Constructive Negation and Constraint Logic Programming with Sets

Agostino DOVIER

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

Enrico PONTELLI

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

Gianfranco ROSSI

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

Received 29 September 1998

Abstract The aim of this paper is to extend the *Constructive Negation* technique to the case of $CLP(\mathcal{SET})$, a Constraint Logic Programming (CLP) 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 concerning the relationships between CLP and constructive negation.

We propose a new constraint satisfaction algorithm, capable of correctly handling constructive negation for large classes of $CLP(\mathcal{SET})$ programs; we also provide a syntactic characterization of such classes of programs. The resulting algorithm provides a novel constraint simplification procedure to handle constructive negation, suitable to theories where unification admits multiple most general unifiers. We also show, using a general result, that it is impossible to construct an interpreter for $CLP(\mathcal{SET})$ with constructive negation which is guaranteed to work for any arbitrary program; we identify classes of programs for which the implementation of the constructive negation technique is feasible.

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

§1 Introduction

The most popular technique for handling negative literals in logic programs is the *Negation as Failure* (NAF) rule [13]. In NAF, given a logic program P and a *ground* atom A, failure to infer an atom allows one to infer its negation, i.e.,

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

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 [24]. Negation as Failure has the advantage of simplicity, which leads to the development of efficient implementations. On the contrary, soundness and completeness of the resolution procedure extended with Negation as Failure (i.e., SLDNF-resolution [35]) can be obtained only by applying suitable syntactical restrictions to programs and goals [1], which are necessary to avoid application of negation to non-ground subgoals. These restrictions may be acceptable in some specific application areas (e.g., deductive databases), but may lead to severe lack of expressive power in general-purpose programming languages.

A viable alternative to Negation as Failure, that overcomes many of these problems, is *Constructive Negation (CN)* [11, 5, 20, 21, 37]. 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, i.e., they can generate constraints on the variables occurring in the goal—from here comes the *constructive* nature of this approach. The term constructive negation was first introduced by Chan [11], as a technique for extending the NAF rule in the context of Logic Programming. Constructive negation has been extended to the more general context of Constraint Logic Programming (CLP) by Stuckey [37], who provided the first completeness result for this method.

The aim of this paper is to analyze the problem of adding constructive negation to $CLP(\mathcal{SET})$, an instance of the general CLP scheme incorporating a powerful theory of sets.

The long term goal of this work is to allow $CLP(\mathcal{SET})$ to directly encode *set grouping* mechanisms. Set grouping [6, 33] is the ability to collect in a set S all the values of X satisfying a given property $p(X, \bar{Y})$, that is,

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

We have demonstrated that set grouping can be implemented using $CLP(\mathcal{SET})$ clauses, but this construction demands the presence of negative literals. Negative literals 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. These set grouping mechanisms are in turn needed to support *intensional set formers*, i.e., set abstractions of the type $\{X : p(X)\}$, according to the solution described in [17]. Various proposals have argued the equivalence between intensional set constructions and negation in logic programming [17, 6].

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 on the form of admissible intensional set formers. For instance, the grouped set cannot be empty whenever the intensional set construction 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 will allow the set grouping operation to determine the constraints on the free variables $\bar{Y}$ which are necessary to guarantee that the set corresponds to the set of values satisfying $p(X, \bar{Y})$.

Example 1.1
Given the following program and goal

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

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

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

This type of flexibility in the operational behavior of intensional set formers cannot be obtained using NAF, but can be achieved by adopting the CN approach to implement negation in set grouping clauses.

Since $CLP(\mathcal{SET})$ is an instance of the CLP scheme, we could try to directly adopt the general scheme for constructive negation in CLP developed by Stuckey in [37]. The applicability of Stuckey's technique, however, depends on the nature of

- the structure $\mathcal{A}$ on which the specific instance of the CLP framework is based, and
- the set of *admissible constraints* in such instance of the CLP framework.

Intuitively, the *admissible constraints* [37] are those formulae that the constraint solver can directly deal with. In particular it is decidable to verify whether $\mathcal{A} \models \exists\varphi$ for each admissible constraint φ. For a *decidable structure*—i.e., a structure $\mathcal{A}$ in which it is decidable to determine whether $\mathcal{A} \models \varphi$ for any first order formula φ—we could choose the set of all formulae as admissible constraints. Nevertheless, even in presence of decidable structures, the class of admissible constraints is typically restricted to guarantee efficient decision procedures—e.g., by keeping it as close as possible to the set of *primitive* constraints.

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* [37] when there is a way to rewrite a *negative* formula of this kind to an equivalent *positive* formula, i.e., a (finite) disjunction of conjunctions of admissible constraints, where universal quantifications are applied only to primitive constraints. Admissible closed structures are well suited to CN as they allow to make explicit the effect of negating a set of solutions. Indeed, the majority of the structures used in CLP are admissible closed [37].

The structure of $CLP(\mathcal{SET})$ is that of *hereditarily finite (and hybrid) well-founded sets.* The primitive constraints are composed by the (positive and negative) literals involving the predicate symbols $=$ and $\in$.

The minimal set of axioms that must hold in every theory of sets includes axiom (N), guaranteeing the existence of the empty set, and axiom (W), describing the operation of adding an element to a set [7]. Every structure we want to use for sets, therefore, must be able to express (N) and (W). The simplest structure for sets is that of *hereditarily finite sets* [10]. As a matter of fact, every model of the axioms (N) and (W) contains at least one sub-model which is isomorphic to the structure of hereditarily finite sets. Vaught proved [38] the *essential undecidability* of the theory consisting of the axioms (N) and (W). This implies that, no matter how we select a structure for sets, it will turn out to be undecidable. In particular, this allows us to conclude that the *theory* of the structure of hereditarily finite sets, being a consistent extension of (NW), is undecidable. As we will discuss in this paper, this result has important consequences with respect to the management of constructive negation: whenever the admissible constraints include all the literals, it is impossible to develop a sound and complete interpreter handling CN over such structures [19].

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 [11] to $CLP(\mathcal{SET})$. The paper focuses mainly on the foundations of this extension.

The main technical contribution of this paper is the development of a new constraint simplification procedure, which can cope with the more general framework of a theory T where unification admits multiple maximally general unifiers. Moreover, we prove a negative result about the possibility of extending the existing approaches to CN [37] to the case of $CLP(\mathcal{SET})$, because of the inherent undecidability of the underlying set theory.

Even though the CN technique we propose is unable to cover all possible cases when used in the context of $CLP(\mathcal{SET})$, we show that it still represents a true enhancement with respect to similar extensions based on the NAF rule. In particular, the technique fulfills to a large extent our needs for designing intensional set formers within the language itself.

The paper is organized as follows. Section 2 briefly reviews the basic ideas—adapted from [11]—underlying the introduction of constructive negation into a standard CLP framework over finite trees $CLP(\mathcal{FT})$ (i.e., a generalization of Prolog). Section 3 provides a brief overview of the $CLP(\mathcal{SET})$ framework. Section 4 extends the rewriting techniques of CN to the case of non-unitary theories, namely theories in which a unification problem may admit more than one (independent) unifier to express all possible solutions. In Section 5 we provide arguments demonstrating 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 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 $CLP(\mathcal{FT})$ [11]. $CLP(\mathcal{FT})$ is the instance of CLP based on the structure of finite trees and generalizing the structure of Prolog. 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. In this regard, most of the standard CLP notations and results [22, 23, 37] are assumed.

Let Σ denote a first-order *signature* which defines a set of predicate symbols Π, a set of function and constant symbols $\mathcal{F}$, and a denumerable collection of variables $\mathcal{V}$. Π is composed by two disjoint sets of symbols: the set of *constraint predicate symbols* (Π_C) and the set of *program predicate symbols* (Π_P). $\bar{t}$ 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. With a slight abuse of notation, we will sometime use $vars(\varphi_1, \ldots, \varphi_n)$ to denote the set $vars(\varphi_1) \cup \cdots \cup vars(\varphi_n)$. 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 $\bar{X} \setminus (\bar{Y} \cap \bar{Z})$. $\varphi[\bar{X}_1, \ldots, \bar{X}_n]$ represents a first-order formula φ where $vars(\varphi) = \bar{X}_1 \cup \cdots \cup \bar{X}_n$.

A *constraint* is, in general, a first-order formula on the signature $\langle \Pi_c, \mathcal{F}, \mathcal{V} \rangle$. `false` is used as a shorthand for an unsatisfiable constraint (e.g., $X \neq X$). Similarly `true` is a shorthand for a tautology constraint. A CLP program P with negation is a set of clauses of the form $A \mathrel{:-} c \,|\, \bar{B}$, where A (the *head*) is a $\langle \Pi_P, \mathcal{F}, \mathcal{V} \rangle$ atom, c is a conjunction of constraints and $\bar{B}$ is a (possibly empty) conjunction of $\langle \Pi_P, \mathcal{F}, \mathcal{V} \rangle$ literals. A *goal* is a clause with empty head.

An instance of the CLP scheme is determined by the following components: a signature Σ, a Σ-structure $\mathcal{A}$ which determines the meaning of the predefined predicate and function symbols, and a first-order Σ-theory $\mathcal{T}$, used to axiomatize the properties of the symbols in Σ. A class AC of *admissible* *constraints* can be introduced to restrict the set of constraints that can be used in programs and goals. Given a structure $\mathcal{A}$ and a collection of admissible constraints AC, the structure is *admissible closed with respect to* AC [37] if, for each conjunction $\varphi[\bar{X}, \bar{Y}]$ of formulae from AC, there exists a formula $\psi[\bar{X}, \bar{Z}]$ such that

- ψ is a disjunction of conjunctions of formulae from AC, and
- $\mathcal{A} \models \forall \bar{X} (\neg \exists \bar{Y} \varphi[\bar{X}, \bar{Y}] \leftrightarrow \exists \bar{Z} \psi[\bar{X}, \bar{Z}])$.

If ψ can be effectively computed, then $\mathcal{A}$ is said to be *constructively admissible closed with respect to* AC.

$CLP(\mathcal{FT})$ is the instance of the CLP framework which considers Σ to contain an infinite collection of function symbols—interpreted as tree constructors—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 domain of $\mathcal{A}$ corresponds to the *Herbrand Universe* used in traditional logic programming. The *primitive constraints* are equations and disequations between terms. Clark's Equality Theory (CET) [13, 28] has been shown to be a complete axiomatization of this structure [27].

Definition 2.1
An *admissible constraint* [11] 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 constraints.

Example 2.1
Assume p is defined by the clause

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

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 $\forall Z(V \neq f(Z) \vee W \neq g(Z))$.[*2]

Fortunately, such kind of formulae can be transformed into equivalent disjunctions of conjunctions of admissible constraints by using the following simplification rule [11]:

Lemma 2.1
Let t be a term, D be a formula, and let $\bar{X} = vars(t)$ and $\bar{Y} = vars(D) \setminus \bar{X}$. Then

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

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

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

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

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

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

Definition 2.2
A *CN constraint* in $CLP(\mathcal{FT})$ with CN is a conjunction of admissible constraints and $\forall$-quantified disjunctions of admissible constraints.

Example 2.2
The following formula is a CN constraint:

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

According to Lemma 2.1 it is possible to rewrite it as:

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

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

As a corollary of Lemma 2.1, we have that the structure of $CLP(\mathcal{FT})$ with CN is constructively admissible closed with respect to the set of admissible constraints of $CLP(\mathcal{FT})$. The constraint satisfiability procedure used in $CLP(\mathcal{FT})$ with CN must be able to check the satisfiability of formulae of the form $\vee_{i=1}^{h} c_i$, where each c_i is a CN 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 canonical form (here called *solved form*). A constraint in solved form different from `false` can be proved to be always satisfiable with respect to the given structure $\mathcal{A}$ (the Herbrand Universe in this case).

The procedure $SAT_{\mathcal{FT}}$ is articulated in two components. The first component handles equality constraints, and can be implemented using a standard unification algorithm [24)]. The second component is used to simplify $\neq$-constraints. The complete definition of this second component of $SAT_{\mathcal{FT}}$—called notequal_FT—along with the definition of solved form for $CLP(\mathcal{FT})$ with CN, is presented in Appendix A. $SAT_{\mathcal{FT}}$ incorporates the simplification rule of Lemma 2.1, whose aim is to deal with universally quantified disjunctions of CN 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); or
- during execution of rule (3) of the notequal_FT procedure—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 scheme by Stuckey [37)]—obtained by taking as the frontier of the sub-derivation for a negative goal $:-\ \neg G$ the finite collection of solutions for $:-\ G$. Hence, it requires that the refutation tree has no infinite branches—though relaxations of this condition are possible [12, 20, 34, 37)]. Assume that the negative literal $\neg p(\bar{t})$ is selected in a goal

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

As in [37, 34)], a sub-derivation for $c \mid p(\bar{t})$ is developed—i.e., we determine the solutions for $p(\bar{t})$ which are consistent with all the constraints in c[*3] This technique allows to reduce the number of possible solutions for $:-p(\bar{t})$ by pruning as soon as possible some branches of its derivation tree (i.e., all those which are

[*3] Note that the goal $:-\ c \mid \neg A, \bar{B}$ (i.e., $\neg c \vee A \vee \neg \bar{B}$) is logically equivalent to $:-\ c \mid \neg(c \mid A), \bar{B}$ (i.e., $\neg c \vee (c \wedge A) \vee \neg \bar{B}$).

not compatible with the constraints in c).

Let the goal $:- c \mid p(\bar{t})$ have k solutions, where each solution is a conjunction 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 holds [37]:

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

where $comp(P)$ is the *completion* of P [13]. Intuitively, the completion of a program is obtained by transforming the implications expressed by the program's rules into bi-implications; e.g., if all the rules defining predicate p in a program P are

$$\begin{array}{lcl} p(X_1, \ldots, X_n) & :- & \Gamma_1 \\ & \ldots & \\ p(X_1, \ldots, X_n) & :- & \Gamma_m \end{array}$$

then $comp(P)$ will contain the formula

$$p(X_1, \ldots, X_n) \leftrightarrow \bigvee_{i=1}^{m} \exists \bar{Y}_i \Gamma_i$$

where $\bar{Y}_i = vars(\Gamma_i) \setminus \{X_1, \ldots, X_n\}$. Note that we assume that $comp(P)$ contains also the selected first-order Σ-theory—in this case *Clark's Equational Theory (CET)* for the structure of finite trees. In the successive sections dealing with $CLP(\mathcal{SET})$ we will also assume that $comp(P)$ contains the theory *Set* for the structure of hereditarily finite sets.

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 $\wedge_{i=1}^{k} \forall \bar{Y}_i \, \neg c_i$ can be simplified via a rewriting based on Lemma 2.1, which transforms it to an equivalent disjunction of CN constraints

$$\bigvee_{i=1}^{h} \exists \bar{Z}_i \, d_i.$$

This formula is in a form suitable for manipulation by $SAT_{\mathcal{FT}}$. Appendix B provides a complete description of the rewriting process for $CLP(\mathcal{FT})$.

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

Example 2.3
Consider the following simple $CLP(\mathcal{FT})$ program and goal:

$$\begin{array}{l}
\mathtt{p(a,a).} \\
\mathtt{p(f(Y),f(Y)).} \\
:\!-\ \neg\mathtt{p(X_1,X_2).}
\end{array}$$

To compute the answer for the goal $:\!- \neg p(X_1, X_2)$ we first determine the answer for $:\!-\ p(X_1, X_2)$, which corresponds to:

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

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 rewriting described in Appendix B transforms the second conjunct, by applying to it the simplification rule of Lemma 2.1:

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

By applying distributivity we obtain the four disjuncts:

$$\begin{array}{l}
(X_1 \neq a \wedge \forall Y \, (X_1 \neq f(Y))) \ \vee \\
(X_1 \neq a \wedge X_1 = f(Y) \wedge X_2 \neq f(Y)) \ \vee \\
(X_2 \neq a \wedge \forall Y \, (X_1 \neq f(Y))) \ \vee \\
(X_2 \neq a \wedge X_1 = f(Y) \wedge X_2 \neq f(Y)).
\end{array}$$

This formula is finally communicated to $SAT_{\mathcal{FT}}$, which returns the following disjunction of solved form constraints

$$\begin{array}{l}
(X_1 \neq a \wedge \forall Y \, (X_1 \neq f(Y))) \ \vee \\
(X_1 = f(Y) \wedge X_2 \neq f(Y)) \ \vee \\
(\forall Y \, (X_1 \neq f(Y)) \wedge X_2 \neq a) \ \vee \\
(X_1 = f(Y) \wedge X_2 \neq a \wedge X_2 \neq f(Y)).
\end{array}$$

This formula represents the computed answer for the given goal.

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

$CLP(\mathcal{SET})$ [16, 8] is an instance of the general CLP scheme which supports extensional set formers and basic set-theoretical operations. In this section, we introduce the basic syntactic components of the language and we precisely characterize their meaning by specifying the interpretation domain $\mathcal{S}$ and its associated interpretation function $I^{\mathcal{S}}$.

From the syntactic point of view, sets are represented in $CLP(\mathcal{SET})$ using the binary function symbol $\{\cdot\,|\,\cdot\}$ as set constructor, while the constant $\emptyset$ is used to denote the empty set. Therefore, Σ is required to contain at least the two elements $\{\cdot\,|\,\cdot\}$ and $\emptyset$. For the sake of readability, we will use $\{\cdot\,|\,\cdot\}$ as an infix operator, thus writing $\{s\,|\,t\}$ instead of $\{\cdot\,|\,\cdot\}(s,t)$. Intuitively, $\{t\,|\,s\}$ denotes the set $\{t\} \cup s$.

Definition 3.1
A *set term* is recursively defined as follows:

- $\emptyset$ is a set term;
- a variable X is a set term;
- $\{t\,|\,s\}$ is a set term if t is a term and s a set term.

Hereafter, we adopt the simplified notation $\{t_0, \ldots, t_n|\, X\}$ to describe the term $\{t_0\,|\,\{t_1\,|\,\cdots\{t_n\,|\,X\}\cdots\}\}$, and, when X is $\emptyset$, we simply write it 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 $\in$; thus, all 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), are primitive constraints. A *constraint* is a conjunction of primitive constraints. The *admissible constraints* of $CLP(\mathcal{SET})$ are exactly the primitive constraints.

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 $\Sigma = \{\emptyset, \{\cdot\,|\,\cdot\}, \ldots\}$. Given an arbitrary total ordering $\preceq$ on U_H, we consider the smallest congruence relation $\equiv$ over U_H that fulfills the *left permutativity property*

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

and the *left absorption property*

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

for the symbol $\{\cdot \mid \cdot\}$.

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

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

Given a term t, let $t^{\mathcal{S}}$ denote the representative term of the congruence class containing t.

The interpretation function $I^{\mathcal{S}}$ is defined as follows. Any ground 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 $\{s_1, \ldots, s_{n-1}, t_1^{\mathcal{S}} \mid s_n\}$, $n > 0$, for some terms $s_1, \ldots, s_n$.

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—represents the domain of *hereditarily finite hybrid sets*, that is the collection of sets whose elements are uninterpreted Herbrand terms or other (finite) hybrid sets.

Such a domain can be also characterized via a simple axiomatic first-order set theory, which has been called *Set* [17, 18]. *Set* includes, besides the standard equality axioms, the axioms for the existence of the empty set and for the element insertion operation, and suitable versions of the CET axioms and of the extensionality axiom. Detailed description of these axioms has been presented elsewhere [17, 18]. It has been proved [16] that the structure $\langle \mathcal{S}, I^{\mathcal{S}} \rangle$ and the theory *Set correspond* on the class of admissible constraints considered above. Furthermore, this constraint domain is *solution compact* [22].

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 it is a conjunction of primitive constraints, each of the form:

- $X = t$, where X does not occur neither in t nor in the rest of c, or

- $X \neq t$, and X does not occur in t, or
- $t \notin X$, and X does not occur in t.

It is interesting to observe that a solved form constraint does not contain any constraint of the form $s \in t$. In fact, these constraints are eliminated by replacing them with equivalent equality constraints. In particular, the following equivalence can be proved to hold in *Set*

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

The constraint satisfiability test is performed by a procedure—called $SAT_{\mathcal{SET}}$—which non-deterministically computes, for each constraint c, a finite collection of constraints in solved form $\{c'_1, \ldots, c'_n\}$ that can be proved to be *equisatisfiable* to c. This result is stated in the following theorem.

Theorem 3.1
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 $SAT_{\mathcal{SET}}$ [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 3.1
Let c be a constraint in solved form, other than `false`. Then c is $\mathcal{S}$-solvable and *Set*-satisfiable. □

Corollary 3.1
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 $\mathsf{notin\_Set}$ 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 3.1
Given the equation $\{X \mid R\} = \{Y \mid S\}$, where X, Y, R, and S are 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. $[R/\{Y \mid N\}, S/\{X \mid N\}]$, where N is a new variable.

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

$$\mathsf{notequal\_Set}(\mathsf{notin\_Set}(\mathsf{equal\_Set}(\mathsf{in\_Set}(c))))$$

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

Example 3.2
Consider the constraint

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

$SAT_{\mathcal{SET}}$ first rewrites it as 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 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 transformations apply to the other constraint, $X = B \wedge \{B\} \neq \{A, B\}$, which leads to the second solution for the initial constraint, namely

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

§4 CN for Theories with Multiple Most General Unifiers

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. In general, a unification problem in the context of $CLP(\mathcal{SET})$ admits a *finite* number of independent solutions. 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*. Let us generalize the discussion to the case of unification with respect to an arbitrary equational theory E [36].

Given an equational theory E, we can define the concept of E-equality ($=_E$). The E-equality $=_E$ is the *least congruence relation* over the set of Σ-terms which contains E and which is closed under substitution [4]. A substitution σ is an *E-unifier* of two terms s, t if $s\sigma =_E t\sigma$—i.e., $s\sigma$ and $t\sigma$ belong to the same E-congruence class. Given two substitutions σ_1 and σ_2, we say that $\sigma_1 \leq_E \sigma_2$ if there exists a substitution λ such that $\sigma_1(X) =_E \lambda(\sigma_2(X))$ for all variables X.

Definition 4.1
Given two terms s, t and an equational theory E, a *Complete Set of E-unifiers* for s and t ($CSU_E(s,t)$) is a collection of E-unifiers of s, t such that for any E-unifier σ of s, t there exists an element θ in $CSU_E(s,t)$ such that $\sigma \leq_E \theta$.

A *complete set of maximally general E-unifiers for s and t* $\mu CSU_E(s,t)$ is a complete set of E-unifiers such that for all unifiers σ and τ in $\mu CSU_E(s,t)$, if $\sigma \neq \tau$ then $\sigma \not\leq_E \tau$.

The rewriting process used by the constructive negation technique—as Chan suggests [11]—is strongly based on the inherent properties of the underlying constraint domain. In particular, Lemma 2.1 is based on the assumption that for every pair of terms s, t there is a set $\mu CSU_E(s,t)$ that 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*—since the latter may admit multiple maximally general unifiers.

4.1 Failure of Lemma 2.1

To better understand why Lemma 2.1 fails, we exhibit a situation in which the $\leftarrow$ part of the bi-implication of Lemma 2.1 does not hold—conversely, the $\rightarrow$ part of this bi-implication is a simple theorem of first-order predicate calculus.

Example 4.1
Consider the formula

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

If Lemma 2.1 holds, then (i) can 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 2.1 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 implies that also the left-hand side is satisfied.
2. On the other hand, if $U\theta$ and t do unify, then since $U\theta$ is ground 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.

4.2 Handling Quantified Disjunctions of Constraints

In this section, we define a new rewriting rule that can replace the rule of Lemma 2.1 in those cases where unification can produce multiple maximally general unifiers—i.e., $\mu CST_E(s,t)$ has cardinality greater than one.

The availability of this new rule will allow us (Section 6) to generalize the CN technique to the $CLP(\mathcal{SET})$ framework. The intuitive solution is the following:

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

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 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 (i.e., the variables in t and D have been renamed). Moreover, with a slight abuse of notation, we will use θ_i to denote both a substitution as well as its equational representation as a system of equations in solved form (i.e., if X/t is part of the substitution θ_i then the equation $X = t$ is part of the equational representation of θ_i). 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 4.1
Let t be a term and D be a formula, and let

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

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,

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

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

$$\exists \bar{X}\,(U = t \wedge \neg \exists \bar{X}'\bar{Y}\,(t = t' \wedge \neg D')). \tag{2}$$

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

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

where $\{A_1, \dots, A_{h_i}\} = dom(\theta_i) \cap vars(t)$ and $\{B_1, \dots, B_{k_i}\} = dom(\theta_i) \cap \bar{X}'$. The completeness of the set of unifiers allows us to state

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

Using this equivalence we can rewrite formula (2) as follows:

$$\exists \bar{X}\,\left(U = t \wedge \neg \exists \bar{X}'\bar{Y}\,\left(\bigvee_{i=1}^{k} \exists \bar{Z}_i\,\theta_i \;\wedge\; \neg D'\right)\right). \tag{3}$$

Observe that the new variables $\bar{Z}_i$ are free under the substitutions produced by the unification process. Thus, $\bar{Z}_i\theta_i = \bar{Z}_i$. (3) can be written as:

$$\exists \bar{X}\,\left(U = t \wedge \neg \left(\exists \bar{X}'\bar{Y}\bar{Z}_1\,(\theta_1 \wedge \neg D') \vee \cdots \vee \exists \bar{X}'\bar{Y}\bar{Z}_k\,(\theta_k \wedge \neg D')\right)\right). \tag{4}$$

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

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

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 (4) we have

$$\exists \bar{X} \left(U = t \;\wedge \left(\begin{array}{l} \neg \exists \bar{X}_1'\bar{Y}\bar{Z}_1\,(\theta_1|_{vars(t)} \wedge \neg D'\theta_1) \\ \vee\ \cdots\ \vee \\ \neg \exists \bar{X}_k'\bar{Y}\bar{Z}_k\,(\theta_k|_{vars(t)} \wedge \neg D'\theta_k) \end{array} \right) \right)$$

which, finally, can be rewritten as

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

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 $A_j\theta_i|_{vars(t)}$. ■

Proposition 4.1
Consider the formula (1) of Lemma 4.1. 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 $\forall \bar{Y}\,D$. ■

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 2.1. 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 2.1 is a special instance of Lemma 4.1.

Remark 4.1
As can be seen in the proof of Lemma 4.1, 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 4.1 are illustrated by the following examples.

Example 4.2
Consider the same formula as in Example 4.1

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

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 4.3
Consider the formula

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

By applying Lemma 4.1, 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 4.1, the last part of the formula can be rewritten as:

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

By applying distributivity, we obtain:

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

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

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

§5 CN in an Undecidable Theory: $CLP(\mathcal{SET})$

The aim of the rewriting rule of Lemma 4.1 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

$$\forall \bar{X}_i' \bar{Z}_i (\neg\theta_i|_{vars(t)} \vee \forall \bar{Y} D' \theta_i) \tag{5}$$

are somehow "simpler" to rewrite than the original formula. This has been shown to work properly in the examples presented before. The availability of such a rewriting rule would allow us to conclude—according to the definitions in [37]—that the structure of $CLP(\mathcal{SET})$ with CN is *admissible closed*. In turn this would allow one to use the approach of [37] 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 would allow us to devise decision procedures for the undecidable theory. The following example shows a case in which the rewriting rule of Lemma 4.1 applied to a $CLP(\mathcal{SET})$ formula leads to a non-terminating process.

Example 5.1
Consider the formula

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

The unification $\{X_1 \,|\, X_2\} = \{X_1' \,|\, X_2'\}$ admits four maximally general unifiers (see Example 3.1):

1. $[X_1'/X_1, X_2'/X_2]$,
2. $[X_1'/X_1, X_2'/\{X_1 \,|\, X_2\}]$,
3. $[X_1'/X_1, X_2/\{X_1' \,|\, X_2'\}]$,
4. $[X_2/\{X_1' \,|\, N\}, X_2'/\{X_1 \,|\, N\}]$.

Thus, the resulting equivalent formula from Lemma 4.1 is

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

In this case the rewriting produces two sub-formulae that must be rewritten whose "complexity" is greater than that of the original formula (note that the new formula contains instances of formulae of the same form as the original disjunction).

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 $\{\cdot \,|\, \cdot\}$, is just one of these sub-structures. Therefore, according to Vaught [38], this structure is undecidable. It is possible to use this result to show that if the class of admissible constraints contains all the constraint literals and their conjunctions (which is a rather minimal requirement) and the structure is undecidable, then it is not possible to build an effective interpreter for constructive negation [19].

Let us summarize here the key points of this result [19] applied to the case of $CLP(\mathcal{SET})$. Let us consider a generic formula Ψ for this structure; without loss of generality, assume Ψ is written in the 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 $vars(\psi_1, \ldots, \psi_k) = \bar{X}_1 \cup \cdots \cup \bar{X}_n$. Using the transformation procedure from [19], it is possible to translate Ψ into a $CLP(\mathcal{SET})$ program with negation

P and a goal G. For instance, the formula

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

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

$$\begin{array}{l}
\mathtt{p}_0 :- \mathtt{p}_1(\mathtt{X}_1). \\
\mathtt{p}_1(\mathtt{X}_1) :- \neg\mathtt{p}_2(\mathtt{X}_1). \\
\mathtt{p}_2(\mathtt{X}_1) :- \mathtt{p}_3(\mathtt{X}_1, \mathtt{X}_2). \\
\mathtt{p}_3(\mathtt{X}_1, \mathtt{X}_2) :- \neg\mathtt{p}_4(\mathtt{X}_1, \mathtt{X}_2). \\
\mathtt{p}_4(\mathtt{X}_1, \mathtt{X}_2) :- \mathtt{p}_5(\mathtt{X}_1, \mathtt{X}_2, \mathtt{X}_3). \\
\mathtt{p}_5(\mathtt{X}_1, \mathtt{X}_2, \mathtt{X}_3) :- \mathtt{X}_1 \in \mathtt{X}_2, \mathtt{X}_1 = \mathtt{X}_3. \\
\mathtt{p}_5(\mathtt{X}_1, \mathtt{X}_2, \mathtt{X}_3) :- \mathtt{X}_1 \in \mathtt{X}_3.
\end{array}$$

together with the goal

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

where $\mathtt{p}_0, \ldots, \mathtt{p}_5$ are new predicate symbols. Intuitively, the construction of the program inductively decomposes the original formula, introducing a new predicate for each level of decomposition.

It is possible to prove that if P_Ψ is the program generated from the formula Ψ and $G_\Psi[\bar{Y}]$ the goal, then

$$comp(P_\Psi) \models_3 \forall\bar{Y}(\Psi \leftrightarrow G_\Psi[\bar{Y}])$$

where $\models_3$ is the logical consequence operator in three valued logic [37].

Furthermore, the $CLP(\mathcal{SET})$ programs obtained in this way are always recursion-free. This implies that they are consistent in two valued logic and the refutation trees are always finite: non-termination can not be introduced by the resolution process. Thus, an interpreter of $CLP(\mathcal{SET})$ with negation will return a computed answer for G_Ψ with respect to the program P_Ψ if and only if the formula is satisfiable in the structure $\langle \mathcal{S}, I^{\mathcal{S}} \rangle$.

Therefore, the constructive negation technique for $CLP(\mathcal{SET})$—even if applied to the simple case of non-recursive programs—would allow us to decide the satisfiability problem with respect to NW for quantified formulae, contradicting [38]—which is an undecidable problem. This result allows us to argue that in general it is not possible to construct a sound and complete system which handles arbitrary $CLP(\mathcal{SET})$ programs with CN.

5.2 A Class of Decidable $CLP(\mathcal{SET})$ Formulae

The open question is then to identify which are the classes of $CLP(\mathcal{SET})$ formulae that can be safely rewritten using the rule of Lemma 4.1. As we have seen, starting from the formula

$$\forall \bar{X}\,(U \neq t \vee D) \tag{6}$$

we obtain a formula that contains a certain number of occurrences of disjuncts of the form $\forall \bar{X}_i' \bar{Z}_i\,(\neg\theta_i|_{vars(t)} \vee D'\theta_i)$.[*4] 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 (6) is selected and transformed using the rule of Lemma 4.1. 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 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.

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 4.1) and, hence, one additional step is sufficient to completely eliminate the disjunctions 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, may lead to a potentially unbound growth of the terms generated after each rewriting step.

Theorem 5.1
The formula $\forall \bar{X}\,(U \neq t \vee D)$, where $vars(t) = \bar{X} \cup \bar{Q}$, can be rewritten using Lemma 4.1 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.

[*4] Hereafter we will assume, without loss of generality, that the $\forall$-quantified variables $\bar{Y}$ occurring only in the sub-formula D of formula (1) have been moved into D itself.

On the contrary, if t contains some set term, we must analyze the sub-formulae 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 4.1. 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 sub-formula

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

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 $\forall \bar{X}(U \neq t \vee \cdots)$, 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 $\{\cdot \mid \cdot\}$, 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 $vars(t_j) \cap \bar{X}' \subsetneq \bar{X}'$ 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 process 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 sub-term 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:

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

have size strictly smaller than that of t.

■

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

$CLP(\mathcal{SET})$ does not provide any form of negation in its 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 restrictions 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 [37], 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 [25].

6.1 Admissible Constraints and Constraint Satisfiability

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

Definition 6.1

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

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

CN constraints are defined from admissible constraints as in Definition 2.2.

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. $X = t$, and the variable X does not occur neither in t nor in the rest of c;
2. $\forall \bar{Z}(X \neq t)$, the variable X does not occur in t, $\bar{Z} \subseteq vars(t)$, $X \notin \bar{Z}$, and $t \notin \bar{Z}$;
3. $\forall \bar{Z}(t \notin X)$, the variable 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, $SAT_{\mathcal{SET}}$ has to check, in general, the satisfiability of a disjunction of CN constraints $C = \vee_{i=1}^{h} c_i$. 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 Figure 1 (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 under Clark's Equality Theory).

- 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

$$\begin{array}{ll} \forall \bar{Z}\,(& \exists W(\, W \in \{s_1 \,|\, s\} \wedge W \notin \{t_1 \,|\, t\}\,) \vee \\ & \exists W(\, W \in \{t_1 \,|\, t\} \wedge W \notin \{s_1 \,|\, s\})\,). \end{array} \qquad (7)$$

Unfortunately, this gives rise to a new kind of formulae which are likely to be much harder to deal with than the original one and, consequently, can lead to non-termination of the rewriting process. Therefore, we prefer to avoid to deal with the more general case of the input formula and, rather, we single out three possible sub-cases 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 (7) 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 (7) (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 $\{t_1,\ldots,t_n \,|\, X\}$ and $\{s_1,\ldots,s_m \,|\, X\}$. We can proceed like in (i), obtaining in addition two 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 4.1.

6.3 notin_Set

Figure 2 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:

- $\forall \bar{Z}(t \notin \{s_1 \,|\, s\} \vee D)$ (cf. rule (7)): 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.
- $\forall \bar{Z}(t \notin f(t_1, \ldots, t_n) \vee D)$, and $f \not\equiv \{\cdot \,|\, \cdot\}$ (cf. rule (5)): in this case the first disjunct is certainly true (disregarding $\bar{Z}$) and the whole formula can be simply removed.
- $\forall \bar{Z}\,(t \notin X \vee D)$, and $X \notin vars(t)$ (cf. rule (6)): 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.

It is important to observe that, in the general case, non-termination of the rewriting process may arise from any input formula that, after rewriting, contains a sub-formula of the form

$$\forall \bar{Z}\,(s \neq t \vee D)$$

Let C be a disjunction of CN constraints, c, c', and D be CN 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 $\{\cdot \mid \cdot\}$.

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 $\forall \bar{Z}\,(X \neq \{t_1,\ldots,t_n \mid \Delta\}) \wedge c'$, $\Delta \equiv \emptyset$ or $\Delta \equiv Y$, and $X \in vars(t_1,\ldots,t_n)$	$\mapsto$	c'
(7″)	$\forall \bar{Z}\,(X \neq \{t_1,\ldots,t_n \mid X\}) \wedge c'$, 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′)	$\forall \bar{Z}\,(\{s_1,\ldots,s_m \mid \Delta\} \neq \{t_1,\ldots,t_n \mid \Delta\}) \wedge c'$, and $\Delta \equiv \emptyset$ or $\Delta \equiv X$	$\mapsto$	$\forall \bar{Z}\,(s_1 \notin \{t_1,\ldots,t_n \mid \Delta\} \vee \cdots \vee s_m \notin \{t_1,\ldots,t_n \mid \Delta\} \vee t_1 \notin \{s_1,\ldots,s_m \mid \Delta\} \vee \cdots \vee t_n \notin \{s_1,\ldots,s_m \mid \Delta\}) \wedge c'$
(8″)	$\{s_1 \mid s\} \neq \{t_1 \mid t\} \wedge c'$	$\mapsto$	$(W \in \{s_1 \mid s\} \wedge W \notin \{t_1 \mid t\} \wedge c') \vee (W \in \{t_1 \mid t\} \wedge W \notin \{s_1 \mid s\} \wedge c')$.
(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 $\bar{X}'_i = \bar{X}' \setminus dom(\theta_i)$	$\mapsto$	$(\forall \bar{X}\,(s \neq t) \wedge c') \vee \left(s = t \wedge \bigwedge_{i=1}^{k} \forall \bar{X}'_i \bar{N}_i (\neg\theta_i\vert_{vars(t)} \vee \forall \bar{Z}_D\, D'\theta_i) \wedge c'\right)$

Fig. 1 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)	$\forall \bar{Z}(t \notin \{s_1 \mid s\}) \wedge c'$	$\mapsto$	$\forall \bar{Z}(t \neq s_1) \wedge \forall \bar{Z}(t \notin s) \wedge c'$
(5)	$\forall \bar{Z}(t \notin f(t_1, \ldots, t_n) \vee D) \wedge c'$	$\mapsto$	c'
(6)	$\forall \bar{Z}\,(t \notin X \vee D) \wedge c'$	$\mapsto$	$\forall \bar{Z} N\,(X \neq \{t \mid N\} \vee D) \wedge c'$
(7)	$\forall \bar{Z}(t \notin \{s_1 \mid s\} \vee D) \wedge c'$	$\mapsto$	$\forall \bar{Z}\,(t \neq s_1 \vee D) \wedge \forall \bar{Z}\,(t \notin s \vee D) \wedge c'$

Fig. 2 Rewriting function for $\notin$ (the notational conventions are those of Figure 1)

where t contains a set term of the form $\{t_1, \ldots, t_n \mid V\}$, with V variable. This is a direct consequence of the property stated by Theorem 5.1. The problem may arise, for instance, with the formulae dealt with by rule (6) of notin_Set, which are always rewritten to $\forall \bar{Z} N\,(X \neq \{t \mid N\} \vee D)$, as well as with the formulae dealt with by rule ($7''$) of notequal_Set, whenever n is greater than 1.

Furthermore, we have assumed that input formulae of the form

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

are not dealt with by $SAT_{\mathcal{SET}}$ in the general case but only in the three subcases treated by rule (8) of notequal_Set. Once again, this is done to avoid non-termination of the rewriting process.

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, independently from 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 syntactic characterizations of various classes of $CLP(\mathcal{SET})$ programs for which the constraint satisfiability procedures presented in Section 6 are guaranteed to be well-defined and terminating for every input formula.

7.1 Bounded Constraints

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 $\{t_0, \ldots, t_n \,|\, V\}$, 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 $\forall \bar{Z}\,(\{s_0, \ldots, s_m \,|\, s\} \neq \{t_0, \ldots, t_n \,|\, t\})$, $\bar{Z}$ 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.

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 $\{t_0, \ldots, t_n \,|\, X\}$, X variable, then we could avoid all the undefined situations for $SAT_{\mathcal{SET}}$. 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 7.1
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 $\{t_0, \ldots, t_n \,|\, X\}$, X variable.[*5]

Lemma 7.1
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.

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

- 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 $\{t_0, \ldots, t_n \mid X\}$, 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).

$\blacksquare$

7.2 Allowed and r-Allowed Programs

The bounded condition is exactly what is needed to ensure that the rewriting procedures developed for implementing constructive negation produce always a result. In this section we identify syntactic conditions that guarantees that unbounded constraints can not be generated.

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 $H :- B_1, \ldots, B_n$ is said to be *allowed* in the standard case [1)] if every variable X appearing in the clause appears in at least one positive literal in the body of the clause. When a constraint c is in solved form and different from `false` (cf. Def. 6.2), it is always of the form $X_1 = t_1 \wedge \cdots \wedge X_n = t_n \wedge c'$ where c' does not contain equations and $\{X_1, \ldots, X_n\} \cap \mathit{vars}(c') = \emptyset$. The first part of a constraint identifies a substitution $\theta = [X_1/t_1, \ldots, X_n/t_n]$; in this case we refer to $X_1 = t_1 \wedge \cdots \wedge X_n = t_n$ as $\hat{\theta}$.

Definition 7.2

Given a program P, we build the program `norm`(P) as follows: for every clause

$$A :- c \mid \bar{B}$$

in P, add the clauses

$$(A :- c_1 \mid \bar{B})\theta_1, \ldots, (A :- c_k \mid \bar{B})\theta_k$$

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 $SAT_{\mathcal{SET}}(c)$, where $\hat{\theta}_i$ represents the equational part and c_i represents the negative constraint part of the i-th constraint.

Definition 7.3
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 $:\!-\, c \,|\, \bar{B}$ is allowed if $\mathtt{norm}(p(X_1, \ldots, X_n) :\!-\, c \,|\, \bar{B})$ is allowed in the standard sense, where $\{X_1, \ldots, X_n\} = \mathit{vars}(c \,|\, \bar{B})$, and p is a new predicate symbol.

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

Example 7.1
Consider the $CLP(\mathcal{SET})$ clause

$$\mathtt{p(X,Y)} :\!-\, \{\mathtt{X,Y}\} = \{\mathtt{V} \,|\, \mathtt{Z}\},\, \mathtt{V} \notin \mathtt{Z}, \mathtt{q(V)}, \neg\mathtt{r(Z)}\,. \qquad (8)$$

$SAT_{\mathcal{SET}}(\{X, Y\} = \{V \,|\, Z\} \wedge V \notin Z)$ 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:

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

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 clause (8) is not allowed.

Corollary 7.1
Let P a $CLP(\mathcal{SET})$ program with negation, G a goal, and let us assume that P and G are allowed. Then any computed answer for G with respect to P does

not contain any unbound set term.

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 unbound constraints occur during the resolution process, allowedness ensures that every variable generating such situations will be eventually instantiated to a ground term in all terminating derivations. This ensures that such undesired constraints eventually disappear: the proof follows from Lemma 7.1.

Remark 7.1
Note that there are allowed programs that are not bounded and vice versa. For instance, the clause $\mathtt{q(X,Z)} :- \mathtt{X} \notin \mathtt{Z}, \mathtt{p(X,Z)}$ is allowed and unbound, whereas the clause $\mathtt{q(X,Z)} :- \mathtt{X} \neq \mathtt{Z}$ is bounded but not allowed.

It is evident that the class of allowed programs is too narrow for our purposes. As a matter of fact, for this class of programs one could use Negation as Failure 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 specialized notion of allowedness for $CLP(\mathcal{SET})$ programs.

Hereafter, we assume that all the programs and goals used have been previously normalized using the procedure described in Definition 7.2.

Definition 7.4

- A goal $\bar{B}$ is *r-allowed* ("r" stands for restricted) if, for every variable X that occurs in a term $\{t_0, \ldots, t_n \,|\, X\}$ or in a constraint $t \in X$ or $t \notin X$ 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 $H :- \bar{B}$ 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.

Example 7.2
Consider the program

$$\begin{array}{l} \mathtt{p(\{A\,|\,R\})} :- \neg\mathtt{q(A)}, \mathtt{r(R)}.\\ \mathtt{r(\{a\})}.\\ \mathtt{r(\{b\})}.\\ \mathtt{q(a)}. \end{array}$$

This program is r-allowed, but it is not allowed since the variable A in the first clause does not occur in a positive literal of the body. The rewriting procedures of the previous section allow us to compute the answers:

$$\begin{array}{l} \mathtt{X} = \{\mathtt{A}, \mathtt{a}\}, \mathtt{A} \neq \mathtt{a}\\ \mathtt{X} = \{\mathtt{A}, \mathtt{b}\}, \mathtt{A} \neq \mathtt{a} \end{array}$$

to the non-ground goal $:- \mathtt{p(X)}$. Observe that no answer can be computed using Negation as Failure, since this technique cannot handle subgoals made only by non-ground negative literals (floundering [1]).

Lemma 7.2
Let P be an r-allowed program and G an r-allowed goal. Then, any computed answer for G with respect to P does not contain any term of the form $\{t_0, \ldots, t_n \,|\, X\}$, where X is a variable.

Proof The proof follows immediately from the definition of allowedness for $CLP(\mathcal{SET})$ programs and from Lemma 7.1. ■

7.3 $CLP(\mathcal{SET})$ Programs with Set Grouping

As mentioned earlier, one of the main goals behind the introduction of negation in $CLP(\mathcal{SET})$ is the support for intensional set formers. According to [8], intensional set formers can be introduced in $CLP(\mathcal{SET})$ as simple syntactic extensions which are automatically replaced by suitable $CLP(\mathcal{SET})$ definitions which implement the needed set grouping facilities. As shown in [8], set grouping requires the ability to deal with negation in clause bodies.

When restricting to this particular usage of negation, however, we can focus our attention to those $CLP(\mathcal{SET})$ programs that are obtained by eliminating all the intensional set formers that possibly occur in them. Accordingly, we can replace the notions of allowedness and r-allowedness with a more specific notion of *set-allowedness*, which allows us to significantly enlarge the class of admissible programs, so as to encompass a large class of $CLP(\mathcal{SET})$ programs which make use of intensional set formers.

Let us precisely define the notion of intensional set formers and show the rewriting technique for eliminating them from $CLP(\mathcal{SET})$ programs. Then we will define the set-allowedness property.

Definition 7.5
An *intensional set former* is a syntactic object of the kind:

$$\{X \,:\, \exists \bar{Z}\, G[X, \bar{Y}, \bar{Z}]\}$$

where G is a $CLP(\mathcal{SET})$-goal. Intensional set formers can be used in $CLP(\mathcal{SET})$ programs and goals wherever ordinary terms can.

The intended semantics of an intensional set former $S = \{X \,:\, \varphi\}$ is expressed by:

$$\forall X (X \in S \leftrightarrow \varphi)\,.$$

or, equivalently, by

$$\forall X\,(X \in S \rightarrow \varphi) \quad \wedge \quad \neg\exists X\,(X \notin S \wedge \varphi)\,. \tag{9}$$

Computing the set S requires the ability to perform *set grouping*, that is to collect in a set S all the values of X satisfying the property φ. This mechanism can be easily implemented in $CLP(\mathcal{SET})$ with negation. In fact, the first conjunct of formula (9) can be handled in $CLP(\mathcal{SET})$ through the use of recursion (or through the use of *Restricted Universal Quantifiers* [17]), while the second conjunct can be easily expressed in $CLP(\mathcal{SET})$ but using negation.

The following algorithm eliminates intensional set formers introducing new $CLP(\mathcal{SET})$ clauses (and one negation for each intensional set former removed).

Elimination of intensional set formers

- Without loss of generality, we can assume that all intensional set formers occur in constraints of the form $\mathbb{I} = \{X \,:\, \exists \bar{Z}\, G[X, \bar{Y}, \bar{Z}]\}$ (this result can easily be achieved by introducing new variables).
- Replace each atomic constraint of the form $\mathbb{I} = \{X \,:\, \exists \bar{Z}\, G[X, \bar{Y}, \bar{Z}]\}$ with the constraint $\mathbb{I} = \{X \,:\, \delta(X, \bar{Y})\}$ where δ is a newly generated predicate symbol (one for each intensional set former), and add to the program the new clause (14) of Figure 3.

$$\begin{array}{rcll}
\mathtt{setof}_\delta(I,\bar{Y}) & :- & \mathtt{subsetof}_\delta(I,\bar{Y}), \neg\mathtt{partial}_\delta(I,\bar{Y}). & (10)\\
\mathtt{subsetof}_\delta(\emptyset,\bar{Y}). & & & (11)\\
\mathtt{subsetof}_\delta(\{X \mid S\},\bar{Y}) & :- & X \notin S, \delta(X,\bar{Y}), \mathtt{subsetof}_\delta(S,\bar{Y}). & (12)\\
\mathtt{partial}_\delta(I,\bar{Y}) & :- & V \notin I, \delta(V,\bar{Y}). & (13)\\
\delta(X,\bar{Y}) & :- & G. & (14)
\end{array}$$

Fig. 3 Set grouping clauses

- For each predicate symbol δ generated by the previous normalization step, generate the new predicate symbols setof_δ, $\mathtt{subsetof}_\delta$, and $\mathsf{partial}_\delta$, and introduce their definitions in the program as in Figure 3 (clauses (10)—(13)) and replace the constraint $\mathbb{I} = \{X : \delta(X,\bar{Y})\}$ by the atom $\mathtt{setof}_\delta(\mathbb{I},\bar{Y})$.

A $CLP(\mathcal{SET})$ *program with set grouping* is a $CLP(\mathcal{SET})$ program with negation obtained by eliminating intensional set formers from it, according to the above transformation rules. Note that in a $CLP(\mathcal{SET})$ program with set grouping, negation is allowed to occur only in *set grouping clauses*, namely in clauses defining $\mathtt{setof}_\delta$ predicates.

Example 7.3
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.

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

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

$$
\begin{array}{l}
\texttt{possibly\_lovers}(\texttt{I},\texttt{Y}) \mathrel{:-} \texttt{setof}_\delta(\texttt{I},\texttt{Y}).\\
\texttt{setof}_\delta(\texttt{I},\texttt{Y}) \mathrel{:-}\\
\quad \texttt{subsetof}_\delta(\texttt{I},\texttt{Y}), \neg\texttt{partial}_\delta(\texttt{I},\texttt{Y}).\\
\texttt{subsetof}_\delta(\emptyset,\texttt{Y}).\\
\texttt{subsetof}_\delta(\{\texttt{X} \mid \texttt{S}\},\texttt{Y}) \mathrel{:-}\\
\quad \texttt{X} \notin \texttt{S}, \delta(\texttt{X},\texttt{Y}), \texttt{subsetof}_\delta(\texttt{S},\texttt{Y}).\\
\texttt{partial}_\delta(\texttt{I},\texttt{Y}) \mathrel{:-}\\
\quad \texttt{V} \notin \texttt{I}, \delta(\texttt{V},\texttt{Y}).\\
\delta(\texttt{X},\texttt{Y}) \mathrel{:-}\\
\quad \texttt{X} \neq \texttt{Y}, \texttt{likes}(\texttt{X},\texttt{Y}).
\end{array}
$$

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 (finite) 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 at `www.math.unipr.it/~gianfr/setlog.Home.html/interpreter`.

Unfortunately, a $CLP(\mathcal{SET})$ program with set grouping turns out to be neither allowed nor r-allowed. Clause (13) of Figure 3, for instance, is clearly neither allowed nor r-allowed due to the presence of the constraint $V \notin I$ and the fact that I does not occur in a positive literal of the body. Therefore we introduce a special notion of allowedness—called set-allowedness—that, taking into account the particular usage of negation which is performed within set grouping clauses, allows a large class of $CLP(\mathcal{SET})$ programs with set grouping to be considered as admissible programs.

Definition 7.6
Let P be a $CLP(\mathcal{SET})$ program with set grouping. Assume that P is normalized (i.e., $\texttt{norm}(P) = P$).

- A clause of P defining a predicate p different from $\texttt{setof}_\delta, \texttt{subsetof}_\delta$, and $\texttt{partial}_\delta$, is *set-allowed* if it is r-allowed.
- A clause of P defining a predicate $\texttt{setof}_\delta$, $\texttt{subsetof}_\delta$, or $\texttt{partial}_\delta$, is *set-*

allowed if all the clauses used to define δ (at all levels) are set-allowed.

- P is *set-allowed* if all its clauses are set-allowed.

Without loss of generality, we can also assume that calls to predicates $\mathtt{setof}_\delta(\mathtt{S}, \bar{\mathtt{Y}})$ in the translated program always occur with $\mathtt{S}$ and $\bar{\mathtt{Y}}$ variables.

Theorem 7.1
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 rule that selects first $\mathtt{subsetof}_\delta$ atoms is adopted.

Proof We can start observing that, even if the clauses for $\mathtt{partial}_\delta$ (clauses (11) and (12) of Figure 3) are only set-allowed (instead of r-allowed), they cannot generate any $\{t_0, \ldots, t_n \,|\, X\}$ term. In fact, what prevents these clauses to be r-allowed is only the variable S in the second clause of $\mathtt{subsetof}_\delta$ (observe that the first clause for $\mathtt{partial}_\delta$ is not allowed—unless $\bar{Y}$ is empty). By the definition of $\mathtt{subsetof}_\delta$ itself, however, it is evident that S will be always instantiated to a set term with no rest variable.

According to Lemma 7.2, since all the clauses for all the δ predicates are set-allowed by hypothesis, they cannot generate any $\{t_0, \ldots, t_n \,|\, X\}$.

On the other hand, the hypothesis that in any call to $\mathtt{setof}_\delta(\mathtt{I}, \bar{\mathtt{Y}})$ $\mathtt{I}$ and $\bar{\mathtt{Y}}$ are variables, along with the additional hypothesis on the selection rule, ensure that no term $\{t_0, \ldots, t_n \,|\, X\}$ can be passed to $\mathtt{setof}_\delta$ predicates (and, hence, to the $\mathtt{partial}_\delta$ predicates) from outside, i.e., as a parameter of the atom $\mathtt{setof}_\delta(\mathtt{I}, \bar{\mathtt{Y}})$. Moreover, by the hypothesis on the selection rule, when $\mathtt{partial}_\delta$ is called, its first parameter $\mathtt{I}$ is always instantiated to a term of the form:

$$\emptyset, \{s_0\}, \{s_0, s_1\}, \ldots$$

This ensures that the constraint $V \notin I$, that could generate undefined situations, is simply rewritten as:

$$\mathtt{true}, V \neq s_0, V \neq s_0 \wedge V \neq s_1, \ldots$$

Thus, no undefined situations are generated even if the original program is unbound. Hence, the thesis follows immediately from Lemma 7.1. ■

Example 7.4
Consider the program of Example 7.3. First of all, notice that this program is set allowed if the clauses defining `likes` are r-allowed. Let us assume that the

program of Example 7.3 is augmented with the following clauses:

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

The second clause above is not allowed, but it is r-allowed. Thus, the whole program is set-allowed. The computed answers for the goal

$$:- \texttt{possibly\_lovers}(\texttt{S}, \texttt{Y}).$$

are:

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

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

$$\mathit{Any} \neq \mathit{john} \wedge \neg \mathit{partial}_\delta(\{\mathit{john}\}, \mathit{Any}).$$

Since the computed answer for the positive version is $\mathit{Any} = \mathit{ann}$ we get, by constructive negation, the answer $\mathit{Any} \neq \mathit{ann}$ 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 $\{t_0, \ldots, t_n \,|\, X\}$, 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.

7.4 Programming Examples

Let us consider a directed graph $\langle V, E \rangle$ represented by a collection of clauses of the type

$$\texttt{edge}(\alpha,\beta).$$

for each edge from α to β in the graph, and by a clause

$$\texttt{nodes}(V).$$

where V is the set of nodes in the graph.

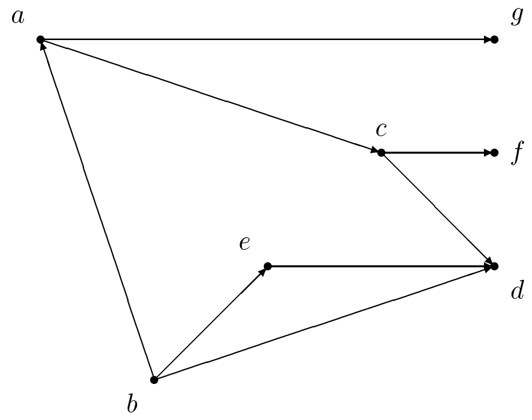

Fig. 4 Example Graph

The problem of computing the set S of nodes which are reachable from a given node Y can be expressed using an intensional set former as follows:

$$\texttt{reach}(\texttt{Y},\texttt{S}) :- \texttt{S} = \{\texttt{X} : \texttt{connected}(\texttt{Y},\texttt{X})\}.$$
$$\texttt{connected}(\texttt{A},\texttt{B}) :- \texttt{edge}(\texttt{A},\texttt{B}).$$
$$\texttt{connected}(\texttt{A},\texttt{B}) :- \texttt{edge}(\texttt{A},\texttt{C}), \texttt{connected}(\texttt{C},\texttt{B}).$$

The translation of this program produces:

$$\texttt{reach}(\texttt{Y},\texttt{S}) :- \texttt{setof}_\delta(\texttt{S},\texttt{Y}).$$
$$\delta(\texttt{X},\texttt{Y}) :- \texttt{connected}(\texttt{Y},\texttt{X}).$$

plus the standard clauses (10)—(13) of Figure 3, where $\bar{Y}$ is simply Y. The resulting program is set-allowed since the definition of δ is r-allowed.

Consider the graph of Figure 4.

- A goal of the type $:-\texttt{reach}(a, S)$ will be successfully executed and produce the expected solution $S = \{c, d, f, g\}$ (note that the first call to $\texttt{partial}_\delta$, $\neg\texttt{partial}_\delta(\emptyset, Y)$, cannot be executed safely using NAF because Y is non-ground).
- A goal of the type $:- \texttt{reach}(Y, \emptyset)$ will be also handled properly. The $\texttt{subsetof}_\delta$ goal will succeed for any value in the second argument, thanks to the first clause of $\texttt{subsetof}_\delta$. The $\neg\texttt{partial}_\delta(\emptyset, Y)$ will succeed only for those values of Y which are nodes of the graph which have no outgoing

edges. Therefore, the following answer will be produced:

$$\mathtt{Y} \neq \mathtt{a}, \mathtt{Y} \neq \mathtt{b}, \mathtt{Y} \neq \mathtt{c}, \mathtt{Y} \neq \mathtt{e}.$$

As another example, we show a case where negation is allowed to occur also outside set grouping clauses. Consider an undirected graph where each edge is represented as a set of two nodes. The following predicate

$$\mathtt{intersect}(\mathtt{E}_1, \mathtt{E}_2) :- \mathtt{E}_1 = \{\mathtt{X}, \mathtt{Y}\}, \mathtt{E}_2 = \{\mathtt{X}, \mathtt{Z}\}, \mathtt{Y} \neq \mathtt{Z}.$$

expresses the fact that the two distinct edges E_1, E_2 have one node in common. Observe that this program is r-allowed.

Let us consider the goal

$$:- \neg\mathtt{intersect}(\mathtt{E}_1, \mathtt{E}_2).$$

where $\mathtt{E}_1$ and $\mathtt{E}_2$ are unbound variables. The constructive negation procedure first computes the answer for the corresponding positive goal, that is

$$\exists XYZ(E_1 = \{X, Y\} \wedge E_2 = \{X, Z\} \wedge Y \neq Z).$$

The negation of this formula will produce

$$\forall XYZ(E_1 \neq \{X, Y\} \vee E_2 \neq \{X, Z\} \vee Y = Z).$$

This is a good example of a situation in which the rewriting of Lemma 2.1 would not provide the correct answer. After appropriate simplifications, the outcome of a constructive negation computation is composed by the following disjunction:

$$\forall XY\,(E_1 \neq \{X, Y\}) \;\vee \qquad (15)$$

$$\exists XY\,(E_1 = \{X, Y\} \wedge \forall Z\,(E_2 \neq \{X, Z\}) \wedge \forall Z\,(E_2 \neq \{Y, Z\})) \;\vee \qquad (16)$$

$$\exists XY\,(E_1 = \{X, Y\} \wedge E_2 = \{X, Y\}) \qquad (17)$$

Intuitively, (15) states that E_1 is not an edge. (16) states that E_1 is an edge and we have that either E_2 is not an edge or, if it is an edge, it has no nodes in common with E_1. Observe that two universally quantified formulae are necessary to state this property: they are generated by the rewriting technique of Lemma 4.1. Case (17) states that E_1 and E_2 are the same edge.

For instance, a goal like

$$:- \neg\mathtt{intersect}(\mathtt{E}_1, \mathtt{E}_2), \mathtt{edge}(\mathtt{E}_1), \mathtt{edge}(\mathtt{E}_2).$$

where $\mathtt{edge}(\{\alpha,\beta\})$ is true if there is an undirected edge between α and β, can be solved safely using constructive negation, disregarding the order in which the literals occurring in the goal are selected.

Finally, observe that the composition of intensional sets—i.e., the use of intensional set formers in the construction of other intensional set formers—can potentially create problems. These arise from the fact that the notion of set-allowedness presented earlier requires the "body" of an intensional set former to be r-allowed. And this becomes a problem when such body includes the definition of another intensional set former. In most cases, however, there are simple ways of avoiding this problem. Providing a *domain predicate* in the intensional set definition (in a fashion similar to the use of domain predicates in non-monotonic reasoning [32]) allows us to restrict variables that otherwise would violate the allowedness conditions. For example we can construct a set which contains pairs of the type $\langle X, S\rangle$ where S is the non-empty set of nodes reachable from X:

$$\texttt{reachpairs(T)} :- \texttt{T} = \{\texttt{A} : \exists \texttt{X}, \texttt{S}\,(\texttt{A} = (\texttt{X}, \texttt{S}), \texttt{S} \neq \emptyset, \texttt{reach(S,X)})\}.$$

and modify the definition of `reach` as

$$\texttt{reach(S,X)} :- \texttt{nodes(V)}, \texttt{S} = \{\texttt{A} : \texttt{X} \in \texttt{V}, \texttt{connected(A,X)}\}.$$

It is easy to see that by using the condition $\texttt{X} \in \texttt{V}$ throughout the construction of the intensional set the allowedness property can be maintained.

§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.[*6] To this purpose, we have precisely identified syntactic conditions for programs that ensure that NAF can be used safely. When translating intensional set formers 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

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

[*6] A similar approach is adopted in [30] where NAF, CN, and also Intensional Negation, are integrated into a single Prolog compiler.

with the goal $:-\ \mathtt{S} = \{\mathtt{X} : \mathtt{p}(\mathtt{X},\mathtt{Y})\}$, which is translated into a $CLP(\mathcal{SET})$ program which is not set-allowed—and, thus, non-allowed—can instead be dealt with safely using NAF (except for the case of the empty set), provided $\mathtt{subsetof}_\delta$ within set grouping clauses is executed before $\neg\mathtt{partial}_\delta$. Also note that whenever an intensional set former 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 involving set grouping, to determine, through the use of abstract interpretation techniques [9)], properties like groundness [14, 31)], non-failure [15)], 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 former denotes the empty set—e.g., as in Example 1.1. In fact, whenever the execution of $\mathtt{subsetof}_\delta(\mathtt{S},\mathtt{Y})$ instantiates $\mathtt{S}$ to the empty set, $\mathtt{Y}$ is left uninstantiated; in turn, this may cause a non-safe use of the negation rule for $\neg\mathtt{partial}_\delta(\mathtt{S},\mathtt{Y})$.

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 $\{t_1,\ldots,t_n \,|\, X\}$ do not necessarily imply an undefined situation for $SAT_{\mathcal{SET}}$. We could easily add simple specialized rules to the constraint satisfiability functions presented in Section 6, in order to guarantee termination of the rewriting process for these special cases. In particular,

- if the input formula c has the form

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

  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

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

  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 $\{t_1,\ldots,t_n \,|\, X\}$, 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 of gaining efficiency during set grouping.

Regarding 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 former. Therefore, better performance results for set grouping could be obtained by combining the two all-solutions collection operations. This can be achieved 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 Prolog built-in predicate `setof/3`, suitably adapted to work with $CLP(\mathcal{SET})$ programs. Note that the list `L` of computed answers for `p(X,Y)` collected by `setof(X,p(X,Y),L)` is just an "approximation" of the set $\{X : p(X, Y)\}$. For instance, if `p` is defined by the two clauses

```
p(a).                  p(f(X)).
```

the result of `setof` is `L = [a,f(X)]` whereas the intensional set former actually denotes the infinite set $\{a, f(\alpha_1), f(\alpha_2), \ldots\}$ with the constraints $\alpha_1 \neq X_2, \ldots$ Moreover `setof` is unable to deal with the case of empty set. Given the set AS, we can use it:

- to compute all the solutions for the variables that possibly occur in the goal $\texttt{partial}_p(S, \bar{Y})$, 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 $\texttt{subsetof}_p(S, \bar{Y})$.

More in detail, for each SS_i the set of solutions for $\texttt{partial}_p(SS_i, \bar{Y})$ (see the definition of $\texttt{partial}_\delta$ in Example 7.3) is computed as follows

1. remove 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 $X)$ occurs in SS_i;
2. use 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

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 prevents the application of the existing results presented in the literature [37].

The 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 proposed 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 restricted framework, CN strictly subsumes NAF.

The results have been developed in the context of the $CLP(\mathcal{SET})$ framework, an instance of the constraint logic programming framework for managing hereditarily hybrid finite sets. Nevertheless, the results and techniques are sufficiently general to provide foundations for the development of similar studies in the context of other instances of CLP for complex structures.

Acknowledgment The research presented in this paper has benefited from discussions with P. Bruscoli, E. G. Omodeo, C. Piazza, 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 Grants CCR-9875279, CCR-9900320, CCR-982052, EIA-9810732, and HRD-9906130. A. Dovier and G. Rossi are partially supported by the MURST project *Certificazione automatica di programmi mediante interpretazione astratta.*

References

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

2) Arenas-Sánchez, P., and Dovier, A. A Minimality Study for Set Unification. *Journal of Functional and Logic Programming,* 7:1–49, 1997.

3) Arni, N., Greco, S., and Saccà, D. Matching of Bounded Set Terms in the Logic Language LDL++. *Journal of Logic Programming,* 27(1):73–87, 1996.

4) Baader, F., and Schulz, K.U. Unification Theory. In W. Bibel and P.H.

Schmidt, editors, *Automated Deduction - A Basis for Applications, Vol. I: Foundations - Calculi and Methods*, Vol. 8 of Applied Logic Series, pages 225-263. Kluwer Academic Publishers, 1998.

5) BARBUTI, R., MANCARELLA, P., PEDRESCHI, D., AND TURINI, F. A Transformation Approach to Negation in Logic Programming. *Journal of Logic Programming,* 8:201–228, 1990.

6) BEERI, C., NAQVI, S., SHMUELI, O., AND TSUR., S. Set Constructors in a Logic Database Language. *Journal of Logic Programming,* 10(3):181–232, 1991.

7) BERNAYS, P. A system of axiomatic set theory. Part I. *Journal of Symbolic Logic*, 2:65–77, 1937.

8) 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., pp. 647–661.

9) BRUYNOOGHE, M. A Framework for the Abstract Interpretation of Logic Programs. *Journal of Logic Programming*, 10(1–4):91-124, 1991.

10) CANTONE, D., FERRO, A., AND OMODEO., E. G. *Computable Set Theory, Vol. 1.* International Series of Monographs on Computer Science. Clarendon Press, Oxford, 1989.

11) 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., pp. 111–125.

12) 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., pp. 477–493.

13) CLARK, K. L. Negation as Failure. In *Logic and Databases*, H. Gallaire and J. Minker, Eds. Plenum Press, 1978, pp. 293–321.

14) 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, pp. 52–61.

15) 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, pp. 48–62.

16) 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., pp. 540–556.

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):1–44, 1996.

18) DOVIER, A., POLICRITI, A., AND ROSSI, G. A Uniform Axiomatic View of Lists, Multisets, and Sets, and the Relevant Unification Algorithms. *Fundamenta Informaticae,* 36(2/3):201–234, 1998.

19) Dovier, A., Pontelli, E., and Rossi, G. A Necessary Condition for Constructive Negation in Constraint Logic Programming. *Information Processing Letters*, To appear.

20) Drabent, W. What is Failure? An Approach to Constructive Negation. *Acta Informatica,* 32(1):27–29, 1995.

21) 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, pp. 68–89.

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

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

24) Lloyd, J. *Foundations of Logic Programming.* Springer Verlag, 1986.

25) Kunen, K. Negation in logic programming. *Journal of Logic Programming*, 4:289–308, 1987.

26) Lassez, J. L., Maher, M. J., and Marriot, K. Unification revisited. In J. Minker, ed., *Foundations of Deductive Databases and Logic Programming*, pp. 587–625, Morgan Kaufmann, 1988.

27) Maher, M. J. Complete Axiomatizations of the Algebras of Finite, Rational, and Infinite Trees. In *Proceedings of Third Symposium Logic in Computer Science,* 1988, pp. 349–357.

28) Mal'cev, A. Axiomatizable Classes of Locally Free Algebras of Various Types. In *The Metamathematics of Algebraic Systems*, North Holland, 1971, ch. 23.

29) Martelli, A., and Montanari, U. An Efficient Unification Algorithm. *ACM Transactions on Programming Languages and Systems* 4:258–282, 1982.

30) Moreno-Navarro J. J., and Muñoz-Hernández S. How to Incorporate Negation in a Prolog Compiler. *Proceedings International Workshop on Practical Aspects of Declarative Languages*, E. Pontelli and V. Santos Costa, Eds., vol. 1753 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 2000, pp.124–139.

31) 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, 1991, pp. 49–63.

32) Niemela, I. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. In *Annals of Mathematics and AI*, (to appear).

33) Özsoyoglu,G., Özsoyoglu, Z., and Matos, V. Extending Relational Algebra and Relational Calculus with Set Valued Attributes and Aggregate Functions. In *Transactions on Database Systems*, 12(4):566–592, 1987.

34) Przymusinski, T. On Constructive Negation in Logic Programming. *Proceedings of North American Conference on Logic Programming*, MIT Press, 1990.

35) Shepherdson, J. C. Negation in Logic Programming. In J. Minker ed., *Foundations of Deductive Databases and Logic Programming*, pp. 19–88, Morgan Kaufmann, 1988.

36) Siekmann, J. H. Unification Theory. In C. Kirchner, editor, *Unification*. Academic Press, 1990.

37) Stuckey, P. J. Negation and Constraint Logic Programming. *Information and Computation,* 1:12–33, 1995.

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

§1 Appendix A: Constraint Satisfiability Test in $CLP(\mathcal{FT})$

Definition 1.1

A constraint c is in *solved form* if it is either the constant `false` or a conjunction of admissible 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 $t \notin \bar{Z}$.[*7]

Assuming that the signature Σ has an infinite number of function symbols (as, for instance, Chan implicitly does [11]), 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

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

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

$$\{V_1, \ldots, V_k\} = vars(\forall \bar{Z}_1 \, (Y_1 \neq s_1) \wedge \cdots \wedge \forall \bar{Z}_n \, (Y_n \neq s_n)),$$

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 situation is different if the signature is finite. Assume, for instance, that Σ contains only the constant symbol a and the unary function symbol f

[*7] 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.

(besides =). Then the constraint

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

is unsatisfiable.

The $SAT_{\mathcal{FT}}$ procedure uses both the standard *unification* algorithm unify (see, e.g., [29, 26]), and a negative constraint manager, here named notequal_FT, whose complete definition is shown in Figure 5.

Given a disjunction of CN constraints $\vee_{i=1}^{h} c_i$, $SAT_{\mathcal{FT}}$ first applies unify to each disjunct c_i, obtaining for each of them either `false` or a (unique) constraint d_i'. Note that unify does not generate any new $\neq$-constraint and leaves untouched the formulae that are not based on =-constraints, with the exception of the variable substitution rule which is applied to the whole input formula. 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 $\vee_{j=1}^{k} c_j'$, 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 1.1
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 2.1 for rule (8). ■

§2 Appendix B: Constraint Simplification Process in $CLP(\mathcal{FT})$

Let C be a disjunction of CN constraints, c, c', and D be CN 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 $\bar{X} = \bar{Z} \cap vars(s \neq t), \bar{Z}_D = \bar{Z} \setminus \bar{X}$	$\mapsto$	$(\forall \bar{X}\,(s \neq t) \wedge c')\ \vee (s = t \wedge \forall \bar{Z}_D\, D \wedge c')$

Fig. 5 Rewriting function for $\neq$ in $CLP(\mathcal{FT})$

The conjunction $\wedge_{i=1}^{k} \forall \bar{Y}_i \, \neg c_i$ is simplified via a rewriting process that transforms it into an equivalent disjunction of CN constraints $\vee_{i=1}^{h} \exists \bar{Z}_i \, d_i$ 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 $\wedge_{i=1}^{k} \forall \bar{Y}_i \, \neg c_i$ 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 $c'[Y/X]$.
   - 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*:

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

     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

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

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 $\neg c'_i$:

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

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:

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

Thus, the existentially quantified variables $\bar{Z}$ (that can be chosen so as to be distinct from variables in $X_{\alpha_1} = t_1 \wedge \cdots \wedge X_{\alpha_m} = t_m$) 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 obtain a disjunction of CN constraints $\vee_{i=1}^{h} \exists \bar{Z}_i\, d_i$.

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

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 3.1

If the signature of the language is infinite, then a solved form constraint c different from `false` admits at least one solution with respect to the structure $\langle \mathcal{S}, I^{\mathcal{S}} \rangle$.

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 $\sigma(X) = t\sigma$.

■

From the proof of the Lemma 3.1 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 3.1
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 3.1 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

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

This constraint does not admit any solution if we restrict the signature of the language to contain only $\emptyset$ and $\{\cdot \mid \cdot\}$.