A uniform approach to constraint-solving for lists, multisets, compact lists, and sets

AGOSTINO DOVIER[*] CARLA PIAZZA[†] GIANFRANCO ROSSI[‡]

Abstract

Lists, multisets, and sets are well-known data structures whose usefulness is widely recognized in various areas of Computer Science. These data structures have been analyzed from an axiomatic point of view with a parametric approach in [12] and the relevant unification algorithms have been also parametrically developed. In this paper we extend these results considering more general constraints including not only equality but also membership constraints as well as their negative counterparts. This amounts to define the privileged structures for the considered axiomatic theories and to solve the relevant constraint satisfaction problems in each of the theories. Like in [12], moreover, we adopt a highly parametric approach which allows all the results obtained separately for each single theory to be easily combined so as to obtain a general framework where it is possible to deal with more than one data structure at a time. **Keywords:** Constraints, Computable Set and Multiset Theory.

1 Introduction

Programming and specification languages allow the user to specify aggregation of elementary data objects and, in turn, aggregation of aggregates. Besides the well-known example of arrays, also lists, multisets, and sets are other important forms of data aggregates whose usefulness is widely recognized in various areas of Computer Science. Lists are the "classical" example used to introduce dynamic data structures in imperative programming languages. They are the fundamental data structure in the functional language LISP, and list predicates, such as *member* and *append*, are among the first predicates that are taught to students of the logic programming language PROLOG.

Sets are the main data structure used in specification languages (e.g., in Z [25]) and in high-level declarative programming languages [5, 13, 18, 16]; but also imperative programming languages can take advantage from the set data abstraction (e.g., SETL [26]).

Multisets emerge as the most natural data structure in several interesting applications. 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 [28], *sets came to mean types of objects, while multisets*

[*] 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

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 [28]. Multisets are the fundamental data structure of the Gamma coordination language [3], based on the chemical metaphor, and of the Chemical Abstract Machine [4]: a multiset can be seen as a *solution* containing molecules that can react inside it. Using this metaphor it is natural to write parallel algorithms. For instance, assume that a multiset contains all the numbers between 2 and n and consider the multiset rewriting rule 'x destroys one of its multiples'. Several process can run in parallel inside the multiset applying the rule; at the end of the execution, only the prime numbers from 2 to n remain in the multiset [3]. Some issues on the relevance of multisets in Databases and the related complexity problems can be found in [17].

The basic difference between lists, multisets, and sets 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.

These three data structures have been analyzed from an axiomatic point of view in [12]. The axiomatizations provided in that paper induce a lattice of four points, having sets as top and lists as bottom, as shown in the figure below. In this lattice,

$$\begin{tikzcd} & \text{Sets} & \\ \text{Multisets} \arrow[ur] & & \text{Compact lists} \arrow[ul] \\ & \text{Lists} \arrow[ul] \arrow[ur] & \end{tikzcd}$$

between sets and lists, we find both multisets and a new data structure, called *compact lists*. Compact lists are lists in which contiguous occurrences of the same element are immaterial: a property complementary to that characterizing multisets. Their practical usage in programming has not been explored yet, although some possible examples are suggested in [12].

Lists, multisets, compact lists and sets have been studied in the context of (Constraint) Logic Programming (CLP) languages. In this context all these data structures are conveniently represented as terms, using four different data aggregate constructors endowed with the proper interpretations. The 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 (technically, we are in a *general* context). [12], however, focuses only on *equality* between terms in each of the four theories. This amounts to solve the relevant problems of unification in the equational theories describing the properties of the four considered data structures. Unification algorithms for these four data structures are provided in [12]; NP-unification algorithms for sets and/or multisets are also presented in [1, 10].

In this paper we extend the results of [12] to the case of more general constraints. The constraints we consider are arbitrary conjunctions of literals (i.e., positive and negative atoms) based on both equality and membership predicate symbols. The problem of dealing with such kind of constraints in the context of CLP languages has been already faced in [14], but limited to the set data structure. In this paper, in contrast, we face the same problem for all the four data structures mentioned above. We identify the privileged models for the axiomatic theories used to describe the considered data structures. We define a notion of (satisfiable) solved form for constraints that are conjunctions of positive and negative equality and membership constraints. We develop the rewriting algorithms which map these constraints into solved form constraints—proved to be correct and terminating—for all the four theories.

The whole presentation will be parametric with respect to the considered axiomatic theories, high-lightening differences and similarities between the four aggregates. As a consequence, the proposed solutions (axiomatic theories, structures, constraint satisfiability procedures) can be easily combined so as to account for more than one data structure at a time.

The paper is organized as follows. In Section 2 we fix the overall notation and we recall from [12] the (parametric) axiomatic presentation of the first-order theories we deal with. In Section 3 we define the privileged models and we show that they correspond with the related theories. We also present a global notion of solved form for constraints that ensures satisfiability in the four models. In Section 4 we briefly discuss constraint solving when constraints are conjunctions of equality atoms (unification problem) and we recall the results from [12]. In Section 5 we describe, for each kind of data structure, the constraint rewriting procedures used to eliminate the literals not in solved form possibly occurring in a given constraint, while in the next section, Section 6, we show how to solve parametrically the general satisfiability problem for the admissible constraints. In Section 7 we show how it is possible (and simple) to combine the procedures developed in order to obtain a unique general framework. Finally, some conclusions are drawn in Section 8. Due to lack of space, we omit all the proofs. They will be available in a forthcoming technical report of the University of Parma toghether with the analysis of Bag theories with multi-membership.

2 Preliminaries

We assume basic knowledge of first-order logic (e.g., [15, 7]). A first-order language $\mathcal{L} = \langle \Sigma, \mathcal{V} \rangle$ is defined by a *signature* $\Sigma = \langle \mathcal{F}, \Pi \rangle$ composed by a set $\mathcal{F}$ of constant and function symbols and a set Π of predicate symbols, and by a denumerable set $\mathcal{V}$ of logical variables.

Usually, capital letters X, Y, Z, etc. will be used to represent variables, f, g, etc. to represent function symbols, and p, q, etc. to represent predicate symbols. We will use $\bar{X}$ to denote a (possibly empty) sequence of variables. $T(\mathcal{F}, \mathcal{V})$ ($T(\mathcal{F})$) denotes the set of first-order terms (resp., ground terms) built from $\mathcal{F}$ and $\mathcal{V}$ (resp., $\mathcal{F}$). Given a sequence of terms $t_1, \ldots, t_n$, $FV(t_1, \ldots, t_n)$ will be used to denote the set of all the variables which occur in at least one of the terms t_i. When the context is clear, we will use $\bar{t}$ to denote a sequence $t_1, \ldots, t_n$ of terms.

An *atomic formula* (atom) is an object of the form $p(t_1, \ldots, t_n)$, where $p \in \Pi$, $ar(p) = n$ and $t_i \in T(\mathcal{F}, \mathcal{V})$. The *formulae* are built up from the atomic ones using first-order connectives $(\wedge, \vee, \neg, \ldots)$ and quantifiers $(\exists, \forall)$. We assume the standard notion of *free* variables and we use $FV(\varphi)$ to denote the set of free variables in the first-order formula φ. If $FV(\varphi) = \emptyset$, then the formula is said to be *closed.* $\vec{\exists}\varphi$ ($\vec{\forall}\varphi$) denotes the existential (universal) closure of the formula φ, namely $\exists X_1 \cdots X_n\,\varphi$ ($\forall X_1 \cdots X_n\,\varphi$), where $\{X_1, \ldots, X_n\} = FV(\varphi)$.

A Σ-*structure* (or, simply, a *structure*) $\mathcal{A}$ is composed by a non-empty domain A and by an interpretation function $(\cdot)^{\mathcal{A}}$ which assigns functions and relations on A to the symbols of Σ. A *valuation* σ is a function from a subset of $\mathcal{V}$ in A. Each valuation can be exteded to a function from $T(\mathcal{F}, \mathcal{V})$ in A and to a function from the set of formulae over $\mathcal{L}$ on the set $\{\mathtt{false}, \mathtt{true}\}$. A valuation σ is said a *successful valuation* of φ if $\sigma(\varphi) = \mathtt{true}$.

A *(first-order) theory* $\mathcal{T}$ on $\mathcal{L}$ 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 the 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.

A *substitution* is a mapping $\theta : \mathcal{V} \longrightarrow T(\mathcal{F}, \mathcal{V})$. A substitution is then extended inductively to terms as usual. With ε we denote the empty substitution, namely the substitution such that $\varepsilon(x) = x$ for all variables x. A substitution θ is a $\mathcal{T}$-*unifier* of two terms t, t' if $\mathcal{T} \models \vec{\forall}(\theta(t) = \theta(t'))$ [27].

A *constraint (admissible constraint)* is a conjunction of *literals*, namely atomic formulae or negation of atomic formulae. When C is a constraint, $|C|$ is used to denote the number of occurrences of variables, constant, function, and predicate symbols in C.

Given a theory $\mathcal{T}$ on $\mathcal{L}$ and a structure $\mathcal{A}$, $\mathcal{T}$ and $\mathcal{A}$ *correspond* on the set of admissible constraints Adm [19] if, for each constraint $C \in Adm$, we have that $\mathcal{T} \models \vec{\exists}(C)$ if and only if $\mathcal{A} \models \vec{\exists}(C)$. This property guarantees that $\mathcal{A}$ is a canonical model of $\mathcal{T}$ with respect to Adm: if C is an element of Adm and we know that C is satisfiable in $\mathcal{A}$ then it will be satisfiable in all the models of $\mathcal{T}$.

The following binary function symbols are introduced to denote lists, multisets, compact lists, and sets:

$[\cdot\,|\,\cdot]$	for lists,	$\{\!\!\{\cdot\,|\,\cdot\}\!\!\}$	for multisets,
$[\![\cdot\,|\,\cdot]\!]$	for compact lists,	$\{\cdot\,|\,\cdot\}$	for sets.

The empty list, multiset, compact list, and set are all denoted by the constant symbol $\mathtt{nil}$. We use simple syntactic conventions and notations for terms built using these symbols. In particular, the list $[\,s_1\,|\,[\,s_2\,|\,\cdots\,[\,s_n\,|\,t\,]\cdots]]$ will be denoted by $[s_1, \ldots, s_n\,|\,t]$ or simply by $[s_1, \ldots, s_n]$ when t is $\mathtt{nil}$. The conventions used for lists will be exploited also for multisets, compact lists, and sets.

In [12] it is proposed a uniform parametric axiomatization of the data structures lists, multisets, compact lists, and sets that we briefly recall below. In each axiomatic theory $\mathcal{T}$ used to describe these data structures we have that $\Pi_{\mathcal{T}} = \{=, \in\}$ and $\mathcal{F}_{\mathcal{T}}$ contains the constant symbol $\mathtt{nil}$, exactly one among $[\cdot\,|\,\cdot]$, $\{\!\!\{\cdot\,|\,\cdot\}\!\!\}$, $[\![\cdot\,|\,\cdot]\!]$, or $\{\cdot\,|\,\cdot\}$, plus possibly other (free) constant and function symbols.

These theories, therefore, are *hybrid theories*: the objects they deal with are built out of interpreted as well as uninterpreted symbols. In particular, lists (multisets, compact lists, sets) may contain uninterpreted Herbrand terms as well as other lists (resp., multisets, compact lists, sets). Moreover, all the data aggregates 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 data structures (namely, lists, multisets, compact lists, and sets).

2.1 Lists

Let us consider a first-order language $\mathcal{L}_{List} = \langle \Sigma_{List}, \mathcal{V} \rangle$ over a signature $\Sigma_{List} = \langle \mathcal{F}_{List}, \Pi \rangle$ such that the binary function symbol $[\cdot\,|\,\cdot]$ and the constant symbol `nil` are in $\mathcal{F}_{List}$, and $\Pi = \{=, \in\}$. A first-order theory for lists over the language $\mathcal{L}_{List}$—called *List*—is shown in the figure below:

(K)	$\forall x\, y_1 \cdots y_n$	$(x \notin f(y_1, \ldots, y_n))$	$f \in \mathcal{F}_{List}, f \not\equiv [\cdot\,|\,\cdot]$
(W)	$\forall y\, v\, x$	$(x \in [\,y\,|\,v\,] \leftrightarrow x \in v \vee x = y)$	
(F_1)	$\forall x_1 \cdots x_n y_1 \cdots y_n$	$\left(\begin{array}{c} f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \\ \rightarrow x_1 = y_1 \wedge \cdots \wedge x_n = y_n \end{array} \right)$	$f \in \mathcal{F}_{List}$
(F_2)	$\forall x_1 \cdots x_m y_1 \cdots y_n$	$f(x_1, \ldots, x_m) \neq g(y_1, \ldots, y_m)$	$f, g \in \mathcal{F}_{List}, f \not\equiv g$
(F_3)	$\forall x$	$(x \neq t[x])$	
	where $t[x]$ denotes a term, having x as proper subterm		

The three axiom schemata (F_1), (F_2), and (F_3) (called freeness axioms, or Clark's equality axioms—see [8]) have been originally introduced by Mal'cev in [22]. Observe that axiom (F_1) holds for $[\cdot\,|\,\cdot]$ as a particular case. Axiom (F_3) states that there does not exist a 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* (see, e.g., [21]) which has the aim, among others, of guaranteeing the *acyclicity* of membership. Note that (K) implies that $\forall x\,(x \notin \texttt{nil})$.

2.2 Multisets

Let $\mathcal{L}_{Bag} = \langle \Sigma_{Bag}, \mathcal{V} \rangle$ be a language over a signature $\Sigma_{Bag} = \langle \mathcal{F}_{Bag}, \Pi \rangle$ such that the binary function symbol $\{\!\!\{ \cdot\,|\,\cdot \}\!\!\}$ and the constant symbol `nil` are in $\mathcal{F}_{Bag}$, and $\Pi = \{=, \in\}$. A hybrid theory of multisets—called *Bag*—can be simply obtained from the theory of lists shown above. The constructor $[\cdot\,|\,\cdot]$ used for lists is replaced by the binary function symbol $\{\!\!\{ \cdot\,|\,\cdot \}\!\!\}$. The behavior of this new symbol is regulated by the following equational axiom

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

which, intuitively, states that the order of elements in a multiset is immaterial (*permutativity property*). Axioms (K), (W), (F_2), and (F_3) of *List*—with $[\cdot\,|\,\cdot]$ replaced by $\{\!\!\{ \cdot\,|\,\cdot \}\!\!\}$ and $\mathcal{F}_{List}$ replaced by $\mathcal{F}_{Bag}$—still hold. Conversely, axiom schema (F_1) does not hold for multisets, when f is instantiated to $\{\!\!\{ \cdot\,|\,\cdot \}\!\!\}$.

The same is true for compact lists and sets. Thus, in the general case—that is, assuming that also the symbols for compact lists and sets are introduced—axiom schema (F_1) is replaced by:

$$\boxed{\begin{array}{lll} (F_1') & \forall x_1 \cdots x_n y_1 \cdots y_n & \left(\begin{array}{c} f(x_1,\ldots,x_n) = f(y_1,\ldots,y_n) \\ \to x_1 = y_1 \wedge \cdots \wedge x_n = y_n \end{array} \right) \\ & \multicolumn{2}{l}{\textit{for any } f \in \mathcal{F}_{Bag} \cup \mathcal{F}_{CList} \cup \mathcal{F}_{Set},} \\ & \multicolumn{2}{l}{f \textit{ distinct from } \{\!\{ \cdot | \cdot \}\!\},\ [\![\cdot | \cdot]\!],\ \{ \cdot | \cdot \}} \end{array}}$$

In $KWE_p^m F_1' F_2 F_3$, however, we lack in a general criterion for establishing equalities and disequalities between multisets. To obtain it, the following *multiset extensionality* property is introduced:

> *Two (hybrid) multisets are equal if and only if they have the same number of occurrences of each element, regardless of their order.*

The axiom proposed in [12] to force this property is the following:

$$\boxed{\begin{array}{lll} (E_k^m) & \forall y_1 y_2 v_1 v_2 & \left(\begin{array}{l} \{\!\{ y_1 \,|\, v_1 \}\!\} = \{\!\{ y_2 \,|\, v_2 \}\!\} \ \leftrightarrow \\ \qquad (y_1 = y_2 \wedge v_1 = v_2) \vee \\ \qquad \exists z\, (v_1 = \{\!\{ y_2 \,|\, z \}\!\} \wedge v_2 = \{\!\{ y_1 \,|\, z \}\!\}) \end{array} \right) \end{array}}$$

Observe that (E_k^m) implies (E_p^m). (E_k^m) is needed for establishing disequalities between bags.

2.3 Compact lists

Let $\mathcal{L}_{CList} = \langle \Sigma_{CList}, \mathcal{V} \rangle$ be a first-order language over a signature $\Sigma_{CList} = \langle \mathcal{F}_{CList}, \Pi \rangle$ such that the binary function symbol $[\![\cdot | \cdot]\!]$ and the constant symbol `nil` are in $\mathcal{F}_{CList}$, and $\Pi = \{=, \in\}$. Similarly to bags, a hybrid theory of *compact lists*—called *CList*—can be obtained from the theory of lists with only a few changes. The list constructor symbol is replaced by the binary function symbol $[\![\cdot | \cdot]\!]$, to be used as the compact list constructor. The behavior of this symbol is regulated by the equational axiom

$$\boxed{(E_a^c) \qquad \forall xy\, [\![x, x \,|\, y]\!] = [\![x \,|\, y]\!]}$$

which, intuitively, states that contiguous duplicates in a compact list are immaterial (*absorption property*). An example showing usefulness of compact lists comes from formal languages: let $s_1, \ldots, s_m, t_1, \ldots, t_n$ be elements of an alphabet, then

$$\begin{array}{ll} s_1^+ \cdots s_m^+ \text{ and } t_1^+ \cdots t_n^+ & \text{are the same regular expression} \\ \text{if and only if} & [\![s_1, \ldots, s_m]\!] = [\![t_1, \ldots, t_n]\!]\,. \end{array}$$

As for multisets, a general criterion for establishing both equality and disequality between compact lists is needed. This is obtained by introducing the following axiom:

$$\boxed{\begin{array}{lll} (E_k^c) & \forall y_1 y_2 v_1 v_2 & \left(\begin{array}{l} [\![y_1 \,|\, v_1]\!] = [\![y_2 \,|\, v_2]\!] \ \leftrightarrow \\ \qquad (y_1 = y_2 \wedge v_1 = v_2) \vee \\ \qquad (y_1 = y_2 \wedge v_1 = [\![y_2 \,|\, v_2]\!]) \vee \\ \qquad (y_1 = y_2 \wedge [\![y_1 \,|\, v_1]\!] = v_2) \end{array} \right) \end{array}}$$

Note that axiom (E_a^c) is implied by (E_k^c). Axioms (K), (W), (F_2)—with $[\cdot|\cdot]$ replaced by $[\![\cdot | \cdot]\!]$ and $\mathcal{F}_{List}$ replaced by $\mathcal{F}_{CList}$—and axiom (F_1') introduced for multisets, still hold. The freeness axiom (F_3), instead, needs to be suitably modified. As opposed to lists and multisets, an equation such as $X = [\![\, \mathtt{nil} \mid X \,]\!]$ admits a finite tree solution, namely a solution that binds X to the term $[\![\, \mathtt{nil} \mid t \,]\!]$, where t is any term. Therefore, axiom (F_3) is replaced by

$$\boxed{\begin{array}{ll} (F_3^c) & \forall x \quad (x \neq t[x]) \\ & \textit{unless } t \textit{ has the form } [\![\, t_1, \ldots, t_n \mid x \,]\!], \\ & x \textit{ not occurring in } t_1, \ldots, t_n, \textit{ and } t_1 = \cdots = t_n \end{array}}$$

2.4 Sets

Let $\mathcal{L}_{Set} = \langle \Sigma_{Set}, \mathcal{V} \rangle$ be a first-order language over a signature $\Sigma_{Set} = \langle \mathcal{F}_{Set}, \Pi \rangle$ such that the binary function symbol $\{\cdot \mid \cdot\}$ and the constant symbol $\mathtt{nil}$ are in $\mathcal{F}_{Set}$, and $\Pi = \{=, \in\}$. The last theory we consider is the simple theory of sets *Set*. Sets have both the *permutativity* and the *absorption properties* which, in the case of $\{\cdot \mid \cdot\}$, can be rewritten as follows:

$$\boxed{\begin{array}{lrcl} (E_p^s) & \forall xyz \, \{x, y \mid z\} & = & \{y, x \mid z\} \\ (E_a^s) & \forall xy \, \{x, x \mid y\} & = & \{x \mid y\} \end{array}}$$

A criterion for testing equality (and disequality) between sets is obtained by merging the multiset equality axiom (E_k^m) and the compact list equality axiom (E_k^c):

$$\boxed{(E_k^s) \qquad \forall y_1 y_2 v_1 v_2 \left(\begin{array}{r} \{y_1 \mid v_1\} = \{y_2 \mid v_2\} \;\leftrightarrow \qquad\qquad \\ (y_1 = y_2 \wedge v_1 = v_2) \vee \\ (y_1 = y_2 \wedge v_1 = \{y_2 \mid v_2\}) \vee \\ (y_1 = y_2 \wedge \{y_1 \mid v_1\} = v_2) \vee \\ \exists k \, (v_1 = \{y_2 \mid k\} \wedge v_2 = \{y_1 \mid k\}) \end{array} \right)}$$

According to (E_k^s) duplicates and ordering of elements in sets are immaterial. Thus, (E_k^s) implies the equational axioms (E_p^s) and (E_a^s). In [12] it is also proved that they are equivalent when domains are made by terms.

Axioms (K), (W), (F_2)—with $[\cdot \mid \cdot]$ replaced by $\{\cdot \mid \cdot\}$, and $\mathcal{F}_{List}$ replaced by $\mathcal{F}_{Set}$—and axiom (F_1') introduced for multisets, still hold. The modification of axiom (F_3) for sets, instead, simplifies the one used for compact lists:

$$\boxed{\begin{array}{ll} (F_3^s) & \forall x \quad (x \neq t[x]) \\ & \textit{unless } t \textit{ has the form } \{t_1, \ldots, t_n \mid x\}, \; x \textit{ not occurring in } t_1, \ldots, t_n \end{array}}$$

Figure 1 summarizes the four theories. The two right-most axioms, Perm. (Permutativity) and Abs. (Absorption) are implied by (E_k^*) axioms and so they are actually superfluous. However, they are sufficient to characterize the theories from an equational point of view (see Section 3.1).

3 Privileged structures and solved form

In this section we briefly recall the privileged structures proposed in [12] for the four theories of the previous section. Then, we show that these structures and the theories correspond on the class of constraints analyzed. Moreover, we give a general notion of solved form that holds for constraints in all the four theories, and we show that a constraint in solved form is satisfiable in the corresponding privileged model (hence, in all the models of the theory, thanks to the correspondence result).

Name	empty	with	Equality		Herbrand	Acycl.	Perm.	Abs.
List	(K)	(W)	(F_1)		(F_2)	(F_3)		
Bag	(K)	(W)	(E_k^m)	(F_1')	(F_2)	(F_3)	•	
CList	(K)	(W)	(E_k^c)	(F_1')	(F_2)	(F_3^c)		•
Set	(K)	(W)	(E_k^s)	(F_1')	(F_2)	(F_3^s)	•	•

Figure 1: Axioms for the four theories

3.1 Privileged structures

In Section 2 we have presented four first-order hybrid theories for aggregates. For each of them, the behavior of a particular function symbol—the relevant aggregate constructor—is precisely characterized by an *equational theory*:

- E_{List}, the empty theory for *List*,
- E_{Bag}, the theory consisting of the *Permutativity* axiom (E_p^m) for *Bag*,
- E_{CList}, the theory consisting of the *Absorption* axiom (E_a^c) for *CList*,
- E_{Set}, the theory consisting of both the *Permutativity* (E_p^s) and *Absorption* (E_a^s) axioms for *Set*.

Using the appropriate equational theory we can define for each different kind of aggregate a privileged model for the relevant first-order theory. Let $\mathbb{T}$ be either *List* or *Bag* or *CList* or *Set*.

- The *domain* of the model is the quotient $T(\mathcal{F}_{\mathbb{T}})/\equiv_{\mathbb{T}}$ of the ordinary Herbrand Universe $T(\mathcal{F}_{\mathbb{T}})$ over the smallest congruence relation $\equiv_{\mathbb{T}}$ induced by the equational theory $E_{\mathbb{T}}$ on $T(\mathcal{F}_{\mathbb{T}})$.
- The interpretation of a term t is its equivalence class $[t]$.
- $=$ is interpreted as the identity on the domain $T(\mathcal{F}_{\mathbb{T}})/\equiv_{\mathbb{T}}$.
- 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 $[t_1, \ldots, t_n, t \,|\, r]$ ($\{\!\!\{ t_1, \ldots, t_n, t \,|\, r \}\!\!\}$, $[\![t_1, \ldots, t_n, t \,|\, r]\!]$, or $\{t_1, \ldots, t_n, t \,|\, r\}$) for some terms $t_1, \ldots, t_n, r$.

Remark 3.1 *When $[s]$ is a multiset or a set, since the permutativity property holds, the requirement for $[t] \in [s]$ to be* `true` *can be simplified to: $[s]$ contains a term of the form $\{\!\!\{ t \,|\, r \}\!\!\}$ or $\{t \,|\, r\}$, respectively.*

These structures—named $\mathcal{LIST}, \mathcal{BAG}, \mathcal{CLIST}$, and $\mathcal{SET}$—are important models for the theories of aggregates we are studying, as it ensues from the following theorem.

Theorem 3.2 *The structures $\mathcal{LIST}, \mathcal{BAG}, \mathcal{CLIST}$, and $\mathcal{SET}$ and the theories List, Bag, CList, and Set correspond on the class of admissible constraints.*

3.2 Solved form

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

Definition 3.3 *A constraint C is in* pre-solved form *if all its literals are in pre-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
- $t \in X$ *and* X *does not occur in* t
- $X \neq t$ *and* X *does not occur in* t
- $t \notin X$ *and* X *does not occur in* t.

In order to establish satisfiability of a constraint in pre-solved form we need to introduce two further conditions that must be satisfied by the constraint, in particular by membership literals. If both conditions are satisfied we will say that the constraint is in *solved form*. Solved form constraints will be proved to be always satisfiable in the corresponding structure.

The first condition is informally motivated by the following example. Consider the constraint $X \in Y \wedge Y \in X$. It is in pre-solved form but it is clearly unsatisfiable in the structures $\mathcal{LIST}, \mathcal{BAG}, \mathcal{CLIST}$, and $\mathcal{SET}$. These constraints could be satisfiable in non well-founded models of membership. This topic is studied in [1] for equality constraints in the theory *Set*.

The first condition takes care of these situations and is precisely defined as follows. Let C be a pre-solved form constraint and $C^{\in}$ be the part of C containing only $\in$-atoms. Build the directed graph $\mathcal{G}_{C^{\in}}$ as follows:

nodes: Associate a distinct node to each $X \in FV(C^{\in})$.

edges: If $t \in X$ is in $C^{\in}$, $\nu_1, \ldots, \nu_n$ are the nodes associated with the variables in t, and μ is the node associated with the variable X, then add the edges $\langle \nu_1, \mu \rangle, \ldots, \langle \nu_n, \mu \rangle$.

A pre-solved form constraint C is *acyclic* if $\mathcal{G}_{C^{\in}}$ is acyclic.

The second condition for pre-solved form constraints is intuitively motivated by the following observations. Consider the constraint $a \in X \wedge a \notin X$. It is in pre-solved form and acyclic but unsatisfiable. Conversely, the constraint $\{A\} \in X \wedge \{a\} \notin X$ is satisfiable in $\mathcal{SET}$—take, for instance, any value of $A \neq a$ and $X = \{\{A\}\}$.

More in general, it is easy to see that whenever there are two literals $t \in X$ and $t' \notin X$ in C and t and t' unify in the considered theory $E_{\mathbb{T}}$ with the empty substitution ε, the constraint C is unsatisfiable. For example, the constraint $\{A, B\} \in X \wedge \{B, A\} \notin X$ in $\mathcal{L}_{Set}$ is unsatisfiable (indeed, terms $\{A, B\}$ and $\{B, A\}$ unify in *Set* with the empty substitution ε).

This condition, however, does not cover all the possible cases in which an acyclic constraint in pre-solved form is unsatisfiable, as it ensues from the following example. Let C be the $\mathcal{L}_{Set}$-constraint $a \in X \wedge X \in Y \wedge \{a \,|\, X\} \notin Y$. Observe that there are no pairs of terms t, t' of the form singled out above. Nevertheless, since $a \in X$ is equivalent to $\exists N\, (X = \{a \,|\, N\})$, by applying the substitution for X we get the pair of literals $\{a \,|\, N\} \in Y$ and $\{a, a \,|\, N\} \notin Y$. $\{a \,|\, N\}$ and $\{a, a \,|\, N\}$ unify in *Set* with ε: the latter constraint (hence, the former since it has been obtained by equivalent rewritings) is unsatisfiable.

To formally define the second condition for pre-solved constraints, taking into account all the possible cases informally described above, we introduce the following definitions and the subsequent lemma.

Definition 3.4 *Given a substitution $\theta \equiv [X_1/t_1, \ldots, X_n/t_n]$ and a natural number $m \geq 0$ we define by induction on m the substitution θ^m as:*

$$\begin{array}{rcl} \theta^0 & \equiv & \varepsilon \\ \theta^{m+1} & \equiv & [X_1/\theta^m(t_1), \ldots, X_n/\theta^m(t_n)] \end{array}$$

If there exists $m > 0$ such that $\theta^{m+1} \equiv \theta^m$ we say that θ is convergent. *Given a convergent substitution θ the* closure *θ^* of θ is the substitution θ^m such that $\forall k > m$ it holds $\theta^k \equiv \theta^m$.*

Definition 3.5 *Let C be a constraint in pre-solved form over the language $\mathcal{L}_{List}$ ($\mathcal{L}_{Bag}, \mathcal{L}_{CList}, \mathcal{L}_{Set}$) and let*

$$p_1^1 \in X_1, \ldots, p_1^{k_1} \in X_1, \ldots, p_q^1 \in X_q, \ldots, p_q^{k_q} \in X_q$$

be all the membership atoms of C. We define the member substitution *σ_C as follows:*

$$\sigma_C \equiv [X_1/[F_1, p_1^1, \ldots, p_1^{k_1} \mid M_1], \ldots, X_q/[F_q, p_q^1, \ldots, p_q^{k_q} \mid M_q]]$$

(resp., $\sigma_C \equiv [X_1/\{\!\{ F_1, p_1^1, \ldots, p_1^{k_1} \mid M_1 \}\!\}, \ldots,]$, $\sigma_C \equiv [X_1/[\![F_1, p_1^1, \ldots, p_1^{k_1} \mid M_1]\!], \ldots,]$, $\sigma_C \equiv [X_1/\{F_1, p_1^1, \ldots, p_1^{k_1} \mid M_1\}, \ldots,]$) where F_i and M_i are new variables not occurring in C.

Lemma 3.6 *If C is a constraint in pre-solved form and acyclic, and σ_C is its member substitution, then σ_C is convergent and $\sigma_C^* \equiv \sigma_C^{q-1}$, where q is the number of variables which occur in the right-hand side of membership atoms.*

As an example, let C be the pre-solved form and acyclic $\mathcal{L}_{Set}$-constraint

$$a \in Y \wedge Y \in X \wedge X \in Z \wedge \{\{a \mid Y\} \mid X\} \notin Z \tag{1}$$

It holds that:

$$\begin{array}{rcl} \sigma_C & \equiv & [Y/\{F_Y, a \mid M_Y\}, X/\{F_X, Y \mid M_X\}, Z/\{F_Z, X \mid M_Z\}]\,, \\ \sigma_C^* & \equiv & [Y/\{F_Y, a \mid M_Y\}, X/\{F_X, \{F_Y, a \mid M_Y\} \mid M_X\}, \\ & & \;Z/\{F_Z, \{F_X, \{F_Y, a \mid M_Y\} \mid M_X\} \mid M_Z\}] \end{array}$$

We are now ready to introduce the definition of solved form.

Definition 3.7 *Let $E_{\mathbb{T}}$ be one of the four equational theories associated with the four kinds of aggregates. A constraint C in pre-solved form and acyclic is in* solved form *if for each pair of literals of the form $t \notin X, t' \in X$ in C we have that:*

$$E_{\mathbb{T}} \not\models \vec{\forall}(\sigma_C^*(t) = \sigma_C^*(t')).$$

The condition in the Definition 3.7 requires the ability to perform the test $E_{\mathbb{T}} \models \vec{\forall}(s = s')$ for any pair of terms s and s' in $\mathcal{L}_{\mathbb{T}}$. This test is connected with the availability of a unification algorithm for the theory $E_{\mathbb{T}}$. As a matter of fact, this test is equivalent to check if the empty substitution ε is a $E_{\mathbb{T}}$-unifier of $s = s'$. Since in [12] it is proved that the four theories we are dealing with are finitary (i.e., they admit a finite set of mgu's that covers all possible unifiers), this can be done using a unification algorithm for the theory at hand.

As an example, consider again the constraint (1). It holds that

$$\begin{array}{rcl}\sigma^*_C(X) & \equiv & \{F_X, \{F_Y, a \,|\, M_Y\} \,|\, M_X\} \\ \sigma^*_C(\{\{a \,|\, Y\} \,|\, X\}) & \equiv & \{\{a, F_Y, a \,|\, M_Y\}, F_X, \{F_Y, a \,|\, M_Y\} \,|\, M_X\}\end{array}$$

Hence, the constraint is not in solved form since $E_{Set} \models \vec{\forall}(\sigma^*_C(X) = \sigma^*_C(\{\{a \,|\, Y\} \,|\, X\}))$. Observe that using σ_C instead of σ^*_C the situation would not be detected, since $\{F_X, Y \,|\, M_X\} = \{\{a, F_Y, a \,|\, M_Y\}, F_X, Y \,|\, M_X\}$ is not satisfied, for instance, when $F_X \neq \{F_Y, a \,|\, M_Y\} \wedge Y \neq \{F_Y, a \,|\, M_Y\} \wedge \{F_Y, a \,|\, M_Y\} \notin M_X$.

Remark 3.8 *The solved form considered in [14], where only sets are taken into account, differs from the one considered in this paper in that the former does not include any atom of the form $t \in X$. As a matter of fact, in the theory Set it holds that (see also Remark* 3.1*) $s \in t \leftrightarrow \exists N(t = \{s \,|\, N\})$. Thus, in Set all membership constraints can be always replaced by equivalent equality constraints. This in turn implies that the additional conditions on the pre-solved form are not required at all. Cycle detection, for instance, is simply delegated to the unification algorithm used by the constraint rewriting procedure. The same holds also for multisets, but unfortunately it does not hold for lists and compact lists. In fact, $t \in X$ in the theory List (as well as in CList) cannot be replaced by a finite number of equality constraints. Therefore, since we want to have a single solved form which is adequate for all the four theories considered in this paper—in view of the combination of them into a single theory—we need to keep also atoms of the form $t \in X$ as irreducible constraints which can therefore occur in the solved form. Consequently, we added the further conditions on the literals to characterize solved forms in order to guarantee satisfiability.*

Theorem 3.9 *Let C_{List} (C_{Bag}, C_{CList}, C_{Set}) be a constraint in solved form over the language $\mathcal{L}_{List}$ (resp., $\mathcal{L}_{Bag}$, $\mathcal{L}_{CList}$, $\mathcal{L}_{Set}$). C_{List} (C_{Bag}, C_{CList}, C_{Set}) is satisfiable in $\mathcal{LIST}$ (resp., $\mathcal{BAG}$, $\mathcal{CLIST}$, $\mathcal{SET}$).*

4 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 the four theories discussed in Section 2 have been proposed in [12]. They have been proved to terminate and to be sound and complete with respect to the corresponding axiomatic theories (namely, *List*, *Bag*, *CList*, and *Set*). It has been shown that the equality constraints are parametric with respect to these theories and that it is easy to merge them and to work in the combined theory that takes into account the four proposed data structures simultaneously.

The unification algorithms proposed in [12], namely:

- Unify_lists for lists,
- Unify_bags for multisets,

- Unify_clists for compact lists, and
- Unify_sets for sets,

will be used unaltered in the four global constraint solvers that we are going to propose in this paper (Section 6).

The output of the algorithms is either `false`, when the constraint is unsatisfiable, or a disjunction of solved form constraints (Def. 3.7) composed only by equality atoms. The complexity results for unification problems have been studied and proved to require linear time for lists, and to be NP-complete for the other forms of data aggregates.

5 Constraint rewriting procedures

We now extend the results presented in [12] for equality constraints to the whole classes of admissible constraints for the four constraint domains. We describe the constraint rewriting procedures used to eliminate all the literals not in pre-solved form possibly occurring in a given constraint C. We provide a different procedure for each kind of constraint literals—except for equality constraints whose constraint simplification procedures are constituted by the unification algorithms mentioned in the previous section. In the next section we will show how these procedures can be combined to test satisfiability, in the corresponding privileged structure, of any constraint written in one of the considered languages.

As done with the unification algorithms we will stress the parametric nature of all the procedures we define, by keeping their presentation as independent as possible from the kind of data aggregate they deal with. This will serve to let the merging of these procedures into a single general procedure be a straightforward step.

5.1 Lists

We begin the investigation with the theory *List*. If a constraint is a conjunction of equality atoms, then the decision problem for satisfiability can be solved in linear time, since it is simply a standard unification problem ([23, 24]).

If a constraint C is a conjunction of equalities and disequalities, then the satisfiability problem for *List* is still solvable in polynomial time $O(n^2)$ where $n = |C|$ [2, 9]. As far as disequalities are concerned, they can be managed by the procedure neq-list of Figure 2.

Lemma 5.1 *Let C be a constraint. Then $List \models \vec{\forall}(C \leftrightarrow \mathsf{neq\text{-}list}(C))$. Moreover,* neq-list($C$) *can be implemented so as to run in time $O(n)$, where $n = |C|$.*

These polynomial results can not be extended to all the admissible constraints.

Theorem 5.2 *The satisfiability problem for conjunctions of $\in$-atoms and $\neq$-literals in List is NP-hard.*

We have seen how to reduce equality constraints (algorithm Unify_lists, Section 4) and disequality constraints (algorithm neq-list, Figure 2). In Figure 3 we show the

function neq-list(C)
    while there is a $\neq$-constraint c not in pre-solved form in C do
    case c of

(1)	$d \neq d$ d is a constant	$\mapsto$	`false`
(2)	$f(s_1, \ldots, s_m) \neq g(t_1, \ldots, t_n)$ $f \not\equiv g$	$\mapsto$	`true`
(3)	$t \neq X$ t is not a variable	$\mapsto$	$X \neq t$
(4)	$X \neq X$ X is a variable	$\mapsto$	`false`
(5)	$f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_n)$ $n > 0, f \not\equiv [\cdot \mid \cdot]$	$\mapsto$	$s_1 \neq t_1 \vee$ (i) $\vdots$ $\vdots$ $s_n \neq t_n$ (n)
(6)	$[s_1 \mid s_2] \neq [t_1 \mid t_2]$	$\mapsto$	$s_1 \neq t_1 \vee$ (i) $s_2 \neq t_2$ (ii)
(7)	$X \neq f(t_1, \ldots, t_n)$ $X \in FV(t_1, \ldots, t_n)$	$\mapsto$	`true`

Figure 2: Rewriting procedure for disequations over lists

rewriting procedures in-list and nin-list for membership and negated membership literals over lists.

Theorem 5.3 *Given a constraint C, $\mathcal{LIST} \models \vec{\forall}(C \leftrightarrow \textsf{nin-list}(\textsf{in-list}(C)))$.*

In the proof of Theorem 5.3 no one of the axioms that distinguish the four theories is involved. Thus, the rewriting procedures for $\in$ and $\notin$ constraints over bags, compact lists and sets can be obtained from in-list and nin-list by replacing $[\cdot \mid \cdot]$ with the corresponding aggregate constructor symbol. When useful, we will refer to these procedures with the generic names in-$\mathbb{T}$ and nin-$\mathbb{T}$, where $\mathbb{T}$ is any of the considered theories.

Corollary 5.4 *Let C be a constraint and $\mathbb{T}$ be one of the theories Bag, CList, and Set. Then $\mathcal{A} \models \vec{\forall}(C \leftrightarrow \textsf{nin-}\mathbb{T}(\textsf{in-}\mathbb{T}(C)))$ where $\mathcal{A}$ is the structure corresponding to the theory $\mathbb{T}$.* □

The following lemma will be useful to prove soundness and completeness of the global constraint solving procedure for *List*.

Lemma 5.5 *Let t, t' be two terms and C a solved form constraint over the language $\mathcal{L}_{List}$, such that $FV(t) \cup FV(t') \subseteq FV(C)$. If $\mathcal{LIST} \not\models \vec{\forall}(t = t')$, then $E_{List} \not\models \vec{\forall}(\sigma^*_C(t) = \sigma^*_C(t'))$.*

5.2 Multisets

We already know from [12] that the decision problem for multiset unification is NP-complete. Thus, the global satisfiability test is NP-hard. We know also that the

function in-list(C)
while there is a $\in$-constraint c in C not in pre-solved form do
case c of

(1)	$\left.\begin{array}{r} r \in f(t_1,\dots,t_n) \\ f \not\equiv [\cdot\,\vert\,\cdot] \end{array}\right\}$	$\mapsto$	`false`
(2)	$r \in [t\,\vert\,s]\ \}$	$\mapsto$	$r = t \lor$ (a) $r \in s$ (b)
(3)	$\left.\begin{array}{r} r \in X \\ X \in FV(r) \end{array}\right\}$	$\mapsto$	`false`

function nin-list(C)
while there is a $\notin$-constraint c in C not in pre-solved form do
case c of

(1)	$\left.\begin{array}{r} r \notin f(t_1,\dots,t_n) \\ f \not\equiv [\cdot\,\vert\,\cdot] \end{array}\right\}$	$\mapsto$	`true`
(2)	$r \notin [t\,\vert\,s]\ \}$	$\mapsto$	$r \neq t \land r \notin s$
(3)	$\left.\begin{array}{r} r \notin X \\ X \in FV(r) \end{array}\right\}$	$\mapsto$	`true`

Figure 3: Rewriting procedures for $\in$ and $\notin$ constraints over lists

same complexity results hold for compact list and set unification. Thus, the global satisfiability test will be NP-hard for all the considered data structures.

Equality constraints are managed by Unify_bags (see Section 4). Furthermore, thanks to Corollary 5.4, we know that the rewriting procedures in-list and nin-list developed for lists (see Figure 3) can be used almost unaltered also for bags.

As far as disequality constraints are concerned, a rewriting procedure—called neq-bag—capable of eliminating disequality constraints not in pre-solved form from the input constraint is presented in Figure 4.

In this procedure we make use of the functions tail and untail which are defined as follows:[1]

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

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

$$\begin{array}{rcl} \{\!\!\{t_1\,|\,s_1\}\!\!\} \neq \{\!\!\{t_2\,|\,s_2\}\!\!\} & \leftrightarrow & (t_1 \neq t_2 \lor s_1 \neq s_2)\land \\ & & \forall N\,(s_2 \neq \{\!\!\{t_2\,|\,N\}\!\!\} \lor s_1 \neq \{\!\!\{t_1\,|\,N\}\!\!\}) \end{array}$$

An 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

[1]Function tail is easily adapted to work with sets as well, assuming $\{\!\!\{\cdot\,|\,\cdot\}\!\!\}$ is replaced by $\{\cdot\,|\,\cdot\}$.

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

Figure 4: Rewriting procedure for $\neq$-constraints over bags

version of multiset equality and disequality. In particular, we have:

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

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

The rewriting rule shown in Figure 4 (rule (6.2)) avoids these difficulties introducing only existential quantification. Its correctness and completeness are proved in the following lemma.

Lemma 5.6 *Let C be a constraint. Then $\mathcal{BAG} \models \vec{\forall}(C \leftrightarrow \exists \bar{N}\, \mathsf{neq\text{-}bag}(C))$ where $\bar{N} = FV(\mathsf{neq\text{-}list}(C)) \setminus FV(C)$.*

Remark 5.7 *The procedure* in-bag *could safely be extended by the rule:*

(4)	$r \in X \quad \mapsto \quad X = \{\!\{ r \mid N \}\!\}$

where N is a new variable. One can add this rewriting rule, justified by the model $\mathcal{BAG}$, to reach a solved form that removes all occurrences of $\in$-constraints (see Remark 3.8) without affecting termination and completeness. As a matter of fact, none of the rewriting procedures Unify_bags, neq-bag, nin-bag *introduces $\in$-constraints. Thus, if we add rule (4) we are sure to completely remove $\in$-constraints from the constraints. Termination of this modified version of the algorithm follows trivially.*

The same considerations and results hold for sets but, as already observed in Remark 3.8, they do not hold for lists and compact lists. Therefore, when dealing with a theory at a time one could add rule (4) where appropriate. But when dealing with the global combined theory (see Section 7), since we assume that $\in$ is a polymorphic operator (i.e., it applies indistinctly to all the four types of data structures) and that there are no type declarations, we are no longer able to distinguish whether $t \in X$ can be rewritten using rule (4)—that is X is a set or a bag—or not. This is the

function neq-clist(C) while there is a $\neq$-constraint c in C not in pre-solved form do case c of			
(1)—(5) as in neq-list			
(6)	$[\![t_1 \mid s_1]\!] \neq [\![t_2 \mid s_2]\!] \} \mapsto$	$t_1 \neq t_2 \vee$ $s_1 \neq s_2 \wedge [\![t_1 \mid s_1]\!] \neq s_2 \wedge s_1 \neq [\![t_2 \mid s_2]\!]$	(a) (b)
(7.1)	$\left.\begin{array}{r} X \neq t \\ X \in FV(t), \\ X \text{ is not } [\![t_1, \ldots, t_n \mid X]\!], \\ n > 0 \text{ and } X \notin FV(t_1, \ldots, t_n) \end{array}\right\} \mapsto$	`true`	
(7.2)	$\left.\begin{array}{r} X \neq [\![t_1, \ldots, t_n \mid X]\!] \\ X \notin FV(t_1, \ldots, t_n) \end{array}\right\} \mapsto$	$t_1 \neq t_2 \vee$ $\vdots$ $t_1 \neq t_n \vee$ $X = \mathtt{nil} \vee$ $X = [\![N_1 \mid N_2]\!] \wedge N_1 \neq t_1$	$(a.1)$ $\vdots$ $(a.n)$ (b) (c)

Figure 5: Rewriting procedure for disequations over compact lists

reason why we prefer to not introduce this rule neither when dealing with bags and sets alone.

The following lemma will be useful to prove soundness and completeness of the global constraint solving procedure for *Bag*.

Lemma 5.8 *Let t, t' be two terms and C a solved form constraint over the language $\mathcal{L}_{Bag}$, such that $FV(t) \cup FV(t') \subseteq FV(C)$. If $\mathcal{BAG} \not\models \vec{\forall}(t = t')$, then $E_{Bag} \not\models \vec{\forall}(\sigma^*_C(t) = \sigma^*_C(t'))$.*

5.3 Compact Lists

As far as equality constraints are concerned, we can use the unification algorithm Unify_clists for compact lists (cf. Section 4). $\in$ and $\notin$ constraints are dealt with by the procedures in-clist and nin-clist trivially adapted from the same procedures for lists shown in Figure 3. It remains to deal with $\neq$-constraints. The rewriting procedure for this kind of constraints—called neq-clist—is shown in Figure 5.

Lemma 5.9 *Let $C_1, \ldots, C_k$ be the constraints non-deterministically returned by* neq-clist(C) *and $\bar{N}_i = FV(C_i) \setminus FV(C)$. Then $\mathcal{CLIST} \models \vec{\forall}\left(C \leftrightarrow \bigvee_{i=1}^{k} \exists \bar{N}_i C_i\right)$.*

Observe that, differently from multisets, the rewriting rule for disequality of compact lists mimics perfectly the axiom (E^c_k). This has been possible since this axiom does not introduce (new) existentially quantified variables.

The following lemma will be useful to prove soundness and completeness of the global constraint solving procedure for *CList*.

function neq-set(C) while there is a $\neq$-constraint c in C not in pre-solved form do case c of	
(1)—(5) as in neq-list	
(6)	$\{t_1\,\|\,s_1\} \neq \{t_2\,\|\,s_2\}\ \} \mapsto$ $Z \in \{t_1\,\|\,s_1\} \wedge Z \notin \{t_2\,\|\,s_2\} \vee \quad (a)$ $Z \in \{t_2\,\|\,s_2\} \wedge Z \notin \{t_1\,\|\,s_1\} \quad (b)$
(7.1)	$\left.\begin{array}{r} X \neq t \\ X \in FV(t), \\ X \text{ is not } \{t_1,\dots,t_n\,\|\,X\}, \\ n > 0 \text{ and } X \notin FV(t_1,\dots,t_n) \end{array}\right\} \mapsto \ \texttt{true}$
(7.2)	$\left.\begin{array}{r} X \neq \{t_1,\dots,t_n\,\|\,X\} \\ X \notin FV(t_1,\dots,t_n) \end{array}\right\} \mapsto \ t_1 \notin X \vee \quad (i)$ $\vdots \qquad \vdots$ $t_n \notin X \quad (n)$

Figure 6: Rewriting procedure for disequations over sets

Lemma 5.10 *Let t, t' be two terms and C a solved form constraint over the language $\mathcal{L}_{CList}$, such that $FV(t) \cup FV(t') \subseteq FV(C)$. If $\mathcal{CLIST} \not\models \vec{\forall}(t = t')$, then $E_{CList} \not\models \vec{\forall}(\sigma^*_C(t) = \sigma^*_C(t'))$.*

5.4 Sets

The handling of equalities involving sets is governed by the unification algorithm Unify_sets (cf. Section 4). Procedures in-set and nin-set—adapted from the corresponding procedures for lists shown in Figure 3—are used for membership literals involving sets. The remaining constraints, namely, $\neq$-constraints, are managed by the rewriting procedure neq-set shown in Figure 6.

Some remarks are needed regarding rule (6). As for multisets, axiom (E^s_k) introduces an existentially quantified variable to state equality. Thus, its direct application for stating disequality requires universally quantified constraints that go outside the language.

The rewriting rule (6.2) used for multisets can not be used in this context. In fact, the property that $s_1 \neq N$ implies $\{\!\{ t_1 \,|\, s_1 \}\!\} \neq \{\!\{ t_1 \,|\, N \}\!\}$, that holds for finite multisets does not hold for sets. For instance, $\{a\} \neq \{a, b\}$ but $\{b, a\} = \{b, a, b\}$. Thus, this rewriting rule would be not correct for sets.

A rewriting rule for set-disequalities can be obtained by taking the negation of the standard extensionality axiom extended to deal with hybrid colored sets:

$$\boxed{\begin{array}{llll} (E_k) & x = y & \leftrightarrow & \forall z\, (z \in x \leftrightarrow z \in y) \wedge \\ & & & \ker(x) = \ker(y) \end{array}}$$

$\ker(t)$ identifies the kernel of a ground term t (operationally, it is the same as function tail of Section 5.2). Intuitively, $\ker(t)$ is what remains of a set when all its elements have been removed. In "standard" sets, $\ker(s) = \texttt{nil}$. In colored sets, $\ker(s)$ can be any ground term of the form $f(t_1, \dots, t_n)$, with $f \not\equiv \{\cdot\,|\,\cdot\}$ (axiom (K) ensures that such terms—called kernels—do not contain any element). For instance,

$$
\begin{array}{lcl}
\mathsf{SAT}_{\mathbb{T}}(C) & = & \mathsf{repeat} \\
& & \quad C' := C; \\
& & \quad C := \mathsf{Unify\_\mathbb{T}s}(\mathsf{neq\text{-}\mathbb{T}}(\mathsf{nin\text{-}\mathbb{T}}(\mathsf{in\text{-}\mathbb{T}}(C)))); \\
& & \mathsf{until}\ C = C'; \\
& & \mathsf{return}(\mathsf{is\_solved}_{\mathbb{T}}(C)).
\end{array}
$$

Figure 7: The satisfiability procedure, parametric with respect to $\mathbb{T}$

$\ker(\{a \mid f(b)\}) = f(b)$. Axiom (E_k) has been proved in [12] to be equivalent to (E_k^s) in models whose domains are terms: in particular it holds in $\mathcal{SET}$.

This is the approach followed in [14]. Unfortunately this solution introduces some technical complications that require further special controls to check that a constraint possibly involving ker terms is satisfiable. We prefer to skip this issue here and refer the interested reader to [14].

In rule (6) of neq-set, therefore, we assume that all sets have the same kernel. If this would not be the case, then the neq-set procedure could be not correct. For example, $\{a \mid b\} \neq \{a \mid c\}$ is false according to rule (6), whereas it is true if also the kernels are taken into account. This simplification, however, is further motivated by the fact that in the combined theory (see Section 7) it will turn out to be convenient adding sorts to our underlying logic in order to avoid "mixed" aggregates—i.e., aggregates built using different aggregate constructors in the same term. The addition of sorts would provide also an immediate solution to the problem of colored sets, since sorts could force all sets to be based only on the empty set.

Lemma 5.11 *Let $C_1, \ldots, C_k$ be the constraints non-deterministically returned by* neq-set(C) *and $\bar{N}_i = FV(C_i) \setminus FV(C)$. Then $\mathcal{SET} \models \vec{\forall}\left(C \leftrightarrow \bigvee_{i=1}^{k} \exists \bar{N}_i C_i\right)$, provided rule* (6) *is never fired by two terms with different* ker.

6 Constraint solving

In this section we address the problem of establishing if a constraint C written in one of the languages studied in this paper is satisfiable in the related privileged structure—and, thus, in any structure that models the corresponding theory.

We show how to produce solution constraints, namely, returning an equisatisfiable disjunction of solved form constraints—for each of the four theories.

Constraint satisfiability for the theory $\mathbb{T}$ is checked by the non-deterministic rewriting procedure $\mathsf{SAT}_{\mathbb{T}}$ shown in Figure 7. Its definition is completely parametric with respect to the theory involved. $\mathsf{SAT}_{\mathbb{T}}$ uses iteratively the various rewriting procedures presented in the previous section. Each disjunction generated by the rewriting rules of these procedures is interpreted as a (don't know) non-deterministic choice. Thus, $\mathsf{SAT}_{\mathbb{T}}(C)$ returns a collection $C_1, \ldots, C_k$ of constraints. Each of them is either in solved form or `false`. The two conditions that guarantee that a constraint in pre-solved form is in solved form are tested by function $\mathsf{is\_solved}_{\mathbb{T}}$ shown in Figure 8. By Theorem 3.9 a constraint in solved form is guaranteed to be satisfiable in the corresponding structure.

```
function is_solved_T(C)
    build the directed graph G_{C∈}
    if G_{C∈} has a cycle
        then return false
        else
            compute σ*_C
            if there is a pair t ∈ X, t' ∉ X in C s.t. T ⊨ ∀⃗(σ*_C(t) = σ*_C(t'))
                then return false
                else return C.
```

Figure 8: Final check for solved form constraints

Theorem 6.1 (Termination) *Let $\mathbb{T}$ be one of the theories List, Bag, CList, and Set. Each non-deterministic execution of* $\mathsf{SAT}_{\mathbb{T}}(C)$ *terminates in a finite number of steps. Moreover, the constraint returned is either* `false` *or a solved form constraint.*

Lemma 6.2 *Let $\mathbb{T}$ be one of the theories List, CList, Bag and Set, and C a constraint in pre-solved form over the language of $\mathbb{T}$. If* $\mathsf{is\_solved}_{\mathbb{T}}(\mathsf{C})$ *returns* `false`*, then C is not satisfiable in the structure $\mathcal{A}$ which corresponds to $\mathbb{T}$.*

Theorem 6.3 (Soundness and Completeness) *Let $\mathbb{T}$ be one of the theories List, Bag, CList, and Set, and $C_1, \ldots, C_k$ be the solved form constraints non-deterministically returned by* $\mathsf{SAT}_{\mathbb{T}}(C)$*, and $\bar{N}_i = FV(C_i) \setminus FV(C)$. Then $\mathcal{A} \models \vec{\forall}\left(C \leftrightarrow \bigvee_{i=1}^{k} \exists \bar{N}_i C_i\right)$, where $\mathcal{A}$ is the structure corresponding to the considered theory $\mathbb{T}$.*[2]

Corollary 6.4 *Given a constraint C, it is decidable whether $\mathcal{A} \models \vec{\exists} C$, where $\mathcal{A}$ is one of the structures $\mathcal{LIST}$, $\mathcal{BAG}$, $\mathcal{CLIST}$, $\mathcal{SET}$.*

7 Combining Theories

The four theories presented in the paper can be combined in order to provide more general frameworks where to deal with several of the proposed data structures simultaneously. As a matter of fact, the axioms of the four theories have been defined so as to make this combination a straightforward task. All the data structures are built in the same way (as regulated by axioms (W)), using the same kind of elements. Each axiom involves at most one aggregate constructor symbol, so that the theory for one aggregate is not influenced by the presence of axioms for the other aggregates. The combined theory is therefore obtained by simply taking the union of the sets of axioms of the four individual theories.

As regards the interpretation domain of the privileged structure for the combined theory a simple solution is obtained by: taking the union of the four equational theories considered in the individual cases (axioms (E_p^m) and (E_p^s), as well as axioms (E_a^c) and (E_a^s) must be considered different); taking the union of the set of terms for the individual theories; using the combined equational theory to compute the quotient of the combined Herbrand Universe $T(\mathcal{F}_{List} \cup \mathcal{F}_{Bag} \cup \mathcal{F}_{CList} \cup \mathcal{F}_{Set})$. This

[2]With the small exceptions for sets (see Lemma 5.11) that can however easily be overtaken.

simply causes some equivalence classes that are distinct in the individual cases to be merged in the same class in the combined case. Thus, for instance, terms $\{\!\!\{ a, b \}\!\!\}$ and terms $\{\!\!\{ b, a \}\!\!\}$ which are put into different classes if we consider only the equational axioms for Set, are instead members of the same equivalence class when considering the combined equational theory.

Theorem 3.9 ensures the satisfiability of a solved form constraint for all the theories: an effective way to find a successful valuation is given. It is easy to extend the result to the combined theory. The crucial point is that for variables X occurring only in constraints $X \neq t, t \notin X, t \in X$ the solution is found in $\mathcal{SET}$.

As regards constraint solving, also the various constraint rewriting procedures can be easily combined in order to obtain a general constraint solver for the combined theory. As a matter of fact, all rewriting rules used in these procedures have been obtained in a quite direct way from the relevant axioms and thus they inherit from the latter their parametric definition. Parametricity of the rewriting rules has been made evident throughout the presentation in previous sections. Specific instances of these rules are obtained by simply replacing one aggregate constructor with a different one. The global satisfiability procedure SAT for the combined case is obtained from the generic definition of $\mathsf{SAT}_{\mathbb{T}}$ (see Figure 7) by replacing each call to a generic procedure $p_{\mathbb{T}}(C)$ with the composition of the four specific calls $p_{Set}(p_{Bag}(p_{CList}(p_{List}(C))))$.

Since, for each theory $\mathbb{T}$, all the rewriting procedures do not generate any constraint not belonging to the theory itself, termination of the satisfiability procedure SAT for the combined case is immediately obtained from the termination of the satisfiability procedures for the individual theories. Similarly, soundness and completeness of the global satisfiability procedure is also preserved.

The language obtained by the combination of the four theories allows one to write terms that freely mix various kinds of different data structures. Thus, for instance, we can write a term like $\{a \mid [\![b, c]\!]\}$, which is in part constructed as a set and in part as a compact list.

To avoid the existence of such terms, for which it is hard to find a "natural" interpretation and which are likely to be of little practical utility, an elegant solution is to introduce a notion of sort—hence moving to the context of multi-sorted first-order languages.

Roughly speaking, in this context, one can associate a different sort with every symbol in the language. Thus, for instance, one can introduce the sort Set which is intuitively the sort of all the terms which denote sets. In the term $\{t_1 \mid t_2\}$, t_2 is required to be of sort Set, while t_1 can be of any sort. Thus, the sort of $\{\cdot \mid \cdot\}$ is $\mathsf{any} \times \mathsf{set} \to \mathsf{set}$. Only terms that respect their sorts are allowed to occur in admissible constraints.

This way, different data structures—i.e., data structures of different sorts—can not be mixed within the same term. Also the problem of colored aggregates disappears (provided a constant `nil` with the proper sort is assumed to exist for each distinct data structure, e.g., `nil`, $\{\!\!\{ \}\!\!\}$, $[\![ ]\!]$, $\emptyset$).

A detailed discussion of this topic is outside the scope of this paper. Indeed, the aim of this section is to show that the choices made in the axiomatic definition of the theories for the considered data aggregates, as well as the parametric definition of the relevant constraint rewriting procedures, make their combination into a single general framework immediately feasible, with only a very limited effort.

Conversely, turning this proposal into a concrete CLP programming language that provides all the four data structures altogether requires a few technical matters, such as those concerning the use of sorts, to be further refined.

8 Conclusions

In this paper we have extended the results of [12] studying the constraint solving problem for four different aggregate theories: the theories of lists, multisets, compact lists, and sets. The analyzed constraints are conjunctions of literals based on equality and membership predicate symbols. We have identified the privileged models for these theories by showing that they correspond with the theories on the set of admissible constraints. We have developed a notion of solved form (proved to be satisfiable) and presented the rewriting algorithms which allow to use this notion to decide the satisfiability problems in the four contexts.

In particular, we have shown how constraint solving can be developed parametrically for these theories and we have pointed out the differences and similarities between the four aggregate data structures. Moreover, we have faced complexity problems and we have discussed the issue of combining the independent results obtained.

As further work it could be interesting to analyze parametrically the behavior of the four data structures in presence of append-like operators (*append* for lists, $\cup$ for sets, $\uplus$ for multisets). It has been recently proved that these operators can not be defined without using universal quantifiers (or recursion) with the languages analyzed in this paper [11].

Acknowledgments

The authors wish to thank Ashish Tiwari and Silvia Monica for useful discussions on the topics of this paper. This work is partially supported by MURST project *Certificazione automatica di programmi mediante interpretazione astratta*.

References

[1] D. Aliffi, A. Dovier, and G. Rossi. From Set to Hyperset Unification. *Journal of Functional and Logic Programming*, 1999(10):1–48. The MIT Press, September 1999.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, Cambridge, 1998.

[3] J. Banatre and D. Le Metayer. 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] C. Beeri, S. Naqvi, O. Shmueli, and S. Tsur. Set Constructors in a Logic Database Language. *Journal of Logic Programming 10*, 3 (1991), 181–232.

[6] D. Cantone, E. G. Omodeo, and A. Policriti. The Automation of Syllogistic. II. Optimization and Complexity Issues. *Journal of Automated Reasoning*, 6:173–187, 1990.

[7] C. C. Chang and H. J. Keisler. *Model Theory.* Studies in Logic. North Holland, 1973.

[8] K. L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–321. Plenum Press, 1978.

[9] J. Corbin and M. Bidoit. A rehabilitation of Robinson's unification algorithm. In R.Mason ed., *Information Processing 1983*, Elevisier (North Holland), pp. 909–914.

[10] 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.

[11] A. Dovier, C. Piazza, and A. Policriti. Comparing expressiveness of set constructor symbols. In H. Kirchner and C. Ringeissen, eds., *FROCOS'00*, LNCS No. 1794, pp. 275–289, 2000.

[12] 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.

[13] 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.

[14] A. Dovier and G. Rossi. Embedding Extensional Finite Sets in CLP. In D. Miller, editor, *Proc. of International Logic Programming Symposium, ILPS'93*, pages 540–556. The MIT Press, Cambridge, Mass., October 1993.

[15] H. B. Enderton. *A mathematical introduction to logic*. Academic Press, 1973. 2^{nd} printing.

[16] C. Gervet. Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. *Constraints*, 1:191–246, 1997.

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

[18] P. M. Hill, and J. W. Lloyd. *The Gödel Programming Language*. The MIT Press, Cambridge, Mass., 1994.

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

[20] D. Kapur and P. Narendran. NP-Completeness of the Set Unification and Matching Problems, In J. H. Siekmann ed., 8th CADE, LNCS n. 230, pp. 489–495, 1986.

[21] K. Kunen. *Set Theory. An Introduction to Independence Proofs*. Studies in Logic. North Holland, 1980.

[22] 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.

[23] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.

[24] M. S. Paterson and M. N. Wegman. Linear unification. *Journal of Computer System Science*, 16(2):158–167, 1978.

[25] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*, Second Edition. Prentice Hall, 1996.

[26] J. T. Schwartz, R. B. K. Dewar, E. Dubinsky, and E. Schonberg. *Programming with sets, an introduction to SETL*. Springer-Verlag, Berlin, 1986.

[27] J. H. Siekmann. Unification theory. In C. Kirchner, editor, *Unification*. Academic Press, 1990.

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