Multiset rewriting by multiset constraint solving*

Agostino Dovier † Carla Piazza ‡
Gianfranco Rossi §

Abstract

Term-like theories of multisets have been recently studied from an axiomatic point of view and constraint solving algorithms have been developed. In this paper we summarize these results and we show how they can be exploited to develop multiset rewriting based applications. Specifically, we present a simple yet powerful declarative implementation of GAMMA, a computational framework based on multiset rewriting, using a CLP language embedding the constraint solving mechanism previously developed.

1 Introduction

Multisets emerge as the most natural data structure in several interesting applications. For instance, solutions to the equation $x^4 - 2x^2 + 1 = 0$ are better described by the multiset $\{\!\{ -1, -1, 1, 1 \}\!\}$ rather than by the set $\{-1, -1, 1, 1\}$ which is equivalent to $\{-1, 1\}$. As explained in [19], *sets came to mean types of objects, while multisets are based on tokens.* This justifies the use of multisets in describing processes which consume resources. In particular, multisets over some set of basic elements (urelements) can be perfectly connected to fragments of linear logic [19]. Multisets are the fundamental data structure of the GAMMA coordination language [2] and of the Chemical Abstract Machine [3]. Multiset rewriting is also the primitive operation in P-systems, modeling *membrane (and DNA) computing* [18]. Some issues on the relevance of multisets in Databases and the related complexity issues are discussed in [12].

In [8], multisets—along with other well-known data aggregates, such as lists and sets—are analyzed from an axiomatic point of view, in the context of (Constraint) Logic Programming languages [14]. In this context multisets, as well as

*The work is partially supported by MURST project *Certificazione automatica di programmi mediante interpretazione astratta.*

†Dip. Scientifico-Tecnologico, Univ. di Verona. Strada Le Grazie 15, 37134 Verona (Italy). dovier@sci.univr.it

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

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

the other data aggregates, are conveniently represented as terms, using suitable aggregate constructors endowed with the proper interpretations. The axiomatic theories studied are *hybrid*, i.e. they can deal with interpreted function symbols as well as with an arbitrary number of free constant and function symbols. The basic difference between multisets and the other data aggregates lies in the importance of order and/or repetitions of their elements: in lists both order and repetitions of elements are important; in multisets the order is immaterial, whereas the repetitions are important; in sets order and repetitions are not taken into account. In *compact lists*, instead, contiguous occurrences of the same element are immaterial: a property complementary to that characterizing multisets.

In [9] satisfiability algorithms for constraints in the above-mentioned theories are developed. Constraints are conjunctions of positive and negative literals based on the equality ($=$) and the *membership* ($\in$) predicate symbols. A constraint solving procedure (the *constraint solver*) is defined parametrically with respect to the considered aggregate theory. This procedure is able to reduce a satisfiable constraint to a solved form that describes its solutions; if the constraint is unsatisfiable, instead, it is reduced to `false`. Constraint satisfiability is tested on a suitable interpretation structure: unsatisfiability over this structure implies unsatisfiability in all models of multisets. The availability of the constraint solver procedure, in turn, makes it a trivial task to obtain a *constraint logic programming* (CLP) language dealing with the considered constraints as an instance of the general CLP scheme [14]. One such constraint domain for the case of sets, along with the specific CLP language based on it, is presented in [11].

In the first part of the paper (Sections 2 and 3), we survey the axiomatization of multisets and the constraint solving results of [8, 9], trying to simplify as much as possible the presentation that is rather technical in the original papers and complicated by the desire of presenting parametrically four theories instead of only one. In the second part (Sections 4 and 5), we address the problem of multiset rewriting and we show how a CLP language embedding the multiset constraint solver presented in the first part can be easily exploited to implement systems based on the multiset rewriting paradigm. This is done, specifically, by describing a simple implementation of the GAMMA language, and presenting a number of simple examples using the GAMMA multiset rewriting approach. In Section 6 we compare our approach with other works in the same area. Finally, some conclusions and possible lines for future work are drawn in Section 7.

Throughout the paper we assume basic knowledge of first-order logic. Moreover, we will adopt the following notation conventions: variables are usually represented by capital letters X, Y, Z, etc.; small letters f, g, etc. usually represent function symbols, whereas p, q, etc. represent predicate symbols; finally, the notation $\bar{X}$ will be used to denote a (possibly empty) sequence of variables.

2 Multiset Theory and Constraints

The distinguishing properties of multisets can be precisely stated via a suitable first-order axiomatic theory.[1] In this paper we refer to the parametric axiomatization given in [8] which is used to provide a uniform characterization of several related aggregate data structures, namely, lists, multisets, compact lists, and sets. In particular, multisets are defined by the theory named *Bag*.

The language upon which *Bag* is built—called $\mathcal{L}_{Bag}$—is a first-order language 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 symbol $\{\!\!\{ \cdot \,|\, \cdot \}\!\!\}$, plus possibly other (free) constant and function symbols, whereas Π_{Bag} is the set of predicate symbols containing exactly the two predicate symbols $=$ and $\in$.

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. Moreover, a multiset can be built by 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 aggregate. 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`.

Example 2.1 ($\mathcal{L}_{Bag}$ terms)

1. *$\{\!\!\{ a, b, a \}\!\!\}$ (i.e., $\{\!\!\{ a \,|\, \{\!\!\{ b \,|\, \{\!\!\{ a \,|\, \mathtt{nil} \}\!\!\} \}\!\!\} \}\!\!\}$) 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.*

A *multiset constraint (admissible 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 built from $\mathcal{F}_{Bag}$ and $\mathcal{V}$.

Example 2.2 ($\mathcal{L}_{Bag}$ constraints)

[1] A *(first-order) theory* $\mathcal{T}$ on a language $\mathcal{L}$ is a set of closed first-order formulae of $\mathcal{L}$, such that each closed formula of $\mathcal{L}$ which can be deduced from $\mathcal{T}$ is in $\mathcal{T}$. A *(first-order) set of axioms* Ax on $\mathcal{L}$ is a set of closed first-order formulae of $\mathcal{L}$. A set of axioms Ax is said to be an *axiomatization* of $\mathcal{T}$ if $\mathcal{T}$ is the smallest theory such that $Ax \subseteq \mathcal{T}$. Sometimes we use the term *theory* also to refer to an axiomatization of the theory.

1. *$X \in \{\!\!\{ a, b \,|\, Y \}\!\!\} \wedge X \notin Y \wedge X \neq a$;*

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. *$\{\!\!\{ e, e, e, e, e \}\!\!\} = \{\!\!\{ X \,|\, R \}\!\!\} \wedge X \notin \{\!\!\{ e \,|\, S \}\!\!\}$.*

The hybrid theory of multisets *Bag* is shown in Fig. 1. The three axiom schemata $(F_1), (F_2)$, and (F_3) (called freeness axioms, or Clark's equality axioms) have been originally introduced by Mal'cev in [17]. (F_1) has been modified to (F_1') in order to take care separately of the multiset constructor symbol. Axiom schema (F_3) states that there is no term which is also a subterm of itself. In particular if $X = \{\!\!\{ X \}\!\!\}$ had solutions, then, by (W), $X \in X$ would also have solutions. Thus, axiom schema (F_3) is a weak form of the *foundation axiom* which has the aim, among others, of guaranteeing the *acyclicity* of membership. Note that (K) implies that $\forall x\, (x \notin \texttt{nil})$. Axiom schema (F_3^m) forces that a property, very natural for *finite* multisets, holds for all multisets.

(K)	$\forall x\, \bar{y}$	$(x \notin f(y_1, \ldots, y_n))$	$f \in \mathcal{F}_{Bag}, f \not\equiv \{\!\!\{ \cdot \,|\, \cdot \}\!\!\}$
(W)	$\forall y\, v\, x$	$(x \in \{\!\!\{ y \,|\, v \}\!\!\} \leftrightarrow x \in v \vee x = y)$	
(F_1')	$\forall \bar{x}\bar{y}$	$\left(\begin{array}{l} f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \\ \to x_1 = y_1 \wedge \cdots \wedge x_n = y_n \end{array}\right)$	$f \in \mathcal{F}_{Bag}, f \not\equiv \{\!\!\{ \cdot \,|\, \cdot \}\!\!\}$
(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)	$\forall \bar{x}\bar{y}z$	$\left(\begin{array}{l} \{\!\!\{ x_1, \ldots, x_m \,|\, z \}\!\!\} = \{\!\!\{ y_1, \ldots, y_n \,|\, z \}\!\!\} \\ \to \{\!\!\{ x_1, \ldots, x_m \}\!\!\} = \{\!\!\{ y_1, \ldots, y_n \}\!\!\} \end{array}\right)$	$m, n > 0$
(E_k^m)	$\forall \bar{y}\bar{v}$	$\left(\begin{array}{l} \{\!\!\{ y_1 \,|\, v_1 \}\!\!\} = \{\!\!\{ y_2 \,|\, v_2 \}\!\!\} \;\leftrightarrow \\ (y_1 = y_2 \wedge v_1 = v_2) \vee \\ \exists z\, (v_1 = \{\!\!\{ y_2 \,|\, z \}\!\!\} \wedge v_2 = \{\!\!\{ y_1 \,|\, z \}\!\!\}) \end{array}\right)$	

Figure 1: The theory *Bag*

Axiom (E_k^m) is motivated by the following observations. The behavior of the multiset constructor symbol $\{\!\!\{ \cdot \,|\, \cdot \}\!\!\}$ is regulated by the following *equational axiom*

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

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 $KWF_1'F_2F_3$. What is needed is a way to characterize the following *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.*

This is done in [8] by the axiom (E_k^m) that implies (E_p^m) but has a negative flavor, as well.

Let E_{Bag} be the equational theory containing the equational axiom (E_p^m). Using E_{Bag} we can define the *initial model* $\mathcal{BAG}$ for the theory *Bag* 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 the ordinary Herbrand Universe $T(\mathcal{F}_{Bag})$ 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.

Given a theory $\mathcal{T}$ on a first-order language $\mathcal{L}$ and a structure $\mathcal{A}$, $\mathcal{T}$ and $\mathcal{A}$ *correspond* on the set of admissible constraints *Adm* [14] if, for each constraint $C \in \mathit{Adm}$, we have that $\mathcal{T} \models \vec{\exists}(C)$ if and only if $\mathcal{A} \models \vec{\exists}(C)$. It can be proved that this property holds also for the theory *Bag* and the structure $\mathcal{BAG}$.

Theorem 2.3 (Correspondence) *The structure $\mathcal{BAG}$ and the theory Bag correspond on the class of admissible constraints.* $\square$

This property guarantees that if C is an admissible constraint, its satisfiability (unsatisfiability) in $\mathcal{BAG}$ implies that it will be satisfiable (resp., unsatisfiable) in all the models of *Bag*.

The notion of satisfiability is based on the notion of valuation. A *valuation* σ is a function from a subset of $\mathcal{V}$ to A. Each valuation can be extended to a function from $T(\mathcal{F}, \mathcal{V})$ to A and to a function from the set of formulae over $\mathcal{L}$ to the set $\{\texttt{false}, \texttt{true}\}$. A valuation σ is said to be a *successful valuation* of φ if $\sigma(\varphi) = \texttt{true}$.

Example 2.4 (Valuations)

1. *Consider the constraint $a \in X \wedge B \in X \wedge X = \{\!\{\, Y, Z \,\}\!\} \wedge B \neq a$. The following successful valuation ensures that it is satisfiable in $\mathcal{BAG}$ (hence, in all models of Bag):*

$$\begin{array}{lll l lll} B & \mapsto & \texttt{nil}\ \textit{(to be precise, the class of}\ \texttt{nil}\textit{)} & \quad & Y & \mapsto & a \\ X & \mapsto & \{\!\{\, a, \texttt{nil} \,\}\!\} & & Z & \mapsto & \texttt{nil} \end{array}$$

2. *The constraint $a \in X \wedge B \in X \wedge X = \{\!\{\, Y \,\}\!\} \wedge B \neq a$ is instead unsatisfiable in $\mathcal{BAG}$ (hence, in all models).*

3. *Consider the constraints of Example 2.2. The first two are satisfiable. In particular, any successful valuation for constraint* 1 *needs to map* X *to* b*, 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.*

3 Multiset constraint solving

The technique used to check satisfiability of multiset constraints is based on the availability of 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`).

In this section we briefly review the overall constraint solving technique as presented in [9] (where it has been parametrically developed for the four theories considered). The single rewriting procedures by which satisfiable constraints are rewritten to solved form constraints are shown in the Appendix.

3.1 Solved form

A particular form of constraints—called *solved form*—plays a fundamental rôle in establishing satisfiability of constraints in the corresponding structures.

A constraint C is in *solved form* if all its literals are in solved form, that is, they 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.

Observe that no membership literals are present in the solved form. For multisets (but this is not true for lists and compact lists [9]) it holds that:

$$s \in t \leftrightarrow \exists N(t = \{\!\{ s \,|\, N \}\!\}).$$

Thus, in *Bag* all membership constraints can be always replaced by equivalent equality constraints.

Importance of solved form constraints ensues from the following result.

Theorem 3.1 (Satisfiability) *Let* C *be a constraint in solved form over the language* $\mathcal{L}_{Bag}$*. 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, following the lines of the proof of Theorem 3.1, 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.

$$\begin{array}{lcl}
\mathsf{SAT}_{Bag}(C) & = & \mathsf{repeat} \\
& & \quad C' := C; \\
& & \quad C := \mathsf{Unify_bags}(\mathsf{neq\text{-}bag}(\mathsf{nin\text{-}bag}(\mathsf{in\text{-}bag}(C)))); \\
& & \mathsf{until}\ C = C'; \\
& & \mathsf{return}(C).
\end{array}$$

Figure 2: The satisfiability procedure

3.2 Global Constraint Solving

We assume that each kind of constraint literal in the theory *Bag* is dealt with a proper rewriting procedure: Unify_bags for equality constraints, neq-bag for inequalities, in-bag and nin-bag for membership and negated membership constraints, respectively. Their complete definitions are shown in the Appendix.

Constraint satisfiability for the theory *Bag* is checked by the non-deterministic procedure SAT_{Bag} shown in Figure 2. SAT_{Bag} uses iteratively the various constraint rewriting procedures to eliminate all the literals not in solved form possibly occurring in a given constraint C.

Each disjunction generated by the rules of these rewriting procedures is interpreted as a (don't know) non-deterministic choice. Thus, $\mathsf{SAT}_{Bag}(C)$ returns a collection $C_1, \dots, C_k$ of constraints. Each of them is either in solved form or `false`. By Theorem 3.1 a constraint in solved form is guaranteed to be satisfiable in $\mathcal{BAG}$. Each C_i in solved form represents implicitly a family of solutions of C.

Theorem 3.2 (Termination) $\mathsf{SAT}_{Bag}(C)$ *terminates in a finite number of steps. Moreover, the constraint returned is either* `false` *or a solved form constraint.* □

Theorem 3.3 (Soundness and Completeness) *Let $C_1, \dots, C_k$ be the constraints returned by $\mathsf{SAT}_{Bag}(C)$, and $\bar{N}_i = FV(C_i) \setminus FV(C)$, where $FV(\varphi)$ denotes the set of free variables in the formula φ. Then*

$$\mathcal{BAG} \models \vec{\forall}\left(C \leftrightarrow \bigvee_{i=1}^{k} \exists \bar{N}_i C_i\right)$$

□

Thus, as an important corollary, given a constraint C, it is decidable whether $\mathcal{BAG} \models \vec{\exists} C$.

Example 3.4 ($\mathsf{SAT}_{\mathcal{BAG}}$ computations) *Consider again the constraint $C \equiv a \in X \wedge B \in X \wedge X = \{\!\!\{Y, Z\}\!\!\} \wedge B \neq a$ in Example 2.4. From that example we know that this constraint is satisfiable in $\mathcal{BAG}$. Hence, the constraint solving procedure $\mathsf{SAT}_{\mathcal{BAG}}(C)$ terminates successfully, returning the solved form constraint*

$$X = \{\!\!\{a, Z\}\!\!\} \wedge B = Z \wedge Y = a \wedge Z \neq a.$$

Note that this solution subsumes the valuation for the given constraint shown in Example 2.4.

For the second constraint in Example 2.4, $a \in X \wedge B \in X \wedge X = \{\!\{ Y \}\!\} \wedge B \neq a$, *which is instead unsatisfiable,* $\mathsf{SAT}_{\mathcal{BAG}}(C')$ *returns* `false`.

3.3 Examples

To give the taste of the expressiveness of $\mathcal{L}_{Bag}$-constraints we present two further more elaborated examples.

Example 3.5 (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 to the following multiset unification problem:

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

where the variables Y_i *take the place of* $\neg X_i$. *Each solution to this equation computed by* $\mathsf{SAT}_{\mathcal{BAG}}$ *is also a solution to the given 3-SAT problem and, vice versa, from each solution to the SAT problem it is immediate to assign values to the variables* Y_i *and* R_i *in order to obtain a solution to the equation. One such solution is, for instance:*

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

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

Example 3.6 (Map Coloring) *Consider the following instance of the well-known problem of the coloring of a geographical map. Assume that we wish to use the four colors* white, yellow, red, *and* blue *to color the countries* Austria, France, Italy, Slovenia, *and* Swiss. *This problem can be rendered by the following* $\mathcal{L}_{Bag}$*-constraint*

$$\begin{array}{lclc} \{\!\{ \mathsf{Italy}, \mathsf{Austria} \}\!\} \in \mathsf{X} & \wedge & \{\!\{ \mathsf{Italy}, \mathsf{France} \}\!\} \in \mathsf{X} & \wedge \\ \{\!\{ \mathsf{Italy}, \mathsf{Slovenia} \}\!\} \in \mathsf{X} & \wedge & \{\!\{ \mathsf{Italy}, \mathsf{Swiss} \}\!\} \in \mathsf{X} & \wedge \\ \{\!\{ \mathsf{Austria}, \mathsf{Swiss} \}\!\} \in \mathsf{X} & \wedge & \{\!\{ \mathsf{Austria}, \mathsf{Slovenia} \}\!\} \in \mathsf{X} & \wedge \\ \{\!\{ \mathsf{France}, \mathsf{Swiss} \}\!\} \in \mathsf{X} & \wedge & & \\ \mathsf{X} \;\; = \;\; \{\!\{ \{\!\{ \mathsf{white}, \mathsf{yellow} \}\!\}, \{\!\{ \mathsf{white}, \mathsf{red} \}\!\}, \{\!\{ \mathsf{white}, \mathsf{blue} \}\!\}, & & & \\ \qquad \{\!\{ \mathsf{yellow}, \mathsf{red} \}\!\}, \{\!\{ \mathsf{yellow}, \mathsf{blue} \}\!\}, \{\!\{ \mathsf{red}, \mathsf{blue} \}\!\} \}\!\} & & & \end{array}$$

where names with capital initial are variables and all other names are constants. Observe that X *is a multiset containing all possible unordered pairs of colors which can be formed using the given four colors. One of the (many) solutions for this constraint computed by the* $\mathsf{SAT}_{\mathcal{BAG}}(C)$ *is:*

Italy = white ∧ Swiss = red ∧ France = yellow ∧ Austria = yellow ∧ Slovenia = red

4 $CLP(\mathcal{BAG})$

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 [14, 15] by instantiating it over the language, the structure, and the theory 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 the operational, algebraic and logical semantics of $CLP(\mathcal{BAG})$, are those of the general CLP scheme [14, 15] suitably instantiated on the specific constraint domain of multisets. This is very similar to what is described in greater detail in [11] for the case of sets. A working prototype of the CLP language which embeds both sets and multisets is available at http://prmat.math.unipr.it/~gianfr/setlog.Home.html.

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 4.1 ($CLP(\mathcal{BAG})$ **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*:*

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

Other examples of $CLP(\mathcal{BAG})$ programs will be presented in the next section. $CLP(\mathcal{BAG})$ defines a simple, yet very expressive, framework that allows us to program using multisets in a very flexible way. In particular, algorithms which are related to the general idea of multiset rewriting can be quite naturally implemented as $CLP(\mathcal{BAG})$ programs. We will further investigate on this topic in the next section, by showing how $CLP(\mathcal{BAG})$ can be used to implement a simple interpreter of the GAMMA language and how this, in turn, provides us with a very straightforward way to code multiset rewriting in $CLP(\mathcal{BAG})$.

5 Multiset rewriting in $CLP(\mathcal{BAG})$

Multiset rewriting (or transformation) is a powerful problem solving paradigm. Multiset rewriting systems specify solutions to problems in terms of successive transformations over multisets, starting from an initial multiset to a final multiset of values. A transition from a multiset to the next one is specified by a multiset rewriting rule, which, in its simplest form, can be seen as a way to establish a relationship between two multisets.

Applications of multiset rewriting are many and of various kinds, ranging from membrane computing [18] to security protocol analysis [5], from communicating object models [4] to constraint multiset grammars [16]. Various, more or less specialized, programming languages support multiset rewriting as a primitive mechanism (e.g., GAMMA [2], GammaLog [6]) or provide multiset processing capabilities through which it is possible to implement some form of multiset rewriting (e.g., SETA [1]).

In this section we will focus on the notion of multiset rewriting provided by the coordination language GAMMA, probably the best-known system based on multiset rewriting.

5.1 The GAMMA model of multiset rewriting

GAMMA [2] is a high-level parallel programming paradigm based on multiset rewriting. The idea behind GAMMA is to provide a formalism to write parallel programs without taking care of control constraints imposed by a particular physical parallelism. The state of a program in GAMMA is a multiset whose elements can interact with each other according to some rule, and the result of enforcing this rule is a new multiset. GAMMA's model of execution can be seen as a generalization of *chemical solutions* in which *molecules* can interact with each other according to *reaction rules* [3].

A GAMMA program is composed by a collection of non-deterministic rules, each expressed as a rewriting of multisets. For example, the following rule can be used to compute the maximum element of a multiset s.

$$\begin{array}{l} \mathit{max\_elements}(s) = \Gamma((R, A))(s) \ \ \textsf{where} \\ \quad R(x, y) = x \leq y \\ \quad A(x, y) = \{\!\!\{\, y \,\}\!\!\} \end{array}$$

R is a *reaction* condition, that is, a boolean function which specifies when the rule is applicable. A is an *action*, that is, a multiset rewriting which takes place if the condition R holds. The Γ operator defines the basic building block for the operational semantics of a GAMMA program. Evaluation of $\Gamma((R, A))(s)$ proceeds as follows: arbitrarily select, if it exists, a subset of s, $\{\!\!\{\, x_1, \ldots, x_n \,\}\!\!\}$ such that $R(x_1, \ldots, x_n)$ holds, removing $x_1, \ldots, x_n$ from s, then execute $A(x_1, \ldots, x_n)$, adding the new elements possibly generated by it to the multiset s; otherwise, i.e., if no such a subset exists, computation of $\Gamma((R, A))(s)$ terminates and the result is the current value of s.

Thus, for instance, the following is a possible sequence of the multisets which are generated by executing $\mathit{max\_elements}(\{\!\!\{\, 4, 6, 5 \,\}\!\!\})$

$$\{\!\!\{\, 4, 6, 5 \,\}\!\!\} \rightarrow \{\!\!\{\, 6, 5 \,\}\!\!\} \rightarrow \{\!\!\{\, 6 \,\}\!\!\}.$$

No subset of $\{\!\!\{\, 6 \,\}\!\!\}$ can be selected which satisfies the given R condition; so, the computation terminates.

GAMMA provides also a few operators to build complex programs by combining simple ones [13].

5.2 Implementing GAMMA in $CLP(\mathcal{BAG})$

Execution of a GAMMA program can be described in terms of successive transformations on multisets. These transformations can be easily simulated by using the powerful and flexible multiset processing facilities offered by $CLP(\mathcal{BAG})$. Proving this claim is the goal of the rest of this section.

The starting point is the $CLP(\mathcal{BAG})$ implementation of the Γ operator. This is obtained by defining a predicate $\mathsf{gamma}(\mathsf{M}_1, \mathsf{M}_2)$ that establishes a relationship between two multisets, the initial multiset M_1 and the final multiset M_2.

```
gamma(Input, Input) :-
    not rule(Input, _).
gamma(Input, Output) :-
    rule(Input, Intermediate),
    gamma(Intermediate, Output).
```

This is a very natural, though executable, specification of the operational semantics of the Γ operator. The multiset transformation process terminates as soon as no rule can be applied successfully to the input multiset.

According to this specification, GAMMA rewriting rules can be simply defined as $CLP(\mathcal{BAG})$ rules of the form:

```
rule(InputMultiset, OutputMultiset) :- Condition
```

where InputMultiset and OutputMultiset are multiset terms and Condition is the conjunction of atomic predicates representing the reaction condition of the rule, namely the condition that must be satisfied in order to apply the rule.

Observe that the call to rule(Input,Intermediate) in the second rule of predicate gamma implies—among others—to solve the equation between two multisets Input = InputMultiset. If Input has the form $\{\!\{ t_1, \ldots, t_k \}\!\}$ and InputMultiset has the form $\{\!\{ x_1, \ldots, x_n | R \}\!\}$, with $n \leq k$, then solving this equation amounts to arbitrarily select a subset $\{\!\{ x_1, \ldots, x_n \}\!\}$ of Input, as required by the definition of the Γ operator.

The availability of the gamma predicate makes it straightforward to write $CLP(\mathcal{BAG})$ programs according to the GAMMA programming paradigm. We show this by presenting the solution of a number of simple problems, most of which are adapted from [2].

Prime Numbers. Compute all the prime numbers that are not greater than N.

Add to the definition of gamma the rule

```
rule({{X, Y | Z}}, {{X | Z}}) :-
    Rmdr is Y mod X,
    Rmdr = 0.
```

The answer to the goal

$$:- \mathsf{gamma}(\{\!\![\, 2,3,4,\ldots,\mathsf{N} \,]\!\!\}, \mathsf{Output}).$$

will return as Output the desired result.

Factorial. Compute the factorial of a number N.
Add to the definition of gamma the rule

$$\begin{array}{l}\mathsf{rule}(\{\!\![\, \mathsf{X},\mathsf{Y} \mid \mathsf{Z} \,]\!\!\}, \{\!\![\, \mathsf{V} \mid \mathsf{Z} \,]\!\!\}) :- \\ \quad \mathsf{V\ is\ Y} * \mathsf{X}.\end{array}$$

The answer to the goal

$$:- \mathsf{gamma}(\{\!\![\, 1,2,3,\ldots,\mathsf{N} \,]\!\!\}, \mathsf{Output}).$$

will be (the solved form constraint) $\mathsf{Output} = \{\!\![\, \mathsf{N!} \,]\!\!\}$.

Convex Hull. Given a multiset of (more than 3) points in the Euclidian plane, we wish to find the vertices of the smallest convex polygon including all of them. Assuming that in the given multiset there are not 3 points on the same line, the problem can be solved by adding to the gamma predicate the following $CLP(\mathcal{BAG})$ rules:

$$\begin{array}{l}
\mathsf{rule}(\{\!\![\, \mathsf{P1},\mathsf{P2},\mathsf{P3},\mathsf{P} \mid \mathsf{Z} \,]\!\!\}, \{\!\![\, \mathsf{P1},\mathsf{P2},\mathsf{P3} \mid \mathsf{Z} \,]\!\!\}) :- \\
\quad \mathsf{in_triangle}(\mathsf{P}, \{\!\![\, \mathsf{P1},\mathsf{P2},\mathsf{P3} \,]\!\!\}). \\
\mathsf{in_triangle}(\mathsf{P}, \{\!\![\, \mathsf{P}_1,\mathsf{P}_2,\mathsf{P}_3 \,]\!\!\}) :- \\
\quad \mathsf{same_side}(\mathsf{P}_1,\mathsf{P}_2,\mathsf{P}_3,\mathsf{P}), \\
\quad \mathsf{same_side}(\mathsf{P}_1,\mathsf{P}_3,\mathsf{P}_2,\mathsf{P}), \\
\quad \mathsf{same_side}(\mathsf{P}_2,\mathsf{P}_3,\mathsf{P}_1,\mathsf{P}). \\
\mathsf{same_side}((\mathsf{X}_1,\mathsf{Y}_1),(\mathsf{X}_2,\mathsf{Y}_2),(\mathsf{X}_3,\mathsf{Y}_3),(\mathsf{X}_4,\mathsf{Y}_4)) :- \\
\quad (((\mathsf{X}_4-\mathsf{X}_1)*(\mathsf{Y}_2-\mathsf{Y}_1)-(\mathsf{X}_2-\mathsf{X}_1)*(\mathsf{Y}_4-\mathsf{Y}_1))* \\
\quad ((\mathsf{X}_3-\mathsf{X}_1)*(\mathsf{Y}_2-\mathsf{Y}_1)-(\mathsf{X}_2-\mathsf{X}_1)*(\mathsf{Y}_3-\mathsf{Y}_1))) \geq 0.
\end{array}$$

Chemical reactions. We show two examples of this class.

- Suppose to have a multiset containing atoms of hydrogen (h) and oxygen (o). The rule

  $$\mathsf{rule}(\{\!\![\, \mathsf{h},\mathsf{h},\mathsf{o} \mid \mathsf{Z} \,]\!\!\}, \{\!\![\, \mathsf{w} \mid \mathsf{Z} \,]\!\!\}).$$

  describes the chemical reaction for generating molecules of water (w). The output to a goal of the form:

  $$:- \mathsf{gamma}(\{\!\![\, \mathsf{o},\mathsf{o},\mathsf{h},\mathsf{h},\mathsf{h},\mathsf{o},\mathsf{h},\mathsf{h},\mathsf{h} \,]\!\!\}, \mathsf{Output}).$$

  will be $\mathsf{Output} = \{\!\![\, \mathsf{w},\mathsf{w},\mathsf{w} \,]\!\!\}$.

- Assume molecules are represented as multisets of basic elements, and a substance as a multiset of its component molecules. Assume one wants to filter, for example, mineral water in order to completely remove from it anything different from pure water H_2O. This process can be modeled by the following rule (to be added to the definition of gamma):

$$\begin{array}{l}\mathsf{rule}(\{\!\!\{\, \mathsf{X} \mid \mathsf{W} \,\}\!\!\}, \mathsf{W}) :- \\ \quad \mathsf{X} \neq \{\!\!\{\, \mathsf{h}, \mathsf{h}, \mathsf{o} \,\}\!\!\}.\end{array}$$

  A sample goal is:

$$:- \mathsf{gamma}(\{\!\!\{\, \{\!\!\{\, \mathsf{h}, \mathsf{h}, \mathsf{o} \,\}\!\!\}, \{\!\!\{\, \mathsf{si}, \mathsf{o}, \mathsf{o} \,\}\!\!\}, \{\!\!\{\, \mathsf{o}, \mathsf{h}, \mathsf{h} \,\}\!\!\} \,\}\!\!\}, \mathsf{Output}).$$

  whose answer will be $\mathsf{Output} = \{\!\!\{\, \{\!\!\{\, \mathsf{h}, \mathsf{h}, \mathsf{o} \,\}\!\!\}, \{\!\!\{\, \mathsf{o}, \mathsf{h}, \mathsf{h} \,\}\!\!\} \,\}\!\!\}$.
  Note that in this example we use a negative constraint (namely, $\neq$) and that the input multiset contains other (nested) multisets. Therefore, the constraint solver has to solve—and actually it is able to do—inequalities involving multisets, which are possibly syntactically different terms but denoting "semantically" equivalent objects (e.g., $\{\!\!\{\, \mathsf{h}, \mathsf{h}, \mathsf{o} \,\}\!\!\} \neq \{\!\!\{\, \mathsf{o}, \mathsf{h}, \mathsf{h} \,\}\!\!\}$).

6 Discussion and related work

A solution similar to the one described in the previous section, but dealing with sets instead of multisets, is sketched in [11]. Although many of the problems dealt with here can be solved using sets as well (e.g., the prime numbers problem, provided the element Y is forced to not belong to Z by a constraint $\mathsf{Y} \notin \mathsf{Z}$) there are other problems where multiset rewriting turns out to be much more suitable than set rewriting. For example, in the rule for computing the factorial of a given number the output multiset may contain more than one occurrence of the same element at a moment, and all of them must be accounted for (e.g., the element 12 could be obtained both as $3 * 4$ and $2 * 6$ and $1 * 12$).

An implementation of the Γ operator similar to ours is also presented in [6]. This implementation however relies on a preliminary version of the logic language with multisets considered in this paper. Indeed, the language in [6]—called Gammalog—is based on the extended logic language described in [10], which in turn has evolved to the CLP language $CLP(\mathcal{SET})$ described in [11]. The adaptation of these languages to the case of multisets turns out to be less trivial than expected. The constraint solving procedures for set and multiset constraints have been extensively studied in the last few years and are presented in detail in [9]. The language $CLP(\mathcal{BAG})$ used in this paper is based on these procedures and is completely specified by the relevant constraint structure we have briefly recalled in Section 2. In contrast, no precise specification of the procedure used to solve multiset equalities is provided in [6].

Moreover, Gammalog deals with only equations on multisets—dealt with as an ad-hoc extension to a Horn clause language—whereas $CLP(\mathcal{BAG})$ provides a complete treatment of equality and membership, as well as their negative counterparts, as constraints in the more general CLP setting. While these constraints are not used in the simple examples of GAMMA programs shown in this paper their presence adds extra expressive power and flexibility to the language.

Example 6.1 (Negative Constraints) *Compute the number of occurrences of element* a *in a multiset* s*. Add to the definition of predicate* gamma *the rule*

$$\begin{array}{l}\mathsf{rule}(\{\!\!\{\, \mathsf{X} \mid \mathsf{R} \,\}\!\!\}, \mathsf{R}) :- \\ \quad \mathsf{X} \neq \mathsf{a}.\end{array}$$

The goal

$$:- \mathsf{gamma}(\{\!\!\{\, \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{c}, \mathsf{a} \,\}\!\!\}, \mathsf{M}), \mathsf{size}(\mathsf{M}, \mathsf{N})$$

where size(M,N) *is true if* N *is the cardinality of multiset* M*, returns in* N *the desired result.*

The rule for this definition uses the negative constraint $\neq$. Note that if some elements of the given multiset s *are not ground, then syntactic disequality of Prolog does not work properly, whereas constraint $\neq$ still continues to work. For example, the goal*

$$:- \mathsf{gamma}(\{\!\!\{\, \mathsf{a}, \mathsf{b}, \mathsf{X} \,\}\!\!\}, \mathsf{M}), \mathsf{size}(\mathsf{M}, \mathsf{N})$$

(non-deterministically) returns the following two results:

$$\begin{array}{l}\mathsf{X} = \mathsf{a}, \mathsf{N} = 2 \\ \mathsf{X} \neq \mathsf{a}, \mathsf{N} = 1\end{array}$$

Thus, the availability of constraints in $CLP(\mathcal{BAG})$ allows GAMMA-like rewriting rules to be used not only to process completely specified multisets but also to compute possible values for partially specified multisets.

On the other hand, [6] addresses also the translation of the sequential and parallel composition operators which is instead not considered in this paper. The main goal of this paper, however, is not to provide a complete translation of GAMMA programs to $CLP(\mathcal{BAG})$ programs, rather to show the naturalness of the implementation of languages based on multiset rewriting using the framework of $CLP(\mathcal{BAG})$.

7 Conclusions

The main technical contributions of this paper can be summarized as follows:

- giving a precise axiomatic characterization of multisets—adapted from the general parametric presentation of [8]—along with a short overview of the constraint language and the constraint solving technique proposed in [9];

- showing how a CLP language endowed with these multiset manipulation facilities can be used to implement a general computational framework based on the multiset rewriting paradigm, namely, GAMMA.

This work may represent a first step towards a deeper connection between the area of constraint programming and that of multiset processing. One future line of work is to show how to implement membrane computing within our framework and, on the other hand, how to derive DNA processes and algorithms from problems encoded as multiset constraints.

References

[1] 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.

[2] J. Bânatre and D. Le Métayer. Programming by Multiset Transformation. *Communications of the ACM*, 36(1):98–111. January 1993.

[3] G. Berry and G. Boudol. The Chemical Abstract Machine. *Theoretical Computer Science*, vol. 96 (1992) 217–248.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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).

[10] 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.

[11] A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. Sets and constraint logic programming. To appear in *ACM Transaction on Programming Language and Systems*, 2001.

[12] S. Grumbach and T. Milo. Towards tractable algebras for bags. *Journal of Computer and System Sciences*, 52(3):570–588, 1996.

[13] 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.

[14] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19–20:503–581, 1994.

[15] 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.

[16] 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.

[17] 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.

[18] G. Pāun. Computing with Membranes. Journal of Computer and System Science, 61, 2000.

[19] A. Tzouvaras. The Linear Logic of Multisets. *Logic Journal of the IGPL*, Vol. 6, No. 6, pp. 901–916, 1998.

A Multiset constraint rewriting procedures

A.1 Equality constraints

Equality constraints are conjunctions of atomic formulae based on the predicate symbol '=' (i.e., equations). Unification algorithms for verifying the satisfiability and producing the solutions of equality constraints in *Bag* have been proposed in [8, 7]. In [8] we prove that our multiset unification algorithm terminates and it is sound and complete with respect to *Bag*. The output of the algorithms is either `false`, when the constraint is unsatisfiable, or a disjunction of solved form constraints composed only by equality atoms. The multiset unification problem has been proved to be NP-complete.

The procedure for dealing with equalities over multisets—called Unify_bags—is shown in Figures 3 and 4. The following two auxiliary functions are used by Unify_bags:

$$
\begin{array}{lcll}
\mathsf{tail}(f(t_1,\ldots,t_n)) & = & f(t_1,\ldots,t_n) & f \not\equiv \{\!\!\{ \cdot \,|\, \cdot \}\!\!\} \\
\mathsf{tail}(X) & = & X & \\
\mathsf{tail}(\{\!\!\{ t \,|\, s \}\!\!\}) & = & \mathsf{tail}(s) & \\
\mathsf{untail}(X) & = & \mathtt{nil} & \\
\mathsf{untail}(\{\!\!\{ t \,|\, s \}\!\!\}) & = & \{\!\!\{ t \,|\, \mathsf{untail}(s) \}\!\!\} \,. &
\end{array}
$$

A.2 Membership and negated membership constraints

The rewriting procedures in-bag and nin-bag used to deal with membership literals in *Bag*are shown in Figure 5.

A.3 Disequality constraints

Disequality constraints not in solved form are eliminated from the input constraint by using the rewriting procedure—called neq-bag—presented in Figure 6.

Special attention must be devoted to the management of disequalities between multisets (rule (6.2) of neq-bag). If we use directly axiom (E_k^m), we have that:

$$
\begin{array}{lcl}
\{\!\!\{ t_1 \,|\, s_1 \}\!\!\} \neq \{\!\!\{ t_2 \,|\, s_2 \}\!\!\} & \leftrightarrow & (t_1 \neq t_2 \vee s_1 \neq s_2) \wedge \\
 & & \forall N \, (s_2 \neq \{\!\!\{ t_2 \,|\, N \}\!\!\} \vee s_1 \neq \{\!\!\{ t_1 \,|\, N \}\!\!\})
\end{array}
$$

A universal quantification is introduced: this is no longer a constraint according to our definition and, in any case, this is a quite complex formula to deal with.

Alternatively, we could use the intuitive notion of multi-membership: $x \in^i y$ if x belongs at least i times to the multiset y. This way, one can write an alternative version of multiset equality and disequality. In particular, we have:

$$
\begin{array}{lcl}
\{\!\!\{ t_1 \,|\, s_1 \}\!\!\} \neq \{\!\!\{ t_2 \,|\, s_2 \}\!\!\} & \leftrightarrow & \exists X \exists n \, (n \in \mathbb{N} \wedge \\
 & & (X \in^n \{\!\!\{ t_1 \,|\, s_1 \}\!\!\} \wedge X \notin^n \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}) \vee \\
 & & (X \in^n \{\!\!\{ t_2 \,|\, s_2 \}\!\!\} \wedge X \notin^n \{\!\!\{ t_1 \,|\, s_1 \}\!\!\}\{\!\!\{ t_2 \,|\, s_2 \}\!\!\}))
\end{array}
$$

In this case, however, we have a quantification on natural numbers: we are outside the language we are studying.

The rewriting rule (6.2) shown in Figure 6 avoids these difficulties introducing only existential quantification.

```
Unify_bags(ℰ):
ℰ₂ := ∅;
while ℰ₁ is not in solved form or ℰ₂ ≠ ∅ do
    while ℰ₂ ≠ ∅ do
        e := pop(ℰ₂);
        if e is in solved form with respect to ℰ₁ ∧ ℰ₂
        then ℰ₁ := ℰ₁ ∧ e
        else Unify_bag_actions(ℰ, e);
    select arbitrarily and remove from ℰ₁ an equation e not in solved form;
    Unify_bag_actions(ℰ, e).
```

Figure 3: Main part of the unification algorithm for bags

Unify_bag_actions($\mathcal{E}$,e):
case e of

$(m1)$	$X = X$	$\mapsto$ $\mathcal{E}_1 := \mathcal{E}_1$
$(m2)$	$t = X$ t is not a variable	$\mapsto$ $\mathsf{push}(X = t, \mathcal{E}_2)$
$(m3)$	$X = t$ X does not occur in t X occurs in $\mathcal{E}$	$\mapsto$ $\mathcal{E}_1 := (X = t) \wedge \mathcal{E}_1[X/t]; \mathcal{E}_2 := \mathcal{E}_2[X/t]$
$(m4)$	$X = t$ $X \not\equiv t$ and X occurs in t	$\mapsto$ `fail`
$(m5)$	$f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)$ $f \not\equiv g$	$\mapsto$ `fail`
$(m6)$	$f(s_1, \ldots, s_m) = f(t_1, \ldots, t_m)$ $f \not\equiv \{\!\!\{ \cdot \mid \cdot \}\!\!\}$	$\mapsto$ $\mathcal{E}_1 := (s_1 = t_1 \wedge \ldots \wedge s_m = t_m) \wedge \mathcal{E}_1$
$(m7)$	$\{\!\!\{ t \mid s \}\!\!\} = \{\!\!\{ t' \mid s' \}\!\!\}$ $\mathsf{tail}(s)$ and $\mathsf{tail}(s')$ are not the same variable	$\mapsto$ (i) $\mathcal{E}_1 := (t = t') \wedge \mathcal{E}_1; \mathsf{push}(s = s',\ \mathcal{E}_2)$ (ii) $\mathsf{push}(s = \{\!\!\{ t' \mid N \}\!\!\}\ ,\ \mathcal{E}_2); \mathsf{push}(\{\!\!\{ t \mid N \}\!\!\} = s'\ ,\ \mathcal{E}_2)$
$(m8)$	$\{\!\!\{ t \mid s \}\!\!\} = \{\!\!\{ t' \mid s' \}\!\!\}$ $\mathsf{tail}(s)$ and $\mathsf{tail}(s')$ are the same variable	$\mapsto$ $\mathsf{push}(\mathsf{untail}(\{\!\!\{ t \mid s \}\!\!\}) = \mathsf{untail}(\{\!\!\{ t' \mid s' \}\!\!\})\ ,\ \mathcal{S}\mathcal{E}_2)\ .$

Figure 4: Multiset unification rewriting rules

function in-bag(C) while there is a $\in$-constraint c in C not in solved form do case c of			
(1)	$r \in f(t_1, \dots, t_n)$ $f \not\equiv \{\!\{ \cdot \mid \cdot \}\!\}$	$\mapsto$	`false`
(2)	$r \in \{\!\{ t \mid s \}\!\}$	$\mapsto$	$r = t \vee$ (a) $r \in s$ (b)
(3)	$r \in X$ X occurs in r	$\mapsto$	`false`
(4)	$r \in X$	$\mapsto$	$X = \{\!\{ r \mid N \}\!\}$

function nin-bag(C) while there is a $\notin$-constraint c in C not in solved form do case c of			
(1)	$r \notin f(t_1, \dots, t_n)$ $f \not\equiv \{\!\{ \cdot \mid \cdot \}\!\}$	$\mapsto$	`true`
(2)	$r \notin \{\!\{ t \mid s \}\!\}$	$\mapsto$	$r \neq t \wedge r \notin s$
(3)	$r \notin X$ X occurs in r	$\mapsto$	`true`

Figure 5: Rewriting procedures for $\in$- and $\notin$-constraints over multisets

function neq-bag(C) while there is a $\neq$-constraint c in C not in solved form do case c of			
(1)—(5) as in neq-list			
(6.1)	$\{\!\{ t_1 \mid s_1 \}\!\} \neq \{\!\{ t_2 \mid s_2 \}\!\}$ tail(s_1) and tail(s_2) are the same variable	$\mapsto$	untail($\{\!\{ t_1 \mid s_1 \}\!\}$) $\neq$ untail($\{\!\{ t_2 \mid s_2 \}\!\}$)
(6.2)	$\{\!\{ t_1 \mid s_1 \}\!\} \neq \{\!\{ t_2 \mid s_2 \}\!\}$ tail(s_1) and tail(s_2) are not the same variable	$\mapsto$	$(t_1 \neq t_2 \wedge t_1 \notin s_2) \vee$ (a) $(\{\!\{ t_2 \mid s_2 \}\!\} = \{\!\{ t_1 \mid N \}\!\} \wedge s_1 \neq N)$ (b)
(7)	$X \neq t$ $X \not\equiv t, X$ occurs in t	$\mapsto$	`true`

Figure 6: Rewriting procedure for $\neq$-constraints over multisets