Constructive Negation and Constraint Logic Programming with Sets

Agostino Dovier[♯] Enrico Pontelli[♡] Gianfranco Rossi[♣]

Abstract

The aim of this paper is to extend the *Constructive Negation* technique to the case of $CLP(\mathcal{SET})$, a Constraint Logic Programming language based on hereditarily (and hybrid) finite sets. The challenging aspects of the problem originate from the fact that the structure on which $CLP(\mathcal{SET})$ is based is not *admissible closed*, and this does not allow to reuse the results presented in the literature.

We propose a new constraint satisfaction algorithm, capable of dealing with large classes of $CLP(\mathcal{SET})$ programs, and we present a syntactic characterization of such classes. The resulting algorithm provides a novel constraint simplification procedure to handle constructive negation, suitable to those theories where unification admits multiple maximally general unifiers. We also show, using undecidability arguments, that it is impossible to implement a Constructive Negation working on arbitrary $CLP(\mathcal{SET})$ programs with negation—thus proving that classical results regarding Constructive Negation do not scale to the $\mathcal{SET}$ domain.

Keywords: Constraint Logic Programming, Constructive Negation, Programming with Sets.

1 Introduction

The best known form of negation adopted in logic programming is the *Negation as Failure* (NAF) rule [12], i.e., given a logic program P and a ground atom A

$$P \not\vdash A \rightarrow P \vdash \neg A.$$

In practice this is implemented as negation as *finite* failure—i.e., $P \vdash \neg A$ if A belongs to the finite failure set of P. Negation as Failure has the valuable advantage of simplicity, which leads to devise efficient implementations for it. On the contrary, soundness and completeness of the resolution procedure extended with Negation as Failure (i.e., SLDNF-resolution [31]) can be obtained only by applying suitable syntactical restrictions to programs and goals (cf., [1]), which are necessary to avoid application of negation to non-ground subgoals. These restrictions may be acceptable in some specific frameworks

[♯]Università di Verona, Dipartimento Scientifico-Tecnologico, Strada Le Grazie, 3. 37134 Verona (IT). dovier@sci.univr.it

[♡]New Mexico State University, Dept. of Computer Science, Box 30001, Dept. CS, Las Cruces, NM 88003 (USA). epontell@cs.nmsu.edu

[♣]Università di Parma, Dipartimento di Matematica. Via M. D'Azeglio, 85/A. 43100 Parma (IT). gianfr@prmat.math.unipr.it

(e.g., deductive databases), but may lead to severe lack of expressivity in general-purpose programming languages.

A viable alternative to Negation as Failure, that overcomes most of these problems, is *Constructive Negation* (CN) [10, 5, 19, 20, 32]. In Constructive Negation, negative literals in a goal are not restricted to be ground. Moreover, they are not limited to return true/false but they can construct answers, possibly generating constraints on variables occurring in the goal—from here comes the *constructive* nature of this approach.

The term Constructive Negation was first introduced in [10], as a technique for extending the NAF rule in the context of Logic Programming. Constructive Negation was then applied to the more general context of Constraint Logic Programming (CLP) by Stuckey [32], who gave the first completeness result for this method.

The aim of this paper is analyzing the problem of adding Constructive Negation to $CLP(\mathcal{SET})$, an instance of the general CLP scheme incorporating a simple—though powerful—theory of sets.

The long-term goal of this extension is to allow $CLP(\mathcal{SET})$ to express in the language itself *set grouping* mechanisms, needed to support *intensional set formers*, according to the solution described in [17].

Set grouping [6, 29] is the ability to collect in a set S all the values of X satisfying the property $p(X, \bar{Y})$, that is,

$$\forall X(X \in S \leftrightarrow p(X, \bar{Y})).$$

Implementing set grouping by $CLP(\mathcal{SET})$ clauses demands negation to come into play to express the fact that $p(X, \bar{Y})$ is both a *necessary* and *sufficient* condition for X to belong to S. Indeed, the above definition for S can be rewritten as

$$\forall X\,(X \in S \rightarrow p(X, \bar{Y})) \wedge \neg\exists X\,(X \notin S \wedge p(X, \bar{Y}))\,.$$

The first conjunct of this formula can be easily handled in $CLP(\mathcal{SET})$ through the use of recursion (or through the use of *Restricted Universal Quantifiers* [17]), while the second conjunct requires the use of a form of negation.

Negation as Failure requires a number of syntactic restrictions in order to guarantee soundness and completeness of the method. These restrictions translate into severe limitations of the form of admissible intensional set formers—e.g., the grouped set cannot be empty whenever the intensional set term contains free variables in its definition.[1] If our aim is to develop a general framework for computing with sets (as opposed to a language for some specific application areas, e.g., deductive databases, as in the case of $\mathcal{LDL}$), then we should try to relax as much as possible these syntactic restrictions. Moreover, a more "constructive" approach to set grouping (and, hence, to negation) is highly desirable: this would allow the set grouping operation to determine the constraints on the free variables $\bar{Y}$ required to guarantee that the collected set corresponds to the set of values satisfying $p(X, \bar{Y})$.

[1] Similar restrictions have been used in $\mathcal{LDL}$ [6] programs, where set grouping is a built-in feature of the language.

Example 1.1 *Given the following program and goal*

$$\begin{array}{l} \mathtt{p(a,a)}. \\ \mathtt{p(a,f(Z))}. \\ ? - \{\mathtt{X} : \mathtt{p(X,Y)}\} = \emptyset. \end{array}$$

the expected answer is $\mathtt{Y} \neq \mathtt{a}, \forall \mathtt{Z}(\mathtt{Y} \neq \mathtt{f(Z)})$.

Such kind of flexibility in the form of the acceptable intensional set formers can be obtained by using the CN approach—in place of NAF—to implement negation in set grouping clauses.

Since $CLP(\mathcal{SET})$ is an instance of the CLP scheme we could try to use directly the general scheme for Constructive Negation in CLP given by Stuckey in [32]. The applicability of Stuckey's technique, however, depends on the characteristics of the structure $\mathcal{A}$ on which the specific instance of the CLP framework is based, and on the set of admissible constraints in such CLP instance.

Intuitively, *admissible constraints* [32] represent the formulae that the constraint solver can deal with (and, hence, they are all decidable formulae). For a *decidable structure*, we could choose all formulae as admissible constraints, but usually they are chosen to be as close to the primitive constraints as possible, for the sake of efficiency of the constraint solver.

The CN technique also requires the ability to deal with formulae of the form $\neg\exists\bar{X}\psi$, where ψ is a conjunction of admissible constraints. Intuitively, a structure is said to be *admissible closed* [32] when there is a way to rewrite a *negative* formula of this kind to an equivalent *positive* formula that is a (finite) disjunction of conjunctions of admissible constraints, where universal quantifications are applied only to primitive constraint literals. Therefore, admissible closed structures are well suited to be used in Constructive Negation. Indeed, many of the structures used in CLP are admissible closed.

The structure of $CLP(\mathcal{SET})$ is that of *hereditarily finite (and hybrid) sets*, with conjunctions of (positive and negative) literals involving the predicate symbols $=$ and $\in$ as the primitive constraints.

The minimal set of axioms that must hold in every theory of sets includes axiom (N), assuring the existence of the empty set, and axiom (W), describing the operation of adding an element to a set [4]. Every structure we want to use for sets, therefore, must be able to express (N) and (W). The simpler structure for sets is that of *hereditarily finite sets* (cf., e.g., [9]).[2] Vaught, in [33] proves the *essential undecidability* of the theory consisting of axioms (N) and (W). This implies, in particular, that the *theory* of the structure of hereditarily finite sets, being a consistent extension of (NW), is undecidable. Hence, no matter how we select a structure for sets, it will turn out to be undecidable.

Undecidability of structures for sets implies that, no matter how we choose the set of admissible constraints, the structure cannot be admissible closed. Indeed, if the structure would be an admissible closed one, we would be able to reduce a formula of the form $\neg\exists\bar{X}\psi$, ψ being a conjunction of admissible constraints (and, hence, any formula of the

[2] As a matter of fact, every model of (NW) contains at least one submodel which is isomorphic to the structure of hereditarily finite sets.

language), to an equivalent disjunction of conjunctions of admissible constraints, that is to a formula which could be easily analyzed for satisfiability using existing procedures.

The challenging problem is to establish if, and to what extent, the CN approach can be adapted to the $CLP(\mathcal{SET})$ framework, i.e., to an instance of CLP involving an undecidable structure. Because of the peculiarity of the structure at hand we prefer to face the problem directly, trying to extend Chan's constructive negation technique to $CLP(\mathcal{SET})$. The paper focuses mainly on the foundations for such an extension.

The main technical contributions of this paper are (i) the development of a new constraint simplification rule which can cope with the more general framework of a theory T where unification admits multiple maximally general unifiers; (ii) the (negative result about the) unpracticality of the approach of Stuckey [32] to handle CN in $CLP(\mathcal{SET})$, because of the inherent undecidability of the underlying set theory.

Even though the CN technique does not cover all possible cases when used in the context of $CLP(\mathcal{SET})$, it will turn out that it represents a true enhancement with respect to the similar extension based on the NAF rule. Hence, it (partially) fulfills our long-term goal of providing a better support for the implementation of intensional sets within the language itself.

The paper is organized as follows. Section 2 briefly reviews the basic ideas—adapted from [10]—underlying the introduction of Constructive Negation into $CLP(\mathcal{FT})$. Section 3 reviews the $CLP(\mathcal{SET})$ proposal. Section 4 extends the rewriting techniques of CN to non-unitary theories, namely theories in which a unification problem could require more than one unifier to cover all the possible solutions. In Section 5 we prove an undecidability result, which is used to argue about the impossibility of applying CN to arbitrary $CLP(\mathcal{SET})$ programs. Section 6 extends the CN approach to the case of $CLP(\mathcal{SET})$, presenting a new constraint satisfiability procedure. In Section 7 we identify some syntactic conditions which are proved to be sufficient (though not necessary) to produce successful $CLP(\mathcal{SET})$ computations in presence of CN. Finally, Section 8 discusses some more practical issues concerning the implementation of the procedures described in this paper.

2 Constructive Negation in $CLP(\mathcal{FT})$

In this section we briefly review the basic ideas underlying the introduction of Constructive Negation in $CLP(\mathcal{FT})$, the instance of CLP based on the structure of finite trees, as adapted from [10]. Most of these results and techniques will be applied unaltered to $CLP(\mathcal{SET})$. This section also serves as an introduction to the basic definitions and notation that will be used throughout the paper. To this regard, most of the standard CLP notations and results [21, 22, 32] are assumed. In particular, Σ indicates an infinite signature, $\mathcal{A}$ is the Σ-structure which determines the meaning of the predefined predicate and function symbols, and $\mathcal{T}$ is a first-order Σ-theory. $\bar{t}$ is used to denote a (possibly empty) list of terms $t_1, \ldots, t_n$, $[X_1/t_1, \ldots, X_n/t_n]$ (abbreviated $[\,\bar{X}/\bar{t}\,]$) denotes a substitution, and *vars* is a function that returns the set of free variables in a formula or in a Σ-term (by abuse of notation, we shall sometimes use $vars(\varphi_1, \ldots, \varphi_n)$ to denote the set $vars(\varphi_1) \cup \ldots \cup vars(\varphi_n)$). Moreover, when the context is clear, we will use $\bar{X}$

to denote a (possibly empty) set of variables. Thus, for instance, we will write expressions like $\bar{X} \setminus (\bar{Y} \cap \bar{Z})$. Finally, we use the symbol '|' to separate the constraint from the non-constraint part in a goal/clause.

$CLP(\mathcal{FT})$ fixes Σ to contain an infinite collection of function symbols, and the binary predicate symbol $=$. $\mathcal{A}$ is the set of *finite trees*, with the usual interpretation of function symbols (as used in Prolog). The *primitive constraints* are equations between terms. Clark's Equality Theory (CET) [12] is a complete axiomatization of this structure [25].

Definition 2.1 *[10] An* admissible primitive constraint *in $CLP(\mathcal{FT})$ with CN is a constraint of the form:*

- $s = t$, *or*
- $\forall \bar{Z}\,(s \neq t)$,

where s and t are terms, and $\bar{Z}$ is a (possibly empty) set of variables.

The operational semantics of $CLP(\mathcal{FT})$ with CN requires that, whenever a negative subgoal is selected, the corresponding positive version of the subgoal is proved. The negation of the disjunction of all the computed answers is then used to replace the original negative subgoal. Since each answer can be an existentially quantified conjunction of admissible constraints, negating the answers requires the ability to deal with formulae of the form

$$\forall \bar{X} \neg c$$

(or, equivalently, $\neg \exists \bar{X} c$), where c is a conjunction of admissible primitive constraints.

Example 2.2 *Assume p defined by the clause*

$$\mathtt{p(X, Y)} :- \mathtt{X = f(Z), Y = g(Z)}.$$

and consider the goal

$$? - \mathtt{p(V, W)}.$$

This goal produces the answer $\exists Z(V = f(Z) \wedge W = g(Z))$, whose negation is $\forall Z(V \neq f(Z) \vee W \neq g(Z))$.[3]

Fortunately, such kind of formulae can be transformed into equivalent disjunctions of conjunctions of admissible primitive constraints by using the following simplification rule [10]:

Lemma 2.3 *Let t be a term, D be a formula, and let $\bar{X} = vars(t)$ and $\bar{Y} = vars(D) \setminus \bar{X}$. Then*

$$CET \models \forall U(\forall \bar{X} \bar{Y}(U \neq t \vee D) \leftrightarrow (\forall \bar{X}(U \neq t) \vee \exists \bar{X}(U = t \wedge \forall \bar{Y} D))).$$

[3] Note that, like in traditional logic programming, we can view the variables appearing only in goals and clause bodies as existentially quantified.

Proof. The formula $\forall \bar{X}\bar{Y}(U \neq t \vee D)$ can be transformed into the equivalent formula

$$(\forall \bar{X}(U \neq t) \wedge \forall \bar{X}\bar{Y}(U \neq t \vee D)) \vee (\exists \bar{X}(U = t) \wedge \forall \bar{X}\bar{Y}(U = t \wedge \neg D))$$

by adding the tautology $\forall \bar{X}(U \neq t) \vee \exists \bar{X}(U = t)$. The first disjunct can be reduced to $\forall \bar{X}(U \neq t)$. The second disjunct can be rewritten as

$$\exists \bar{X}(U = t \wedge \forall \bar{X}'\bar{Y}(U = t' \wedge \neq D'))$$

where t', D' are variants of t and D, respectively, with $\bar{X}$ replaced by $\bar{X}'$. Since $U = t'$ can be substituted with $t = t'$, and this is in turn equivalent to $\bar{X} = \bar{X}'$, we can obtain $\exists \bar{X}(U = t \wedge \forall \bar{Y}(D))$.

2.3 □

Hence, according to Stuckey's definition [32], the structure of $CLP(\mathcal{FT})$ with CN is admissible closed.

Definition 2.4 *An* admissible constraint *in $CLP(\mathcal{FT})$ with CN is a conjunction of admissible primitive constraints and $\forall$-quantified disjunctions of admissible primitive constraints.*

Example 2.5 *The following formula is an admissible constraint:*

$$\forall X(U \neq f(X) \vee V \neq f(X)).$$

According to Lemma 2.3 it is possible to rewrite it as:

$$\forall X(U \neq f(X)) \vee \exists X(U = f(X) \wedge V \neq f(X)).$$

The constraint satisfiability procedure used in $CLP(\mathcal{FT})$ with CN must be able to check the satisfiability of formulae of the form $\bigvee_{i=1}^{h} c_i$ where each c_i is an admissible constraint. Let us call this procedure $SAT_{\mathcal{FT}}$.

The satisfiability test is performed by rewriting the given formula to an equivalent disjunction of constraints in a simplified canonical form (here called *solved form*). A constraint is solved form can be proved to be always satisfiable with respect to the given structure $\mathcal{A}$ (the Herbrand Universe in this case).

The complete definition of the component of $SAT_{\mathcal{FT}}$ which deals with the simplification of $\neq$-constraints—here called notequal_FT—along with the definition of solved form for $CLP(\mathcal{FT})$ with CN, is shown in Appendix A. $SAT_{\mathcal{FT}}$ incorporates the simplification rule of Lemma 2.3, whose aim is to deal with universally quantified disjunctions of admissible constraints. Such kind of formulae can be generated either:

- during the process of negating the answers of a sub-computation generated by a negative subgoal (as required by the CN approach);
- during execution of rule (3) of the notequal_FT of $SAT_{\mathcal{FT}}$—used to negate an equality of the form $f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)$.

The operational semantics of $CLP(\mathcal{FT})$ with CN can be viewed as an instance of the general schema by Stuckey [32]. As a matter of fact, it is specific to the Herbrand Universe domain, and it always takes as the frontier of the sub-derivation for a negative goal $\leftarrow \neg G$ the finite collection of the solutions for $\leftarrow G$. Hence, it requires that the refutation tree has no infinite branches—though relaxations of this condition are possible [11, 19, 30]. Assume that the negative literal $\neg p(\bar{t})$ is selected in a goal

$$\leftarrow c \,|\, D_1, \ldots, D_{i-1}, \neg p(\bar{t}), D_{i+1}, \ldots, D_n \,.$$

We use the "constraint approach" [32, 30] which assumes that both positive (equational) and negative primitive constraints are used in the sub-derivation for $p(\bar{t})$—i.e., to compute the sub-derivation for $c \,|\, p(\bar{t})$.[4] This technique allows to reduce the number of possible solutions for $\leftarrow p(\bar{t})$ by pruning as soon as possible some branches of its derivation tree.

Let the goal $\leftarrow c \,|\, p(\bar{t})$ have m solutions, where each solution is a disjunction of constraints in solved form. Let $c_1, \ldots, c_k$ be all the solved form constraints occurring in these solutions, and let $\bar{X} = vars(c, \bar{t})$ and $\bar{Y}_i = vars(c_i) \setminus \bar{X}$, for $i = 1, \ldots, k$. The following formula holds [32]

$$comp(P) \models \forall \bar{X} \left(c \wedge p(\bar{t}) \leftrightarrow \bigvee_{i=1}^{k} \exists \bar{Y}_i \, c_i \right)$$

where $comp(P)$ is the *completion* of P [12]. Note that we assume that $comp(P)$ contains also the selected first-order Σ-theory $\mathcal{T}$, namely, *Clark's Equational Theory* for the structure of finite trees, and *Set* for the structure of hereditarily finite sets considered in the successive sections.

Therefore,

$$comp(P) \models \forall \bar{X} \left(\neg(c \wedge p(\bar{t})) \leftrightarrow \bigwedge_{i=1}^{k} \forall \bar{Y}_i \, \neg c_i \right).$$

The conjunction $\bigwedge_{i=1}^{k} \forall \bar{Y}_i \, \neg c_i$ is simplified via a preliminary rewriting that transforms it to an equivalent disjunction of admissible constraints $\bigvee_{i=1}^{h} \exists \bar{Z}_i \, d_i$ which is in a suitable form to be passed to $SAT_{\mathcal{FT}}$ (see Appendix B for a complete description of the preliminary rewriting process for $CLP(\mathcal{FT})$).

If at least one disjunct returned by $SAT_{\mathcal{FT}}$ is different from `false` then $\bigvee_{i=1}^{h} \exists \bar{Z}_i \, d_i$ is satisfiable and the output of $SAT_{\mathcal{FT}}$ represents the computed answer at the current derivation step.

Example 2.6 *Consider the following simple $CLP(\mathcal{FT})$ program and goal:*

```
p(a, a).
p(f(Y), f(Y)).
? − ¬p(X1, X2).
```

[4] Note that the goal $\leftarrow c \,|\, \neg A, \bar{B}$ (i.e., $\neg c \vee A \vee \neg \bar{B}$) is logically equivalent to $\leftarrow c \,|\, \neg(c \,|\, A), \bar{B}$ (i.e., $\neg c \vee (c \wedge A) \vee \neg \bar{B}$).

To compute the answer for the goal $\leftarrow \neg p(X_1, X_2)$ *we first find the answer for* $\leftarrow p(X_1, X_2)$*, obtaining:*

$$(X_1 = a \wedge X_2 = a) \vee$$
$$\exists Y\ (X_1 = f(Y) \wedge X_2 = f(Y)).$$

The negation of the formula is:

$$(X_1 \neq a \vee X_2 \neq a) \wedge$$
$$\forall Y\ (X_1 \neq f(Y) \vee X_2 \neq f(Y)).$$

The preliminary rewriting (see Appendix B) transforms the second conjunct by applying to it the simplification rule of Lemma 2.3:

$$\forall Y\ (X_1 \neq f(Y)) \vee$$
$$(X_1 = f(Y) \wedge X_2 \neq f(Y)).$$

By applying distributivity we get the four disjuncts:

$$(X_1 \neq a \wedge \forall Y\ (X_1 \neq f(Y))) \vee$$
$$(X_1 \neq a \wedge X_1 = f(Y) \wedge X_2 \neq f(Y)) \vee$$
$$(X_2 \neq a \wedge \forall Y\ (X_1 \neq f(Y))) \vee$$
$$(X_2 \neq a \wedge X_1 = f(Y) \wedge X_2 \neq f(Y)).$$

This formula is then passed to $SAT_{\mathcal{FT}}$ *which returns the following disjunction of solved form constraints*

$$(X_1 \neq a \wedge \forall Y\ (X_1 \neq f(Y))) \vee$$
$$\texttt{false} \vee$$
$$(\forall Y\ (X_1 \neq f(Y)) \wedge X_2 \neq a) \vee$$
$$(X_1 = f(Y) \wedge X_2 \neq a \wedge X_2 \neq f(Y))$$

which represents the computed answer for the given goal (the second disjunct can be omitted).

3 An Overview of $CLP(\mathcal{SET})$

$CLP(\mathcal{SET})$ [15, 16, 7] is an instance of the general CLP scheme which supplies (extensional) set terms and basic set-theoretical operations.

In this section, we briefly introduce the basic syntactic entities of the language and we precisely characterize them by specifying the domain $\mathcal{S}$ and its related interpretation function $I^{\mathcal{S}}$.

Sets are represented using $\{\cdot \mid \cdot\}$ as the set constructor, while $\emptyset$ is used to denote the empty set. Therefore, Σ is required to contain at least the two function symbols $\{\cdot \mid \cdot\}$ and $\emptyset$. Intuitively, $\{t \mid s\}$ denotes the set $\{t\} \cup s$. A *set term* is recursively defined as follows:

Definition 3.1 *(set term)*

- $\emptyset$ *is a set term;*
- *a variable* X *is a set term;*

- $\{t \mid s\}$, *where* t *is a term and* s *a set term, is a set term.*

Hereafter, we represent $\{t_0 \mid \{t_1 \mid \cdots \{t_n \mid X\} \cdots\}\}$ as $\{t_0, \ldots, t_n \mid X\}$, and, when X is $\emptyset$, simply as $\{t_0, \ldots, t_n\}$.

Definition 3.2 *The* primitive constraints *in* $CLP(\mathcal{SET})$ *are the (positive and negative) literals based on the predicate symbols* $=$ *and* $\in$*, i.e., literals of the form,* $s = t$*,* $s \neq t$*,* $s \in t$*, and* $s \notin t$*, where* s *and* t *are terms (possibly, set terms). A* constraint *is any conjunction of primitive constraints. Note that, in this case, the admissible constraints are the primitive constraints themselves.*

The definition of the *interpretation domain* $\mathcal{S}$ is such that each set term is mapped to an equivalent set term in a normalized form, where duplicates have been removed and elements are listed in a predefined order. Precisely, let U_H contain the elements of the ordinary Herbrand Universe over the function symbols in $\Sigma = \{\emptyset, \{\cdot \mid \cdot\}, \ldots\}$. Given a total ordering $\preceq$ on U_H, we consider the smallest congruence relation $\equiv$ over U_H that fulfills the *left permutativity property*

$$\{x, y \mid z\} = \{y, x \mid z\}$$

and the *left absorption property*

$$\{x, x \mid z\} = \{x \mid z\}$$

of the constructor $\{\cdot \mid \cdot\}$.

The ordering $\preceq$ can be used to identify a *representative* term from each congruence class in $U_H/_{\equiv}$ (see [17]). For instance, $\{a, b\}$ is the representative term of the class containing $\{b, a\}$, $\{a, a, b\}$, $\{a, b, b\}$, and so on. Then, we define

$$\mathcal{S} = \{t : t \in U_H \text{ and } t \text{ is a representative term}\}.$$

The interpretation function $I^{\mathcal{S}}$ is defined as follows. Any term t is mapped to its representative $t^{\mathcal{S}}$. Given two terms, t_1 and t_2,

- $t_1 = t_2$ if and only if $t_1^{\mathcal{S}}$ and $t_2^{\mathcal{S}}$ are the same element in $\mathcal{S}$;
- $t_1 \in t_2$ if and only if $t_2^{\mathcal{S}}$ is of the form $\{s_1, \ldots, s_n, t_1^{\mathcal{S}} \mid r\}$ for some $n \geq 0$ and some $s_1, \ldots, s_n, r$.

Hence, the constraint domain of $CLP(\mathcal{SET})$—i.e., the structure $\langle \mathcal{S}, I^{\mathcal{S}} \rangle$, along with the class of constraints defined above—is the domain of *hereditarily finite hybrid sets*, that is sets whose elements are uninterpreted Herbrand terms as well as other (finite) hybrid sets.

Such a domain has been also characterized via a simple axiomatic first-order set theory, named *Set* [17, 18]. Basically, *Set* includes, besides the standard equality axioms, the axioms for the existence of the empty set and for the element insertion operation, a suitable version of the classical extensionality axiom, and the *CET* axioms.

It has been proved [16] that the structure $\langle \mathcal{S}, I^{\mathcal{S}} \rangle$ and the theory *Set* correspond on the class of constraints considered above. Furthermore, this constraint domain is *solution compact* [21].

In order to check satisfiability of a given constraint c in $CLP(\mathcal{SET})$, c is transformed to an equivalent disjunction of constraints in solved form, which is proved to be always satisfiable.

Definition 3.3 *A constraint c is in* solved form *if it is either* `false` *or a conjunction of primitive constraints of one of the following forms:*

- $X = t$, *where X does not occur neither in t nor in the rest of c;*
- $X \neq t$, *and X does not occur in t;*
- $t \notin X$, *and X does not occur in t.*

Observe that membership constraints of the form $s \in t$ can be eliminated by replacing them with equality constraints. In particular, the following equivalence can be proved to hold in *Set*

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

The constraint satisfiability test is performed by a procedure ($SAT_{\mathcal{SET}}$) which non-deterministically computes, for a given constraint c, a finite collection of constraints in solved form $\{c'_1, \ldots, c'_n\}$ that is provable to be *equisatisfiable* to c, as stated by the following theorem.

Theorem 3.4 *For any constraint c,*

$$Set \models c \leftrightarrow \bigvee_{c'_i \text{ is returned by } SAT_{\mathcal{SET}}(c)} (\exists vars(c'_i) \setminus vars(c))\ \ c'_i$$

□

The termination of $SAT_{\mathcal{SET}}$ and the finiteness of the number of non-deterministic choices generated during its computation, ensure the finiteness of the number of constraints non-deterministically returned by $SAT_{\mathcal{SET}}$ [15, 16].

The notion of solved form plays a fundamental role in the definition of the constraint satisfiability procedure, as ensues from the following lemma (see [16] for complete proofs).

Lemma 3.5 *Let c be a constraint in solved form, other than* `false`. *Then c is $\mathcal{S}$-solvable and Set-satisfiable.* □

Corollary 3.6 *A constraint c is Set-satisfiable if and only if there exists a non-deterministic choice such that $SAT_{\mathcal{SET}}(c)$ does not return the constraint* `false`. *Furthermore, Set is satisfaction complete.* □

$SAT_{\mathcal{SET}}$ uses four different constraint simplification functions, one for each class of primitive constraints: equal_Set for $=$, notequal_Set for $\neq$, in_Set for $\in$, and notin_Set for $\notin$. In particular, equalities are managed by the set unification algorithm defined in [17]. For any given set of equations $\mathcal{E}$, possibly involving set terms, this algorithm is able to compute through non-determinism each element of a complete set of solutions for $\mathcal{E}$.

Example 3.7 *Given the equation* $\{X \mid R\} = \{Y \mid S\}$, X, Y, R, *and* S *variables, the set unification algorithm returns the four solutions:*

1. $[X/Y, R/S]$,
2. $[X/Y, S/\{X \mid R\}]$,
3. $[X/Y, R/\{Y \mid S\}]$,
4. $[R/\{Y \mid N\}, S/\{X \mid N\}]$, N *a new variable.*

Given a constraint c, $SAT_{\mathcal{SET}}$ repeatedly performs the following rewritings on c

$$\mathsf{notequal_Set(notin_Set(equal_Set(in_Set(c))))}$$

until a fixpoint is reached—i.e., the resulting constraint is in solved form and no further rewriting applies to it.

Example 3.8 *Consider the constraint*

$$X \in \{A, B\} \wedge \{X\} \neq \{A, B\}.$$

$SAT_{\mathcal{SET}}$ *first replaces it with either* $X = A \wedge \{A\} \neq \{A, B\}$ *or* $X = B \wedge \{B\} \neq \{A, B\}$. *The first constraint,* $\{A\} \neq \{A, B\}$, *is further rewritten to the two simpler constraints* $Z \in \{A\} \wedge Z \notin \{A, B\}$ *or* $Z \in \{A, B\} \wedge Z \notin \{A\}$, *where* Z *is a new variable. The former is easily rewritten to* `false`, *whereas the latter is transformed to the constraint in solved form*

$$X = A \wedge A \neq B.$$

Similar rewritings apply to the other constraint, $X = B \wedge \{B\} \neq \{A, B\}$, *getting the second solution for the given initial constraint, namely*

$$X = B \wedge A \neq B.$$

4 CN for Theories with Non-unique Mgu's

As noted before, one of the distinguishing features of $CLP(\mathcal{SET})$ is the use of *set unification*, i.e., unification under permutativity and absorption. Set unification is characterized by the loss of the uniqueness of the most general unifier (mgu) property. Thus, in this context we need to replace the notion of unique *most general unifier* with the notion of *complete set* of *maximally general unifiers.*

Definition 4.1 *Given two terms* s, t *and an equational theory* E, *the* complete set of unifiers *for* t *and* s, $\bigcup_E(s, t)$, *is the collection of all the maximally general* E*-unifiers of* s, t *such that for any* E*-unifier* σ *of* s, t *there exists an element* θ *in* $\bigcup_E(s, t)$ *such that* $\sigma \leq_E \theta$.

On the other hand, the rewriting process used by the Constructive Negation technique—as Chan himself suggests in [10]—is strongly based on the inherent properties of the underlying constraint domain. In particular, Lemma 2.3 is based on the assumption that for every pair of terms s, t the set $\bigcup_E(s,t)$ contains at most one element, i.e., there exists a *unique* mgu.

As a consequence, this lemma does not work correctly when we move from the conventional domain of finite trees—described by the theory CET—to the domain of finite sets—described by the theory Set.

4.1 Failure of Lemma 2.3

To better understand why Lemma 2.3 fails, we exhibit a case in which the $\leftarrow$ part of the implication of Lemma 2.3 does not hold—conversely, the $\rightarrow$ part of this implication is a simple theorem of first-order predicate calculus.

Example 4.2 *Consider the formula*

$$(i) \quad \forall X_1 X_2 \, (U \neq \{X_1, X_2\} \vee X_1 = \emptyset).$$

If Lemma 2.3 would hold, then (i) *could be transformed to the equivalent formula*

$$\begin{array}{ll} (ii) & \forall X_1 X_2 \, (U \neq \{X_1, X_2\}) \vee \\ & \exists X_1 X_2 \, (U = \{X_1, X_2\} \wedge X_1 = \emptyset). \end{array}$$

Taking $U = \{\emptyset, \{\emptyset\}\}$, $X_1 = \{\emptyset\}$, *and* $X_2 = \emptyset$, *then* (i) *is* `false`, *whereas* (ii) *is* `true`.

The reason why Lemma 2.3 does not work correctly in our framework is the use of set unification in place of standard unification.

In fact, let us consider the original formulation of the lemma

$$CET \models \forall U (\forall \bar{X} \bar{Y} \, (U \neq t \vee D) \leftrightarrow (\forall \bar{X} (U \neq t) \vee \exists \bar{X} (U = t \wedge \forall \bar{Y} D))).$$

Considering an arbitrary value assignment θ to U, and assuming that the right-hand side of the implication is satisfied, then two cases are possible.

1. In the first case, the first disjunct is satisfied, which means that $U\theta$ and t do not unify. This clearly implies that also the left-hand side is satisfied.

2. On the other hand, if $U\theta$ and t do unify, then being $U\theta$ ground implies that there is exactly only one possible assignment σ to the variables $\bar{X}$ such that $U\theta = t\sigma$. Thus, for all the assignments different from σ we will have $U\theta \neq t$, while for the remaining case the second disjunct will guarantee that $D\sigma$ is satisfied.

The whole reasoning is based on one basic assumption: for each assignment θ for U, there is *at most* one assignment σ for $\bar{X}$ such that $U\theta = t\sigma$.

If unification between two terms allows for more than one solution, then the validity of the right-hand side of the implication will not necessarily imply the validity of the left-hand side—we are guaranteed that for one of the solutions of the unification D will be true, but we do not have any guarantee regarding the remaining solutions.

In the next subsection we will present a solution to the above problem.

4.2 Handling quantified disjunctions of constraints

In this section, we define a new rewriting rule that can replace the rule of Lemma 2.3 in those cases where unification can result in multiple maximally general unifiers—i.e., $\bigcup_E(s,t)$ has cardinality greater than one.

The availability of this new rule will allow us in Section 6 to generalize the CN technique to the $CLP(Set)$ framework. The intuitive solution is the following:

$$\mathcal{T} \models \forall U(\forall \bar{X}\bar{Y}(U \neq t \vee D) \leftrightarrow (\forall \bar{X}(U \neq t) \vee \forall \bar{X}(U = t \rightarrow \forall \bar{Y} D)))$$

that is, we try to force D to be valid whenever a positive solution to the unification $U = t$ is detected. Unfortunately, this rewriting will not solve in general our problem, since the implication in the second disjunct is implicitly hiding a disjunction—which means that we are failing to achieve the objective of moving the disjunctions outside of the universal quantification.

Let us first introduce some notational conventions that will be useful in the next discussion. t' and D' are used to represent $t[\bar{X}/\bar{X}']$ and $D[\bar{X}/\bar{X}']$, respectively, where $\bar{X}'$ is a fresh new set of variables. Moreover, with a slight abuse of notation, we will use θ_i to denote both the substitution and its equational representation as a system of equations in solved form. Similarly, $\neg\theta_i$ will be used to denote a disjunction of disequations of the form $x_i \neq t_i$, where each x_i is a variable not occurring elsewhere in the disjunction. Finally, if θ is a substitution, we will denote by $\theta|_{\bar{X}}$ the restriction of θ to the set of variables $\bar{X}$.

Lemma 4.3 *Let t be a term and D be a formula, and let*

$$\bar{X} \subseteq vars(t),\ \bar{Q} = vars(t) \setminus \bar{X},\ and\ \bar{Y} = vars(D) \setminus \bar{X}.$$

Moreover, let $\theta_1, \ldots, \theta_k$ be a complete set of maximally general unifiers for $t = t'$, $\bar{Z}_i$ be the new variables (possibly) introduced by θ_i, and let $\bar{X}'_i = \bar{X}' \setminus dom(\theta_i)$. Then,

$$\forall U \bar{Q} \left(\begin{array}{c} \forall \bar{X}\bar{Y}\,(U \neq t \vee D) \\ \leftrightarrow \\ \forall \bar{X}(U \neq t) \ \vee \exists \bar{X} \left(U = t \ \wedge \bigwedge_{i=1}^{k} \forall \bar{X}'_i \bar{Z}_i \,(\neg\theta_i|_{vars(t)} \vee \forall \bar{Y}\, D'\theta_i) \right) \end{array} \right)$$

Proof. Given

$$\forall \bar{X}\bar{Y}\,(U \neq t \vee D)$$

we can add the tautology $\forall \bar{X}\,(U \neq t) \vee \exists \bar{X}\,(U = t)$, and by distributing we obtain

$$\underbrace{(\forall \bar{X}\bar{Y}\,(U \neq t \vee D) \wedge \forall \bar{X}\,(U \neq t))}_{(i)} \quad \wedge \quad \underbrace{(\forall \bar{X}\bar{Y}\,(U \neq t \vee D) \wedge \exists \bar{X}\,(U = t))}_{(ii)}$$

Observe that *(i)* can be written as $\forall \bar{X}\bar{Y}\,((U \neq t \vee D) \wedge U \neq t)$ which is equivalent to $\forall \bar{X}\,(U \neq t)$.

Let us focus on part *(ii)* of the formula. This can be rewritten in the equivalent form

$$\exists \bar{X}\,(U = t) \wedge \neg\exists \bar{X}\bar{Y}\,(U = t \wedge \neg D)$$

which in turn can be expressed, by adding a fresh new set $\bar{X}'$ of variables, as

$$\exists \bar{X}\,(U = t \wedge \neg\exists \bar{X}'\bar{Y}\,(U = t' \wedge \neg D'))$$

Observe that the variables $\bar{Q}$ possibly occurring in t are not affected by the substitution $[\bar{X}/\bar{X}']$. Substituting U with t we get

$$\exists \bar{X}\,(U = t \wedge \neg \exists \bar{X}'\bar{Y}\,(t = t' \wedge \neg D')) \tag{1}$$

From the assumptions of the lemma, $(t = t')$ has a complete set of maximally general unifiers $\theta_1, \ldots, \theta_k$. Each θ_i is of the form

$$A_1 = A_1\theta_i \wedge \ldots \wedge A_{h_i} = A_{h_i}\theta_i \wedge B_1 = B_1\theta_i \wedge \ldots \wedge B_{k_i} = B_{k_i}\theta_i$$

where $\{A_1, \ldots, A_{h_i}\} = dom(\theta_i) \cap vars(t)$ and $\{B_1, \ldots, B_{k_i}\} = dom(\theta_i) \cap \bar{X}'$.

The completeness of the set of unifiers allows us to state

$$(t = t' \leftrightarrow \bigvee_{i=1}^{k} \exists \bar{Z}_i\,\theta_i).$$

Using this equivalence we can rewrite formula (1) as follows:

$$\exists \bar{X}\,(U = t \wedge \neg \exists \bar{X}'\bar{Y}\,(\bigvee_{i=1}^{k} \exists \bar{Z}_i\,\theta_i \ \wedge \ \neg D')). \tag{2}$$

Observe that the new variables $\bar{Z}_i$ are free under the substitutions produced by the unification process. Thus, $\bar{Z}_i\theta_i = \bar{Z}_i$. After distributing the conjunction in (2) we obtain

$$\exists \bar{X}\,(U = t \wedge \neg(\exists \bar{X}'\bar{Y}\,\bar{Z}_1\,(\theta_1 \wedge \neg D') \vee \ldots \vee \exists \bar{X}'\bar{Y}\,\bar{Z}_k\,(\theta_k \wedge \neg D'))). \tag{3}$$

Each component of the disjunction has the form $\exists \bar{X}'\bar{Y}\,\bar{Z}_i\,(\theta_i \wedge \neg D')$, which means

$$\exists \bar{X}'\bar{Y}\,\bar{Z}_i\,(A_1 = A_1\theta_i \wedge \ldots \wedge A_{h_i} = A_{h_i}\theta_i \wedge B_1 = B_1\theta_i \wedge \ldots \wedge B_{k_i} = B_{k_i}\theta_i \wedge \neg D')$$

which can also be written as

$$\exists \bar{X}'\bar{Y}\,\bar{Z}_i\,(\theta_i|_{vars(t)} \wedge \theta_i|_{\bar{X}'} \wedge \neg D').$$

By applying the substitution θ_i we get

$$\exists \bar{X}'\bar{Y}\,\bar{Z}_i\,(\theta_i|_{vars(t)} \wedge \theta_i|_{\bar{X}'} \wedge \neg D'\theta_i).$$

Observe also that the part $\theta_i|_{\bar{X}'}$ is irrelevant. In fact, the variables in $dom(\theta_i) \cap \bar{X}'$ have been replaced and do not occur in $\theta_i|_{vars(t)} \wedge \neg D'\theta_i$. Since, in general $\exists K\,(K = s \wedge \varphi) \leftrightarrow \varphi$ whenever $K \notin vars(\varphi)$, then we can obtain the formula

$$\exists \bar{X}_i'\bar{Y}\,\bar{Z}_i\,(\theta_i|_{vars(t)} \wedge \neg D'\theta_i)$$

where $\bar{X}_i' = \bar{X}' \setminus dom(\theta_i)$. Going back to the formula (3) we have

$$\exists \bar{X}\,(U = t \wedge \neg \exists \bar{X}_1'\bar{Y}\,\bar{Z}_1\,(\theta_1|_{vars(t)} \wedge \neg D'\theta_1) \wedge \ldots \wedge \neg \exists \bar{X}_k'\bar{Y}\,\bar{Z}_k\,(\theta_k|_{vars(t)} \wedge \neg D'\theta_k))$$

which, finally, can be rewritten as

$$\exists \bar{X}\,(U = t \wedge \forall \bar{X}_1'\,\bar{Z}_1\,(\neg\theta_1|_{vars(t)} \vee \forall \bar{Y}\ D'\theta_1) \wedge \ldots \wedge \forall \bar{X}_k'\,\bar{Z}_k\,(\neg\theta_k|_{vars(t)} \vee \forall \bar{Y}\ D'\theta_k))$$

where each $\neg\theta_i|_{vars(t)}$ is a disjunction of disequations $A_j \neq t_j$, with A_j variables in $vars(t)$, and t_j terms obtained as $A_j\theta_i|_{vars(t)}$.

4.3 □

Proposition 4.4 *Let φ be the formula obtained by application of Lemma 4.3. Then there exists i, $1 \leq i \leq k$, such that the i-th conjunct $\forall \bar{X}_i' \bar{Z}_i \, (\neg \theta_i|_{vars(t)} \vee \forall \bar{Y} \, D'\theta_i)$ can be replaced by $\forall \bar{Y} \, D$.*

Proof. The various substitutions θ_i are obtained as solutions of the unification problem $t = t'$. Since one term is a variant of the other, then $\bar{X}'/\bar{X}$ is one of the solutions. Thus, there exists i such that $\theta_i = [X_1'/X_1, X_2'/X_2, \ldots]$. The corresponding element in the conjunction can be rewritten as $\forall \bar{X}' \bar{Y} \, \bar{Z}_i \, D'\theta_i$. Since no additional variables have been introduced in this solution, then $\bar{Z}_i = \emptyset$. Furthermore, $D'\theta_i$ is nothing else than D. Thus, the conjunct is equivalent to $\forall \bar{Y} \, D$.

4.4 □

This last observation is important to create the connection between the new rewriting rule and the original one used in $CLP(\mathcal{FT})$ and presented in Lemma 2.3. In fact, if the term t in the expression does not contain any set term, then the unification will be guaranteed to behave exactly as standard unification, returning only one most general unifier $[\bar{X}'/\bar{X}]$. Thus, Lemma 2.3 is a special instance of Lemma 4.3.

Remark 4.5 *As can be seen in the proof of Lemma 4.3, the universally quantified disjunctions that are left are mainly obtained from the $\theta_i|_{vars(t)}$ part of each unifier of $t = t'$. The objective of the rewriting process is that of removing as quickly as possible the disjunctions occurring inside universal quantifications. For this reason, the rewriting process can be optimized by using a more "directed" version of the unification process, in which bindings for the variables belonging to $\bar{X}'$ are preferred to the bindings for the variables in $vars(t)$.*

Applications of Lemma 4.3 are illustrated in the following examples.

Example 4.6 *Consider the same formula as in Example 4.2*

$$(i) \quad \forall X_1 X_2 \, (U \neq \{X_1, X_2\} \vee X_1 = \emptyset).$$

Since

$$\{X_1, X_2\} = \{X_1', X_2'\}$$

admits the two unifiers $\theta_1 = [X_1'/X_1, X_2'/X_2]$ and $\theta_2 = [X_1'/X_2, X_2'/X_1]$, the given formula is equivalent to:

$$\forall X_1 X_2 \, (U \neq \{X_1, X_2\}) \;\; \vee \;\; \exists X_1 X_2 \, (U = \{X_1, X_2\} \wedge X_1 = \emptyset \wedge X_2 = \emptyset)$$

Example 4.7 *Consider the formula*

$$\forall X_1 X_2 \, (U \neq \{X_1, f(X_2)\} \vee \varphi(X_1, X_2)).$$

By applying Lemma 4.3, we get:

$$\begin{array}{ll} \forall X_1 X_2 & (U \neq \{X_1, f(X_2)\}) \vee \\ \exists X_1 X_2 & (U = \{X_1, f(X_2)\} \wedge \\ & \varphi(X_1, X_2) \wedge \forall X_2' \, (X_1 \neq f(X_2') \vee \varphi(f(X_2), X_2')). \end{array}$$

Applying again Lemma 4.3, the last part of the formula can be rewritten as:

$$\forall X_2' \, (X_1 \neq f(X_2')) \vee$$
$$\exists X_2' \, (X_1 = f(X_2') \wedge \varphi(f(X_2), X_2'))).$$

By applying distributivity, we obtain:

(1) $\forall X_1 X_2 \, (U \neq \{X_1, f(X_2)\}) \vee$
(2) $\exists X_1 X_2 \, (U = \{X_1, f(X_2)\} \wedge \varphi(X_1, X_2) \wedge \forall X_2' \, (X_1 \neq f(X_2'))) \vee$
(3) $\exists X_1 X_2 X_2' \, (U = \{X_1, f(X_2)\} \wedge X_1 = f(X_2') \wedge \varphi(X_1, X_2) \wedge \varphi(f(X_2), X_2'))$.

The last disjunct in this formula can be further simplified using the redundant variable elimination *rule (see Appendix B), producing the following final disjunction:*

(1) $\forall X_1 X_2 \, (U \neq \{X_1, f(X_2)\}) \vee$
(2) $\exists X_1 X_2 \, (U = \{X_1, f(X_2)\} \wedge \varphi(X_1, X_2) \wedge \forall X_2' \, (X_1 \neq f(X_2'))) \vee$
(3) $\exists X_2 X_2' \, (U = \{f(X_2'), f(X_2)\} \wedge \varphi(f(X_2'), X_2) \wedge \varphi(f(X_2), X_2'))$.

5 Handling CN in an Undecidable Theory: The Case of $CLP(\mathcal{SET})$

The aim of the rewriting rule of Lemma 4.3 is to reduce a universally quantified disjunction to a formula which does not contain disjunctions in the scope of any universal quantifiers. The intuition is that the various

$$\forall \bar{X}_i' \bar{Z}_i (\neg \theta_i|_{vars(t)} \vee \forall \bar{Y} D' \theta_i) \tag{4}$$

are somehow "simpler" to rewrite than the original formula. And this has been shown to work properly in the examples presented before.

The availability of such a rewriting rule then would allow us to conclude—according to the definitions in [32]—that the structure of $CLP(\mathcal{SET})$ with CN is *admissible closed*. In turn this allows one to use the approach of [32] to handle CN in $CLP(\mathcal{SET})$.

Unfortunately, the inherent undecidability of the theory *Set*, that will be proved in the next subsection, prevents us from designing such a general rewriting rule. In fact, the ability to rewrite any formula will allow us to devise decision procedures for the undecidable theory.

The following example shows a case in which the rewriting rule of Lemma 4.3 applied to a $CLP(\mathcal{SET})$ formula leads to a non-terminating process.

Example 5.1 *Consider the formula*

$$\forall X_1 X_2 \, (U \neq \{X_1 \mid X_2\} \vee D).$$

The unification $\{X_1 \mid X_2\} = \{X_1' \mid X_2'\}$ *admits four maximally general unifiers (see example 3.7):*

1. $[X_1'/X_1, X_2'/X_2]$,
2. $[X_1'/X_1, X_2'/\{X_1 \mid X_2\}]$,

3. $[X_1'/X_1, X_2/\{X_1' \mid X_2'\}]$,

4. $[X_2/\{X_1' \mid N\}, X_2'/\{X_1 \mid N\}]$.

Thus, the resulting equivalent formula from Lemma 4.3 is

$$\begin{array}{ll} \forall X_1 X_2 & (U \neq \{X_1 \mid X_2\}) \vee \\ \exists X_1 X_2 & (U = \{X_1 \mid X_2\} \wedge D \wedge D[X_2/\{X_1 \mid X_2\}] \wedge \\ \forall X_1' X_2' & (X_2 \neq \{X_1' \mid X_2'\} \vee D[X_2/X_2']) \wedge \\ \forall X_1' X_2' N & (X_2 \neq \{X_1' \mid N\} \vee D[X_1/X_1', X_2/\{X_1 \mid N\}])). \end{array}$$

In this case the rewriting produces two disjunctions whose "complexity" is greater than that of the original formula.

5.1 Undecidability Results

In this section we show that it is not possible to develop a general operational framework for $CLP(\mathcal{SET})$ with negation which is always complete with respect to the selected semantics.

As already noticed in Section 1, each structure for a set theory contains a sub-structure that is isomorphic to the structure of hereditarily finite sets. The structure $\langle \mathcal{S}, I^{\mathcal{S}} \rangle$ of Section 3, where Σ contains $\emptyset$ and $\{\cdot \mid \cdot\}$, is just one of these sub-structures. Therefore, according to Vaught [33], this structure is undecidable.

Let us consider a generic formula Ψ for this structure, written in prenex form,

$$\Psi = (\neg)\exists \bar{X}_1 \, \neg\exists \bar{X}_2 \, \neg\exists \bar{X}_3 \, \cdots \neg\exists \bar{X}_n \, (\psi_1 \vee \cdots \vee \psi_k),$$

where $vars(\psi_1, \ldots, \psi_k) = \bar{X}_1 \cup \cdots \cup \bar{X}_n$. By Vaught's result this formula is undecidable in the general case.

Fig. 1 shows a procedure—named MakeProg—that is able to translate any prenex formula of the form shown above into the corresponding $CLP(\mathcal{SET})$ program with negation P and goal G. By calling MakeProg with initial parameters 0 and the given formula Ψ, MakeProg returns the corresponding initial goal G and the (stored) set of clauses for the program P (using $\leq 2 \cdot n$ new predicate symbols).

For instance, the formula

$$\exists X_1 \, \neg\exists X_2 \, \neg\exists X_3 \, ((X_1 \in X_2 \wedge X_1 = X_3) \vee (X_1 \in X_3))$$

is translated into the following $CLP(\mathcal{SET})$ program with negation

$$\begin{array}{l} \mathtt{p_0} :\!- \mathtt{p_1(X_1)}. \\ \mathtt{p_1(X_1)} :\!- \neg \mathtt{p_2(X_1)}. \\ \mathtt{p_2(X_1)} :\!- \mathtt{p_3(X_1, X_2)}. \\ \mathtt{p_3(X_1, X_2)} :\!- \neg \mathtt{p_4(X_1, X_2)}. \\ \mathtt{p_4(X_1, X_2)} :\!- \mathtt{p_5(X_1, X_2, X_3)}. \\ \mathtt{p_5(X_1, X_2, X_3)} :\!- \mathtt{X_1 \in X_2, X_1 = X_3}. \\ \mathtt{p_5(X_1, X_2, X_3)} :\!- \mathtt{X_1 \in X_3}. \end{array}$$

```
MakeProg(i,Psi):
    let X̄ be vars(Psi) (i.e., free variables in Psi);
    case Psi of
        psi_1 ∨ ··· ∨ psi_k:        %% quantifier free formula
            for j := 1 to k do assert(p_i(X̄) :- psi_j.);
        ¬ Phi:                      %% negated formula
            let Phi' be MakeProg(i+1,Phi);
            assert(p_i(X̄) :- ¬Phi'.);
        ∃Ȳ Phi:                     %% existentially quantified formula
            let Phi' be MakeProg(i+1,Phi);
            assert(p_i(X̄) :- Phi'.)
    end case;
    return(p_i(X̄))
```

Figure 1: Translation from prenex formula to $CLP(\mathcal{SET})$

together with the goal

$$? - \mathtt{p}_0.$$

where $\mathtt{p}_0, \ldots, \mathtt{p}_5$ are new predicate symbols.

The $CLP(\mathcal{SET})$ programs obtained in this way are always recursion-free, and this ensures that their refutation trees are always finite. Hence, $CLP(\mathcal{SET})$ with negation will return a computed answer for G with respect to the program P if and only if the formula is satisfiable in the structure $\langle \mathcal{S}, I^{\mathcal{S}} \rangle$.

Therefore, $CLP(\mathcal{SET})$ would allow us to solve the satisfiability problem with respect to NW for quantified formulae, contradicting [33].

5.2 A class of decidable $CLP(\mathcal{SET})$ formulae

The open question is then to identify which are the classes of formulae that can be safely rewritten using the rule of Lemma 4.3. As we have seen, starting from the formula

$$\forall \bar{X} \, (U \neq t \vee D) \tag{5}$$

we obtain a formula that contains a certain number of occurrences of disjuncts of the form $\forall \bar{X}'_i \bar{Z}_i \, (\neg \theta_i|_{vars(t)} \vee D'\theta_i)$.[5] Each one of such disjuncts is in the proper form for a new application of the rewriting rule. Thus, we can design the rewriting process as an iterative procedure, in which at each step a formula of the form (5) is selected and transformed using the rule of Lemma 4.3. The risk is that of generating a non-terminating process—i.e., a process in which the formulae produced do not approach the desired normalized form (namely, disjunctions of conjunctions of admissible primitive constraints), as in example 5.1.

However, in the context of $CLP(\mathcal{SET})$ with CN, we can state some general sufficient conditions to guarantee the termination of the rewriting process.

[5] Hereafter we will assume, without loss of generality, that the $\forall$-quantified variables $\bar{Y}$ occurring only in the subformula D of formula (9) have been moved into D itself.

The general objective is to iterate the rewriting process with the aim of eliminating set terms. As a matter of fact, once set terms are removed the behavior of the rewriting process is equivalent to that used in $CLP(\mathcal{FT})$ (as illustrated in Proposition 4.4) and, hence, one additional step is sufficient to completely eliminate the disjunction inside the universal quantification.

It can be shown that, if only *completely specified sets* (i.e., those denoted by set terms of the form $\{t_0, \ldots, t_n\}$) are involved, then after a finite number of steps each set term will be completely eliminated from the computation. On the other hand, the presence of tail variables (i.e., of set terms of the form $\{t_0, \ldots, t_n \,|\, X\}$), as in Example 5.1, leads to a potentially unbound growth of the terms generated after each rewriting step.

Theorem 5.2 *The formula $\forall \bar{X}\,(U \neq t \vee D)$, where $vars(t) = \bar{X} \cup \bar{Q}$, can be rewritten using Lemma 4.3 a finite number of times, provided t does not contain any set term of the form $\{t_1, \ldots, t_n \,|\, V\}$, with V variable.*

Proof. If t contains no occurrences of the function symbol $\{\cdot \,|\, \cdot\}$ then the rewriting technique by Chan can be used and the result follows directly.

On the contrary, if t contains some set term, we must analyze the subformulae of the form

$$\forall \bar{X}'_i \bar{Z}_i\,(\neg\theta_i|_{vars(t)} \vee D'\theta_i)$$

returned by a single application of Lemma 4.3. Substitutions θ_i are (maximally general) unifiers of the unification problem $t = t[\,\bar{X}/\bar{X}'\,]$. Since, by hypothesis, there are no set terms with tail variables, it is always the case that no new variables are introduced during the unification process (see [17, 2] for details of the set unification algorithm). Thus, there are no variables $\bar{Z}_i$. In the following, we show that the *"complexity"* of a subformula

$$\forall \bar{X}'_i(A_1 \neq t_1 \vee \ldots \vee A_h \neq t_h \vee D'\theta_i)$$

with A_j variables ranging over $\bar{X} \cup \bar{Q}$, and $\bar{X}'_i = \bar{X}' \setminus dom(\theta_i)$, is strictly smaller than that of the original formula $\forall \bar{X}(U \neq t \vee \ldots)$, for $i = 1, \ldots, h$. Hence, the process terminates after a finite number of steps. Let us consider the different possible cases:

- If t_j contains no occurrences of $\{\cdot \,|\, \cdot\}$, then with a single further application of the rewriting rule the process terminates;
- if there is a variable X' in $dom(\theta_i)$ then $\bar{X}'_i$ has less elements than $\bar{X}$: the number of quantified variables decreases.
- Assume now that no variable X' belongs to $dom(\theta_i)$. Each equation of θ_i can be either of the form $X_j = t_j$ or $V_j = t_j$.
  - If t_j is a variable $X' \in \bar{X}'$, then we can swap the equation and consider the equivalent substitution in which the variable $X' \in dom(\theta_i)$: see the case above.
  - If t_j is not a variable from $\bar{X}'$ and $vars(t_j) \cap \bar{X}' \subset \bar{X}'$ then the quantified variables in $\bar{X}'_i \setminus vars(t_j)$ can be moved to the right. Again, the number of quantified variables decreases.

– assume that t_j is not a variable from $\bar{X}'$, and, moreover, $\bar{X}' \subseteq vars(t_j)$.
The substitution $W_j = t_j$ has been built during the unification procedure as:

$$\begin{array}{rcll} W_j & = & r_1 & A_1 \in vars(r_1) \\ A_1 & = & r_2 & A_2 \in vars(r_2) \\ A_2 & = & r_3 & A_3 \in vars(r_3) \\ & \vdots & & \\ A_{n-1} & = & r_n & \end{array}$$

with $W_j, A_1, \ldots, A_{n-1} \in \bar{X} \cup \bar{Q}$, and

$$t_j = r_1[A_1/r_2][A_2/r_3]\cdots[A_{n-1}/r_n] \; .$$

Each of the r_k's is a proper subterm of t or of $t[\bar{X}/\bar{X}']$. If we consider the substitution θ_i' as the non-idempotent unifier obtained from θ_i as shown above (θ_i' can be built in parallel to θ_i during the unification process), then all the terms on the r.h.s. in the formula:

$$\forall \bar{X}'(W_j \neq r_1 \vee A_1 \neq r_2 \vee \ldots \vee A_{n-1} \neq r_n \vee \ldots)$$

have size strictly smaller than that of t.

5.2 □

6 Constructive Negation in $CLP(\mathcal{SET})$

$CLP(\mathcal{SET})$ does not provide any form of negation in its original definition—apart for the limited negative knowledge expressible through the $\neq$ and $\notin$ constraints. Nevertheless, as already pointed out in Section 1, negation is required (at least) to support the definition of intensional sets in the language itself [17].

In this section we show how CN—suitably extended in the way described in Section 4 and with the limitations pointed out in Section 5—can be added to $CLP(\mathcal{SET})$.

We focus on the operational aspects of $CLP(\mathcal{SET})$ with CN, by showing the (core part of the) constraint satisfiability procedure used in $CLP(\mathcal{SET})$ with CN. On the other hand, since $CLP(\mathcal{SET})$ with CN can be seen as an instance of the general Constructive Negation scheme [32], we can assume that its declarative semantics can be provided using any of the standard approaches to semantics of CLP with negation, such as $\Phi_P \uparrow \omega$ or Kunen's three valued semantics [23].

6.1 Admissible constraints and constraint satisfiability

First, we define the admissible primitive constraints and the solved form of the constraints in $CLP(\mathcal{SET})$ with CN.

Definition 6.1 *An* admissible primitive constraint *in $CLP(\mathcal{SET})$ with CN has the form:*

- $s = t$, *or*

- $\forall \bar{Z}\,(s \neq t)$, *or*
- $s \in t$, *or*
- $\forall \bar{Z}\,(s \notin t)$.

Definition 6.2 *A constraint c is in* solved form *if it is either* `false` *or it is a conjunction of constraints of one of the following forms:*

1. $X = t$, *X does not occur neither in t nor in the rest of c;*
2. $\forall \bar{Z}(X \neq t)$, *$X$ does not occur in t, $\bar{Z} \subseteq vars(t)$, $X \notin \bar{Z}$, and $t \notin \bar{Z}$;*
3. $\forall \bar{Z}(t \notin X)$, *$X$ does not occur in t, $\bar{Z} \subseteq vars(t)$, and $X \notin \bar{Z}$.*

As done for $CLP(\mathcal{SET})$ [16], we can prove that a constraint in solved form other than `false` is always $\mathcal{S}$-solvable and Set-satisfiable (assuming that the signature Σ contains an infinite number of function symbols). The complete proof is presented in Appendix C.

The constraint satisfiability procedure $SAT_{\mathcal{SET}}$ has to be modified in order to manage the new kind of formulae introduced by the CN extension. Moreover, $SAT_{\mathcal{SET}}$ needs now to be able to check, in general, the satisfiability of a disjunction of admissible constraints $C = \bigvee_{i=1}^{h} c_i$. For each c_i, $SAT_{\mathcal{SET}}$ non-deterministically computes a finite collection of constraints in solved form $\{c'_1, \ldots, c'_n\}$, until a disjunction of only solved form constraints is obtained which can be proved to be equisatisfiable to C.

In the next subsection we will give some hints on how to define the functions notequal_Set and notin_Set for the new version of $SAT_{\mathcal{SET}}$, taking into account the new kind of formulae introduced by the CN extension. The functions equal_Set and in_Set, also used within $SAT_{\mathcal{SET}}$ to deal with equality and membership constraints, are not affected by the CN extension.

6.2 notequal_Set

The complete definition of notequal_Set is shown in Fig. 2 (contrast it with the definition of notequal_FT in Appendix A). The most notable actions in notequal_Set are those dealing with set terms, namely rules (7), (8), and (9).

- Rule (7) is split into two separate cases, the second of which takes care of the fact that, in our theory Set, an equation of the form

  $$X = \{t_1, \ldots, t_n \,|\, X\}$$

  admits the solution

  $$[X/\{t_1, \ldots, t_n \,|\, N\}]$$

  where N is a new variable (note that this equation has no solution in CET).

- Rule (8) deals with disequations involving set terms on both sides

$$\forall \bar{Z}\,(\{s_1 \,|\, s\} \neq \{t_1 \,|\, t\}).$$

In the theory *Set*, the above formula is logically equivalent to

$$\begin{array}{ll} \forall \bar{Z}\,(& \exists W(\, W \in \{s_1 \,|\, s\} \wedge W \notin \{t_1 \,|\, t\}\,) \vee \\ & \exists W(\, W \in \{t_1 \,|\, t\} \wedge W \notin \{s_1 \,|\, s\}\,)\,). \end{array} \tag{6}$$

Unfortunately, this gives rise to a new kind of formulae which are likely to be much harder to deal with than the original one. Therefore, we prefer to avoid to deal with the more general case of the input formula and, rather, we single out three possible subcases that, on the contrary, can be managed successfully:

(i) Both set terms have no unbound tail variable—i.e., they are both of the form $\{r_1, \ldots, r_k\}$. By expanding the $\in$-constraints into the corresponding equalities, distributing $\vee$ over $\wedge$, and applying all the resulting substitutions for W, formula (6) is transformed to the equivalent disjunction (cf. rule $(8')$):

$$\begin{array}{ll} \forall \bar{Z}\,(& s_1 \notin \{t_1, \ldots, t_n\} \vee \cdots \vee s_m \notin \{t_1, \ldots, t_n\} \vee \\ & t_1 \notin \{s_1, \ldots, s_m\} \vee \cdots \vee t_n \notin \{s_1, \ldots, s_m\}). \end{array}$$

This formula will be subsequently simplified by rule (7) of notin_Set.

(ii) $\bar{Z} = \emptyset$. Without $\forall$-quantified variables, the disjunction in formula (6) (cf. rule $(8'')$) can be easily handled by subsequent applications of notin_Set and in_Set.

(iii) The two set terms have the *same* tail variable—i.e., the two sets have the form $\{t_1, \ldots, t_n \,|\, W\}$ and $\{s_1, \ldots, s_m \,|\, W\}$. We can proceed like in (i), obtaining in addition two additional disjuncts

$$\begin{array}{l} \exists W\ (W \in X \wedge W \notin \{t_1, \ldots, t_n \,|\, X\}) \\ \exists W\ (W \in X \wedge W \notin \{s_1, \ldots, s_m \,|\, X\}). \end{array}$$

Since these formulae are always false—no W can belong to X and at the same time not belong to a superset of X—they can be simply removed from the resulting formula (cf. rule $(8')$ with Δ variable).

- Finally, rule (9) is needed for dealing with quantified disjunctions of constraints using the new simplification rule defined in Lemma 4.3.

6.3 notin_Set

Fig. 3 shows the complete definition of the function notin_Set for dealing with negative membership constraints. In particular, the last three rules are devoted to manage quantified disjunctions of constraints whose first disjunct involves a $\notin$ operation. We can distinguish three cases, depending on the form of the input formula:

Let C be a disjunction of admissible constraints, c, c', and D be admissible constraints, t, s, t_i, and s_i be terms, f and g be distinct function symbols, and X and Y be distinct variables. Moreover, let W be a new variable, and f be distinct from $\{\cdot \,|\, \cdot\}$.

notequal_Set(C) :

while at least one disjunct c in C is not in solved form

rewrite c according to one of the following rules:

(1)	$\forall X \bar{Z}\,(D) \wedge c'$, and $X \notin vars(D)$	$\mapsto$	$\forall \bar{Z}\,(D) \wedge c'$
(2)	$\forall \bar{Z}\,(f(s_1,\ldots,s_m) \neq g(t_1,\ldots,t_n)) \wedge c'$	$\mapsto$	c'
(3)	$\forall \bar{Z}\,(f(s_0,\ldots,s_n) \neq f(t_0,\ldots,t_n)) \wedge c'$	$\mapsto$	$\forall \bar{Z}\,(t_0 \neq s_0 \vee \cdots \vee t_n \neq s_n) \wedge c'$
(4)	$f \neq f \wedge c'$	$\mapsto$	`false`
(5)	$\forall \bar{Z}\,(t \neq X) \wedge c'$, and t is not a variable	$\mapsto$	$\forall \bar{Z}\,(X \neq t) \wedge c'$
(6)	$\forall \bar{Z}\,(X \neq X) \wedge c'$ or $\forall Y\,(X \neq Y) \wedge c'$ or $\forall X \bar{Z}\,(X \neq t) \wedge c'$, and $X \notin vars(t)$	$\mapsto$	`false`
(7′)	$\forall \bar{Z}\,(X \neq f(t_1,\ldots,t_n)) \wedge c'$, and $X \in vars(t_1,\ldots,t_n)$ or $\forall \bar{Z}\,(X \neq \{t_1,\ldots,t_n \,|\, \Delta\}) \wedge c'$, $\Delta \equiv \emptyset$ or $\Delta \equiv Y$, and $X \in vars(t_1,\ldots,t_n)$	$\mapsto$	c'
(7″)	$\forall \bar{Z}\,(X \neq \{t_1,\ldots,t_n \,|\, X\}) \wedge c'$, and $X \notin vars(t_1,\ldots,t_n)$	$\mapsto$	$\forall \bar{Z}\,(t_1 \notin X \vee \cdots \vee t_n \notin X) \wedge c'$
(8′)	$\forall \bar{Z}\,(\{s_1,\ldots,s_m \,|\, \Delta\} \neq \{t_1,\ldots,t_n \,|\, \Delta\}) \wedge c'$, and $\Delta \equiv \emptyset$ or Δ is a variable	$\mapsto$	$\forall \bar{Z}\,(s_1 \notin \{t_1,\ldots,t_n \,|\, \Delta\} \vee \cdots \vee s_m \notin \{t_1,\ldots,t_n \,|\, \Delta\} \vee t_1 \notin \{s_1,\ldots,s_m \,|\, \Delta\} \vee \cdots \vee t_n \notin \{s_1,\ldots,s_m \,|\, \Delta\}) \wedge c'$
(8″)	$\{s_1 \,|\, s\} \neq \{t_1 \,|\, t\} \wedge c'$	$\mapsto$	$(W \in \{s_1 \,|\, s\} \wedge W \notin \{t_1 \,|\, t\} \wedge c') \vee (W \in \{t_1 \,|\, t\} \wedge W \notin \{s_1 \,|\, s\} \wedge c')$.
(9)	$\forall \bar{Z}\,(s \neq t \vee D) \wedge c'$, $\bar{X} = \bar{Z} \cap vars(s \neq t)$, $\bar{Z}_D = \bar{Z} \setminus \bar{X}$, $\bar{Q} = vars(s \neq t) \setminus \bar{X}$, $\theta_1,\ldots,\theta_k$ a complete set of unifiers for t and t', $\bar{N}_i$ new variables in θ_i, and $\bar{X}'_i = \bar{X}' \setminus dom(\theta_i)$	$\mapsto$	$(\forall \bar{X}\,(s \neq t) \wedge c') \vee \left(s = t \wedge \bigwedge_{i=1}^{k} \forall \bar{X}'_i \bar{N}_i (\neg \theta_i|_{vars(t)} \vee \forall \bar{Z}_D\, D'\theta_i) \wedge c'\right)$

Figure 2: Rewriting function for $\neq$

notin_Set(C) : while at least one disjunct c in C is not in solved form rewrite c according to one of the following rules:			
(1)	$\forall X \bar{Z}\,(D) \wedge c'$, and $X \notin vars(D)$	$\mapsto$	$\forall \bar{Z}\,(D) \wedge c'$
(2)	$\forall \bar{Z}\,(t \notin f(t_1, \ldots, t_n)) \wedge c'$ or $\forall \bar{Z}\,(t \notin X) \wedge c'$, and $X \in vars(t)$	$\mapsto$	c'
(3)	$\forall X \bar{Z}(t \notin X) \wedge c'$, and $X \notin vars(t)$	$\mapsto$	`false`
(4)	$\forall \bar{Z}(t \notin \{s_1 \,|\, s\}) \wedge c'$	$\mapsto$	$\forall \bar{Z}(t \neq s_1) \wedge \forall \bar{Z}(t \notin s) \wedge c'$
(5)	$\forall \bar{Z}(t \notin f(t_1, \ldots, t_n) \vee D) \wedge c'$	$\mapsto$	c'
(6)	$\forall \bar{Z}\,(t \notin X \vee D) \wedge c'$	$\mapsto$	$\forall \bar{Z} N\,(X \neq \{t \,|\, N\} \vee D) \wedge c'$
(7)	$\forall \bar{Z}(t \notin \{s_1 \,|\, s\} \vee D) \wedge c'$	$\mapsto$	$\forall \bar{Z}\,(t \neq s_1 \vee D) \wedge \forall \bar{Z}\,(t \notin s \vee D) \wedge c'$

Figure 3: Rewriting function for $\notin$ (the notational conventions are those of Fig. 2)

- $\forall \bar{Z}(t \notin \{s_1 \,|\, s\} \vee D)$: we can exploit the fact that *Set*, as well as any set theory, guarantees that
$$t \notin \{s_1 \,|\, s\} \leftrightarrow t \neq s_1 \wedge t \notin s.$$
By applying this equivalence to the given formula, and then distributing $\vee$, we get the equivalent formula
$$\forall \bar{Z}((t \neq s_1 \vee D) \wedge (t \notin s_1 \vee D)).$$
In this form, the universal quantification can be distributed over the conjunction (cf. rule (7)).

- $\forall \bar{Z}(t \notin f(t_1, \ldots, t_n) \vee D)$, and $f \not\equiv \{\cdot \,|\, \cdot\}$: in this case the first disjunct is certainly true (disregarding $\bar{Z}$) and the whole formula can be simply removed (cf. rule (5)).

- $\forall \bar{Z}\,(t \notin X \vee D)$, and $X \notin vars(t)$: by exploiting the fact that $t \in X \leftrightarrow \exists N(X = \{t \,|\, N\})$, the given formula can be rewritten to
$$\forall \bar{Z} N\,(X \neq \{t \,|\, N\} \vee D).$$
The handling of this formula calls for the subsequent use of rule (9) of notequal_Set.

Observe that all input formulae that are rewritten to a formula containing a subformula of the form
$$\begin{gathered} \forall \bar{Z}\,(s \neq t \vee D) \\ t \text{ containing a set term of the form } \{t_1, \ldots, t_n \,|\, V\}, V \text{ variable} \end{gathered} \tag{7}$$
may lead, in the general case, to non-termination of the rewriting process. For instance, the problem may arise with the formulae dealt with by rule (6) of notin_Set, which are

always rewritten to $\forall \bar{Z} N\,(X \neq \{t \,|\, N\} \vee D)$, as well as with the formulae dealt with by rule $(7'')$ of notequal_Set, whenever n is greater than 1. This is a direct consequence of the property stated by Theorem 5.2.

Furthermore, input formulae of the form

$$\forall \bar{Z}\,(\{s_1 \,|\, s\} \neq \{t_1 \,|\, t\}).$$

are not dealt with by $SAT_{\mathcal{SET}}$ in the more general case to avoid non-termination.

The possibility of having non-terminating computations finds its justification in the undecidability results of Section 5. Such results indicate that there are cases which cannot be dealt with adequately within the $CLP(\mathcal{SET})$ framework, disregarding which technique is used to implement negation.

In the next section we will identify a class of $CLP(\mathcal{SET})$ programs with CN for which $SAT_{\mathcal{SET}}$ is guaranteed to be able to decide satisfiability of any input constraint.

7 Admissible $CLP(\mathcal{SET})$ Programs with CN

The aim of this section is to provide suitable syntactic characterizations of those $CLP(\mathcal{SET})$ programs for which the constraint satisfiability functions presented in Section 6 are guaranteed to be well-defined and terminating for every input formula.

Actually, we are not interested in characterizing general $CLP(\mathcal{SET})$ programs where negation can occur everywhere in goals and clause bodies, but only those $CLP(\mathcal{SET})$ programs where negation has been used to implement set grouping (see Section 1). These programs can be obtained automaticaly from *positive* $CLP(\mathcal{SET})$ programs endowed with intensional set formers by replacing all the intensional set terms with the $CLP(\mathcal{SET})$ definitions which allow to implement the set grouping facility directly in the language itself [7]. In these programs, negation is allowed to occur only in *set grouping clauses*, that is, in those clauses which are specifically aimed at collecting all the elements of intensionally defined sets.[6]

As noted in Section 6, there are two situations for which the constraint satisfiability algorithm $SAT_{\mathcal{SET}}$ is undefined:

(*i*) $SAT_{\mathcal{SET}}$ has to deal with a formula of the form $\forall \bar{Z}\,(s \neq t \vee D)$, t containing a set term of the form $\{t_1, \ldots, t_n \,|\, V\}$, V variable. In this case the rewriting process can be non-terminating;

(*ii*) $SAT_{\mathcal{SET}}$ has to deal with a formula of the form $\forall \bar{Z}\,(\{s_1 \,|\, s\} \neq \{t_1 \,|\, t\})$, $\bar{Z}$ not empty, s and/or t variables, but not the same variable: no rule of $SAT_{\mathcal{SET}}$ applies in this case to the formula.

[6] The problem of dealing with intensional sets through translation to set grouping clauses with negation has been considered also in [17], in the context of the language {log}, an extended Horn clause logic language endowed with (extensional) sets. Actually, $CLP(\mathcal{SET})$ can be viewed just as the CLP counterpart of {log}, and as such the former inherits most of the features of the latter, including the way intensional sets—as well as restricted universal quantifiers—are dealt with. An interpreter of the {log} language has been implemented on the top of SICStus Prolog and is available on the WEB at http://prmat.math.unipr.it/gianfr/setlog.Home.html/interpreter.

Definition 7.1 *Any of the situations* (i) *and* (ii) *listed above is called an* undefined situation *for* $SAT_{\mathcal{SET}}$.

It turns out that, if we could exclude from the computation of $SAT_{\mathcal{SET}}$ the presence of terms of the form $\{t_1, \ldots, t_n \,|\, X\}$, X variable, then we could avoid all the undefined situations of $SAT_{\mathcal{SET}}$. This is formalized by the following notion of bounded constraint (which extends the concept of *bounded set* used by various authors [6, 3]) and by the subsequent lemma.

Definition 7.2 *A constraint* c *is said* bounded *if it does not contain neither terms* $\{t_0, \ldots, t_n \,|\, X\}$ *nor constraints of the form* $t \in X$ *or* $\forall \bar{Z}\,(t \notin X \vee D)$. *Otherwise, the constraint is said to be unbounded.*

Note that we exclude also constraints of the form $t \in X$ and $\forall \bar{Z}\,(t \notin X \vee D)$ because the rewriting of these constraints generates constraints of the form $\{t_1, \ldots, t_n \,|\, X\}$, X variable.[7]

Lemma 7.3 *If a constraint* c *is bounded, then the computation of* $SAT_{\mathcal{SET}}(c)$ *does not generate any undefined situations. Moreover,* $SAT_{\mathcal{SET}}(c)$ *returns only bounded constraints.*

Proof. By case analysis.

- If a constraint $\in$ is bounded, then it is replaced by a disjunction of bounded equations (function in_Set);
- if a conjunction of equations is bounded, then it is replaced by a conjunction of equations in solved form that does not contain any term of the form $\{t_1, \ldots, t_n \,|\, X\}$, X variable (function equal_Set);
- all undefined situations for the formulae dealt with by function notequal_SET are generated only by unbounded disequations; for all other cases, notequal_SET generates only bounded constraints;
- similarly, all undefined situations for notin_SET are generated only by unbounded constraints (specifically, by rule (6) and by rule (7) whenever the involved set term has a variable tail part).

7.3 □

In order to guarantee that only bounded constraints are processed by $SAT_{\mathcal{SET}}$ we introduce a special form of allowedness, called *set-allowedness*, for $CLP(\mathcal{SET})$ programs.

Let us first adapt the usual notion of *allowedness* to $CLP(\mathcal{SET})$ programs, which, as shown in Section 3, may contain positive ($=$ and $\in$) as well as negative ($\neq$ and $\notin$) constraints besides user defined atoms. Remember that a clause $H \leftarrow B_1, \ldots, B_n$ is said to be *allowed* in the standard case [1] if every variable X appearing in a clause appears in at least one positive literal in the body of the clause.

[7] In particular, $t \in X$ is always rewritten to $X = \{t \,|\, N\}$, N new variable.

Definition 7.4 *(program normalization) Given a program P, we build the program* $\mathtt{norm}(P)$ *as follows: for every clause*

$$A \leftarrow c \,|\, \bar{B}$$

in P, add the clauses

$$(A \leftarrow c_1 \,|\, \bar{B})\theta_1, \ldots, (A \leftarrow c_k \,|\, \bar{B})\theta_k$$

to $\mathtt{norm}(P)$*, where*

$$\hat{\theta}_1 \wedge c_1, \ldots, \hat{\theta}_k \wedge c_k$$

are all the (possibly none) solved form constraints returned by $SAT_{\mathcal{SET}}(c)$, with $\hat{\theta}_i$ representing the equational part and c_i representing the negative constraint part of the i-th constraint.

Definition 7.5 *(allowedness for $CLP(\mathcal{SET})$ programs) A $CLP(\mathcal{SET})$ program with negation P is* allowed *if* $\mathtt{norm}(P)$ *is allowed in the standard sense, viewing negative constraints $(\notin, \neq)$ as negative literals. Similarly, a goal $\leftarrow c \,|\, \bar{B}$ is allowed if* $\mathtt{norm}(p(X_1, \ldots, X_n) \leftarrow c \,|\, \bar{B})$ *is allowed in the standard sense, where $\{X_1, \ldots, X_n\} = vars(c \,|\, \bar{B})$, and p is a new predicate symbol.*

Hence, given a constraint c, a goal $\leftarrow c$ is allowed if and only if the solved form constraints returned by $SAT_{\mathcal{SET}}(c)$ are all of the form $X_1 = t_1 \wedge \ldots \wedge X_n = t_n$, where $\{X_1, \ldots, X_n\} = vars(c)$ and $t_1, \ldots, t_n$ are ground terms.

Example 7.6 *Consider the $CLP(\mathcal{SET})$ clause*

$$\mathtt{p(X, Y)} :\!- \mathtt{\{X, Y\} = \{V \,|\, Z\},\ V \notin Z,\ q(V), \neg r(Z)}\,.$$

$SAT_{\mathcal{SET}}(\{X, Y\} = \{V \,|\, Z\} \wedge V \notin Z)$ returns the three solved form constraints:

- $V = X \wedge Z = \{Y\} \wedge X \neq Y$,
- $V = Y \wedge Z = \{X\} \wedge X \neq Y$,
- $Y = X \wedge V = X \wedge Z = \emptyset$.

Hence, the initial clause can be rewritten to the equivalent program:

$$\begin{array}{l} \mathtt{p(X, Y)} :\!- \mathtt{X \neq Y,\ q(X), \neg r(\{Y\})}. \\ \mathtt{p(X, Y)} :\!- \mathtt{X \neq Y,\ q(Y), \neg r(\{X\})}. \\ \mathtt{p(X, X)} :\!- \mathtt{q(X), \neg r(\emptyset)}. \end{array}$$

The first and second clauses are not allowed in the standard sense—e.g., in the first clause the variable `Y` *appears only in negative constraints. Therefore the original clause is not allowed.*

By considering only allowed $CLP(\mathcal{SET})$ programs and goals, and assuming that a selection rule which delays negative subgoals until they are ground is adopted, it would be easy to prove that $SAT_{\mathcal{SET}}$ is always well-defined and terminating. In fact, even if we cannot exclude that some unbounded constraints occur during the resolution process,

allowedness ensures that every variable generating such situations will be eventually instantiated to a ground term. This ensures that such undesired constraints eventually disappear: the proof follows from Lemma 7.3.

It is evident that this class of admissible programs is too narrow for our purposes. As a matter of fact, for this class of programs one could use NAF as well. However, since we are interested in the grounding of only those variables that occur as the tail variables of a set term, we can take advantage of this fact and define a finer notion of allowedness for $CLP(\mathcal{SET})$ programs. Moreover, we can further refine this notion by restricting our attention to $CLP(\mathcal{SET})$ programs where negation is allowed only in set grouping clauses. Both refinements will allow us to significantly enlarge the class of admissible programs.

Hereafter, we assume that all the programs and goals used have been previously normalized using the procedure described in Definition 7.4.

Definition 7.7 *(r-allowedness)*

- *A goal $\bar{B}$ is* r-allowed *("r" stands for restricted) if, for every variable X that occurs in a term $\{t_1, \ldots, t_n \mid X\}$, X occurs also in a positive literal $p(s_1, \ldots, s_m)$, and all the clauses used to define p are allowed;*
- *a clause $H \leftarrow \bar{B}$ is* r-allowed *if $\neg H, \bar{B}$ is* r-allowed*;*
- *a program is* r-allowed *if all its clauses are* r-allowed.

Note that allowedness implies r-allowedness, whereas the vice versa is not true in general.

Lemma 7.8 *Let P be an r-allowed program and G an r-allowed goal. Then, any computed answer for G w.r.t. P does not contain any term of the form $\{t_1, \ldots, t_n \mid X\}$, X variable.*

Proof. The proof follows immediately from the definition of allowedness for $CLP(\mathcal{SET})$ programs and from Lemma 7.3.

7.8 □

Let us now focus on $CLP(\mathcal{SET})$ programs which are obtained by eliminating intensional sets. The following example (informally) introduces the translation technique used to eliminate intensional sets, by showing the generated collection of set grouping clauses.

Example 7.9 *Consider the following clause defining the predicate* possibly_lovers*, which is true if S is the set of people who like person Y, save Y herself.*

$$\begin{array}{l}\texttt{possibly_lovers}(\texttt{S}, \texttt{Y}) :- \\ \quad \texttt{S} = \{\texttt{X} : \texttt{X} \neq \texttt{Y}\,,\ \texttt{likes}(\texttt{X}, \texttt{Y})\}.\end{array}$$

This clause is transformed into the following $CLP(\mathcal{SET})$ program with negation:

$$\begin{array}{l}\texttt{setof}_\delta(\texttt{S}, \texttt{Y}) :- \\ \quad \texttt{subsetof}_\delta(\texttt{S}, \texttt{Y}), \\ \quad \neg\texttt{partial}_\delta(\texttt{S}, \texttt{Y}). \\ \delta(\texttt{X}, \texttt{Y}) :- \\ \quad \texttt{X} \neq \texttt{Y}\,,\ \texttt{likes}(\texttt{X}, \texttt{Y}).\end{array}$$

$$\begin{array}{l}\texttt{subsetof}_\delta(\emptyset, \texttt{Y}). \\ \texttt{subsetof}_\delta(\{\texttt{X} \mid \texttt{S}\}, \texttt{Y}) :- \\ \quad \texttt{X} \notin \texttt{S}\,, \\ \quad \delta(\texttt{X}, \texttt{Y}), \texttt{subsetof}_\delta(\texttt{S}, \texttt{Y}). \\ \texttt{partial}_\delta(\texttt{S}, \texttt{Y}) :- \\ \quad \texttt{Z} \notin \texttt{S}\,,\ \delta(\texttt{Z}, \texttt{Y}).\end{array}$$

Any translation of intensional set formers can be performed in this way. Without loss of generality, we can also assume that calls to predicates $\texttt{setof}_\delta(\texttt{S}, \bar{\texttt{Y}})$ in the translated program always occur with S and $\bar{\texttt{Y}}$ variables; this hypothesis will be useful in the next Theorem 7.11.

Note that the second clause of $\texttt{subsetof}_\delta$ is not r-allowed because of the variable S and because $\texttt{subsetof}_\delta$ is not allowed (at least, whenever there are 'free' variables in the intensional set definition).

Definition 7.10 *(set-allowedness) Let P be a $CLP(\mathcal{SET})$ program with negation obtained by eliminating intensional set terms.*

- *A clause of P defining a predicate* $\texttt{setof}_\delta$ *is* set-allowed *if all the clauses used to define δ (at all levels) are set-allowed;*
- *a clause of P defining a predicate p different from* $\texttt{setof}_\delta$ *is* set-allowed *if it is r-allowed;*
- *P is* set-allowed *if, for all δ, all* $\texttt{setof}_\delta$ *clauses are set-allowed.*

Theorem 7.11 *Let P be a set-allowed program and G a goal. Then, $SAT_{\mathcal{SET}}$ is well-defined and terminating for all possible derivations of G in P, provided a selection rules that select first* $\texttt{subsetof}_\delta$ *atoms is adopted.*

Proof. We can start observing that, even if the $\texttt{setof}_\delta$ clauses are only set-allowed (instead of r-allowed), they cannot generate any $\{t_1, \ldots, t_n \,|\, X\}$ term. In fact, what prevents these clauses to be r-allowed is only the variable S in the second clause of $\texttt{subsetof}_\delta$. By the definition of $\texttt{subsetof}_\delta$ itself, it is evident that S will be always instantiated to a set term with no rest variable.

According to Lemma 7.8, since all the clauses for all the δ predicates are set-allowed by hypothesis, they cannot generate any $\{t_1, \ldots, t_n \,|\, X\}$.

On the other hand, the hypothesis that in any call to $\texttt{setof}_\delta(\texttt{S}, \bar{\texttt{Y}})$ S and $\bar{\texttt{Y}}$ are variables, along with the additional hypothesis on the selection rule, ensure that no term $\{t_1, \ldots, t_n \,|\, X\}$ can be passed to $\texttt{setof}_\delta$ predicates (and, hence, to the $\texttt{partial}_\delta$ predicates) from outside, i.e., as a parameter of the atom $\texttt{setof}_\delta(\texttt{S}, \bar{\texttt{Y}})$.

Hence, the thesis follows immediately from Lemma 7.3.

7.11 □

Example 7.12 *Let us assume that the program of Example 7.9 is augmented with the following clauses:*

$$\texttt{likes}(\texttt{peter}, \texttt{ann}).$$
$$\texttt{likes}(\texttt{john}, \texttt{Any}).$$

The corresponding $CLP(\mathcal{SET})$ program is set-allowed (note that the clauses defining likes *instead are not allowed). The computed answers for the goal*

$$:\!- \texttt{possibly_lovers}(\texttt{S}, \texttt{Y}).$$

are:

$$\texttt{S} = \{\texttt{peter}, \texttt{john}\}, \texttt{Y} = \texttt{ann}$$
$$\texttt{S} = \{\texttt{john}\}, \texttt{Y} \neq \texttt{ann}.$$

The second answer for the goal is computed by solving the (negative) subgoal

$$\neg\ \texttt{partial_likes(\{john\},Any)}.$$

Since the computed answer for the positive version is Any = ann we get, by Constructive Negation, the answer Any ≠ ann for the negated goal.

The class of set-allowed programs *strictly* includes the class of allowed programs. As special cases, all programs without intensional sets and all programs with intensional sets, but with no occurrences of terms of the form $\{t_1, \ldots, t_n \mid X\}$, are set-allowed.

The class of set-allowed programs extends also the class of $\mathcal{LDL}$ acceptable programs [6]. As a matter of fact, the ability to 'collect' empty sets is explicitly forbidden in [6], whereas this does not raise any problem in $CLP(\mathcal{SET})$ with CN, provided set-allowedness is guaranteed. Thus, for instance, the computed answer for the program of Example 1.1 is exactly the expected one.

8 Implementation Issues

Constructive Negation can be quite difficult to implement and very expensive to execute. Conversely, the NAF rule is widely used in practical implementations of Prolog for its simplicity and efficiency.

Therefore, our implementation of $CLP(\mathcal{SET})$ with intensional set formers tries to use both techniques to implement negation, preferring NAF whenever possible.[8] To this purpose, we have precisely identified syntactic conditions for programs that assure that NAF can be used safely. When translating intensional set terms into the $CLP(\mathcal{SET})$ clauses that perform set grouping the translator can select which form of negation must be used in any particular case.

Observe that thanks to our restricted use of negation (in set grouping clauses only) the class of programs for which the use of NAF is admissible turns out to be larger than that of the allowed programs. For example, the program

$$\texttt{p(a,\{a\})}.$$
$$\texttt{p(b,\{b|X\})}.$$

with the goal $:-\ \texttt{S} = \{\texttt{X} : \texttt{p(X,Y)}\}$, which is translated into a $CLP(\mathcal{SET})$ program which is not set-allowed—and, of course, also non-allowed—can instead be dealt with safely using NAF (except for the case of the empty set), provided $\texttt{subsetof}_\delta$ within set grouping clauses is executed before $\neg\texttt{partial}_\delta$. Also note that whenever an intensional set term has no free variables in its goal part, then the NAF rule can be always safely used.

Surely, better results (in the sense of a finer distinction between the different forms of negation required) could be obtained by integrating the translation mechanisms with suitable program analysis tools. These could be applied to those parts of the program

[8] A similar approach is adopted in [27] where NAF, CN, and also Intensional Negation, are integrated into a single Prolog compiler.

involving set grouping, to determine, through the use of abstract interpretation techniques citebruy91, properties like groundness [13, 28], non-failure [14], finiteness and so on.

Only when NAF cannot be applied, then CN is used in its place. For instance, this happens very often when trying to find out for which values of its free variables an intensional set term denotes the empty set (cf., e.g., Example 1.1). In fact, whenever the execution of $\texttt{subsetof}_\delta(\texttt{S},\texttt{Y})$ instantiates `S` to the empty set, `Y` is left uninstantiated; in turn, this may cause a non-safe use of the NAF rule for $\neg\texttt{partial}_\delta(\texttt{S},\texttt{Y})$.

If CN has to be used and the program is set-allowed then set grouping can be safely performed. However, even if the program is not set-allowed there are special cases for which the presence of terms of the form $\{t_1,\ldots,t_n \mid X\}$ do not necessarily imply an indefinite situation for $SAT_{\mathcal{SET}}$. To this purpose, we could easily add simple ad-hoc rules to the constraint satisfiability functions presented in Section 6 to guarantee termination of the rewriting process also for these special cases. Specifically,

- if the input formula c has the form

  $$\forall \bar{Z}\,(X \neq \{t_1,\ldots,t_n \mid V\} \vee D) \wedge c',\ X \not\equiv N,\ \text{and}\ X \in vars(t_1,\ldots,t_n)\,:$$

  the first disjunct of the $\forall$-quantified formula is certainly true, and the whole formula can be rewritten to c';

- if the input formula c has the form

  $$\forall \bar{Z}\,(s \neq t \vee D) \wedge c',\ \text{and}\ vars(s \neq t) \cap vars(D) = \emptyset\ \text{or}\ \bar{Z} = \emptyset\ :$$

  the disjunction can be directly split into its two disjuncts.

Furthermore, better termination behavior can be obtained by adding run-time checks in order to *delay* the rewriting of the formulae containing set terms of the form $\{t_1,\ldots,t_n \mid X\}$, until either the variables in the formulae have reached an adequate level of instantiation or the formulae have been transformed in such a way to allow the application of one of the terminating cases.

All the mentioned implementation techniques have the common effect of enlarging the class of acceptable programs and, possibly, of gaining some efficiency when performing set grouping.

As concerns this last point, one can further observe that both set grouping and CN require the ability to collect all the solutions to a given goal (namely, the goal that occurs in the related intensional set term). Therefore, better performance results for set grouping could be obtained by combining the two all-solutions collection operations.

The idea goes as follows. For every intensional set $S \equiv \{X : p(X,t_1,\ldots,t_n)\}$ that is likely to require CN, first collect the set AS of all the $(n+1)$-tuples that represent the computed answers for the predicate $p(X,t_1,\ldots,t_n)$. This can be done by using the built-in predicate setof/3 suitably adapted to work with CLP(SET) programs.[9] Given

[9] Note that the list `L` of computed answers for $\texttt{p}(\texttt{X},\texttt{Y})$ collected by $\texttt{setof}(\texttt{X},\texttt{p}(\texttt{X},\texttt{Y}),\texttt{L})$ is just an "approximation" of the set $\{X : p(X,Y)\}$. For instance, if `p` is defined by the two clauses `p(a).` and `p(f(X)).` the result of `setof` is `L = [a,f(X)]` whereas the intensional set denotes the infinite set $\{a, f(X_1), f(X_2),\ldots\}$ with the constraint $X_1 \neq X_2,\ldots$. Moreover `setof` is unable to deal with the case of empty set.

the set AS, we can use it both to compute all the solutions for the variables that possibly occur in the goal $partial_p(S, \bar{Y})$, which are subsequently negated according to the CN technique, and to find out all the sets SS_i of $(n+1)$-tuples as computed by the goal $subsetof_p(S, \bar{Y})$. More in detail, for each SS_i the set of solutions for $partial_p(SS_i, \bar{Y})$ (see the definition of $partial_\delta$ in Example 7.9) is computed by removing from AS all the tuples $[X, t_1, \ldots, t_n]$ for which $[X, Z_1, \ldots, Z_n]$, $Z_1, \ldots, Z_n$ new variables distinct from X, occur in SS_i, and using the resulting subset AS' to compute all the instances of $t_1, \ldots, t_n$ for which $[Z, t_1, \ldots, t_n]$, Z new variable, belongs to AS'.

9 Conclusions and Future Work

In this paper we have tackled the problem of dealing with Constructive Negation in the framework of $CLP(\mathcal{SET})$. Constructive Negation in $CLP(\mathcal{SET})$ is required in order to efficiently support more expressive programming constructs, such as intensional set formers.

The challenging aspect of the problem originates from the fact that the structure on which $CLP(\mathcal{SET})$ is based is not *admissible closed*, and this does not allow to reuse the existing results present in the literature [32].

The basic undecidable nature of the axiomatic structure of $CLP(\mathcal{SET})$ has been used to show that a universal scheme supporting CN is infeasible. Nevertheless, we are proposing a general method for managing constraints, suitable for theories where unification admits multiple maximally general unifiers, which allows to obtain correct results for large classes of programs. Some of these classes have been precisely characterized through syntactic conditions—showing in particular how, even in this complex framework, CN strictly subsumes NAF.

Few aspects of this work are still open. The idea of using CN in $CLP(\mathcal{SET})$ originates by the practical need of supporting intensional set definitions. Work is currently in progress (and some preliminary results have been already presented [7]) to estimate the impact of the restrictions necessary to guarantee effectiveness of CN on the intensional set definitions.

Acknowledgments

The research presented in this paper has benefited from discussions with P. Bruscoli, G. Gupta, E. G. Omodeo, A. Policriti, and D. Ranjan, all of whom we would like to thank.

Partial support came from C.N.R. Grant 97.02426.CT12. E. Pontelli is partially supported by NSF Grant CCR 96-25358, HRD 96-28450, and CDA 97-29848.

References

[1] Apt, K. R., and Bol, R. Logic Programming and Negation: a Survey. *Journal of Logic Programming 19/20* (1994), 9–71.

[2] Arenas-Sánchez, P., and Dovier, A. A minimality study for set unification. *Journal of Functional and Logic Programming 1997*, 7 December 1997.

[3] Arni, N., Greco, S., and Saccà, D. Matching of Bounded Set Terms in the Logic Language LDL++. In *Journal of Logic Programming*, (1996), Vol. 4.

[4] Bernays, P. A system of axiomatic set theory. Part I. *The Journal of symbolic logic*, Vol. 2, (1937), 65–77.

[5] Barbuti, R., Mancarella, P., Pedreschi, D., and Turini, F. A transformation approach to negation in logic programming. *Journal of Logic Programming 8* (1990), 201–228.

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

[7] Bruscoli, P., Dovier, A., Pontelli, E., and Rossi., G. Compiling Intensional Sets in CLP. In *Proc. Eleventh Int'l Conf. on Logic Programming* (1994), P. Van Hentenryck, Ed., The MIT Press, Cambridge, Mass., 647–661.

[8] Bruynooghe, M. A Framework for the Abstract Interpretation of Logic Programs. In *Journal of Logic Programming*,Vol. 10, 1991.

[9] Cantone, D., Ferro, A., and Omodeo., E. G. *Computable Set Theory, Vol. 1.* International Series of Monographs on Computer Science. Clarendon Press, Oxford, 1989.

[10] Chan, D. Constructive Negation Based on the Completed Database. In *Proc. Fifth International Conference and Symposium on Logic Programming* (1988), R. Kowalski and K. Bowen, Eds., The MIT Press, Cambridge, Mass., 111–125.

[11] Chan, D. An Extension of Constructive Negation and its Application in Coroutining. In *Proc. North-American Conference on Logic Programming* (1989), E. Lusk and R. Overbeek, Eds., The MIT Press, Cambridge, Mass., 477–493.

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

[13] Cortesi, A., and Filè, G. Abstract Interpretation of Logic Programs: an Abstract Domain for Groundness, Sharing, Freeness and Compoundness Analysis. In *Proc. of SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation*, ACM Press,1991.

[14] Debray, S., and Lopez-Garcia, P., and Hermenegildo, M. Non-failure Analysis for Logic Programs. In *Proc. of the International Conference on Logic Programming*, MIT Press, 1997.

[15] Dovier, A., and Rossi, G. Embedding Extensional Finite Sets in CLP. In *Proc. of Int'l Logic Programming Symposium*, (1993), D. Miller, Ed., The MIT Press, Cambridge, Mass., 540–556.

[16] Dovier, A. A Language with Finite Sets Embedded in the CLP Scheme. In *Selected papers from the* 4^{th} *Int'l Workshop on Extension of Logic Programming* (1994), R. Dyckhoff, Ed., vol. 798 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin, 77–93.

[17] Dovier, A., Omodeo, E. G., Pontelli, E., and Rossi, G. {log}: A Language for Programming in Logic with Finite Sets. *Journal of Logic Programming 28*, 1 (1996), 1–44.

[18] Dovier, A., Policriti, A., and Rossi, G. Integrating lists, multisets, and sets in a logic programming framework. In 1^{st} *Int'l Workshop on Frontiers of Combining Systems* (1996), F. Baader and K. U. Schulz, Eds., vol. 3 of *Applied Logic*, Kluwer Academic Publishers, 213–229.

[19] Drabent, W. What is Failure? An Approach to Constructive Negation. Tech. rep., Linkoping University, October 1993.

[20] Fages, F. Constructive negation by pruning and optimization higher-order predicates for clp and cc languages. In *Constraint Programming: Basic and Trends* (1995), vol. 910 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 68–89.

[21] Jaffar, J., and Lassez, J.-L. Constraint Logic Programming. Tech. rep., Department of Computer Science, Monash University, June 1986.

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

[23] Kunen, K. Negation in logic programming. *The Journal of Logic Programming 4* (1987), 289–308.

[24] Lassez, J. L., Maher, M. J., and Marriot, K. Unification revisited. In *Lecture Notes in Computer Science* (1986), vol. 306.

[25] Maher, M. J. Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees. In *Proceedings of* 3^{rd} *Symposium Logic in Computer Science* (1988), 349–357.

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

[27] Moreno-Navarro J.J., and Muñoz-Hernández S. On the Practical Use of Negation in a Prolog Compiler. *Proceedings of AGP98-International Conference on Declarative Programming*, (J.L. Freire and M. Falaschi eds.), 457–469, 1998.

[28] Muthukumar, K., and Hermenegildo, M. Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation. In *Proc. of International Conference on Logic Programming*, MIT Press, 49–63, 1991.

[29] Ozsooglu,G., Ozsoyoglu, Z., and Matos, V. Extending Relational Algebra and Relational Calculus with Set Valued Attributes and Aggregate Functions. In *Transactions on Database Systems*, Vol. 12, No. 4, ACM Press, 1987.

[30] Przymusinski, T. On constructive negation in logic programming. *Proceedings of North American Conference on Logic Programming*, MIT Press, 1990.

[31] Shepherdson, J. C. Negation in Logic Programming. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, 1988, 19–88.

[32] Stuckey, P. J. Negation and Constraint Logic Programming. *Information and Computation 1* (1995), 12–33.

[33] Vaught, R. L. On a Theorem of Cobham Concerning Undecidable Theories. In *Proceedings of the 1960 International Concress* (1962), E. Nagel, P. Suppes, and A. Tarski, Eds., Stanford University Press, Stanford, 14–25.

Appendix A: Constraint satisfiability test in $CLP(\mathcal{FT})$

Definition 9.1 *A constraint c is in* solved form *if it is either the constant* `false` *or a conjunction of admissible primitive constraints of one of the following forms:*

- $X = t$ *and X does not occur neither in t nor in the remaining part of c;*
- $\forall \bar{Z}(X \neq t)$, *$X$ does not occur in t, $\bar{Z} \subseteq vars(t)$, $X \notin \bar{Z}$, and $t \notin \bar{Z}$.*[10]

Assuming that the signature Σ has an infinite number of function symbols (as, for instance, Chan implicitly does [10]), then a constraint in solved form is always satisfiable with respect to the given structure $\mathcal{A}$ (the Herbrand constraint domain in this case). This fact can be proved as follows. A solved form constraint is of the form

$$X_1 = t_1 \wedge \ldots \wedge X_m = t_m \wedge$$
$$\forall \bar{Z}_1 (Y_1 \neq s_1) \wedge \ldots \wedge \forall \bar{Z}_n (Y_n \neq s_n)$$

where $X_1, \ldots, X_m$ occur only once. If there is a solution for the second part, a solution for $X_1, \ldots, X_m$ can trivially be found. Let $\{V_1, \ldots, V_k\} = vars(\forall \bar{Z}_1 (Y_1 \neq s_1) \wedge \ldots \wedge \forall \bar{Z}_n (Y_n \neq s_n))$, and let $f_1, \ldots, f_k$ be k function symbols of Σ not occurring in $s_1, \ldots, s_n$. Then, any assignment of terms with outermost functor f_i to Y_i is a solution.

The case with a finite signature would be completely different. Assume, for instance, that Σ contains only the constant symbol a and the unary function symbol f (besides $=$). Then the constraint

$$X \neq a \wedge \forall Z (X \neq f(Z))$$

is unsatisfiable. However, in this case, if we would assume to have also DCA, then constraint satisfiability is still decidable (cf. [24]).

The $SAT_{\mathcal{FT}}$ procedure uses both the standard *unification* algorithm unify (see, e.g., [26, 24]), and a negative constraint manager, here named notequal_FT, whose complete definition is shown in Fig. 4.

[10] Observe that conditions $X \notin \bar{Z}$ and $t \notin \bar{Z}$ in the second kind of constraint in solved form are aimed at excluding special cases, like $\forall \bar{Z} X (X \neq t)$ and $\forall Y (X \neq Y)$, which are trivially unsatisfiable.

Given a disjunction of admissible constraints $\bigvee_{i=1}^{h} c_i$, $SAT_{\mathcal{FT}}$ first applies unify to each disjunct c_i, obtaining for each of them either `false` or a (unique) constraint d_i'.[11] Then, notequal_FT is applied to each d_i', getting for each of them an equivalent disjunction of constraints. Since notequal_FT may generate new unification problems (cf. action (8)), unify and notequal_FT need to be iterated, until a disjunction $\bigvee_{j=1}^{k} c_j'$, where all the c_j''s are in solved form, is obtained.

Note that, by assuming that rule (1) has precedence over the other rules, superfluous universal quantifications are always removed before applying any other rule in notequal_FT.

$SAT_{\mathcal{FT}}$ is guaranteed to terminate for every input formula C. Moreover, the disjunction C' obtained by $SAT_{\mathcal{FT}}$ is equivalent to the given disjunction C, as stated by the following

Theorem 9.2 *Given an input formula C of the form $\bigvee_{i=1}^{h} c_i$, if $SAT_{\mathcal{FT}}(C)$ is $\bigvee_{j=1}^{k} c_j'$, then*

$$CET \models \bigvee_{i=1}^{h} c_i \leftrightarrow \bigvee_{j=1}^{k} c_j'$$

Proof. (hint) The key point is to prove soundness and completeness of notequal_FT. These are ensured: by first-order logic for rule (1); by CET for rules (2)–(7); by Lemma 2.3 for rule (8).

9.2 □

Appendix B: Constraint simplification process in $CLP(\mathcal{FT})$

The conjunction $\bigwedge_{i=1}^{k} \forall \bar{Y}_i \neg c_i$ is simplified via a rewriting process that transforms it into an equivalent disjunction of admissible constraints $\bigvee_{i=1}^{h} \exists \bar{Z}_i d_i$ which is in a suitable form to be passed to $SAT_{\mathcal{FT}}$.

The simplification process is accomplished through the following sequential steps.

1. Let X be a variable in $\bar{X}$ (that is, a goal-variable) and Y be a variable in $\bar{Y}_i$ (that is, a variable introduced by the resolution process). For each constraint c_i in $\bigwedge_{i=1}^{k} \forall \bar{Y}_i \neg c_i$ apply the following rules.

   - Remove all the *redundant variables*:

     $$X = Y \wedge c' \mapsto c'[Y/X].$$

     By equality axioms, $\exists Y\, (X = Y \wedge c')$ is equivalent to $c'[Y/X]$.

[11] Note that unify does not generate any new $\neq$-constraint and leaves untouched formulae not based on $=$-constraints, save the variable substitution rule which is applied to the whole input formula.

Let C be a disjunction of admissible constraints, c, c', and D be admissible constraints, t, s, t_i, and s_i be terms, f and g be distinct function symbols, and X and Y be distinct variables.

notequal_FT(C) :

while at least one disjunct c in C is not in solved form

rewrite c according to one of the following rules:

(1)	$\forall X \bar{Z}\,(D) \wedge c'$, and $X \notin vars(D)$	$\mapsto$	$\forall \bar{Z}\,(D) \wedge c'$
(2)	$\forall \bar{Z}\,(f(s_1, \ldots, s_m) \neq g(t_1, \ldots, t_n)) \wedge c'$	$\mapsto$	c'
(3)	$\forall \bar{Z}\,(f(s_0, \ldots, s_n) \neq f(t_0, \ldots, t_n)) \wedge c'$	$\mapsto$	$\forall \bar{Z}\,(t_0 \neq s_0 \vee \cdots \vee t_n \neq s_n) \wedge c'$
(4)	$f \neq f \wedge c'\}$	$\mapsto$	`false`
(5)	$\forall \bar{Z}\,(t \neq X) \wedge c'$, and t is not a variable	$\mapsto$	$\forall \bar{Z}\,(X \neq t) \wedge c'$
(6)	$\forall \bar{Z}\,(X \neq X) \wedge c'$ or $\forall X \bar{Z}\,(X \neq t) \wedge c'$, and $X \notin vars(t)$, or $\forall Y\,(X \neq Y) \wedge c'$	$\mapsto$	`false`
(7)	$\forall \bar{Z}\,(X \neq f(t_1, \ldots, t_n)) \wedge c'$, and $X \in vars(t_1, \ldots, t_n)$	$\mapsto$	c'
(8)	$\forall \bar{Z}\,(s \neq t \vee D) \wedge c'$, and $\bar{X} = \bar{Z} \cap vars(s \neq t), \bar{Z}_D = \bar{Z} \setminus \bar{X}$	$\mapsto$	$(\forall \bar{X}\,(s \neq t) \wedge c') \vee (s = t \wedge \forall \bar{Z}_D\, D \wedge c')$

Figure 4: Rewriting function for $\neq$ in $CLP(\mathcal{FT})$

- Remove all the *redundant equalities*:

$$Y = t \wedge c' \mapsto c'.$$

If a variable Y introduced by the refutation process occurs as $Y = t$, then the substitution $[Y/t]$ has been applied. Hence, Y does not occur in c', and $\exists Y\,(Y = t \wedge c')$ is equivalent to $c' \wedge \exists Y\,(Y = t)$. The latter is itself equivalent to c'.

- Remove the *irrelevant negative constraints*:

$$\left.\begin{array}{r} \forall \bar{Z}\,(V \neq t) \wedge c', \\ \text{and } \exists Y \in vars(\forall \bar{Z}\,(V \neq t)),\ Y \in \bar{Y}_i \\ \text{s.t. } Y \text{ is not in the equality part of } c' \end{array}\right\} \mapsto c'.$$

This step is justified by the assumption that the signature is infinite. The main idea is that if a variable Y introduced by the refutation process is not connected to a goal-variable X, then it is always possible to find a value for it satisfying the negative constraints where it occurs. To show the necessity of a signature containing an infinite number of function symbols, consider the following example. If the only function symbols in Σ are the constant 0 and the one-place function symbol s, then $c' \wedge Y \neq 0 \wedge \forall Z\,(Y \neq s(Z))$ is unsatisfiable even if Y does not occur in c' and c' is satisfiable.

This rewriting transforms each c_i into a new constraint c_i' of the form

$$\exists \bar{Y}_i\,(X_{\alpha_1} = t_1 \wedge \ldots \wedge X_{\alpha_m} = t_m \wedge \forall \bar{Z}_1\,(V_1 \neq s_1) \wedge \ldots \wedge \forall \bar{Z}_j\,(V_j \neq s_j))$$

where $\bar{Y}_i \subseteq vars(t_1, \ldots, t_m)$.

2. For each constraint c_i', apply notequal_FT (specifically, action (8)) to its negation $\neg c_i'$:

$$\forall \bar{Y}_i\,(X_{\alpha_1} \neq t_1 \vee \ldots \vee X_{\alpha_m} \neq t_m \vee \exists \bar{Z}(V_1 = s_1 \vee \ldots \vee V_j = s_j))$$

where $\bar{Z} = \bar{Z}_1 \cup \cdots \cup \bar{Z}_j$.

In this way, for each $\neg c_i'$ we get an equivalent disjunction of conjunctions. In particular, the simplification process at step 1 (which ensures that $\bar{Y}_i \subseteq vars(t_1, \ldots, t_m)$) guarantees that the last disjunct, that contains the equality part of the formula $\neg c_i'$, is of the form:

$$\exists \bar{Y}_i\,(X_{\alpha_1} = t_1 \wedge \ldots \wedge X_{\alpha_m} = t_m \wedge \exists \bar{Z}(V_1 = s_1 \vee \ldots \vee V_j = s_j)).$$

Thus, the existentially quantified variables $\bar{Z}$ (that can be chosen so as to be distinct from variables in $X_{\alpha_1} = t_1 \wedge \ldots \wedge X_{\alpha_m} = t_m$) can be moved in front of the formula and hence implicitly quantified, as usual.

3. By taking the conjunction of all the formulae obtained at step 2, and by distributing $\vee$ over $\wedge$, we get a disjunction of admissible constraints $\bigvee_{i=1}^{h} \exists \bar{Z}_i\, d_i$.

Appendix C:
Satisfiability of Constraints in Solved Form in $CLP(\mathcal{SET})$

A constraint in *solved form* is either false or it is a conjunction of the form $C_= \wedge C_\neq \wedge C_\notin$, where

- $C_=$ is a conjunction of primitive constraints of the form $X = t$, where X occurs only once in the whole constraint $C_= \wedge C_\neq \wedge C_\notin$;
- $C_\neq$ is a conjunction of primitive constraints of the form $\forall \bar{Z}\,(X \neq t)$, where X does not appear in t, $X \notin \bar{Z}$, and $t \notin \bar{Z}$;
- $C_\notin$ is a conjunction of primitive constraints of the form $\forall \bar{Z}\,(t \notin X)$, where X does not appear in t and $X \notin \bar{Z}$.

This choice for the solved form is justified by the following result:

Lemma 9.3 *A solved constraint c different from* false *admits at least one solution with respect to the structure $\langle \mathcal{S}, I^{\mathcal{S}} \rangle$ if the signature of the language is infinite.*

Proof. By definition a constraint c in solved form (different from false) can be decomposed in three parts, $C_=$, $C_\neq$, and $C_\notin$. We can build a solution σ for this constraint as follows:

- for each variable X which appears in a primitive constraint $\forall \bar{Z}\,(X \neq t)$ or $\forall \bar{Z}\,(t \notin X)$, define $\sigma(X) = t_X$, where t_X is any ground term whose main functor is not used anywhere in c.
- for each variable Y appearing free in the right-hand side of a primitive constraint $X = t[Y]$, $\forall \bar{Z}\,(X \neq t[Y])$, or $\forall \bar{Z}\,(t[Y] \notin X)$, define $\sigma(Y) = \emptyset$.
- for each variable X which appears in a primitive constraint of the form $X = t$, define $\sigma(X) = t\sigma$.

σ is indeed a solution of the constraint c.

9.3 □

From the proof of the Lemma 9.3 we can observe that a solution for the constraint c can be designed using only one function symbol in the signature of the language which does not appear in the constraint itself. Thus, we can express the following corollary to the lemma:

Corollary 9.4 *If c is a constraint in solved form different from* false*, and there exists a symbol f in the signature of the language which is not used in c, then c is satisfiable.*

The result expressed in the Corollary 9.4 does not hold if the requirement about the existence of an unused symbol in the signature is relaxed. This can be easily seen from the following example: let c be the constraint

$$Z \neq \emptyset \wedge \forall XY\,(Z \neq \{X \,|\, Y\})$$

This constraint does not admit any solution if we restrict the signature of the language to contain only $\emptyset$ and $\{\cdot \mid \cdot\}$. Furthermore, in presence of finite signatures, the traditional rule of independence of disequations [24] does not hold—e.g., assuming a signature containing only $\emptyset$ and $\{\cdot \mid \cdot\}$, given the constraint $\forall Z_1 Z_2 \, (X \neq \{Z_1 \mid Z_2\}) \wedge X \neq \emptyset$, each disequation is satisfiable but, as discussed before, the whole system does not admit any solutions. The problem of deciding satisfiability of (set) constraint in solved form within a language with finite signature is currently under investigation.