraint (containing equalities, disequalities, memberships, and negation of memberships) can be automatically implemented using a P system.

6 Conclusions

We have described how to use the language $\mathcal{L}_{Bag}$ to define P systems in a fairly natural way. This gives a precise formulation of P systems inside a first-order theory. Moreover, we have shown that the CLP language $CLP(\mathcal{BAG})$ is particularly well-suited to simulate P systems, thanks to its capabilities of manipulating multisets in a very general and flexible way. Then, showing that the constraint solvers of $CLP(\mathcal{BAG})$ can be implemented by P systems (endowed with the membrane division rule), we have suggested an alternative way to encode, in a rather natural way, NP problems which are easily described using $\mathcal{L}_{Bag}$ constraints into P systems.

As a future work it could be interesting to analyze if using a Prolog system that supports some form of parallelism (e.g., [13]) as the execution environment where to embed our CLP language would provide a way to enhance also our simulation of P systems in the direction of a real parallel execution.

References

1. K. R. Apt. *From Logic Programming to Prolog.* International Series in Computer Science. Prentice Hall, 1997.
2. P. Arenas-Sánchez, F. J. López-Fraguas, M. Rodríguez-Artalejo Embedding Multiset Constraints into a Lazy Functional Logic Language In C. Palamidessi, H. Glaser, K. Meinke, editors, *Principles of Declarative Programming*, LNCS 1490, Springer-Verlag, pp. 429–444, 1998.
3. J. Bânatre and D. Le Métayer. Programming by Multiset Transformation. *Communications of the ACM*, 36(1):98–111. January 1993.
4. G. Berry and G. Boudol. The Chemical Abstract Machine. *Theoretical Computer Science*, vol. 96 (1992) 217–248.
5. J.-P. Bodeveix, C. Percebois, S. Majoul. An Object-Oriented Coordination Model based on Multiset Rewriting. In Proc. of *Ninth International Conference on Parallel and Distributed Computing Systems.* Dijon, France, 1996.
6. I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Relating Strands and Multiset Rewriting for Security Protocol Analysis In P. Syverson, ed., *13th IEEE Computer Security Foundations Workshop—CSFW'00*, pp. 35–51, 2000.
7. P. Ciancarini, D. Fogli, and M. Gaspari. A Logic Language Based on GAMMA-like Multiset Rewriting. In R. Dyckhoff, H. Herre, P. Schroeder-Heister eds., *Extensions of Logic Programming*, 5th International Workshop, LNCS 1050, 1996, pp. 83–101.
8. E. Dantsin and A. Voronkov. A Nondeterministic Polynomial-Time Unification Algorithm for Bags, Sets and Trees. In W. Thomas ed., *Foundations of Software Science and Computation Structure*, LNCS Vol. 1578, pages 180–196, 1999.
9. A. Dovier, A. Policriti, and G. Rossi. A uniform axiomatic view of lists, multisets, and sets, and the relevant unification algorithms. *Fundamenta Informaticae*, 36(2/3):201–234, 1998.
10. A. Dovier, C. Piazza, and G. Rossi. A uniform approach to constraint-solving for lists, multisets, compact lists, and sets. Technical Report, Dipartimento di Matematica, Università di Parma, no. 235, July 2000 (available at http://prmat.math.unipr.it/~gianfr/PAPERS/RR_PR235.ps).

11. A. Dovier, E. G. Omodeo, E. Pontelli, and G. Rossi. {log}: A Language for Programming in Logic with Finite Sets. *Journal of Logic Programming*, 28(1):1–44, 1996.
12. A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. Sets and constraint logic programming. *ACM Transaction on Programming Language and Systems* (TOPLAS), 22(5) 2000, pp. 861–931.
13. G. Gupta and E. Pontelli. Optimization Schemas for Parallel Implementation of Nondeterministic Languages. *Int. Parallel Proc. Symposium*, IEEE, pp. 428–435, 1997.
14. C. Hankin, D. Le Métayer, and D. Sands. A Parallel Programming Style and Its Algebra of Programs. In *Proc. Conf. on Parallel Architecture and Languages Europe (PARLE 93)*, vol. 694 of LNCS, 367-378, Springer-Verlag, Berlin, 1993.
15. J. Herbrand. Recherches sur la theorie de la demonstration. Master's thesis, Université de Paris, 1930. Also in *Ecrits logiques de Jacques Herbrand*, PUF, Paris, 1968.
16. J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19–20:503–581, 1994.
17. J. Jaffar, M. J. Maher, K. Marriott, and P. J. Stuckey. The Semantics of Constraint Logic Programs. *Journal of Logic Programming 37*(1–3), 1–46, 1998.
18. K. Marriott, B. Meyer, and K. B. Wittenburg. A survey of visual language specification and recognition. In K. Marriott and B. Meyer, editors, *Visual Language Theory* pages 5–85, Springer, 1998.
19. A. Mal'cev. Axiomatizable Classes of Locally Free Algebras of Various Types. In *The Metamathematics of Algebraic Systems*, Collected Papers, Ch. 23. North Holland, 1971.
20. A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems 4* (1982), 258–282.
21. G. Păun. Computing with Membranes. Journal of Computer and System Science, 61(1):108–143, 2000.
22. G. Păun. Attacking NP Complete Problems. Journal of Automata, Languages and Combinatorics, 6(1):75–90, 2001.
23. G. Rossi. The Languages $CLP(\mathcal{SET})$ and $CLP(\mathcal{BAG})$. User Manuals and Running Interpreters. In http://prmat.math.unipr.it/~gianfr/setlog.Home.html.
24. A. Tzouvaras. The Linear Logic of Multisets. *Logic Journal of the IGPL*, Vol. 6, No. 6, pp. 901–916, 1998.