Disunification in $ACI1$ Theories [*]

AGOSTINO DOVIER   CARLA PIAZZA
Università di Udine
Dip. di Matematica e Informatica
Via Le Scienze 206
33100 Udine (Italy)
dovier|piazza@dimi.uniud.it

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

Abstract

Disunification is the problem of deciding satisfiability of a system of *equations* and *disequations* with respect to a given equational theory. In this paper we study the disunification problem in the context of $ACI1$ equational theories. This problem is of great importance, for instance, for the development of constraint solvers over *sets*. Its solution opens new possibilities for developing automatic tools for static analysis and software verification. In this work we provide a characterization of the interpretation structures suitable to model the axioms in $ACI1$ theories. The satisfiability problem is solved using known techniques for the equality constraints and novel methodologies to transform disequation constraints to manageable *solved forms*. We propose four solved forms, each offering an increasingly more precise description of the set of solutions. For each solved form we provide the corresponding rewriting algorithm and we study the time complexity of the transformation. Remarkably, two of the solved forms can be computed and tested in polynomial time. All these solved forms are adequate to be used in the context of a Constraint Logic Programming system—e.g., they do not introduce universal quantifications, as instead happens in some of the existing solved forms for disunification problems.

Keywords: *Equational theories, disunification, ACI, complexity, CLP, Sets.*

1 Introduction

Equational theories are first-order theories whose axioms are universally quantified equations between first-order terms [38]. A non-empty equational theory E forces certain classes of syntactically different terms to be interpreted as the same object in any model of E. For instance, if E contains the axiom $X + Y = Y + X$ (for the sake of simplicity, we omit the universal quantification in the description of axioms), then the terms $a+b$ and $b+a$ have to be interpreted in the same way in any model of E. On the other hand, equational theories are not sufficiently strong to state that two terms *must* be distinguished in all the models of E. As a matter of fact, the 1-element structure $\underline{\mathbf{1}}$—the structure which maps all terms to the unique domain element—is a model of *any* equational theory, and in $\underline{\mathbf{1}}$ any constraint of the form $s \neq t$ is unsatisfiable. If a "wider" structure is

[*]A preliminary version of this paper, entitled *ACI1 Constraints* by A. Dovier, C. Piazza, E. Pontelli, and G. Rossi, appeared in *Danny De Schreye ed., Proceedings of the 1999 International Conference on Logic Programming. Pages 573–587, The MIT Press, 1999.*

chosen, then the satisfiability problem for a set of positive (equations) and negative (disequations) constraints—also known as the *disunification problem*—becomes meaningful and complex.

In this paper we tackle this problem in the context of equational theories describing the *Associative*, *Commutative*, and *Idempotent* (ACI) nature of a function symbol. We also deal with theories that incorporate an identity element (axiom (1)) for the ACI function symbol. The ultimate goal of our effort is to develop tools for handling $ACI1$ constraints which can be used within a *Constraint Logic Programming (CLP)* framework [27]. Constraints in the context of ACI theories—or of similar theories for set-like structures—have been shown to be very important from the theoretical as well as the practical point of view [32, 25, 24, 13], finding effective applications in a variety of areas, such as software specification [40], databases [31, 7], scheduling [25] and product configuration [39], and planning [41].

The problem of handling positive and negative constraints w.r.t. an equational theory has been explored in the literature. Fernandez [23] explored the use of narrowing strategies to simplify disunification formulae w.r.t. equational theories. Comon [15] developed a general solution to the disunification problem; however, such solution is valid only for *compact* equational theories, and $ACI1$—as discussed later—does not meet this requirement. The general problem of solving disequations with respect to a given equational theory has also been addressed by Bürckert [9]. The technique employed by Bürckert to solve the problem is that of transforming disequations into unification problems, whose sets of solutions—that, for finitary theories, can be finitely represented—are exactly the negation of the solution set of the starting problem. The answer to the satisfiability problem is represented by a pair (σ, Ψ), called *substitution with exceptions*. In this context, θ is a solution of the initial constraints if and only if θ is an instance of the substitution σ and θ is not an instance of any of the substitutions in the set Ψ. Verifying whether θ is a solution and verifying the existence of solutions belonging to (σ, Ψ) are non-trivial problems. Moreover, substitutions with exceptions are essentially solved form constraints containing universally quantified variables, which makes them unsuitable to be used in the context of a CLP system. In particular, in complex theories such as $ACI1$ they can easily lead to undecidable formulae. Baader and Schulz [5] have proposed a general technique capable of combining satisfiability algorithms for disunification in disjoint equational theories. Following this approach, once the satisfiability problems for $ACI1$ constraints involving only constant symbols (discussed in Section 4) and the constraint problem with (free) function symbols (already solved in literature [33]) have been studied separately, then it is possible to automatically obtain a satisfiability algorithm for the combined (general) case. However, the method leads to an exponential explosion of alternatives and there seems to be no practical way to obtain "partial" efficient solutions from such scheme. This is the reason why we prefer to tackle directly the combined problem (Section 5).

In this paper we present constraint solving techniques to handle $ACI1$ equation and disequation constraints in CLP languages. The presentation starts with a characterization of the structures which are suitable to model the axioms in $ACI1$—the *join-semilattices with bottom.* This is used to explore the issue of satisfiability of positive (Section 3) and negative (Section 4) constraints with respect to the different possible signatures of the language. This analysis captures the relationships between the satisfiability of negative constraints and the "shape" of the interpretation structure. The design of these structures suggests the possibility of applying the results described in the paper to the manipulation of abstract domains for static analysis and program verification. A similar approach in this direction has been described in [13], where $ACI1$ positive constraints are used for sharing analysis. Integration of these concepts in an actual abstract interpreter for constraint logic programming is reported to be in progress.

In the context of a CLP system, uninterpreted function symbols are typically manipulated as finite tree constructors [33]. We develop a first-order theory $\mathbb{T}_{ACI1}$ which extends $ACI1$ and *corresponds* to the structure having as domain $T(\Sigma)/=_{ACI1}$—i.e., the Herbrand Universe modulo the least congruence relation imposed by $ACI1$—on the class of conjunctions of positive and negative constraints. This allows us to focus on the canonical domain of Herbrand terms. In this context we present four *solved forms* for disunification problems, as well as the algorithms which allow arbitrary $ACI1$ constraints to be transformed into any of these four forms (Section 5). These solved forms meet the general requirements for *solved form constraints*—e.g., deciding their satisfiability is trivial and efficient. Furthermore, all these solved forms are adequate to be efficiently used in the context of a CLP system—a property which is absent from most solved forms proposed by other researchers [5, 9]. Two of these solved forms (called *implicit* and *intermediate*) can be obtained in polynomial time from any conjunction of disequations. Finally, we show how the results have a direct application to solve set-based constraints, taking advantage of the polynomial nature of the implicit solved form proposed. The results achieved open new possibilities in the *practical* manipulation of $ACI1$ constraints, thus overcoming limitations present in the existing CLP languages over set structures [19]. Section 6 presents various examples aimed at providing a feel for the expressive power of $ACI1$-constraints. Section 7 summarizes our results and relates them to analogous problems and results presented in the literature. A detailed complexity analysis of the problem is discussed in Section 8. Some conclusions are drawn in Section 9.

2 Preliminaries

Throughout the paper we assume the standard notation of first-order logic and constraint logic programming [22, 27]. Let us recall some commonly used terms.

A first-order *signature* Σ is a set of function and constant symbols. We denote with $ar(f)$ the arity of the function symbol f. In this paper we consider first-order signatures with a distinguished binary function symbol $\cup$ (that will be used infixed) and a distinguished constant $\emptyset$. A signature Σ is *general* if it contains at least one function symbol of arity greater than 0 different from $\cup$. We also assume the presence of a countable set $\mathcal{V}$ of *variables*. $T(\Sigma, \mathcal{V})$ ($T(\Sigma)$) denotes the set of first-order terms (resp. *ground* terms) built from Σ and $\mathcal{V}$ (resp. Σ). Given a sequence of terms $t_1, \ldots, t_n$, $vars(t_1, \ldots, t_n)$ denotes the set of all variables that occur in at least one of the terms t_i. A variable X occurs *nested* in a term t if t is of the form: $t_1 \cup (\cdots \cup (f(\cdots X \cdots) \cup (\cdots \cup t_n) \cdots)$, $f \not\equiv \cup$, and $f \in \Sigma$. Given a term or a formula t, $|t|$ denotes the number of occurrences of symbols from $\Sigma \cup \mathcal{V}$ in t.

We will follow the convention of denoting variables with upper-case characters and function symbols and constants with lower-case characters. $s \equiv t$ is used to denote the *syntactic equivalence* between the terms s and t.

A *substitution* is a function from $\mathcal{V}$ to $T(\Sigma, \mathcal{V})$ defined as the identity function except for a finite subset of $\mathcal{V}$—called the *domain* of the substitution. Substitutions can be extended to terms via structural induction [22]. The notation t^σ will be used to represent the application of a substitution σ to the term t. A substitution σ is said to be grounding for a term t if t^σ is a ground term. A substitution μ is an *instance* of a substitution θ if there is a substitution η such that for each variable X $X^\mu = (X^\theta)^\eta$.

Given $\mathcal{L} = \langle \Pi, \Sigma, \mathcal{V} \rangle$, where Π is a collection of *predicate symbols*, we denote with Adm a predefined set of first-order formulae over $\mathcal{L}$, called *admissible constraints*. For the sake of simplicity, we assume that Adm contains the two constraints true and false. We will often informally talk about a *constraint* to indicate a generic element of the class of the admissible constraints.

An $\mathcal{L}$*-structure* (or, simply, a *structure*) $\mathcal{M} = \langle M, (\_)^{\mathcal{M}} \rangle$ is composed of a non-empty set M—the *domain* of the structure—and of an interpretation function $(\_)^{\mathcal{M}}$ which assigns functions and relations on M to the symbols of $\mathcal{L}$.

A $\mathcal{M}$*-valuation* of a constraint $\mathcal{C}$ is an assignment of values from M to the free variables of $\mathcal{C}$. We will often talk simply about *valuations* when the structure $\mathcal{M}$ is clear from the context. The notion of valuation can be inductively extended to terms and formulae (as traditional practice in logic).

A valuation σ is a *successful valuation* of the formula φ if $\sigma(\varphi)$—i.e., φ evaluated according to the variables assignment described by σ—is true in the structure $\mathcal{M}$. We will also denote this fact as $\mathcal{M} \models \sigma(\varphi)$. Given a first-order theory T and a closed formula φ, $T \models \varphi$ represents the fact that for each structure $\mathcal{M}$ such that $\mathcal{M} \models T$, it holds that $\mathcal{M} \models \varphi$. Let us denote with $\vec{\exists}(\varphi)$ the existential closure of the formula φ.

Given a first-order theory T on $\mathcal{L}$ and a structure $\mathcal{M}$ for $\mathcal{L}$, T and $\mathcal{M}$ *correspond on* Adm [27] if, for each constraint $\mathcal{C} \in Adm$, we have that

$$T \models \vec{\exists}(\mathcal{C}) \textit{ iff } \mathcal{M} \models \vec{\exists}(\mathcal{C})$$

Given two structures for $\mathcal{L}$, $\mathcal{M} = \langle M, (\_)^{\mathcal{M}} \rangle$ and $\mathcal{N} = \langle N, (\_)^{\mathcal{N}} \rangle$ an *homomorphism* h from $\mathcal{M}$ to $\mathcal{N}$ is a function $h : M \longrightarrow N$ such that:

$$\forall f \in \Sigma, a_1, \dots, a_n \in M \; (h(f^{\mathcal{M}}(a_1, \dots, a_n)) = f^{\mathcal{N}}(h(a_1), \dots, h(a_n))) \tag{1}$$

and

$$\forall p \in \Pi, a_1, \dots, a_m \in M \; (p^{\mathcal{M}}(a_1, \dots, a_m) \to p^{\mathcal{N}}(h(a_1), \dots, h(a_m)))\,. \tag{2}$$

An homomorphism h is said to be an *isomorphism* if h is bijective and also the opposite of the property (2) (i.e., the implication $\leftarrow$) holds.

Given a first-order language $\mathcal{L}$ and two structures $\mathcal{M}$ and $\mathcal{N}$ over $\mathcal{L}$, an *embedding* of $\mathcal{M}$ in $\mathcal{N}$ is an isomorphism from $\mathcal{M}$ into a substructure of $\mathcal{N}$. If the only predicate symbol of $\mathcal{L}$ is the equality, an embedding of $\mathcal{M}$ in $\mathcal{N}$ can be equivalently defined as an *injective homomorphism* from $\mathcal{M}$ to $\mathcal{N}$ [37].

2.1 E-unification and E-disunification

If $s, t \in T(\Sigma, \mathcal{V})$, then $s = t$ is a Σ*-equation* and $s \neq t$ is a Σ*-disequation*. An *equational theory* is a first-order theory whose axioms are universally quantified Σ-equations.

Given an equational theory E, we can define the concept of E*-equality* ($=_E$) as the *least congruence relation* over $T(\Sigma, \mathcal{V})$ which contains E and which is closed under substitution [6]. The relation $=_E$ induces a partition of $T(\Sigma)$ into congruence classes. The set of these classes will be denoted by $T(\Sigma)/=_E$, while the congruence class containing t will be denoted by $[t]$. $T(\Sigma)/=_E$, together with a mapping (the *interpretation function*) which assigns to each term t the equivalence class $[t]$, is a model of the theory E.

Given a conjunction $\mathcal{C} \equiv (s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ of Σ-equations, the *(decision) E-unification problem* is the problem of deciding whether $E \models \vec{\exists}\mathcal{C}$. If E is an equational theory, Birkoff's completeness theorem [38] ensures that $E \models \vec{\exists}\mathcal{C}$ if and only if $T(\Sigma)/=_E \models \vec{\exists}\mathcal{C}$. From a constraint point of view, Σ-equations can be chosen as admissible constraints. The theory E and the model identified by $T(\Sigma)/=_E$ correspond on the class containing all possible conjunctions of equations [27].

A substitution σ is an *E-unifier* of two terms s, t if $s^\sigma =_E t^\sigma$—i.e., for all grounding substitutions γ for s^σ and t^σ, it holds that $E \models s^{\sigma\gamma} = t^{\sigma\gamma}$ [38].

A collection of E-unifiers for s, t is *complete* if every E-unifier of s, t can be expressed as instance of a unifier in the collection. The elements of a complete and minimal set of unifiers are usually called *most general unifiers.* The *E-unification problem* for s and t is the problem of finding a possibly minimal set of E-unifiers which represents all solutions of the equation $s = t$.

Given a conjunction $\mathcal{C} \equiv (s_1 \neq t_1 \wedge \cdots \wedge s_n \neq t_n)$ of Σ-disequations [29]—i.e., a *disequation constraint*—we call *(decision) E-disequation problem* the problem of establishing whether $E \models \vec{\exists}\mathcal{C}$. For an equational theory E, this test has always a negative answer: the structure $\underline{\mathbf{1}} = \langle \{\perp\}, (\_)^{\underline{\mathbf{1}}} \rangle$, with $(\_)^{\underline{\mathbf{1}}}$ the interpretation which maps each term to the unique element $\perp$, is a model of any equational theory, and makes each constraint of the form $s \neq t$ unsatisfiable. This problem originates from the fact that a non-empty equational theory E forces certain distinct terms to be interpreted in the same way in any model of E—however, it is not strong enough to state that two terms *must* be distinguished in each model of E. As a consequence, the disequation problem is typically stated as the problem of verifying satisfiability of $\vec{\exists}\mathcal{C}$ w.r.t. a *given interpretation structure* $\mathcal{M}$. Typically the chosen structure $\mathcal{M}$ is $T(\Sigma)/=_E$. A related problem is the determination of the structures $\mathcal{M}$ which satisfies $\mathcal{C}$.

Example 2.1 *Let E consist of the unique axiom $X = Y$ and let* Adm *contain all Σ-equations and Σ-disequations. Then E corresponds to $\underline{\mathbf{1}}$; in particular, E is a complete axiomatization of $\underline{\mathbf{1}}$.*

If E corresponds to $\underline{\mathbf{1}}$, then E is said to be *trivial.* In particular:

Proposition 2.2 *Given a non-trivial equational theory E such that the set* Adm *contains all Σ-equations and Σ-disequations, there is no structure corresponding to E.*

Proof. Assume, by contradiction, that there exists a structure $\mathcal{M}$ which corresponds to E. Since E is not trivial, $\mathcal{M} \neq \underline{\mathbf{1}}$. Thus, $|M| \geq 2$ and there are $a, b \in M$ such that $a \neq b$. This implies that $\mathcal{M} \models \exists X, Y(X \neq Y)$; but $E \not\models \exists X, Y(X \neq Y)$, since $\underline{\mathbf{1}}$ is a model of E and $\underline{\mathbf{1}} \not\models \exists X, Y(X \neq Y)$. □

A theory E is *satisfaction complete* [27] if, for each admissible constraint $\mathcal{C}$, either $E \models \vec{\exists}\mathcal{C}$ or $E \models \neg\vec{\exists}\mathcal{C}$. With respect to the concept of satisfaction completeness in presence of non-trivial theories, Proposition 2.2 leads to the following result:

Corollary 2.3 *Given a non-trivial equational theory E such that the set* Adm *contains all Σ-equations and Σ-disequations, E is not satisfaction complete.*

Satisfaction completeness forces each model to be complete w.r.t. the testing of satisfiability of the class of formulae *Adm.* The above result states that this is not possible for equational theories.

As mentioned in Section 1, our ultimate goal is to handle constraints composed of arbitrary conjunctions $\mathcal{C}$ of Σ-equations and Σ-disequations. This class of problems has been typically referred to as *E-disunification problems* [15, 9]. An *E-solution* σ of $\mathcal{C}$ in a structure $\mathcal{M}$ is a valuation $\sigma : \mathcal{V} \longrightarrow M$ such that $\mathcal{M} \models \sigma(s) = \sigma(t)$ for all $s = t$ in $\mathcal{C}$ and $\mathcal{M} \models \sigma(s) \neq \sigma(t)$ for all $s \neq t$ in $\mathcal{C}$.

The technique for handling equations and disequations presented in this paper provides a methodology to tackle disunification problems. As a matter of fact, if E is a finitary theory—i.e., every unification problem admits a finite complete collection of most general unifiers—and a complete set of unifiers can be computed, then a disunification problem $\mathcal{C}$ can be simply solved by:

1. computing a complete set of unifiers for the equations in $\mathcal{C}$, and
2. verifying whether, given any of these unifiers σ, there is a solution for the disequations of $\mathcal{C}^\sigma$ [5].

Clearly, here we are taking advantage of the fact that $ACI1$-unification problems can be not only decided, but a collection of unifiers for such problems can be explicitly computed [4].

2.2 *ACI*1 and its Models

Let us start by recalling some standard definitions from lattice theory [26]. A binary relation $\leq$ over L is a *partial order* on L if $\leq$ is reflexive, antisymmetric, and transitive. Let us denote with $\bot$ the *bottom* of the partial order—i.e., $(\forall x \in L)(\bot \leq x)$. $\langle L, \leq \rangle$ is a *join-semilattice* if the element $x \bigvee y \in L$ exists for each $x, y \in L$, where $x \bigvee y$ (x *join* y) is the unique element satisfying the properties: $x \leq x \bigvee y$, $y \leq x \bigvee y$, and for all $z \in L$ such that $x \leq z$ and $y \leq z$ it holds that $x \bigvee y \leq z$. If $\langle L, \leq \rangle$ is a partial order with bottom $\bot$, then $a \in L$ is an *atom* if $(\forall x \in L)((x \leq a \wedge x \neq a) \rightarrow x = \bot)$. Thus, if a is an atom then the only other element x such that $x \leq a$ is $\bot$.

Let $\Sigma \supseteq \{\emptyset, \cup\}$ be a signature containing the binary function symbol $\cup$ and the constant symbol $\emptyset$, and let $\mathcal{V}$ be a countable set of variables. The following equations describe the theory $ACI1$:

$$\begin{array}{lrcl}
(A) & (X \cup Y) \cup Z & = & X \cup (Y \cup Z) \\
(C) & X \cup Y & = & Y \cup X \\
(I) & X \cup X & = & X \\
(1) & \emptyset \cup X & = & X
\end{array}$$

Let us analyze the structures $\mathcal{M} = \langle M, (\_)^{\mathcal{M}} \rangle$ for $\Sigma = \{\emptyset, \cup\}$ that are models of $ACI1$. By definition of structure, the domain M cannot be empty. Let us indicate with the symbol $\bot$ the element $(\emptyset)^{\mathcal{M}} \in M$. A relation $\leq$ is induced by $(\cup)^{\mathcal{M}}$ (simply denoted by $\cup^{\mathcal{M}}$) on M:

$$x \leq y \leftrightarrow x \cup^{\mathcal{M}} y = y$$

Proposition 2.4 *Let $\mathcal{M} = \langle M, (\_)^{\mathcal{M}} \rangle$ be a model of ACI. Then $\leq$ is a partial order on M. Moreover, if $\mathcal{M}$ is a model of the axiom (1), then $\bot = (\emptyset)^{\mathcal{M}}$ is the bottom of $(M, \leq)$.*

Proof. $\leq$ is reflexive, since, by (I) $x \cup^{\mathcal{M}} x = x$ for all $x \in M$. $\leq$ is antisymmetric, since, by (C), $x \cup^{\mathcal{M}} y = y$ and $y \cup^{\mathcal{M}} x = x$ implies $x = y$.

Assume $x \cup^{\mathcal{M}} y = y$ and $y \cup^{\mathcal{M}} z = z$. Then $x \cup^{\mathcal{M}} z = x \cup^{\mathcal{M}} (y \cup^{\mathcal{M}} z)$. From axiom (A) we obtain that the latter is equal to $(x \cup^{\mathcal{M}} y) \cup^{\mathcal{M}} z = y \cup^{\mathcal{M}} z = z$, which proves the transitivity of $\leq$. Finally, by (1) $(\forall x \in M)((\emptyset)^{\mathcal{M}} \cup^{\mathcal{M}} x = x)$, i.e., $\bot \leq x$. □

It is easy to observe that if $\mathcal{M}$ is not a model of ACI, then $\leq$ is not guaranteed to be a partial order. Given a structure $\mathcal{M}$ for $\{\emptyset, \cup\}$ that is a model of $ACI1$, we denote with $\langle M, \leq \rangle$ the partial order defined above [26].

Proposition 2.5 *Let $\mathcal{M} = \langle M, (\_)^{\mathcal{M}} \rangle$ be a structure for $\{\emptyset, \cup\}$ with $\bot = (\emptyset)^{\mathcal{M}} \in M$. $\mathcal{M}$ is a model of $ACI1$ if and only if $\langle M, \leq \rangle$ is a join-semilattice with bottom $\bot$.*

Proof. Let $\mathcal{M}$ be a model of $ACI1$. Proposition 2.4 ensures that $\leq$ is a partial order on M with bottom $\bot$. Hence, $\langle M, \leq \rangle$ is a join-semilattice, where $x \bigvee y$ is defined as $x \cup^{\mathcal{M}} y$.

Let $\langle M, \leq \rangle$ be a join-semilattice with bottom $\bot$. Now $\mathcal{M} = \langle M, (\_)^{\mathcal{M}} \rangle$ can be defined as $(\emptyset)^{\mathcal{M}} = \bot \in M$ and $\forall a_1, a_2 \in M$ $(a_1 \cup^{\mathcal{M}} a_2 = a_1 \bigvee a_2)$. Observe that $a_1 \bigvee a_2$ exists since $\langle M, \leq \rangle$ is a join-semilattice. We prove that $\mathcal{M}$ is a model of $ACI1$ by using standard properties of join and the definition of bottom:

$$\begin{array}{llrcl}
(A) & \forall a_1, a_2, a_3 \in M & (a_1 \cup^{\mathcal{M}} a_2) \cup^{\mathcal{M}} a_3 & = & (a_1 \bigvee a_2) \bigvee a_3 = a_1 \bigvee (a_2 \bigvee a_3) \\
 & & & = & a_1 \cup^{\mathcal{M}} (a_2 \cup^{\mathcal{M}} a_3) \\
(C) & \forall a_1, a_2 \in M & a_1 \cup^{\mathcal{M}} a_2 & = & a_1 \bigvee a_2 = a_2 \bigvee a_1 = a_2 \cup^{\mathcal{M}} a_1 \\
(I) & \forall a_1 \in M & a_1 \cup^{\mathcal{M}} a_1 & = & a_1 \bigvee a_1 = a_1 \\
(1) & \forall a_1 \in M & \emptyset^{\mathcal{M}} \cup^{\mathcal{M}} a_1 & = & \bot \bigvee a_1 = a_1
\end{array}$$

□

2.3 Term Normalization

The $ACI1$ axioms allow us to design a normalization function $\rho : T(\Sigma, \mathcal{V}) \longrightarrow T(\Sigma, \mathcal{V})$; the objective of this function is to normalize all the occurrences of terms having $\cup$ as main functor. Intuitively, the effect of ρ is to remove repeated occurrences of terms t_i and occurrences of $\emptyset$ from terms of the form $t_1 \cup \cdots \cup t_n$, and reorder the remaining elements according to a predefined lexicographic ordering between terms—i.e., t_1 is "smaller" than t_2, t_2 is "smaller" than t_3, etc.

The normalization proceeds in a number of steps. Let us assume the presence of a predetermined ordering $<_{\#}$ on the symbols in $\Sigma \cup \mathcal{V}$ such that:

- for each $X \in \mathcal{V}$ and for each $f \in \Sigma \setminus \{\emptyset\}$ we assume that $X <_{\#} f$—i.e., all the variables precede any other symbol in the ordering
- for each $X \in \mathcal{V}$, $\emptyset <_{\#} X$
- $f <_{\#} \cup$ for each $f \in \Sigma \setminus \{\cup\}$

This ordering between symbols can be used to create a lexicographic ordering $\prec$ between all the terms in $T(\Sigma, \mathcal{V})$. In the following f and g stands for function symbols in Σ (including $\cup$) and we assume that a term with outermost symbol $\cup$ is written as $\cup(s, t)$. $t \prec t'$ iff

- $t = \emptyset$, and $t' \neq \emptyset$
- $t, t' \in \mathcal{V}$ and $t <_{\#} t'$
- $t \in \mathcal{V}$ and $t' \notin \mathcal{V}$
- $t = f(t_1, \ldots, t_n)$, $t' = g(s_1, \ldots, s_m)$ and $f <_{\#} g$
- $t = f(t_1, \ldots, t_n)$, $t' = f(s_1, \ldots, s_n)$, and there exist $1 \leq i \leq n$ such that $t_j = s_j$ for $j < i$ and $t_i \prec s_i$.

The actual normalization is realized by the procedure ρ described in Figure 1. Given a term $t \in T(\{\emptyset, \cup\}, \mathcal{V})$, then $\rho(t)$ is always of the form $\emptyset$ or $X_1 \cup \cdots \cup X_m$, where $X_1 \preceq \cdots \preceq X_m$.

Observe that the result of $\rho(t)$ is not properly a term, but the associativity of $\cup$ allows us to use this entity as a term.

The function top_sort used in the algorithm in Figure 1 is a lexicographical sorting algorithm that accepts as input lists of terms from $T(\Sigma, \mathcal{V})$, and uses $<_{\#}$ as the ordering criterion to compare basic components of the terms. $\eta(t)$ is an auxiliary function that removes all the outermost parentheses when the term t is of the form $((\cdots(t_1 \cup t_2) \cup \cdots) \cup t_h)$—again, this is possible thanks to the associative nature of the operator $\cup$.

```
function $\rho(t)$:
    if $t$ is of the form $f(t_1, \ldots, t_h)$, $h \geq 0$, $f \not\equiv \cup$
    then return $f(\rho(t_1), \ldots, \rho(t_h))$
    else begin
        $t_1 \cup \cdots \cup t_h := \eta(t)$;
        for $i := 1$ to $h$ do $t_i := \rho(t_i)$;
        $L :=$ top_sort($[t_1, \ldots, t_h]$);
        $[s_1, \ldots, s_m] :=$ remove repetitions from $L$;
        if $m > 1$ and $s_1 = \emptyset$
          then return $s_2 \cup \cdots \cup s_m$
          else return $s_1 \cup \cdots \cup s_m$
    end;
```

```
function $\eta(t)$:
    if $t$ is of the form $(t_1 \cup t_2)$
    then return $\eta(t_1) \cup \eta(t_2)$
    else return $t$;
```

Figure 1: Normalization function

Theorem 2.6 *Let $S = \Sigma \backslash \{\emptyset, \cup\}$. If $t \in T(\Sigma, \mathcal{V})$, $vars(t) = \bar{X}$, $|t| = n$, then $ACI1 \models \forall \bar{X}\ (\rho(t) = t)$ and*

1. *if S is a set of constant symbols, then $\rho(t)$ can be computed in time $O(n \log n)$,*
2. *if S contains a function symbol of arity greater than 0, then $\rho(t)$ can be computed in time $O(n^2)$.*

Proof. The soundness and completeness of the transformation rules is an immediate consequences of the axioms $ACI1$. The following observations can be drawn concerning the time complexity of the simplification process:

1. In the first case, there are no occurrences of function symbols in t. Thus, there is only one list to sort and it is made of strings of length 1. The time complexity is $O(n)$ for η, $O(n \log n)$ for the ordering, and $O(n)$ to remove repetitions; the last instruction has cost $O(1)$.
2. From [2] we know that top_sort has complexity $O(\ell_{tot} + m)$, where m is the cardinality of the alphabet and ℓ_{tot} is the sum of the lengths of the strings to sort. We can consider the alphabet composed of the set of function and constant symbols and variables occurring in t. Determining the alphabet can be achieved in time $O(n \log n)$ before calling ρ (for the sake of simplicity, we do not count parentheses and commas). Thus, $m \leq n$ and $\ell_{tot} \leq n$; the complexity of top_sort is $O(n)$.

   The time complexity $T(n)$ can be determined as follows:

$$\begin{cases} T(1) &= c & \text{when } t \text{ is a constant or a variable term} \\ T(n) &= \sum_{i=1}^{h} T(\ell_i) + c & \text{when } t \text{ is } f(t_1, \ldots, t_h),\ h > 0, \text{ and } \ell_i \text{ is the length of } t_i \\ T(n) &= \sum_{i=1}^{h} T(\ell_i) + O(n) & \text{when } \eta(t) \text{ is } t_1 \cup \cdots \cup t_h,\ h > 0, \text{ and } \ell_i \text{ is the length of } t_i \end{cases}$$

   The worst case is the last one. We prove, by induction that $T(n) = O(n^2)$.

   Base: For $n = 1$ the result is trivial.

   Step: Assume that $T(\ell) = O(\ell^2) \leq k\ell^2$, for some $k \in \mathbb{N}$, for all $\ell < n$. Since

$$\sum_{i=1}^{h} \ell_i \leq n - 1 \tag{3}$$

then we have:

$$\begin{array}{rcll} T(n) & = & \sum_{i=1}^{h} T(\ell_i) + O(n) & \text{by definition} \\ & \leq & k\sum_{i=1}^{h} (\ell_i)^2 + O(n) & \text{by inductive hypothesis} \\ & \leq & k(\sum_{i=1}^{h} \ell_i)^2 + O(n) & \text{by algebraic properties} \\ & \leq & k(n-1)^2 + O(n) & \text{by property (3)} \end{array}$$

Since we can assume k large enough to guarantee that $k(n-1)^2 + O(n) \leq kn^2$ and we have proved the bound using the same constant k of the inductive hypothesis, we have that $T(n) = O(n^2)$.

□

Observe that if Σ is completely known a priori and it is not a general signature, then a radix sort algorithm can be employed for computing ρ, leading to a time complexity of $O(n + |\Sigma|)$.

Corollary 2.7 *Let $t_1, t_2 \in T(\Sigma, \mathcal{V})$. Then $ACI1 \models \vec{\forall}(t_1 = t_2)$ if and only if $\rho(t_1) \equiv \rho(t_2)$.*

Proof. It derives from Theorem 2.6 and from the uniqueness of the results of the lexicographical ordering without repetitions. □

Given a term t of $T(\Sigma)$ we will say that t is *normalized* (or in *canonical form*) if $t \equiv \rho(t)$. $\rho(t)$ can be chosen as canonical representative of the $ACI1$-congruence class $[t]$ in $T(\Sigma)$ or $T(\Sigma, \mathcal{V})$. Observe also that ρ is an *idempotent* operation, thus $\rho(t) \equiv \rho(\rho(t))$.

Given a conjunction $\mathcal{C} \equiv (s_1\pi_1t_1 \wedge \cdots \wedge s_h\pi_ht_h)$ where $\pi_i \in \{=, \neq\}$, its *normalized form* is the formula

$$\rho(\mathcal{C}) \equiv \rho(s_1)\pi_1\rho(t_1) \wedge \cdots \wedge \rho(s_h)\pi_h\rho(t_h).$$

The worst-case time complexity for the computation of the normalized form of $\mathcal{C}$ is $O(n^2)$, where $n = |\mathcal{C}|$.

Corollary 2.8 *If $\mathcal{C} \equiv (s_1\pi_1t_1 \wedge \cdots \wedge s_h\pi_ht_h)$, where $\pi_i \in \{=, \neq\}$, then $ACI1 \models \vec{\forall}(\mathcal{C} \leftrightarrow \rho(\mathcal{C}))$.*

Proof. It follows from Corollary 2.7. □

3 *ACI*1 Equation Constraints — Unification

Given two terms s, t we are interested in the decision problem $ACI1 \models \vec{\exists}(s = t)$ and in computing a complete set of $ACI1$-unifiers. Thanks to Corollary 2.8, we can concentrate on the problem $ACI1 \models \vec{\exists}(\rho(s) = \rho(t))$. We can distinguish three classes of unification problem, according to the content of the signature Σ:

Elementary Unification: Σ contains exclusively the two symbols $\cup$ and $\emptyset$; i.e., $\Sigma = \{\emptyset, \cup\}$. The *decision* problem in this case admits always an affirmative answer—the substitution $[\,V/\emptyset \,:\, V \in vars(s,t)\,]$, which assigns $\emptyset$ to each variable, is always a unifier. $\rho(s) = \rho(t)$ is of the form:[1]

$$X_1 \cup \cdots \cup X_m = Y_1 \cup \cdots \cup Y_n$$

[1]When $m = 0$ the l.h.s. is simply $\emptyset$ and, similarly, if $n = 0$ then the r.h.s. is just $\emptyset$.

If $m = 0$ and $n = 0$, the unique most general unifier is the empty substitution ε. When $m > 0$ and $n = 0$, the unification has the unique most general solution: $[X_1/\emptyset, \ldots, X_m/\emptyset]$ (similarly when $m = 0$ and $n > 0$). The case $m > 0, n > 0$ is more complex. However, it is possible to prove that a *unique* most general unifier exists [4].

Example 3.1 *Consider the unification problem $X \cup Y = Z \cup W \cup T$. The unique most general unifier for this problem can be obtained from the following table:*

X	Y	
$A_{1,1}$	$A_{1,2}$	Z
$A_{2,1}$	$A_{2,2}$	W
$A_{3,1}$	$A_{3,2}$	T

The substitution for the left-hand side variables is obtained from the columns of the table—e.g., $[X/A_{1,1} \cup A_{2,1} \cup A_{3,1}]$—while the substitution for the right-hand side variables is obtained from the row of the table—e.g., $[W/A_{2,1} \cup A_{2,2}]$.

General methods for computing the most general unifier for this class of problems have been presented in [4].

Unification with Constants: Σ contains the two symbols $\cup$ and $\emptyset$, together with a finite collection of constants $\{c_1, \ldots, c_n\}$ distinct from $\emptyset$; i.e., $\Sigma = \{\cup, \emptyset, c_1, \ldots, c_n\}$. An equation of the type $\rho(s) = \rho(t)$ is of the form:

$$X_1 \cup \cdots \cup X_k \cup b_1 \cup \cdots \cup b_h = Y_1 \cup \cdots \cup Y_q \cup d_1 \cup \cdots \cup d_p$$

where X_i, Y_j are variables and b_i, d_j are constants. The decision problem $ACI1 \models \vec{\exists}(\rho(s) = \rho(t))$ can be solved in time $O(n)$ where $n = |\rho(s)| + |\rho(t)|$. First of all we can observe that there are structures for $ACI1$ in which all the constant symbols are mapped to different domain objects and where the union of different constants is mapped to different objects as well. Thus, there are equations that are unsatisfiable in some models. For example, the equation $\emptyset = c_1$ is unsatisfiable in any structure which maps c_1 to an object different from $\perp$. It is possible to verify satisfiability of an equation via a simple inclusion test on the set of constant symbols appearing in the two terms; thanks to the $<_\#$ ordering, this can be done in linear time.

In [4] it is shown how to compute a (minimal) complete set of most general unifiers for this problem. Uniqueness of the most general unifier is lost, due to the presence of constants, but the problem remains *finitary*—i.e., it is possible to describe the complete set of solutions through a finite number of unifiers.

General Unification: Σ contains an arbitrary collection of symbols without restrictions on their number and arity; i.e., $\Sigma = \{\cup, \emptyset, f_1, f_2, \ldots\}$.

The general unification problem $\rho(s) = \rho(t)$ has the following format:

$$X_1 \cup \cdots \cup X_h \cup s_1 \cup \cdots \cup s_k = Y_1 \cup \cdots \cup Y_p \cup t_1 \cup \cdots \cup t_q$$

where s_i, t_j are terms whose main functor is different from $\cup$.

The decision problem is NP-complete. The unification problem remains finitary [28, 21], and non-deterministic polynomial algorithms to enumerate the solutions have been proposed. However, remember that the number of solutions to the unification problem can be exponential. Algorithms to compute complete collections of unifiers for this class of problems have been presented in the literature—either as combination of simpler unification procedures [6] or as direct unification algorithms [21].

4 $ACI1$ Disequation Constraints - Disunification

In this section we will concentrate on the problem of handling conjunctions of disequations. The whole disunification problem—conjunctions of equations and disequations—can be solved in two steps. In the first step, we can apply the unification techniques mentioned in the previous section to the equations present in the constraint. In the second step, each unifier produced can be applied to the constraint, thus allowing to remove all the equations from the constraint, and leaving only the disequations $\rho(s) \neq \rho(t)$. While in the unification case each equation is satisfiable in at least one model of $ACI1$, the same does not hold in the case of disequations. A negative constraint can be unsatisfiable in all models (e.g., $\emptyset \neq \emptyset$). Other constraints (e.g., $X_1 \neq X_2$) are satisfiable in some structures and unsatisfiable in others. Thus, the study of $ACI1$ disequations requires an in-depth analysis of the possible structures for the theory considered.

4.1 Elementary $ACI1$ Disequation Constraints

Let us consider $\Sigma = \{\emptyset, \cup\}$. Given a term t, $\rho(t)$ can be either $\emptyset$ or $X_1 \cup \cdots \cup X_m$. Thus, a disequation in normalized form is described by one of the following cases:

1. $r \neq r$ (i.e., $\emptyset \neq \emptyset$ or $X_1 \cup \cdots \cup X_m \neq X_1 \cup \cdots \cup X_m$)
2. $X_1 \cup \cdots \cup X_m \neq \emptyset$, $m > 0$
3. $X_1 \cup \cdots \cup X_i \cup \cdots \cup X_m \neq Y_1 \cup \cdots \cup Y_n$, $m, n > 0$, such that there exists an X_i which does not occur among $Y_1, \ldots, Y_n$

Cases 2 or 3 can be obtained from $\rho(s) \neq \rho(t)$ by swapping the left-hand side and right-hand side of the disequation.

The first disequation is clearly false in any model of E. A disequation of the form 2 or 3 is false in $\underline{\mathbf{1}}$ but it is satisfiable in any other structure for $ACI1$. To obtain a successful valuation σ for form 2 and 3 we can simply fix an element of the domain distinct from $\perp$, say a, and put $\sigma(X_i) = a$, and $\sigma(Z) = \perp$ for all variables Z different from X_i. As far as a disequation constraint of the form $\mathcal{C} \equiv (s_1 \neq t_1 \wedge \ldots \wedge s_n \neq t_n)$ is concerned, the following theorem gives a sufficient condition for its satisfiability.

Theorem 4.1 *Let $\mathcal{C}$ be a disequation constraint in normalized form containing N primitive constraints of type 3 and K primitive constraints of type 2. Let us define $P = \varsigma(N, K)$, where $\varsigma(N, K) = N$ if $K = 0$ and $\varsigma(N, K) = N + 1$ if $K > 0$. Then $\mathcal{C}$ is satisfiable in any structure which contains a substructure isomorphic to $\langle \wp(\{a_1, \ldots, a_P\}), \subseteq \rangle$.*

Proof. Let X_{δ_i} $i = 1, \ldots, N$ be a variable occurring in the left-hand side but not in the right-hand side of the ith primitive constraint. A solution σ can be built by assigning $\{a_i\}$ to X_{δ_i} if X_{δ_i} has not been assigned yet. Observe that the elements $\{a_i\}$ are the *atoms* of the substructure isomorphic to $\langle \wp(\{a_1, \ldots, a_P\}), \subseteq \rangle$.

If there are also equations of type 2, then $\{a_{N+1}\}$ is assigned to all the remaining variables; otherwise, $\perp$ is assigned to all the remaining variables. $\square$

If, on the other hand, the structure $\mathcal{M}$ is fixed a priori, and possibly not wide enough—according to Theorem 4.1 and Corollary 4.3—then the satisfiability problem becomes NP-complete:

Theorem 4.2 *Let $\mathcal{M} = \langle M, \leq \rangle$ be a join-semilattice with bottom, $|M| = k$. Deciding the satisfiability of an $ACI1$-constraint in $\mathcal{M}$ is a NP-complete problem.*

Proof. Given the structure $\mathcal{M}$, with its interpretation of $\cup$, and a valuation σ of the variables of the constraint $\mathcal{C}$ on M, it is easy to infer in polynomial time if σ is a solution or not.

To prove NP-hardness, consider the following polynomial reduction. Let $G = \langle V, E \rangle$ be a graph. The k-coloring problem for G is known to be NP-complete [17]. Let us consider the constraint:

$$\mathcal{C} = \bigwedge_{X,Y \in V, \{X,Y\} \in E} X \neq Y$$

The satisfiability of the k-coloring problem for G is equivalent to the satisfiability of $\mathcal{C}$ over $\mathcal{M}$. $\square$

The NP-hard nature of the problem in presence of a fixed domain justifies a preliminary polynomial test to check if the domain is "wide enough", as described in Theorem 4.1. This can be formalized in the following corollary.

Corollary 4.3 *Let $\mathcal{C}$ be a disequation constraint in normalized form containing N constraints of type 3 and K constraints of type 2, and let $\mathcal{M}$ be a structure. If the structure is sufficiently "wide"—i.e., it contains a substructure isomorphic to a boolean lattice of size at least $2^{\varsigma(N,K)}$—then the satisfiability problem for a disequation constraint $\mathcal{C}$ can be decided with time complexity $O((N+K)\log(N+K))$.*

4.2 $ACI1$ Disequation Constraints with Constants

Let Σ be $\{\emptyset \equiv c_0, \cup, c_1, \ldots, c_m\}$, where $c_1, \ldots, c_m$ are constants.

Let us characterize the structures suitable to interpret Σ. All models $\mathcal{M}$ studied in the case of elementary $ACI1$—thus, all join-semilattices with bottom, as discussed in Proposition 2.5—are also models of $ACI1$ on Σ, provided an interpretation of the constant symbols $c_1, \ldots, c_m$ over M is given. However, it is natural to focus on structures in which the m constants are interpreted as *distinct objects*, each different from $\perp$. This is equivalent to the introduction of an additional (non-equational) axiom in the theory $ACI1$:

$$(F_2') \quad c_i \neq c_j \quad i, j \in \{0, \ldots, m\}, i \neq j$$

Structures satisfying $ACI1F_2'$ are exactly all the join-semilattices with bottom with a domain of at least $m+1$ objects. (F_2') is actually an instance of the *freeness axiom scheme* (F_2) of *Clark's Equality Theory* [12, 34]—also known as *Clash* [29]. For example, if $m = 4$, all the structures of Figure 2 are models of such extended theory.

Furthermore, in our common intuition we desire the constants to play the role of atoms of such semilattices. Thus, among the possible models, we are interested in those satisfying the additional axiom D_c (D stands for Domain, c for constants)

$$(D_c) \quad (\forall I, J \subseteq \{1, \ldots, m\}) \left(I \neq J \to \bigcup_{i \in I} c_i \neq \bigcup_{j \in J} c_j \right)$$

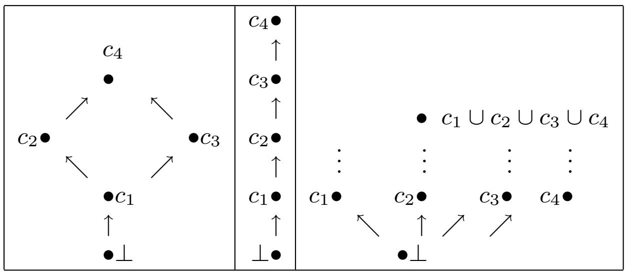

Figure 2: Some models of $ACI1$ when $\Sigma = \{\emptyset, \cup, c_1, c_2, c_3, c_4\}$

where, if $S = \{a_1, \dots, a_n\} \subseteq \{1, \dots, m\}$, then $\bigcup_{i \in S} c_i$ represents the term $c_{a_1} \cup \dots \cup c_{a_n}$. For instance, when $m = 2$, (D_c) becomes:

$$\emptyset \neq c_1 \wedge \emptyset \neq c_2 \wedge \emptyset \neq c_1 \cup c_2 \wedge c_1 \neq c_2 \wedge c_1 \neq c_1 \cup c_2 \wedge c_2 \neq c_1 \cup c_2$$

Among the structures satisfying these requirements, we can find all the Boolean lattices isomorphic to $\langle \wp(\{c_1, \dots, c_m\}), \subseteq \rangle$, i.e., those having $\{c_1\}, \dots, \{c_m\}$ as atoms. Assuming (D_c) we can also ignore (F_2'), since (D_c) implies (F_2').

From an operational point of view, let us assume that $\emptyset <_{\#} c_1 <_{\#} \cdots <_{\#} c_m$. Using ρ we can focus only on the terms of the form:

$$\emptyset \text{ and } X_1 \cup \dots \cup X_k \cup c_{i_1} \cup \dots \cup c_{i_h}$$

where $h + k > 0$ and $i_j < i_{j+1}$ for $j = 1, \dots, h-1$. The disequation $\rho(s) \neq \rho(t)$ gives rise to the following possible cases:

1. $r \neq r$ where r is any normalized term;
2. $c_{i_1} \cup \dots \cup c_{i_h} \neq c_{j_1} \cup \dots \cup c_{j_k}$ and $\{i_1, \dots, i_h\} \neq \{j_1, \dots, j_k\}$
3. $X_1 \cup \dots \cup X_m \cup c_{i_1} \cup \dots \cup c_{i_h} \neq Y_1 \cup \dots \cup Y_n \cup c_{j_1} \cup \dots \cup c_{j_k}$
   where $m > 0$ and either $\{X_1, \dots, X_m\} \neq \{Y_1, \dots, Y_n\}$ or $\{c_{i_1}, \dots, c_{i_h}\} \neq \{c_{j_1}, \dots, c_{j_k}\}$.

Disequations of the first type are false in any model of $ACI1$. Disequations of second type are `true` (and therefore can be removed) in any model of $ACID_c$. In particular, given any join-semilattice with bottom different from $\underline{\mathbf{1}}$, it is possible to satisfy a given disequation of type 2. Similarly, satisfiability of the disequations of the third type depends on the domain considered. In particular, a disequation of type 3 is:

- unsatisfiable in $\underline{\mathbf{1}}$
- satisfiable in any model of $ACI1D_c$ if there is a constant which occurs on one side and not on the other, or if there is a constant in Σ which does not occur in the disequation.

The following theorem holds independently from the presence of (D_c):

Theorem 4.4 *If $\mathcal{C}$ is a disequation constraint in normalized form and it contains r disequations (of type 2 or 3), then $\mathcal{C}$ is satisfiable in any structure for $ACI1$ which contains a substructure isomorphic to $\langle \wp(\{a_1, \dots, a_{r+1}\}), \subseteq \rangle$.*

Proof. Let us assume that the disequations are numbered. The ith disequation contains an element (variable or constant) ν_i which appears on the left-hand side and does not appear on the right-hand side—otherwise it would be an unsatisfiable disequation of type 1. Let us construct an interpretation $(\_)^{\mathcal{M}}$ and a valuation σ as follows: if ν_i is a constant which has not been interpreted yet, then $\nu_i^{\mathcal{M}} = \{a_i\}$, otherwise if ν_i is a variable which has not been assigned yet, then $\sigma(\nu_i) = \{a_i\}$. All remaining constants can be interpreted using $\{a_{r+1}\}$, while the remaining variables can be assigned $\sigma(X) = \emptyset$. It is straightforward to verify that σ is a successful valuation of $\mathcal{C}$ under the interpretation $(\_)^{\mathcal{M}}$. $\square$

In order to handle structures that meet the (D_c) requirement, the size of the substructure in Theorem 4.4 will have to be properly enlarged—including a distinct a_i for each constant.

The proof of the following corollary is a simple generalization of Corollary 4.3 and Theorem 4.4.

Corollary 4.5 *If the structure is sufficiently "wide", then the decision problem for a disequation constraint $\mathcal{C}$ can be solved with worst-case time complexity $O(n \log n)$, where $n = |\mathcal{C}|$. For a given fixed structure, the problem is NP-complete.*

4.3 General $ACI1$ Disequation Constraints

Let us assume that the signature Σ contains at least one function symbol different from $\cup$ and of arity greater than 0. From Section 2.2 we know that, when we consider the partial order $\leq$ induced by the interpretation of $\cup$, the models of $ACI1$ are all the join-semilattices with bottom. However, in presence of a domain M and an interpretation $\cup^{\mathcal{M}}$, the interpretation of the function symbols of Σ introduces a variety of possibilities for building models.

4.3.1 Characterization of a Privileged Model

The most common interpretation of the function symbols different from $\cup$, including the constant symbols, is the one induced by the structure $T(\Sigma)/ =_{ACI1}$, usually denoted by $\mathcal{H}$. In this structure, $(\_)^{\mathcal{H}}$ is defined simply as $(t)^{\mathcal{H}} = [t]$ for any term t. This is equivalent to the interpretation of function symbols as finite tree constructors [33, 27].

In this section we prove some results about this structure and we define a theory $\mathbb{T}_{ACI1}$ such that $\mathcal{H}$ and $\mathbb{T}_{ACI1}$ correspond on the class of formulae we are interested in—namely conjunctions of equations and disequations.

Example 4.6 *Figure 3 shows a representation of $T(\Sigma)/ =_{ACI1}$ when $\Sigma = \{\emptyset, \cup, \{\cdot\}\}$ where $\{\cdot\}$ is a function symbol of arity 1. With a slight abuse of notation, we denote with $\{s, t\}$ the congruence class of $\{s\} \cup \{t\}$. This allows one to interpret the domain of the structure as the set of* hereditarily finite and well-founded *sets. Sets at level i contain exactly i elements.*

$$\begin{tikzcd}
\vdots & \vdots & & \vdots & & \vdots & \vdots \\
\text{lev. } 2 & \{\emptyset, \{\emptyset\}\} & & \{\{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} & & \cdots & \cdots \\
 & & & & & \vdots & \vdots \\
\text{lev. } 1 & \{\emptyset\} \arrow[uu] & & \{\{\emptyset\}\} \arrow[uu] \arrow[uull] & & \{\{\emptyset, \{\emptyset\}\}\} \arrow[uull] & \cdots \\
\text{lev. } 0 & & & \emptyset \arrow[u] \arrow[ull] \arrow[urr] \arrow[urrr] & & & \cdots
\end{tikzcd}$$

Figure 3: $T(\{\emptyset, \cup, \{\cdot\}\})/ =_{ACI1}$

Let us start by observing that the atoms of $\langle \mathcal{H}, \leq \rangle$ are all and only the congruence classes containing terms whose main (outermost) functor is different from $\cup$. This is stated by the following lemma:

Lemma 4.7 *The atoms of $\langle \mathcal{H}, \leq \rangle$ are exactly the congruence classes $[t]$, for some ground term $t \equiv f(t_1, \ldots, t_n)$, $f \not\equiv \cup$, $f \not\equiv \emptyset$.*

Proof. We first prove that the congruence classes induced by the terms of the form $f(t_1, \ldots, t_n)$, with $f \not\equiv \cup$, $f \not\equiv \emptyset$, are atoms. Assume $[X] \leq [t]$, namely $\mathcal{H} \models f(t_1, \ldots, t_n) = f(t_1, \ldots, t_n) \cup X$. This equation has only two solutions in $\mathcal{H}$: $X = \emptyset$ and $X = f(t_1, \ldots, t_n)$ (this can be formally proved using classical results from unification theory).

Consider now two different ground terms,

$$f(s_1, \ldots, s_m) \quad \text{and} \quad g(t_1, \ldots, t_n)$$

and assume that $\mathcal{H} \not\models f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)$. Thus,

$$[f(s_1, \ldots, s_m)] \neq [f(s_1, \ldots, s_m) \cup g(t_1, \ldots, t_n)]$$

By definition of $\leq$ it holds that

$$[f(s_1, \ldots, s_m)] \leq [f(s_1, \ldots, s_m) \cup g(t_1, \ldots, t_n)]$$

and, moreover, $[f(s_1, \ldots, s_m)] \neq \emptyset$. This means that $[f(s_1, \ldots, s_m) \cup g(t_1, \ldots, t_n)]$ is not an atom. □

With a slight abuse of notation, from now on we will call *atom* any term whose main (outermost) function symbol is different from $\cup$. Observe that, by reading $\cup$ as the traditional union operation between sets, then the selected structure $\mathcal{H}$ properly models a form of the extensionality principle for equality between sets, as illustrated in the following lemma.

Lemma 4.8 *Let s, t be two terms, σ be a valuation, and let us assume that $s_1 \cup \cdots \cup s_m \in \sigma(s)$ and $t_1 \cup \cdots \cup t_n \in \sigma(t)$, where all the s_i, t_j are atoms. For all successful valuations σ over $\mathcal{H}$ of the constraint $s \neq t$, there exist an atom a and an index i such that*

- *$s_i = a$ and, for all t_j, $t_j \neq a$ or,*
- *$t_i = a$ and, for all s_j, $s_j \neq a$.*

Proof. Assume $\sigma(s) \neq \sigma(t)$. This means that for all s' term in the class identified by $\sigma(s)$ and t' term in in the class identified by $\sigma(t)$, it holds that $ACI1 \not\models (s' = t')$. But if all the atoms composing s' were in t' and vice versa, then by ACI we would have the equality. □

The structure $\mathcal{H}$ is also a model of the freeness equational axioms (this will be proved in Lemma 4.10):

$$\begin{array}{lll}
(F_1) & f(X_1, \ldots, X_n) = f(Y_1, \ldots, Y_n) \rightarrow X_1 = Y_1 \wedge \cdots \wedge X_n = Y_n & f \in \Sigma, f \not\equiv \cup \\
(F_2) & f(X_1, \ldots, X_m) \neq g(Y_1, \ldots, Y_n) & f \not\equiv g, f, g \not\equiv \cup \\
(F_3) & X \neq t_1 \cup \cdots \cup f(\cdots X \cdots) \cup \cdots \cup t_n & f \not\equiv \cup
\end{array}$$

The standard freeness axioms [12] (also known as *Decomposition*, *Clash*, and *Occur-check* [29]) have been here refined to capture the behavior of $\cup$; in particular, the axiom schema (F_3) deals with nested occurrences of variables in a term.

Similarly to what we did for (D_c) in the previous section, we would like to enforce the property that unions of distinct atoms produce distinct objects of the domain. Instead of extending (D_c)—which would be quite cumbersome in this context—we introduce the following new axiom schema—a form of *Mutation* rule [16]: for all $f \in \Sigma$, $f \not\equiv \cup$, $ar(f) = n$:

$$
\begin{array}{ll}
(D_f) & f(X_1,\dots,X_n) \cup X = Y_1 \cup Y_2 \rightarrow \\
& \exists Z_1, Z_2 \left(\begin{array}{lll}
(Y_1 = f(X_1,\dots,X_n) \cup Z_1 & \wedge & X = Z_1 \cup Y_2) \quad \vee \\
(Y_2 = f(X_1,\dots,X_n) \cup Z_2 & \wedge & X = Y_1 \cup Z_2) \quad \vee \\
(Y_1 = f(X_1,\dots,X_n) \cup Z_1 & \wedge & Y_2 = f(X_1,\dots,X_n) \cup Z_2 \wedge X = Z_1 \cup Z_2)
\end{array} \right)
\end{array}
$$

Observe that the $\leftarrow$ direction is a simple consequence of axioms ACI. This axiom scheme captures the intuitive notion of atoms in the context of $\cup$—i.e., each term constructed using a function symbol f distinct from $\cup$ is an atom of the semilattice—and subsumes the cases covered by (D_c). In particular, it is possible (using a by case proof-theoretic analysis) to prove that $ACI1D_fF_2 \models (D_c)$, while the converse of this result is not true. Consider, for example, $\Sigma = \{\emptyset, \cup, c\}$ and the lattice $\bot < a_1, \bot < a_2, a_1 < c^{\mathcal{M}}, a_2 < c^{\mathcal{M}}$. It satisfies (D_c) but not (D_f): consider

$$
c \cup \underbrace{\bot}_{X} = \underbrace{a_1}_{Y_1} \cup \underbrace{a_2}_{Y_2}.
$$

Instead of proving the above mentioned result, we will prove the weaker result $ACI1D_fF_1F_2F_3 \models (D_c)$ (i.e. $\mathbb{T}_{ACI1} \models (D_c)$).

First we prove the correspondence between the structure $\mathcal{H}$ and the theory

$$
\mathbb{T}_{ACI1} \equiv ACI1F_1F_2F_3D_f
$$

on the class of all the conjunction of equations and disequations. We begin by recalling a result from [37].

Lemma 4.9 *Let $\mathcal{M}$ and $\mathcal{N}$ be two structures of a first-order language $\mathcal{L}$, and let h be an embedding of $\mathcal{M}$ in $\mathcal{N}$. If φ is a quantifier-free (open) formula of $\mathcal{L}$, then for each valuation σ we have $\mathcal{M} \models \sigma(\varphi) \leftrightarrow \mathcal{N} \models h(\sigma(\varphi))$* $\square$

To prove that $\mathcal{H}$ and $\mathbb{T}_{ACI1}$ correspond, we first need to formally prove that $\mathcal{H}$ is a model of $\mathbb{T}_{ACI1}$:

Lemma 4.10 *$\mathcal{H}$ is a model of the theory $\mathbb{T}_{ACI1}$.*

Proof. We prove that the property holds for the various axioms and axiom schema.

$(A)(C)(I)(1)$: $\mathcal{H}$ is a model of $ACI1$, since for any equational theory E, $T(\Sigma)/=_E$ is a model of E [38].

(F_1): If $f(t_1,\dots,t_n)$ and $f(s_1,\dots,s_n)$ are ground terms, then the only way to prove $f(t_1,\dots,t_n) = f(s_1,\dots,s_n)$ from $ACI1$ is:

- to infer $\wedge_{i=1}^{n}(t_i = s_i)$ from $ACI1$;
- to apply the rule derived from classical equality axioms:

$$
\forall X_1 \cdots X_n X_1' \cdots X_n' \left(\bigwedge_{i=1}^{n} X_i = X_i' \rightarrow f(X_1,\dots,X_n) = f(X_1',\dots,X_n') \right)
$$

as last step.

(F_2): It holds trivially, by definition of $\mathcal{H}$, since terms beginning with different free symbols belong to different classes.

(F_3): The fact that $\mathcal{H} \models F_3$ can be proved using a straightforward induction on the complexity of the terms.

(D_f): Let us assume that σ is a valuation such that

$$\mathcal{H} \models \sigma(f(X_1, \dots, X_n) \cup X = Y_1 \cup Y_2)$$

then, from Lemma 4.8, a term in the class $\sigma(f(X_1, \dots, X_n))$ must occur in $\sigma(Y_1)$ or in $\sigma(Y_2)$. This means that

$$\mathcal{H} \models \sigma(Y_1 = f(X_1, \dots, X_n) \cup W_1 \ \vee \ Y_2 = f(X_1, \dots, X_n) \cup W_2).$$

Since $\mathcal{H}$ is a model of $ACI1$, we can assume that no term in $\sigma(f(X_1, \dots, X_n))$ occurs in $\sigma(W_1)$ or in $\sigma(W_2)$.

Let us consider, as first case,

$$\mathcal{H} \models \exists W_1 \sigma(Y_1 = f(X_1, \dots, X_n) \cup W_1) \text{ and } \mathcal{H} \not\models \exists W_2 \sigma(Y_2 = f(X_1, \dots, X_n) \cup W_2).$$

Since $\mathcal{H}$ is a model of (F_1), then according to Lemma 4.8 we have that all the atoms of $\sigma(Y_1 \cup Y_2)$ different from those in $\sigma(f(X_1, \dots, X_n))$ are atoms of $\sigma(X)$.

If the atoms in $\sigma(f(X_1, \dots, X_n))$ are not atoms of $\sigma(X)$, then, once again from Lemma 4.8, we can infer that all the atoms of $\sigma(W_1)$ and $\sigma(Y_2)$ have to be atoms of $\sigma(X)$, and all the atoms of $\sigma(X)$ have to be atoms of $\sigma(W_1)$ or of $\sigma(Y_2)$. Hence, from $ACI1$, $\mathcal{H} \models \sigma(X = W_1 \cup Y_2)$, i.e.,

$$\mathcal{H} \models \sigma(Y_1 = f(X_1, \dots, X_n) \cup W_1 \wedge X = W_1 \cup Y_2).$$

If an atom in $\sigma(f(X_1, \dots, X_n))$ is an atom of $\sigma(X)$, then consider $Z_1 = W_1 \cup f(X_1, \dots, X_n)$. The following statement holds: $\mathcal{H} \models \sigma(Y_1 = f(X_1, \dots, X_n) \cup Z_1)$, and, as in the previous case, $\mathcal{H} \models \sigma(X = Z_1 \cup Y_2)$.

The second case, in which $\mathcal{H} \models \exists W_2 \sigma(Y_2 = f(X_1, \dots, X_n) \cup W_2)$, but $\mathcal{H} \not\models \exists W_1 \sigma(Y_1 = f(X_1, \dots, X_n) \cup W_1)$, can be dealt with similarly.

The third case, in which

$$\mathcal{H} \models \exists W_1 \sigma(Y_2 = f(X_1, \dots, X_n) \cup W_1) \textit{ and } \mathcal{H} \models \exists W_2 \sigma(Y_1 = f(X_1, \dots, X_n) \cup W_2)$$

is similar to $Z_1 = W_1 \cup f(X_1, \dots, X_n)$, when $\sigma(f(X_1, \dots, X_n))$ is an atom of $\sigma(X)$.

□

Lemma 4.11 *If $\mathcal{N}$ is a model of $\mathbb{T}_{ACI1}$, then the function $h : \mathcal{H} \longrightarrow \mathcal{N}$, defined as $h([t]) = t^{\mathcal{N}}$ is an embedding of $\mathcal{H}$ in $\mathcal{N}$.*

Proof. We will prove the following facts:

1. The definition of $h([t])$ does not depend on the choice of the representative of the class;
2. h is an homomorphism;
3. h is injective.

Since we are focusing only on equations and disequation, we can assume that $\mathcal{L}$ does not contain predicate symbols different from $=$; thus, (2) and (3) ensure that h is an embedding.

1. If t_1 and t_2 are two terms such that $[t_1] = [t_2]$, then by definition $ACI1 \models t_1 = t_2$. Since $\mathcal{M} \models t_1 = t_2$ holds for every model $\mathcal{M}$ of $ACI1$, then in particular it holds for $\mathcal{N}$, i.e., $t_1^{\mathcal{N}} = t_2^{\mathcal{N}}$.

2. We need to prove that for all $f \in \Sigma$ and for all terms $t_1, \ldots, t_n \in T(\Sigma)$ it holds that

$$h(f^{\mathcal{H}}([t_1], \ldots, [t_n])) = f^{\mathcal{N}}(h(t_1), \ldots, h(t_n))$$

Now,

$$\begin{array}{rcll} h(f^{\mathcal{H}}([t_1], \ldots, [t_n]) & = & h(f(t_1, \ldots, t_n)) & \text{By property 1. above} \\ & = & (f(t_1, \ldots, t_n))^{\mathcal{N}} & \text{By definition of } h \\ & = & f^{\mathcal{N}}(t_1^{\mathcal{B}}, \ldots, t_n^{\mathcal{N}}) & \text{By definition of structure} \\ & = & f^{\mathcal{N}}(h([t_1]), \ldots, h([t_n])) & \text{By definition of } h \end{array}$$

3. We need to prove that if $h([t_1]) = h([t_2])$, then $[t_1] = [t_2]$. We have already proved that h does not depend on the representative of the class, hence we can assume that t_1 and t_2 are in normalized form. Using this assumption and Corollary 2.7 we have to prove that if t_1 and t_2 are normalized and $h([t_1]) = h([t_2])$, then $t_1 \equiv t_2$.

We prove this fact by structural induction on t_1.

Base: Let $t_1 \equiv c$ be a constant. Since $\mathcal{N}$ is a model of axiom schema (F_2), it can not be that $t_2 \equiv f(s_1, \ldots, s_n)$, with $f \not\equiv \cup, f \not\equiv c$.

Moreover $\mathcal{N}$ is a model of $ACI1D_f$, hence

$$c^{\mathcal{N}} \cup \emptyset^{\mathcal{N}} = s_1^{\mathcal{N}} \cup s_2^{\mathcal{N}} \text{ implies } (s_1^{\mathcal{N}} = c^{\mathcal{N}} \wedge s_2^{\mathcal{N}} = \emptyset^{\mathcal{N}}) \vee (s_1^{\mathcal{N}} = \emptyset^{\mathcal{N}} \wedge s_2^{\mathcal{N}} = c^{\mathcal{N}}) \vee (s_1^{\mathcal{N}} = c^{\mathcal{N}} \wedge s_2^{\mathcal{N}} = c^{\mathcal{N}})$$

and, since $\mathcal{N}$ is a model of (F_2), this implies

$$(s_1 \equiv c \wedge s_2 \equiv \emptyset) \vee (s_1 \equiv \emptyset \wedge s_2 \equiv c) \vee (s_1 \equiv c \wedge s_2 \equiv c)$$

Thus, since t_2 is assumed to be normalized, it cannot be the case that $t_2 \equiv s_1 \cup s_2$. This can be generalized to obtain $t_2 \not\equiv s_1 \cup s_2 \cup \cdots \cup s_n$.

Step: Let $t_1 \equiv f(s_1, \ldots, s_n)$, with $f \not\equiv \cup$. It cannot be $t_2 \equiv g(r_1, \ldots, r_m)$, with $g \not\equiv f, \cup$, since $\mathcal{N}$ is a model of F_2.

As in the previous case we obtain that t_2 cannot be of the form $r_1 \cup \cdots \cup r_m$. Thus, it must be that $t_2 \equiv f(r_1, \ldots, r_n)$, and $s_i^{\mathcal{N}} = r_i^{\mathcal{N}}$, for all $i \leq n$. Using the inductive hypothesis we can infer $t_1 \equiv t_2$.

Let $t_1 \equiv s_1 \cup \cdots \cup s_m$. Since it cannot be that $t_2 \equiv f(r_1, \ldots, r_n)$ (from the previous case applied to t_2), then it must be $t_2 \equiv r_1 \cup \cdots \cup r_n$.

Since t_1 and t_2 are in normalized form, it must be $s_i \equiv f_i(\cdots)$ and $r_j \equiv g_j(\cdots)$, with $f, g \not\equiv \cup$. Proceeding by contradiction, if $t_1^{\mathcal{N}} \not\equiv t_2^{\mathcal{B}}$, since $\mathcal{N}$ is a model of $ACI1$ the following must hold:

$$\exists i \leq m \, \forall j \leq n \, s_i^{\mathcal{N}} \not\equiv t_j^{\mathcal{N}} \vee \exists j \leq n \, i \leq m \, t_j^{\mathcal{N}} \not\equiv s_i^{\mathcal{N}}$$

Without loss of generality we can focus on the first case of the disjunction, and $i = 1$. By inductive hypothesis we have $\forall j \leq n \, h([s_1]) \neq h([t_j])$. Hence, since we have proved that h is an homomorphism, then the following will also hold:

$$\begin{array}{rcccl} h([t_1]) & = & h([s_1]) \cup^{\mathcal{N}} \cdots \cup^{\mathcal{N}} h([s_m]) & = & s_1^{\mathcal{N}} \cup^{\mathcal{N}} \cdots \cup^{\mathcal{N}} s_m^{\mathcal{N}} \\ h([t_2]) & = & h([r_1]) \cup^{\mathcal{N}} \cdots \cup^{\mathcal{N}} h([r_n]) & = & r_1^{\mathcal{N}} \cup^{\mathcal{N}} \cdots \cup^{\mathcal{N}} r_n^{\mathcal{N}} \end{array}$$

From this, since $\mathcal{N}$ is a model of D_f and $s_i \equiv f_1(\cdots)$, we can conclude that $h([t_1]) \neq h([t_2])$.

□

Theorem 4.12 *$\mathcal{H}$ and $\mathbb{T}_{ACI1}$ correspond on the class of all the conjunctions of equations and disequations.*

Proof. From Lemma 4.10 it follows that if $\mathcal{C}$ is a first order formulae and $\mathbb{T}_{ACI1} \models \mathcal{C}$, then $\mathcal{H} \models \mathcal{C}$.

On the other hand, if $\vec{\exists}\mathcal{C}$ is a formula with only existential quantifiers, then $\mathcal{H} \models \vec{\exists}\mathcal{C}$ if and only if there exists σ such that $\mathcal{H} \models \sigma(\mathcal{C})$. Thus, from Lemma 4.11 and Lemma 4.9, we have that $\mathcal{N} \models \vec{\exists}\mathcal{C}$ for all models $\mathcal{N}$ of $\mathbb{T}_{ACI1}$. This implies that $\mathbb{T}_{ACI1} \models \vec{\exists}\mathcal{C}$. □

Corollary 4.13 $\mathbb{T}_{ACI1} \models (D_c)$.

Proof. All the formulae of the axiom schema (D_c) are formed only by one disequation without variables. Hence, by Theorem 4.12, $\mathbb{T}_{ACI1} \models (D_c)$ if and only if $\mathcal{H} \models (D_c)$; the last property is true by definition of $\mathcal{H}$. □

4.3.2 Satisfiability

The theorem we present in this section is fundamental for the satisfiability test of all the solved forms presented in this paper. The goal of the theorem is to relate the normalization of terms produced by ρ to the existence of solutions for a conjunction of disequation constraints. The preliminary lemmas proposed in this section are aimed at analyzing the relationships between syntactic structure of normalized constraints and successful valuations.

A valuation $\sigma : \mathcal{V} \longrightarrow \mathcal{H}$ is said to be *atomic* if for all $X \in \mathcal{V}$, $\sigma(X)$ is an atom of $\mathcal{H}$. Observe that, from Lemma 4.7, we have that if σ is atomic, then $\sigma(X)$ will be a congruence class of the form $[f(t_1, \dots, t_n)]$, with $f \not\equiv \emptyset, f \not\equiv \cup$.

Intuitively, the following lemma shows that if an atomic valuation makes two terms s and t different, then the valuation will also guarantee that s is different from any "set" (i.e., complex term constructed using $\cup$) which contains t:

Lemma 4.14 *Let Σ be a general signature, and let s and $t \equiv t_1 \cup \cdots \cup t_n$ be two normalized terms, where s and t_i are atoms.*

1. *If $s \not\equiv t$, then there exists $i \leq n$ such that $s \not\equiv t_i$.*
2. *If σ is an atomic valuation which satisfies $\bigvee_{i=1}^{n}(s \neq t_i)$, then σ satisfies $s \neq t$.*

Proof.
1) If for all $i \leq n$ we have $s \equiv t_i$, then it is also true that for all $i, j \leq n$ $t_i \equiv t_j$. Hence, since t is normalized, it must be $t \equiv t_1$. From $t \equiv t_1$ and $s \equiv t_1$ we obtain $s \equiv t$, which contradicts the initial assumption $s \not\equiv t$.
2) By axiom (C), without loss of generality, we can assume that σ satisfies $s \neq t_1$. From (D_f) we have

$$s \cup \emptyset = t_1 \cup Y \leftrightarrow \exists A, B \begin{pmatrix} (t_1 = s \cup A \wedge \emptyset = A \cup Y) \vee \\ (Y = s \cup B \wedge \emptyset = B \cup t_1) \vee \\ (t_1 = s \cup A \wedge Y = s \cup B \wedge \emptyset = A \cup B) \end{pmatrix}$$

It is easy to show that the following property holds for all $n > 0$:

$$ACI1 \models \forall X_1 \cdots X_n (X_1 \cup \cdots \cup X_n = \emptyset \leftrightarrow X_1 = \emptyset \wedge \cdots \wedge X_n = \emptyset) \tag{4}$$

In fact, $X_1 \cup \cdots \cup X_n = \emptyset$ implies that $X_1 \cup (X_1 \cup \cdots \cup X_n) = X_1 \cup \emptyset = X_1$, and $(X_1 \cup X_1) \cup X_2 \cup \cdots \cup X_n = X_1 \cup X_2 \cup \cdots \cup X_n = \emptyset$. Similarly for $X_2, \dots, X_n$. The converse is immediate.

From this property we have that $\emptyset = A \cup B \leftrightarrow A = \emptyset \wedge B = \emptyset$. This allows us to rewrite the above formula as

$$s = t_1 \cup Y \leftrightarrow (t_1 = s \wedge Y = \emptyset) \vee (Y = s \wedge t_1 = \emptyset) \vee (t_1 = s \wedge Y = s)$$

Since t is a normalized term, we know that t_1 is normalized, as well, and $t_1 \not\equiv \emptyset$: from (F_2) we can infer that $t_1 = \emptyset$ is unsatisfiable. Hence, we are left with

$$s = t_1 \cup Y \leftrightarrow (t_1 = s \wedge Y = \emptyset) \vee (t_1 = s \wedge Y = s)$$

Since the considerations hold for arbitrary Y, then we can also infer:

$$s = t_1 \cup (t_2 \cup \ldots \cup t_n) \leftrightarrow (t_1 = s \wedge t_2 \cup \ldots \cup t_n = \emptyset) \vee (t_1 = s \wedge t_2 \cup \ldots \cup t_n = s)$$

In our case $\sigma(s) \neq \sigma(t_1)$, which leads to $\sigma(s) \neq \sigma(t_1) \cup \sigma(t_2) \cup \ldots \cup \sigma(t_n)$, hence $\sigma(s) \neq \sigma(t)$. □

The following lemma provides an "extensional" view of disequations between $\cup$-terms:

Lemma 4.15 *Let Σ be a general signature. Let $s \equiv s_1 \cup \cdots \cup s_m$ and $t \equiv t_1 \cup \cdots \cup t_n$ be two normalized terms, s_i, t_j are atoms.*

1. *If $s \not\equiv t$, then:*
   - *there is $j \leq m$ such that, for all $i \leq n$, the relation $s_j \not\equiv t_i$ holds; or*
   - *there is $i \leq n$ such that, for all $j \leq m$, the relation $t_i \not\equiv s_j$ holds.*
2. *Assume that σ is an atomic valuation which satisfies*

$$\left(\bigvee_{j=1}^{m} \bigwedge_{i=1}^{n} s_j \neq t_i \right) \vee \left(\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m} t_i \neq s_j \right),$$

then σ satisfies $s \neq t$.

Proof.
1) Let us assume that for each $j \leq m$ there exists $i_j \leq n$ such that $s_j \equiv t_{i_j}$. Let us also assume that for each $i \leq n$ there exists $j_i \leq m$ such that $t_i \equiv s_{j_i}$. Since s and t are in normalized form, these assumptions lead to the facts

- $m = n$ and
- for all j and i it holds that $i = j_i$ and $j = i_j$.

As a consequence $s \equiv t$ must hold, which contradicts the hypothesis.
2) Let us show this result for $m = 2, n = 2$—the proof can be easily generalized to arbitrary m, n. Without loss of generality, we can assume that σ satisfies $\wedge_{i=1}^{n}(s_1 \neq t_i)$, namely $s_1 \neq t_1 \wedge s_1 \neq t_2$.

From (D_f) we have

$$s_1 \cup Y = t_1 \cup t_2 \leftrightarrow \exists A, B \begin{pmatrix} (t_1 = s_1 \cup A \wedge Y = A \cup t_2) \vee \\ (t_2 = s_1 \cup B \wedge Y = B \cup t_1) \vee \\ (t_1 = s_1 \cup A \wedge t_2 = s_1 \cup B \wedge Y = A \cup B) \end{pmatrix}$$

Observe that $s_1 \cup s_2$ and $t_1 \cup t_2$ are normalized, hence s_1, s_2, t_1, t_2 are normalized and different from $\emptyset$. Since $\sigma(s_1) \neq \sigma(t_1)$, from the proof of Lemma 4.14 (case 2) we have that $\sigma(t_1) \neq \sigma(s_1) \cup A$ holds for all A. In the same way, since $\sigma(s_1) \neq \sigma(t_2)$ we have that $\sigma(t_2) \neq \sigma(s_1) \cup B$ holds for all B. So it must be the case that $\sigma(s_1) \cup Y \neq \sigma(t_1) \cup \sigma(t_2)$ for all Y, and hence, with $Y = \sigma(s_2)$, we obtain $\sigma(s) \neq \sigma(t)$. □

We will define the structure $\mathcal{N}$ having as domain the set of natural numbers, over which we will verify the satisfiability of numeric constraints. The function *depth* : $T(\Sigma, \mathcal{V}) \longrightarrow \mathbb{N}$ will be important to build an homomorphism between $\mathcal{N}$ and $\mathcal{H}$. This function has been defined also on non-ground terms, as this fact will prove useful in Section 8. Intuitively, the function is used to determine the depth of a term interpreted as a tree, where all sub-terms t_i of a term of the type $t_1 \cup \cdots \cup t_n$ are seen as children of the same root. Satisfiability over $\mathcal{N}$ will imply satisfiability over $\mathcal{H}$.

$$\begin{cases} \mathit{depth}(X) & = & 0 & \text{if } X \text{ is a variable} \\ \mathit{depth}(\emptyset) & = & 0 & \\ \mathit{depth}(c) & = & 1 & \text{if } c \text{ is a constant, } c \not\equiv \emptyset \\ \mathit{depth}(t_1 \cup t_2) & = & \max_{i=1}^{2} \mathit{depth}(t_i) & \\ \mathit{depth}(f(t_1, \dots, t_n)) & = & 1 + \max_{i=1}^{n} \mathit{depth}(t_i) & f \not\equiv \cup \end{cases}$$

Given a general signature Σ, we define the structure $\mathcal{N} = \langle \mathbb{N}, (\_)^{\mathcal{N}} \rangle$, where $(\_)^{\mathcal{N}}$ is defined as:

$$(t)^{\mathcal{N}} = \mathit{depth}(t)$$

It is immediate to prove that if $ACI1 \models (s = t)$ for ground terms s and t, then $\mathit{depth}(s) = \mathit{depth}(t)$. Thus, with a slight abuse of notation, we will use *depth* also for the congruence class $[t]$ of a term t.

Lemma 4.16 *The function depth is an homomorphism from $\mathcal{H}$ to $\mathcal{N}$. Moreover, if σ is a valuation on $\mathcal{H}$ for the variables in s, t, and $\mathcal{N} \models \sigma(s) \neq \sigma(t)$, then $\mathcal{H} \models \sigma(s) \neq \sigma(t)$.*

Proof. We prove that *depth* is an homomorphism:

1. $\mathit{depth}(\emptyset^{\mathcal{H}}) = 0 = \emptyset^{\mathcal{N}}$;
2. $\mathit{depth}(c^{\mathcal{H}}) = 1 = c^{\mathcal{N}}$;
3. $\mathit{depth}(t_1 \cup^{\mathcal{H}} t_2) = \max\{\mathit{depth}(t_1), \mathit{depth}(t_2)\} = \mathit{depth}(t_1) \cup^{\mathcal{H}} \mathit{depth}(t_2)$;
4. $\mathit{depth}(f^{\mathcal{H}}(t_1, \dots, t_n)) = 1 + \max_{i=1}^{n} \mathit{depth}(t_i) = f^{\mathcal{N}}(\mathit{depth}(t_1), \dots, \mathit{depth}(t_n))$.

By [37], if σ satisfies $s = t$ in $\mathcal{H}$, since *depth* is an homomorphism, then $\mathit{depth} \circ \sigma$ satisfies $s = t$ in $\mathcal{N}$. This allows to derive the second part of the proposition by contradiction. □

Theorem 4.17 *Let Σ be a general signature. A constraint $\mathcal{C} \equiv (s_1 \neq t_1 \wedge \cdots \wedge s_n \neq t_n)$ such that $\rho(s_i) \not\equiv \rho(t_i)$ for all $i = 1, \dots, n$ is satisfiable in $\mathcal{H}$, and in every model of $\mathbb{T}_{ACI1}$.*

Proof. First of all, according to Corollary 2.8, we have that $ACI1 \models \vec{\forall}(\mathcal{C} \leftrightarrow \rho(\mathcal{C}))$. So, without loss of generality, we can assume that all the terms s_i, t_i in $\mathcal{C}$ are normalized terms.

To achieve the desired result, we proceed as follows: we start by building a new system R; we prove that each atomic valuation σ satisfying R satisfies $\mathcal{C}$, and finally we prove that R admits always at least one atomic solution.

- If $(s_i \neq t_i) \in \mathcal{C}$ and neither s_i nor t_i is of the form $\_ \cup \_$, then let us consider the disequation $s_i \neq t_i$ in R. Obviously each solution of R is also a solution of $s_i \neq t_i$.
- If $(s_i \neq t_{i_1} \cup \cdots \cup t_{i_k}) \in \mathcal{C}$, where the s_i and the t_{i_j}'s are not of the form $\_ \cup \_$, then since $s_i \not\equiv t_{i_1} \cup \cdots \cup t_{i_k}$ (by hypothesis), there must exist $j \leq k$ such that $s_i \not\equiv t_{i_j}$—from Lemma 4.14 case 1. Hence, let us consider the disequation $s_i \neq t_{i_j}$ in R. An atomic solution σ for R is always an atomic solution for $s_i \neq t_i$, since $\sigma(s_i)$ is an atom different from $\sigma(t_{i_j})$ (from Lemma 4.14 case 2).
- If $(s_{i_1} \cup \cdots \cup s_{i_h} \neq t_{i_1} \cup \cdots \cup t_{i_k}) \in \mathcal{C}$, where the s_{i_ℓ} and the t_{i_j} are not of the form $\_ \cup \_$, then from Lemma 4.15 case 1 and without loss of generality we can assume that $\wedge_{j=1}^{k}(s_{i_1} \not\equiv t_{i_j})$. Hence, let us consider the disequations $s_{i_1} \neq t_{i_1}, \dots, s_{i_1} \neq t_{i_k}$ in R. An atomic solution σ for R is a solution for $s_i \neq t_i$, from Lemma 4.15 case 2.

R is a system of disequations involving atoms and variables. To find an atomic solution σ of R, we introduce the auxiliary function $\mathit{find} : T(\Sigma, \mathcal{V}) \times \mathcal{V} \longrightarrow \wp(\mathbb{N})$.

$$\mathit{find}(t, X) = \begin{cases} \emptyset & \text{if } X \text{ does not occur in } t \\ \{0\} & \text{if } t \equiv X \\ \{n \mid n \in \mathit{find}(t_1, X) \vee n \in \mathit{find}(t_2, X)\} & \text{if } t \equiv t_1 \cup t_2 \\ \{n+1 \mid n \in \mathit{find}(t_i, X) \wedge 1 \leq i \leq m\} & \text{if } t \equiv f(t_1, \ldots, t_m) \end{cases}$$

This function produces for each pair (t, X) a set of numbers indicating the depth of the occurrences of X in t. For each variable X in R consider the auxiliary variable N_X, which ranges over integer numbers. Additionally, for each

- pair of terms $\langle s, t \rangle$ occurring in R such that $s \not\equiv t$
- variable X in s
- variable Y in t such that $Y \not\equiv X$
- number $m_{(s,X)}$ in $\mathit{find}(s, X)$
- number $m_{(t,Y)}$ in $\mathit{find}(t, Y)$

we introduce the disequation

$$N_X \neq N_Y + m_{(t,Y)} - m_{(s,X)}.$$

For every variable N_X consider also the test

$$N_X > \max\{\mathit{depth}(v) \mid v \in R, v \text{ ground}\},$$

where the notation $v \in R$ stands for: v is a term which occurs in R. This system admits always a solution. Let us indicate with $\{n_X \mid X \in \mathit{vars}(R)\}$ a solution of such system. Since Σ is general, there is $f \in \Sigma$, $ar(f) > 0$, $f \not\equiv \emptyset$, $f \not\equiv \cup$. We denote by r^h the term constructed as follows:

$$\begin{cases} r^0 & = & \emptyset \\ r^{k+1} & = & f(r^k, \ldots, r^k) \end{cases}$$

and consider the valuation $\sigma = \{X/[r^{n_X}] \mid X \in \mathit{vars}(R)\}$. We can prove that σ satisfies R. It is easy to see that $\mathit{depth}(\sigma(X)) = n_X$.

We prove that if s, t are two terms that occur in R (sub-terms of the s_i, t_i), and $s \not\equiv t$, then $\sigma(s) \neq \sigma(t)$ in $\mathcal{H}$. From this result we can conclude the desired claim, since if $s_i \neq t_i$ is in R then s_i, t_i are two terms which occur in R and $s_i \not\equiv t_i$.

Let us proceed by induction on the structure of s. Since $=$ is symmetric we can avoid to consider the case s, t if the case t, s has already been considered.

If $s \equiv \emptyset$, then, since $t \not\equiv s$, $\mathit{depth}(\sigma(t)) > 0 = \mathit{depth}(\sigma(s))$, from Lemma 4.16, we have $\sigma(s) \neq \sigma(t)$. If, instead $s \equiv c$ with c constant, then, since $t \not\equiv s$, it can be:

- t is a constant different from c, and hence $\sigma(s) = [s] \neq [t] = \sigma(t)$;
- $t \equiv c_1 \cup \cdots \cup c_n$, all the c_i constants, then from $t \not\equiv s$ and Lemma 4.14 case 1, there exists $i \leq n$, such that $s \not\equiv c_i$, hence we have the thesis;
- t is a ground term different from a constant, a union of constants, or a non-ground term. In all these cases, since $n_X > 1$ for all X, $\mathit{depth}(\sigma(y)) > 1 = \mathit{depth}(s)$, from Lemma 4.16 we have the thesis.

If $s \equiv X$, then by induction on t:

- if t is ground, then, since $\mathit{depth}(\sigma(X)) = n_X > \mathit{depth}(t)$, from Lemma 4.16 we have the thesis;
- if $t \equiv Y$, then $n_X \neq n_Y$ is an equation of the system, hence from Lemma 4.16 we have the thesis;
- if $t \equiv f(t_1, \ldots, t_n)$ and X occurs in t, (F_3) leads to the thesis;

- if $t \equiv f(t_1, \ldots, t_n)$ and X does not occur in t, then since we have imposed

$$N_Y > max\{depth(v) \mid v \in R, v \text{ ground}\}$$

for all Y, there must exists a variable Y in t such that $depth(\sigma(t)) = n_Y + m_{(t,Y)}$ with $m_{(t,Y)} \in find(t, Y)$, in the system we have forced: $n_X \neq n_Y + m_{(t,Y)}$, hence from Lemma 4.16 we have the thesis;

- if $t \equiv t_1 \cup \cdots \cup t_n$, then from Lemma 4.14 case 1 there exists t_i such that $t_i \not\equiv X$; hence, by induction $\sigma(t_i) \neq \sigma(X)$. From Lemma 4.14 case 2 we have the desired claim.

If $s \equiv f(s_1, \ldots, s_n)$, then by induction on t:

- the result is trivial if $t \equiv g(t_1, \ldots, t_m)$, $f \not\equiv g$;
- if $t \equiv f(t_1, \ldots, t_n)$, then it must be $s_i \not\equiv t_i$ for some $i \leq n$. By inductive hypothesis we obtain $\sigma(s_i) \neq \sigma(t_i)$, and from (F_1) we have the result;
- if $t \equiv t_1 \cup \cdots \cup t_m$, then, from Lemma 4.14 case 1 there is $t_i \not\equiv s$. By inductive hypothesis on t_i, we have that $\sigma(s) \neq \sigma(t_i)$, and from Lemma 4.14 case 2 we can obtain the result.

If $s \equiv s_1 \cup \cdots \cup s_n$, and $t \equiv t_1 \cup \cdots \cup t_m$ (this is the only case not yet considered), then from Lemma 4.15 case 1 we can focus on $\wedge_{i=1}^m (s_1 \not\equiv t_i)$. By inductive hypothesis $\wedge_{i=1}^m (\sigma(s_1) \neq \sigma(t_i))$ holds, and from Lemma 4.15 case 2 we can obtain the desired result.

Since we have proved that $\mathcal{C}$ is satisfiable in $\mathcal{H}$, from Theorem 4.12, we have that $\mathcal{C}$ is satisfiable in every model of $\mathbb{T}_{ACI1}$. □

Corollary 4.18 *If Σ is a general signature, then the satisfiability of a disequation constraint in $\mathcal{H}$ can be decided in polynomial time.*

Proof. Theorem 2.6 ensures the polynomial nature of the computation of $\rho(\mathcal{C})$ and the equivalence between $\mathcal{C}$ and $\rho(\mathcal{C})$. If one of the literals of $\rho(\mathcal{C})$ is of the form $\rho(s) \neq \rho(s)$ (this test can be performed in linear time on $|\rho(\mathcal{C})|$) then $\rho(\mathcal{C})$ (hence $\mathcal{C}$) is clearly unsatisfiable. If this is not the case, then Theorem 4.17 allows us to conclude that $\mathcal{C}$ is satisfiable. □

5 Solved Forms

Most constraint systems rely on the availability of *constraints simplifiers* to transform constraints into equivalent "simpler" formulae. In particular, it is common to identify a class of formulae, called *solved forms*, which are the target of this simplification. As described in [15], solved forms should meet some intuitive criteria, such as:

- *solvability:* each formula different from the constant false should admit at least one solution;
- *simplicity:* every solution can be easily obtained from a solved form;
- *completeness:* each constraint should be equivalent to a disjunction of solved form constraints.

We propose four solved forms for the $ACI1$-constraints considered in this paper. Each form can be incrementally computed from the previous one. The first form (*implicit*) provides an implicit representation of its set of solutions. Given a constraint, a unique implicit solved form constraint can be computed in polynomial (quadratic) time. The second solved form (*intermediate*) further simplifies a constraint in implicit normal form. In polynomial (quadratic) time it is possible to compute a formula containing a collection of intermediate normal form constraints. Each component of the collection can be computed in polynomial time, but the number of components might be exponential. The third solved form (*explicit*) represents explicitly its solutions. It can be computed from the intermediate form on demand. In this case, determining each disjunct requires exponential time. Finally, the fourth solved form (*subset*) represents its solutions as constraints on individual variables.

All the solved forms are instances of the *compact formulae* defined in [15]. If the theory $ACI1$ were *compact*, the ability to reach a solved form would automatically ensure satisfiability of these constraints over $\mathcal{H}$ [15]. However (see Section 7.2 for a formal proof), $ACI1$ is not compact; thus, a direct proof of satisfiability of the different solved forms is required.

In the following subsections we precisely characterize the four solved forms. However, we anticipate that all of them satisfy the hypothesis of Theorem 4.17—thus, we have:

Corollary 5.1 *An implicit/intermediate/explicit/subset solved form constraint $\mathcal{C}$ different from* false *is satisfiable in $\mathcal{H}$ and $\mathbb{T}_{ACI1} \models \vec{\exists}\mathcal{C}$.*

5.1 Implicit Solved Form

The first solved form we propose refines the constraint $\rho(\mathcal{C})$ by removing useless conjuncts.

Definition 5.2 *A constraint $\mathcal{C}$ is in* implicit solved form *if it is* false*,* true*, or $\mathcal{C} \equiv (s_1 \neq t_1 \wedge \cdots \wedge s_n \neq t_n)$ and for all $i = 1, \ldots, n$*

- *$vars(s_i) \cup vars(t_i) \neq \emptyset$, (i.e., it does not contain ground disequations), and*
- *$s_i \equiv \rho(s_i)$ and $t_i \equiv \rho(t_i)$, (i.e., all terms are normalized), and*
- *$s_i \not\equiv t_i$, (i.e., it does not contain trivial disequations), and*
- *if t_i is a variable, then s_i is a variable as well—i.e., constraints on individual variables are to be written as $X \neq t$—and*
- *if s_i is a variable, then it does not occur nested in t_i (generalization of occur-check—cf. definition of nested in Section 2).*

Example 5.3 *The following constraint is in implicit solved form:*

$$X \neq a \wedge f(X,b) \neq f(a,Y) \wedge X \cup f(X,Y) \neq A \cup f(c,d) \wedge f(A,b) \neq Y \cup Z \wedge X \cup Y \neq Y \cup c \quad (5)$$

Given a constraint $\mathcal{C} \equiv (s_1 \neq t_1 \wedge \cdots \wedge s_n \neq t_n)$, we can obtain an equivalent constraint in implicit solved form, starting from $\rho(\mathcal{C})$ (cf. Section 2.3) and applying the function impl_simpl described in Figure 4.[2]

The result of impl_simpl$(\rho(t))$ is an equivalent constraint in implicit normal form.

Proposition 5.4 *Given a constraint $\mathcal{C}$, then* impl_simpl$(\rho(\mathcal{C}))$ *can be computed in time $O(n^2)$ where $n = |\mathcal{C}|$. Moreover, if $\bar{X} = vars(\mathcal{C})$, then*

$$ACI1F_1F_2F_3 \models \forall \bar{X}(\mathcal{C} \leftrightarrow \mathsf{impl\_simpl}(\rho(\mathcal{C}))$$

Proof. Complexity follows from Theorem 2.6 and the fact that the function impl_simpl analyzes each disequation at most twice.

The correctness of the procedure follows from the Corollary 2.7 and from analyzing the relationships between each rewriting rule and axioms (F_1), (F_2), and (F_3). □

[2]An efficient implementation would require a careful ordering of the reductions—e.g., applying rule (1) and (S2) as soon as possible.

function impl_simpl(φ): while $\varphi = \varphi' \wedge c$ and c is a disequation not in implicit s.f. do			
(1)	$\varphi' \wedge r \neq r$	$\mapsto$	false
(2)	$\left.\begin{array}{r}\varphi' \wedge s \neq t \\ s \not\equiv t, vars(s) = \emptyset \wedge vars(t) = \emptyset\end{array}\right\}$	$\mapsto$	φ'
(3)	$\left.\begin{array}{r}\varphi' \wedge t \neq X \\ t \text{ is not a variable}\end{array}\right\}$	$\mapsto$	$\varphi' \wedge X \neq t$
(4)	$\left.\begin{array}{r}\varphi' \wedge X \neq t_1 \cup \cdots \cup f(\cdots X \cdots) \cup \cdots \cup t_n \\ f \not\equiv \cup\end{array}\right\}$	$\mapsto$	φ'
(5)	$\left.\begin{array}{r}\varphi' \wedge f(\cdots) \neq g(\cdots) \\ f \not\equiv g, f \not\equiv \cup, g \not\equiv \cup\end{array}\right\}$	$\mapsto$	φ'
($S1$)	$\varphi' \wedge$ true	$\mapsto$	φ'
($S2$)	$\varphi' \wedge$ false	$\mapsto$	false

Figure 4: Rewriting procedure for implicit solved form

5.2 Intermediate Solved Form

The second solved form provides an additional level of constraint simplification w.r.t. the implicit solved form. In particular, this solved form simplifies all those disequations where one of the terms involved is a complex term with a main functor different from $\cup$.

The simplification procedure used to generate the intermediate solved form constraints introduces a level of non-determinism, but its complexity remains polynomial.

function int_simpl(φ): while there is a disequation not in intermediate s.f. in φ do replace it as follows:			
(1)–(5) as in impl_simpl			
(6)	$\left.\begin{array}{r}s_1 \cup s_2 \neq f(t_1, \ldots, t_n) \\ f \not\equiv \cup\end{array}\right\}$	$\mapsto$	$f(t_1, \ldots, t_n) \neq s_1 \cup s_2$
(7)	$f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_n)$	$\mapsto$	$(\bigvee_{i=1}^{n} s_i \neq t_i)$
(8)	$f(s_1, \ldots, s_n) \neq t_1 \cup t_2$	$\mapsto$	$\left(\begin{array}{l}\bigvee_{i=1}^{2}(f(s_1, \ldots, s_n) \neq t_i \wedge t_i \neq \emptyset) \vee \\ \vee(f(s_1, \ldots, s_n) \neq t_1 \wedge f(s_1, \ldots, s_n) \neq t_2)\end{array}\right)$

Figure 5: Rewriting procedure for intermediate solved form

Definition 5.5 *An implicit solved form constraint $\mathcal{C}$ is in* intermediate solved form *if it is* false*,* true*, or $\mathcal{C} \equiv (s_1 \neq t_1 \wedge \cdots \wedge s_n \neq t_n)$ and for all $i = 1, \ldots, n$*

- *s_i is a variable which does not occur nested in t_i, or*
- *$s_i \equiv r_1 \cup \cdots \cup r_h$ and $t_i \equiv v_1 \cup \cdots \cup v_k$, $h, k \geq 2$.*

Example 5.6 *The following constraint is in intermediate solved form:*

$$X \neq a \wedge Y \neq b \wedge X \cup f(X,Y) \neq A \cup f(c,d) \wedge Y \neq \emptyset \wedge Y \neq f(A,b) \wedge X \cup Y \neq Y \cup c \qquad (6)$$

The procedure described in Figure 5 produces a formula containing only disequations in intermediate solved form. Moreover, the formula is equivalent to the original constraint.

Proposition 5.7 *Given a constraint $\mathcal{C}$, the formula* $\mathsf{int\_simpl}(\rho(\mathcal{C}))$ *can be computed in time $O(|\mathcal{C}|^2)$. Moreover,* $\mathbb{T}_{ACI1} \models \vec{\forall}(\mathcal{C} \leftrightarrow \mathsf{int\_simpl}(\rho(\mathcal{C})))$.

Proof. The basic idea used to prove the complexity result is that of avoiding duplications of newly generated terms in the step (8). We adopt a tree representation of terms as commonly done in the implementation of Prolog based on Warren Abstract Machine [43]. Let $\mathcal{C} = u_1 \neq v_1 \wedge \cdots \wedge u_p \neq v_p$ be the initial system. Let us assume, without loss of generality, that it is already in implicit solved form. We will use the following data structures:

- An array Terms, whose cells are of the form $\langle \mathsf{Symbol}, \mathsf{Pointer} \rangle$, where Symbol is an element of $\Sigma \cup \mathcal{V}$ and $\mathsf{Pointer}$ is a (possibly empty) list of pointers to other table cells. We will always assume that the entry 0 of the table contains the symbol $\emptyset$ and the pointer nil. For instance, the term $f(g(a), X \cup Y)$ is represented as follows:

0	$\emptyset$	nil
1	f	$(2,3)$
2	g	(4)
3	$\cup$	$(5,6)$
4	a	nil
5	X	nil
6	Y	nil

  Initially, the first $2p$ cells of the table contain the entry points of the representations of the terms $u_1, v_1, \ldots, u_p, v_p$.

- An array Diseq that stores tuples of the type $\langle \mathsf{Done}, \mathsf{Left}, \mathsf{Right}, \mathsf{Pointer} \rangle$ where

  – Left and Right are pointers to the entries in the array Terms, and they represent the left-hand side and the right-hand side of a disequation;

  – Done is a flag describing whether the disequation has been completely processed;

  – $\mathsf{Pointer}$ is of the form $\mathsf{and}(d_1, \ldots, d_h)$, $\mathsf{or}(d_1, \ldots, d_k)$, or $\mathsf{or}(\mathsf{and}(d^1_1, \ldots, d^1_{h_1}), \ldots, \mathsf{and}(d^k_1, \ldots, d^k_{h_k}))$, where d_i are pointers to other disequations.

  At the beginning, only p cells are initialized with values: $\langle \mathsf{false}, 2i-1, 2i, \mathsf{nil} \rangle$ for $i = 1, \ldots, p$. Moreover, assume that $\mathsf{Diseq}[0]$ and $\mathsf{Diseq}[-1]$ are used for the atoms true and false, respectively.

The lists stored in the $\mathsf{Pointer}$ components are used to maintain the representation of the formula φ. Initially $\varphi = \mathsf{and}(1, \ldots, p)$ stating that $\varphi = \bigwedge_{i=1,\ldots,p} \mathsf{Diseq}[i]$. In general, the formula φ is represented by the and-or tree encoded through the $\mathsf{Pointer}$ fields.

The algorithm can be implemented using these data structures as follows:

- Choose a disequation $i > 0$ such that $\mathsf{Diseq}[i].\mathsf{Done} = \mathsf{false}$ and such that one of the rewriting rules (1)–(8) is applicable. The different steps can be applied as follows:

  – (1): set $\mathsf{Diseq}[i].\mathsf{Pointer} = \mathsf{and}(-1)$ and $\mathsf{Diseq}[i].\mathsf{Done} = \mathsf{true}$.

  – (2), (4), and (5): set $\mathsf{Diseq}[i].\mathsf{Pointer} = \mathsf{and}(0)$ and $\mathsf{Diseq}[i].\mathsf{Done} = \mathsf{true}$.

  – (3) and (6): swap the values of $\mathsf{Diseq}[i].\mathsf{Right}$ and $\mathsf{Diseq}[i].\mathsf{Left}$.

– (7): let us denote with S_j the pointer stored in the jth position in the list $\mathsf{Terms}[\mathsf{Diseq}[i].\mathsf{Left}].\mathsf{Pointer}$, and with T_j the pointer stored in the jth position in the list $\mathsf{Terms}[\mathsf{Diseq}[i].\mathsf{Right}].\mathsf{Pointer}$; these represent the entry points for the terms s_j and t_j respectively. Add n new disequations to Diseq $(k+1,\dots,k+n)$, containing $(j=1,\dots,n)$:

$$\mathsf{Diseq}[k+j] = \langle \mathsf{false}, S_j, T_j, \mathsf{nil}\rangle$$

In addition, the field $\mathsf{Diseq}[i].\mathsf{Pointer}$ should be set to $\mathsf{or}(k+1,\dots,k+n)$ and the field $\mathsf{Diseq}[i].\mathsf{Done}$ should be set to true.

– (8): let us denote with T_1 the pointer stored in the first position in the list $\mathsf{Terms}[\mathsf{Diseq}[i].\mathsf{Right}].\mathsf{Pointer}$, and with T_2 the pointer stored in the second position in such list; these represent the entry points for the terms t_1 and t_2 respectively. We need to add 4 new disequations to Diseq $(k+1,\dots,k+4)$, containing:

$$\begin{array}{rcl}\mathsf{Diseq}[k+1] & = & \langle \mathsf{false}, \mathsf{Diseq}[i].\mathsf{Left}, T_1, \mathsf{nil}\rangle \\ \mathsf{Diseq}[k+2] & = & \langle \mathsf{false}, \mathsf{Diseq}[i].\mathsf{Left}, T_2, \mathsf{nil}\rangle \\ \mathsf{Diseq}[k+3] & = & \langle \mathsf{false}, T_1, 0, \mathsf{nil}\rangle \\ \mathsf{Diseq}[k+4] & = & \langle \mathsf{false}, T_2, 0, \mathsf{nil}\rangle\end{array}$$

Finally, $\mathsf{Diseq}[i].\mathsf{Pointer}$ should be set equal to $\mathsf{or}(\mathsf{and}(k+1,k+3),\mathsf{and}(k+2,k+4),\mathsf{and}(k+1,k+2))$ and $\mathsf{Diseq}[i].\mathsf{Done}$ should be set to true.

It is clear that the global number of operations depends linearly on the number of disequations introduced. As a matter of fact, each disequation is processed once and a fixed number of operations (maximum in the cases (7) and (8)) is performed. Thus, the complexity of this procedure corresponds to the number of disequations introduced. Observe that steps (2) and (5) require the ability to detect which variables are present in each subterm. This test can be either implemented in constant time by collecting in each cell of the array Terms additional information (thus requiring $O(|\mathcal{C}|^2)$ total preprocessing time to build Terms) or by making use of standard union-find techniques—thus introducing a small logarithmic overhead to the total complexity.

Given a term t, let us denote with $||t||$ the number obtained by adding the number of occurrences of symbols in t and the number of occurrences of $\cup$ in t—i.e., occurrences of $\cup$ in t are counted twice. Let us prove by induction on $||s||+||t|| \geq 2$ that the number of disequations recursively introduced by a single disequation $s \neq t$ is less than or equal to $||s||\,||t||$.
Base: $||s||+||t|| = 2$: this means that $s \neq t$ is in one of the following forms: $X \neq X, X \neq Y, X \neq c, c \neq X, c \neq c, c \neq d$. In all these cases at most one further equation is introduced.
Step: Let $||s||+||t|| > 2$. Let us consider only the cases (7) and (8). The other cases are trivial.
Rule (7): By inductive hypothesis, each of the n disequations $s_i \neq t_i$ introduces at most $||s_i||\,||t_i||$ disequations. Thus, the considered disequation introduces a number of disequations bounded by

$$n + \sum_{i=1}^{n}(||s_i||\,||t_i||) \leq (1+\sum_{i=1}^{n}||s_i||)(1+\sum_{i=1}^{n}||t_i||)$$

Rule (8): Instead of considering a simple application of rule (8) we consider the whole sequence of application that "remove" all the $\cup$ symbols in the term in the right-hand side. Namely, the disequation:

$$f(s_1,\dots,s_n) \neq t_1 \cup t_2 \cup \cdots \cup t_k$$

where the t_i's are either variables or terms with outermost symbol different from $\cup$, is replaced by:

$$f(s_1,\dots,s_n) \neq t_1, \dots, f(s_1,\dots,s_n) \neq t_k, t_1 \neq \emptyset, \dots, t_k \neq \emptyset$$

The last n disequations do not introduce new disequations (they are either in solved form or they are replaced by true or false). By inductive hypothesis, we have that the disequations $f(s_1,\dots,s_n) \neq t_i$ introduce no more than $||f(s_1,\dots,s_n)||\,||t_i||$ new disequations. Thus, the considered disequation generates at most

$$\sum_{i=1}^{k} ||f(s_1,\dots,s_n)||\,||t_i|| + k \qquad (7)$$

new disequations. Now,

$$\begin{array}{rcl} ||f(s_1,\ldots,s_n)(t_1 \cup \cdots \cup t_k)|| & = & ||f(s_1,\ldots,s_n)||(||t_1|| + \cdots + ||t_k|| + 2(k-1)) \\ & = & \sum_{i=1}^{k} ||f(s_1,\ldots,s_n)||\,||t_i 1|| + 2(k-1)||f(s_1,\ldots,s_n)|| \end{array}$$

Since $||f(s_1,\ldots,s_n)|| \geq 1$ and $k \geq 2$, then this value is greater than (7).

Thus, the global complexity is $O(|\mathcal{C}|^2)$ rule applications.

The correctness of the transformation can be proved by analyzing the various rewriting rules that apply axioms (F_1), (F_2), and (F_3). □

The result produced by the algorithm in Figure 5 can be seen as an and-or tree structure, whose leaves contain true, false, or a disequation in intermediate solved form. The tree can be simplified by removing unnecessary occurrences of true and false, similarly to the effect of steps $(S1)$ and $(S2)$ in Figure 4. The simplification requires a single scan of the tree structure (and thus it does not worsen the previously described complexity result), starting from its leaves and using the obvious simplification rules:

- $\mathsf{or}(\ldots,\mathsf{true},\ldots) \mapsto \mathsf{true}$
- $\mathsf{or}(d_1,\ldots,d_{i-1},\mathsf{false},d_{i+1},\ldots,d_k) \mapsto \mathsf{or}(d_1,\ldots,d_{i-1},d_{i+1},\ldots,d_k)$
- $\mathsf{and}(\ldots,\mathsf{false},\ldots) \mapsto \mathsf{false}$
- $\mathsf{and}(d_1,\ldots,d_{i-1},\mathsf{true},d_{i+1},\ldots,d_k) \mapsto \mathsf{and}(d_1,\ldots,d_{i-1},d_{i+1},\ldots,d_k)$

In spite of this polynomial result, a successive application of the distributivity on the formula obtained can generate an exponential number of disjuncts. It is important to observe that this sort of consideration holds in any general equational theory E. For example, given a binary free function symbol f and the constraint

$$f(X_1,Y_1) \neq f(V_1,Z_1) \wedge \cdots \wedge f(X_n,Y_n) \neq f(V_n,Z_n) \tag{8}$$

After the application of the simplification rules in Figure 5 we obtain (in time $O(n)$):

$$(X_1 \neq V_1 \vee Y_1 \neq Z_1) \wedge \cdots \wedge (X_n \neq V_n \vee Y_n \neq Z_n) \tag{9}$$

The corresponding disjunctive normal form contains 2^n disjuncts. Thus, the presence of an exponential number of disjuncts generated by the int_simpl procedure implies that a complete description of the whole set of solutions as a disjunctive normal form formula requires an exponential amount of time.

Corollary 5.8 *Given a constraint $\mathcal{C}$, an intermediate solved form constraint $\mathcal{C}'$ that implies $\mathcal{C}$ can be computed in time $O(n^2)$, where $n = |\mathcal{C}|$. Moreover, a constraint $\mathcal{C}'$ equivalent to $\mathcal{C}$ and composed by a disjunction of intermediate solved form constraints can be computed in $O(2^{p(n)})$, where $p(n)$ is a polynomial formula.*

Proof. Each disjunct can be computed in from $\mathsf{int\_simpl}(\rho(\mathcal{C}))$ in linear time on its size. The rest of the results derive from the previously made considerations. □

5.3 Explicit Solved Form

In this section we show how a disequation $r_1 \cup \cdots \cup r_m \neq s_1 \cup \cdots \cup s_n$ occurring in a constraint in intermediate solved form can be further simplified *without introducing new variables.* The idea is to reduce disequations having $\cup$ terms on both sides; in the simplified form, the two sides of the disequations contain the same set of variables, with the exception of exactly one variable, which appears only on the left-hand side of the disequation.

Definition 5.9 *An intermediate solved form constraint $\mathcal{C}$ is in* explicit solved form *if it is* false, true, *or $\mathcal{C} \equiv (s_1 \neq t_1 \wedge \cdots \wedge s_n \neq t_n)$, and for all $i = 1, \ldots, n$ the literal $s_i \neq t_i$ is of the following form:*

$$X \cup Y_1 \cup \cdots \cup Y_h \neq Y_1 \cup \cdots \cup Y_h \cup r_1 \cup \cdots \cup r_k$$

where $h \geq 0$ and r_i's are arbitrary terms different from the variable X.

Example 5.10 *The following constraint is in explicit solved form:*

$$X \neq a \wedge Y \neq b \wedge X \neq A \cup Z \wedge Y \neq \emptyset \wedge Y \neq f(A, b) \wedge X \cup Y \neq Y \cup c \tag{10}$$

The process used to convert a constraint $\mathcal{C}$ in intermediate form (and different from false) into an equivalent constraint in explicit solved form is based on the recursive replacement of each conjunct of the form

$$c \equiv \underbrace{X_1 \cup \cdots \cup X_m \cup t_1 \cup \cdots \cup t_h}_{\ell} \neq \underbrace{Y_1 \cup \cdots \cup Y_n \cup s_1 \cup \cdots \cup s_k}_{r},$$

with the formula $\nu(c) \equiv \varphi_\ell \vee \varphi_r$, computed as: $\varphi_\ell \equiv \bigvee_{i=1}^{h} \varphi^i_{term} \vee \bigvee_{i=1}^{m} \varphi^i_{var}$ and

$$\begin{array}{rcl} \varphi^i_{term} & \equiv & \bigwedge_{j=1}^{k}(t_i \neq s_j) \ \wedge \ \bigwedge_{j=1}^{n}(Y_j \neq Y_j \cup t_i) \\ \varphi^i_{var} & \equiv & \bigwedge_{J \subseteq \{1,\ldots,k\}}(X_i \cup Y_1 \cup \cdots \cup Y_n \neq Y_1 \cup \cdots \cup Y_n \cup \bigcup_{r \in J} s_r) \end{array}$$

The definition of φ_r is perfectly symmetrical. Intuitively, φ^i_{term} asserts the fact that t_i is not one of the atoms of the r.h.s., while φ^i_{var} states that X_i is not a subset of the r.h.s.

The replacement of c with $\nu(c)$ can generate a number of new disequations (e.g., $s_1 \neq t_1$ with $s_1 \equiv f(\bar{t}_1)$ and $s_1 \equiv g(\bar{t}_2)$) that can be rewritten by the function int_simpl of Figure 5. However, all the subproblems generated are of *smaller* size and therefore the function expl_simpl described in Figure 6 will eventually terminate.

```
function expl_simpl($\varphi$);
    while there is a disequation $c \equiv r \cup s \neq t \cup u$ in $\varphi$ not in explicit s.f. do
        replace $c$ by $\nu(c)$
```

Figure 6: Rewriting procedure for explicit solved form

The next propositions formalize these results. Propositions 5.11 and 5.12 demonstrate that the transformation described in Figure 6 produces a constraint which is equivalent to the original one—i.e., the solutions space is not modified. Proposition 5.13 shows that the algorithm in Figure 6 terminates for any possible choice of initial constraints.

Proposition 5.11 *For a normalized disequation c of the form $X_1 \cup \cdots \cup X_m \cup t_1 \cup \cdots \cup t_h \neq Y_1 \cup \cdots \cup Y_n \cup s_1 \cup \cdots \cup s_k$ we have that $\mathbb{T}_{ACI1} \models \vec{\forall}(c \leftrightarrow \nu(c))$.*

Proof. (Sketch) Let us analyze the simple case of $e \equiv (X \cup t \neq Y \cup s)$, s and t terms with outermost symbol different from $\cup$. The complete proof can be derived through generalization of this case. For this equation, the formula in the proposition reduces to:

$$\begin{array}{rll} \nu(e) \equiv & (s \neq t \wedge Y \cup t \neq Y) & \vee \\ & (X \cup Y \neq Y \wedge X \cup Y \neq Y \cup s) & \vee \\ & (s \neq t \wedge X \cup s \neq X) & \vee \\ & (X \cup Y \neq X \wedge X \cup Y \neq X \cup t) & \end{array}$$

If we complement this formula we obtain

$$\begin{array}{llll} (s = t \vee Y \cup t = Y) & \wedge & (X \cup Y = Y \vee X \cup Y = Y \cup s) & \wedge \\ (s = t \vee X \cup s = X) & \wedge & (X \cup Y = X \vee X \cup Y = X \cup t) & \end{array} \tag{11}$$

We prove that $X \cup t = Y \cup s$ is indeed equivalent to formula (11).

It is easy to show that the formula (11) implies $X \cup t = Y \cup s$ by the analysis of each of the 16 disjuncts obtained by applying distributivity over (11). For instance, the first disjunct is: $s = t \wedge X \cup Y = Y \wedge s = t \wedge X \cup Y = X$, from which we have: $X \cup t = X \cup Y \cup t = X \cup Y \cup s = Y \cup s$.

To prove the converse, from (D_f) axiom we have that $X \cup t = Y \cup s$ if and only if:

$$\exists A, B \left(\begin{array}{ll} (Y = A \cup t \wedge X = A \cup s) & \vee \\ (s = B \cup t \wedge X = B \cup Y) & \vee \\ (Y = A \cup t \wedge s = t \cup B \wedge X = A \cup B) & \end{array} \right) \tag{12}$$

We can show that each disjunct of the formula (12) implies one disjunct in (11).

- $s = t \vee Y \cup t = Y$:
  1. $Y = A \cup t$ implies $Y \cup t = A \cup t \cup t = A \cup t$ which in turn implies $Y \cup t = Y$ (and, thus, $s = t \vee Y \cup t = Y$)
  2. $s = B \cup t$ implies $s = t$ (axiom (D_f)).
  3. $s = t \cup B$ implies $s = t$ (from axiom (D_f)).

- $X \cup Y = Y \vee X \cup Y = Y \cup s$:
  1. $Y = A \cup t \wedge X = A \cup s$ implies $X \cup Y = A \cup t \cup s = Y \cup s$, and $Y \cup s = A \cup t \cup s$. Thus, $X \cup Y = Y \vee X \cup Y = Y \cup s$
  2. $s = B \cup t \wedge X = B \cup Y$ implies $s = t \wedge (B = \emptyset \vee B = t) \wedge X = B \cup Y$ (axiom (D_f)); this implies $X = Y \vee X = Y \cup s$ which in turn implies $X \cup Y = Y \vee X \cup Y = Y \cup Y \cup s = Y \cup s$
  3. $Y = t \cup A \wedge s = t \cup B \wedge X = A \cup B$ implies

     $$s = t \wedge (B = \emptyset \vee B = t) \wedge X = A \cup B \wedge Y = A \wedge t$$

     this implies

     $$X \cup Y = A \cup t \cup B \wedge s = t \wedge (B = \emptyset \vee B = t) \wedge Y = A \wedge t$$

     which leads to

     $$X \cup Y = Y \vee X \cup Y = Y \cup s$$

- $s = t \vee X \cup s = X$:
  1. $Y = A \cup t \wedge X = A \cup s$ implies $X \cup s = A \cup s \cup s = A \cup s = X$ and thus $s = t \vee X \cup s = X$
  2. $s = B \cup t \wedge X = B \cup Y$ implies $s = t$ and thus $s = t \vee X \cup s = X$
  3. $Y = t \cup A \wedge s = t \cup B \wedge X = A \cup B$ implies $s = t$ and thus $s = t \vee X \cup s = X$

- $X \cup Y = X \vee X \cup Y = X \cup t$:
  1. $Y = A \cup t \wedge X = A \cup s$ implies $X \cup Y = A \cup s \cup t = X \cup t$ and thus $X \cup Y = X \vee X \cup Y = X \cup t$
  2. $s = B \cup t \wedge X = B \cup Y$ implies $s = t \wedge (B = \emptyset \vee B = t) \wedge X = B \cup Y$; this leads to $X \cup Y = Y \cup B \cup Y = Y \cup B = X$
  3. $Y = t \cup A \wedge s = t \cup B \wedge X = A \cup B$ implies $s = t \wedge (B = \emptyset \vee B = t) \wedge Y = A \cup t \wedge X = A \cup B$; this leads to

$$X \cup Y = A \cup B \cup A \cup t = A \cup B \cup t = X \cup B \wedge (B = \emptyset \vee B = t)$$

which finally leads to

$$X \cup Y = X \vee X \cup Y = X \cup t\,.$$

□

Proposition 5.12 *Given an intermediate solved form constraint $\mathcal{C}$, $\mathbb{T}_{ACI1} \models \vec{\forall}(\mathcal{C} \leftrightarrow \mathsf{expl\_simpl}(\mathcal{C}))$.*

Proof. This is an immediate consequence of Propositions 5.11 and 5.7. □

Theorem 5.13 *Given an intermediate solved form constraint $\mathcal{C}$, $\mathsf{expl\_simpl}(\mathcal{C})$ terminates after the application of $O(n^2)$ steps, adopting a structure-sharing technique.*

Proof. The idea is the same as that used in proving Proposition 5.7. When a constraint c is replaced by $\nu(c) \equiv \varphi_\ell \vee \varphi_r$, then a number of disequations that can be further reduced are introduced (those of the form $t_i \neq s_j$). The other disequations are already in explicit solved form and are not source of future computations.[3] We consider only φ_ℓ—in fact φ_r contains exactly the same disequations of φ_ℓ:

$$\begin{array}{rcl} \varphi_\ell & \equiv & \bigvee_{i=1}^{h} \bigwedge_{j=1}^{k} (t_i \neq s_j) \ \wedge \ \bigwedge_{j=1}^{n} (Y_j \neq Y_j \cup t_i) \vee \\ & & \bigvee_{i=1}^{m} \bigwedge_{J \subseteq \{1,\dots,k\}} (X_i \cup Y_1 \cup \cdots \cup Y_m \neq Y_1 \cup \cdots \cup Y_m \cup \bigcup_{r \in J} s_r) \end{array}$$

It is sufficient to extend the proof of Proposition 5.7 by analyzing the rewriting rule ($R3$) defined as

($R3$) for each unprocessed disequation $t_1 \cup \cdots \cup t_h \neq s_1 \cup \cdots \cup s_k$ generate $h \cdot k$ sons labeled $t_i \neq s_j$, $i = 1, \dots, h, j = 1, \dots, k$.

By inductive hypothesis, the disequation $t_i \neq s_j$ introduces at most $||t_i|| \cdot ||s_j||$ new disequations. We need to prove that:

$$h \cdot k + \sum_{i=1,j=1}^{h,k} ||t_i|| \cdot ||s_j|| \leq ||t_1 \cup \cdots \cup t_h \neq s_1 \cup \cdots \cup s_k|| = (\sum_{i=1}^{h} ||t_i|| + 2(h-1))(\sum_{j=1}^{k} ||s_j|| + 2(k-1))$$

that holds trivially since $h > 1$ and $k > 1$. □

Observe that $\nu(c)$ generates an exponential number of disequations. So, even if the number of steps is polynomial, the real complexity of the algorithm is exponential: $O(n^2 2^n)$.

Example 5.14 *Let us consider the constraint $f(X \cup a \cup g(Y), a) \neq f(X \cup Y \cup b, X)$ and assume that $X <_\# Y <_\# a <_\# b <_\# g$. The constraint is in implicit solved form; the corresponding disjunction of constraints in intermediate solved form is*

$$X \neq a \ \vee \ X \cup a \cup g(Y) \neq X \cup Y \cup b$$

[3]Observe that the latter equations, although inactive, are exponential in number.

and the corresponding disjunction of constraints in explicit solved form is

$$\begin{array}{lll} X \neq a \ \vee & X \neq X \cup a \wedge Y \neq Y \cup a \ \vee & X \neq X \cup g(Y) \wedge Y \neq Y \cup g(Y) \ \vee \\ X \neq X \cup b \ \vee & \multicolumn{2}{l}{X \neq X \cup Y \wedge X \cup Y \neq X \cup a \wedge X \cup Y \neq X \cup g(Y) \wedge X \cup Y \neq X \cup a \cup g(Y)} \end{array}$$

Observe that the components of the explicit normal form precisely identify the set of possible solutions. For instance, the second disjunct forces X and Y to be not of the form $a \cup s$ for any term s.

5.4 Subset Solved Form

The goal of this section is to provide an even more explicit representation of the set of solutions of a disequation constraint. In particular, we develop this additional solved form by considering the interpretation of $ACI1$ constraints on the join semilattices described earlier. This view has the advantage of providing a very intuitive interpretation of constraints, e.g., as formulae in a set-theoretical context, where the $ACI1$ operator $\cup$ is interpreted as the set-theoretic union operation. As described in Section 2.2, in each join-semilattice we can define a partial order $\leq$ using the $\bigvee$ operator. In the set theoretical context, the partial order obtained from $\cup$ corresponds to the traditional subset relationship $\subseteq$: $A \subseteq B$ if and only if $A \cup B = B$.[4]

Keeping this in mind, in this section we provide a method which allows to simplify the constraints of the form

$$X \cup Y_1 \cup \cdots \cup Y_h \neq Y_1 \cup \cdots \cup Y_h \cup r_1 \cup \cdots \cup r_k$$

to a collection of constraints of the form

$$Z \not\subseteq t_1 \cup \cdots \cup t_m,\ t \not\subseteq Z$$

where $p \not\subseteq q$ is an abbreviation for $q \neq q \cup p$. This new solved form is very appealing, as it provides a description of the solution in terms of constraints over individual variables (e.g., the variable Z above).

The new rewriting procedure that performs this additional simplification may introduce new variables. These variables are used to make explicit the existence of elements which belong to one set and not to the other—i.e., the complement of the extensionality principle for equality between sets.

Definition 5.15 *An explicit solved form constraint $\mathcal{C}$ is in* subset solved form *if it is* false, true, *or $\mathcal{C} \equiv (s_1 \neq t_1 \wedge \cdots \wedge s_n \neq t_n)$ and if for all $i = 1, \ldots, n$ $s_i \neq t_i$ is in one of the following forms:*

1. *$X \not\subseteq Y_1 \cup \cdots \cup Y_h$, $h > 0$ and $X \not\equiv Y_i$, or*
2. *$f(r_1, \ldots, r_m) \not\subseteq X$, or $X \not\subseteq f(r_1, \ldots, r_m)$, for $f \not\equiv \cup$, and X does not occur in $f(r_1, \ldots, r_m)$.*

Starting from an explicit solved form constraint, the subset solved form can be obtained by the application of the rewriting procedures subset_only and subset_simpl described respectively in Figure 7 and in Figure 8. The first procedure maps all the disequations into negations of inclusions, while the second procedure transforms them in *subset solved form.*

[4]Similarly, we can interpret the ACI operator as $\cap$ (set intersection). In this case, the $\leq$ partial order corresponds to the opposite of the set inclusion—$X \leq Y$ if and only if Y is a subset of X. With this view, the (1) axiom is satisfied, intuitively, by the universe U of all sets.

Example 5.16 subset_only *non-deterministically replaces the disequation in explicit solved form* $X \neq a$ *with the constraints in subset solved form* $X \not\subseteq a$ *and* $a \not\subseteq X$.

Consider now the disequation $X \cup Y \neq Y \cup a \cup g(X)$. subset_only *non-deterministically replaces it into:*

1. $a \not\subseteq X \cup Y$*: Rule* (1) *of* subset_simpl *rewrites it into the constraint in subset solved form:* $a \not\subseteq X \wedge a \not\subseteq Y$.
2. $g(X) \not\subseteq X \cup Y$*: Rule* (1) *of* subset_simpl *rewrites it into* $g(X) \not\subseteq X \wedge g(X) \not\subseteq Y$. *The first constraint is removed (replaced by* true*) by rule (9) of* subset_simpl*; the second is in subset solved form.*
3. $X \not\subseteq Y \cup a \cup g(X)$*: Rule* (2) *of* subset_simpl *rewrites it non-deterministically into:*
   (a) $X = a \cup A \wedge a \not\subseteq A \wedge A \not\subseteq Y \cup g(X)$. *Rule* (3) *applies the substitution* $[X/a \cup A]$ *obtaining* $a \not\subseteq A \wedge A \not\subseteq Y \cup g(a \cup A)$. *Again, rule* (2) *rewrites the latter into:*
      i. $A = g(a \cup A) \cup A' \wedge A' \not\subseteq Y$. *Rule* (8) *leads to failure.*
      ii. $g(a \cup A) \not\subseteq Y$. *We are in subset solved form.*
   (b) $a \not\subseteq X \wedge X \not\subseteq Y \cup g(X)$. *Rule (2) can be applied to the second constraint, leading to:*
      i. $X = g(X) \cup A \wedge g(X) \not\subseteq A \wedge A \not\subseteq Y$. *Rule* (8) *leads to failure.*
      ii. $g(X) \not\subseteq X \wedge X \not\subseteq Y$. *The first constraint is removed (replaced by* true*) by rule (9) of* subset_simpl*; the other is in subset solved form.*

```
function subset_only($\varphi$):
    while there is a disequation $c$ in $\varphi$ do
        replace $c$ according to the corresponding rule:

$X \cup Y_1 \cup \cdots \cup Y_h \neq Y_1 \cup \cdots \cup Y_h \cup r_1 \cup \cdots \cup r_k$ }   $\mapsto$   one of
                                          (1)       $r_1 \not\subseteq X \cup Y_1 \cup \cdots \cup Y_h$
                                           ⋮          ⋮
                                          ($k$)     $r_k \not\subseteq X \cup Y_1 \cup \cdots \cup Y_h$
                                          ($k+1$)   $X \not\subseteq Y_1 \cup \cdots \cup Y_h \cup r_1 \cup \cdots \cup r_k$
```

Figure 7: Rewriting procedure for eliminating $\neq$-constraints

Proposition 5.17 *Given an explicit solved form constraint* $\mathcal{C}$*, let* $\mathcal{C}_1, \ldots, \mathcal{C}_h$ *be the subset solved form constraints returned by* subset_simpl(subset_only($\mathcal{C}$)). *Then* $\mathcal{H} \models \bar{\forall}(C \leftrightarrow \bigvee_{i=1}^{h} \exists \bar{X}_i\, \mathcal{C}_i)$*, where* $\{\bar{X}_i\} = vars(\mathcal{C}_i) \setminus vars(\mathcal{C})$.

Proof. We show that soundness and completeness hold for each rewriting rule.

The function subset_only applies only one rule. Assume, for the sake of simplicity, that the constraint c is of the form:

$$X \cup Y \neq Y \cup r$$

Therefore, the rewriting rule in the function subset_only produces non deterministically the two constraints:

$$X \not\subseteq Y \cup r \tag{13}$$

$$r \not\subseteq X \cup Y \tag{14}$$

Any valuation σ of the variables of c over $\mathcal{H}$ has the general form:

```
function subset_simpl($\mathcal{C}$);
  while there is a $\not\subseteq$-constraint $c$ in $\mathcal{C}$ not in subset solved form do
    replace $c$ according to the corresponding rewriting rule:
```

(1)	$\left.\begin{array}{r} f(\bar{r}) \not\subseteq s_1 \cup \cdots \cup s_n \\ n > 1 \end{array}\right\}$	$\mapsto$	$\bigwedge_{i=1}^{n} f(\bar{r}) \not\subseteq s_i$
(2)	$X \not\subseteq Y_1 \cup \cdots \cup Y_h \cup r_1 \cup \cdots \cup r_k$	$\mapsto$	one of (i) $X = r_1 \cup A \wedge r_1 \not\subseteq A \wedge A \not\subseteq Y_1 \cup \cdots \cup Y_h \cup r_2 \cup \cdots \cup r_k$ (ii) $r_1 \not\subseteq X \wedge X \not\subseteq Y_1 \cup \cdots \cup Y_h \cup r_2 \cup \cdots \cup r_k$
(3)	$\left.\begin{array}{r} X = r \cup A \wedge C \\ X \notin vars(r), X \in vars(C \setminus \{c\}) \end{array}\right\}$	$\mapsto$	$(C \setminus \{c\})[X/A \cup r] \wedge X = A \cup r$
(4)	$r \not\subseteq r$	$\mapsto$	false
(5)	$f(\bar{r}) \not\subseteq g(\bar{s})$	$\mapsto$	true
(6)	$\left.\begin{array}{r} f(r_1, \ldots r_m) \not\subseteq f(s_1, \ldots, s_m) \\ m > 1 \end{array}\right\}$	$\mapsto$	one of $\begin{array}{ll} (i) & r_1 \not\subseteq s_1 \\ \vdots & \vdots \\ (m) & r_m \not\subseteq s_m \end{array}$
(7)	$\left.\begin{array}{r} r_1 \cup \cdots \cup r_m \not\subseteq s \\ m > 1 \end{array}\right\}$	$\mapsto$	one of $\begin{array}{ll} (i) & r_1 \not\subseteq s \\ \vdots & \vdots \\ (m) & r_m \not\subseteq s \end{array}$
(8)	$\left.\begin{array}{r} X = r \cup A \wedge C \\ X \in vars(r) \end{array}\right\}$	$\mapsto$	false
(9)	$f(\cdots X \cdots) \not\subseteq X$	$\mapsto$	true
(10)	$X \not\subseteq f(\cdots X \cdots)$	$\mapsto$	$X \not\subseteq \emptyset$
(S_1)	true $\wedge\, \mathcal{C}$	$\mapsto$	$\mathcal{C}$
(S_2)	false $\wedge\, \mathcal{C}$	$\mapsto$	false

Figure 8: Rewriting procedure for subset solved form

- $\sigma(X) = f_1(\bar{s}_1) \cup \cdots \cup f_a(\bar{s}_a)$, ($\sigma(X) = \emptyset$ when $a = 0$) for some $f_i \in \Sigma \setminus \{\emptyset, \cup\}$
- $\sigma(Y) = g_1(\bar{t}_1) \cup \cdots \cup g_b(\bar{t}_b)$, ($\sigma(Y) = \emptyset$ when $b = 0$) for some $g_i \in \Sigma \setminus \{\emptyset, \cup\}$
- $\sigma(r) = h(\bar{u})$, for some $h \in \Sigma \setminus \{\emptyset, \cup\}$

If σ is a successful valuation for c, then it must be the case that:

1. one term of the form $f_i(\bar{s}_i)$ is different from all terms of the form $g_j(\bar{t}_j)$ and from $h(\bar{r})$, or
2. the term $h(\bar{u})$ is different from all terms of the form $f_j(\bar{s}_j)$ and $g_j(\bar{t}_j)$.

In the first case, the valuation σ satisfies also the constraint (13), while in the second case it satisfies also the constraint (14).

On the other hand, if σ is a successful valuation for the constraint (13), then $\sigma(X)$ contains at least one atom $f_i(\bar{s}_i)$ not occurring in $\sigma(Y \cup r)$. Hence, c is satisfied by σ. If σ is a successful valuation for the constraint (14), the situation is analogous.

We now prove the soundness and completeness of the rules in the function subset_simpl.

(1) This rule is sound and complete, since $f(\bar{r})$ is an atomic element of the domain.

(2) Let us assume that c is of the form $X \not\subseteq Y \cup r$, i.e. $X \cup Y \cup r \neq Y \cup r$. The rule rewrites it into:

$$X = r \cup A \wedge r \not\subseteq A \wedge A \not\subseteq Y \tag{15}$$

$$r \not\subseteq X \wedge X \not\subseteq Y \tag{16}$$

Any valuation σ the terms of c has the form:

- $\sigma(X) = f_1(\bar{s}_1) \cup \cdots \cup f_a(\bar{s}_a)$, ($\sigma(X) = \emptyset$) for some $f_i \in \Sigma \setminus \{\emptyset, \cup\}$
- $\sigma(Y) = g_1(\bar{t}_1) \cup \cdots \cup g_b(\bar{t}_a)$, ($\sigma(Y) = \emptyset$) for some $g_i \in \Sigma \setminus \{\emptyset, \cup\}$
- $\sigma(r) = h(\bar{u})$, for some $h \in \Sigma \setminus \{\emptyset, \cup\}$

Let us assume that σ satisfies c. Two cases are possible: the term $h(\bar{u})$ is equivalent to one of the terms $f_j(\bar{s}_j)$; the term $h(\bar{u})$ is not equivalent to any of the terms $f_j(s_j)$. In the first case we can assume that $h(\bar{u}) \equiv f_1(\bar{s}_1)$. Since σ satisfies c, it must be the case that there is a term of the form $f_j(\bar{s}_j)$, with $j = 2, \ldots, a$, which is not equivalent to any of the terms in $\sigma(Y)$. We can extend the valuation σ to the new variable A defining $\sigma(A) = f_2(\bar{s}_2) \cup \cdots \cup f_a(\bar{s}_a)$. It is immediate to verify that this extended valuation satisfies the constraint (15). In the second case, the atom $h(\bar{u})$ is not between the atoms in $\sigma(X)$, and hence, since σ satisfies c, there must be an atom in $\sigma(X)$ which is not in $\sigma(Y)$. This means that σ satisfies also the constraint (16).

If σ satisfies the constraint (15), then there is an atom in $\sigma(A)$, and hence in $\sigma(X)$, which is different from $h(\bar{u})$, and which is not in $\sigma(Y)$. Hence, σ satisfies also c.

If σ satisfies the constraint (16), then there is an atom in $\sigma(X)$ which is different from $h(\bar{u})$ and which is not in $\sigma(Y)$. This means that σ satisfies also c.

(3) This is justified by equality.

(4) In this case the proof is immediate, since $\cup$ is idempotent.

(5), (6) In these cases the soundness and completeness follow trivially, since the l.h.s. of the rules involve terms which can be only instantiated to atoms.

(7) We can see all the $\sigma(r_i)$ and $\sigma(s_j)$ as collections of atoms and prove the soundness and completeness in a set theoretic way.

(8), (9) They are justified by axiom schema (F_3).

(10) For one direction, observe that, by axiom (1), $\emptyset \cup f(\bar{s}) = f(\bar{s})$ for any term $f(\bar{s})$. Assume now $X \neq \emptyset$. Then $X \cup f(\cdots X \cdots) = f(\cdots X \cdots)$ if and only if $X = f(\cdots X \cdots)$. But this is forced to be false by axiom (F_3).

□

Finally, we need to prove that any non-deterministic execution of $\mathsf{subset\_simpl}(\mathsf{subset\_only}(\mathcal{C}))$, where $\mathcal{C}$ is an explicit form constraint, terminates and it returns a subset solved form constraint.

Without affecting soundness and completeness, we impose a weak form of determinism to the execution process. We assume that each execution of the rule $(2i)$ is followed by all the possible consecutive executions of the rules (3) or (8), (1), and (4) in this order.[5]

Theorem 5.18 *Given an explicit solved form constraint $\mathcal{C}$, then any non-deterministic execution of* $\mathsf{subset\_simpl}(\mathsf{subset\_only}(\mathcal{C}))$ *can be implemented so as to terminate, returning a subset solved form constraint.*

Proof. It is immediate to observe that the function $\mathsf{subset\_only}(\mathcal{C})$ terminates: the function processes each literal of $\mathcal{C}$ once. The literals in $\mathsf{subset\_only}(\mathcal{C})$ are all of one of the two forms: $r \not\subseteq X \cup Y_1 \cup \cdots \cup Y_h$, and $X \not\subseteq Y_1 \cup \cdots \cup Y_h \cup r_1 \cup \cdots \cup r_k$ with $h, k \geq 0$.

The assumption that each execution of the rule $(2i)$ is followed by all the possible executions of the rules (3) (or (8)), (1), and (4) ensures that the maximum number of new variables introduced is $|vars(\mathcal{C})| \cdot m$, where m is the number of terms of $\mathsf{subset\_only}(\mathcal{C})$ in the kth of these literals (If action (8) is fired instead of action (3), then we will have immediate termination by failure). This follows from the following consideration. Each time we introduce a new variable A using the rule $(2i)$ we generate the literal $r_1 \not\subseteq A$. If we try to introduce a new variable A' using the rule $(2i)$ on the variable A and on the term r_1, then the application of (3) gives us the literal $r_1 \not\subseteq A' \cup r_1$, which is mapped by (1) into $r_1 \not\subseteq A' \wedge r_1 \not\subseteq r_1$, and this last produces false using the rule (4).

Hence, the rules $(2i)$ and (3) can be applied only a finite number of times and all the other rules make the complexity of the terms involved in the constraint decrease.

It is easy to prove that all the literals which are not in subset solved form are processed by one of the rules of $\mathsf{subset\_simpl}$, hence the algorithm produces a constraint in subset solved form. □

6 Examples

In this section we present some simple examples constructed using the framework presented in this paper. The first part collects examples which can be directly expressed using disunification problems, while the second part presents short programs developed using a constraint logic programming language embedding $ACI1$ constraints, called $CLP(ACI1)$. In the rest of this section we will use the notation $s \subseteq t$ to denote the fact that $t = s \cup t$.

6.1 Expressive Power of $ACI1$ Constraints

Let us start by considering the traditional map coloring problem. A map can be represented as a graph, where the nodes represent the different regions while the edges connect regions which are adjacent in the map. The objective is to color the graph using a certain number of colors and ensuring that no two adjacent regions receive the same color. We can encode this problem by representing each node in the graph with a distinct variable, e.g.,

$$\begin{array}{rcl} \mathsf{Nodes} & = & \{\mathsf{Czech}, \mathsf{Slovakia}, \mathsf{Poland}, \mathsf{Germany}\} \ \wedge \\ \mathsf{Edges} & = & \{\{\mathsf{Germany}, \mathsf{Czech}\}, \{\mathsf{Germany}, \mathsf{Poland}\}, \{\mathsf{Czech}, \mathsf{Poland}\}, \\ & & \{\mathsf{Poland}, \mathsf{Slovakia}\}, \{\mathsf{Czech}, \mathsf{Slovakia}\}\} \end{array}$$

[5]The restriction is imposed exclusively to keep the termination proof simple.

where the notation $\{t_1, \ldots, t_n\}$ is a short form for $\{t_1\} \cup \cdots \cup \{t_n\}$. The coloring of the map using colors $\mathsf{blue}, \mathsf{red}, \mathsf{white}$ can be obtained by stating that

$$\mathsf{Nodes} = \{\mathsf{blue}, \mathsf{red}, \mathsf{white}\}$$

and by requiring the absence of singleton sets from the colored set of edges:

$$\{\{\mathsf{blue}\}\} \not\subseteq \mathsf{Edges} \wedge \{\{\mathsf{red}\}\} \not\subseteq \mathsf{Edges} \wedge \{\{\mathsf{white}\}\} \not\subseteq \mathsf{Edges}$$

A possible solution is

$$\mathsf{Germany} = \mathsf{blue}, \mathsf{Poland} = \mathsf{white}, \mathsf{Czech} = \mathsf{red}, \mathsf{Slovakia} = \mathsf{blue}$$

Another simple problem that can be directly encoded is the problem of computing all the cycles of length n from a given directed or undirected graph. Let us consider the case of undirected graphs, represented as in the previous case as a set of nodes and a set of edges:

$$\mathsf{Nodes} = \{\mathsf{a}_1, \ldots, \mathsf{a}_n\} \wedge \mathsf{Edges} = \{\{\mathsf{a}_{\mathsf{i}_1}, \mathsf{a}_{\mathsf{j}_1}\}, \ldots, \{\mathsf{a}_{\mathsf{i}_k}, \mathsf{a}_{\mathsf{j}_k}\}\}$$

A cycle is a sequence of contiguous edges which starts and terminates at the same node. E.g., for $n = 3$:

$$\{\{\mathsf{X}_1, \mathsf{X}_2\}, \{\mathsf{X}_2, \mathsf{X}_3\}, \{\mathsf{X}_3, \mathsf{X}_1\}\} \subseteq \mathsf{Edges} \wedge \mathsf{X}_1 \neq \mathsf{X}_2 \wedge \mathsf{X}_2 \neq \mathsf{X}_3 \wedge \mathsf{X}_3 \neq \mathsf{X}_1$$

(assuming that we want to ignore self-loops in the graph).

Let us consider one more graph example. Given a graph, we would like to determine the *subgraph isomorphism* property, i.e., given two graphs G_1 and G_2 we would like to verify whether there exists a subgraph of G_2 which is isomorphic to G_1. This can be easily achieved if we represent the graph G_1 using variables to represent its nodes:

$$\mathsf{Nodes1} = \{\mathsf{N}_1, \ldots, \mathsf{N}_h\} \wedge \bigwedge_{\mathsf{i}<\mathsf{j}} \mathsf{N}_\mathsf{i} \neq \mathsf{N}_\mathsf{j} \wedge \mathsf{Edges1} = \{\{\mathsf{N}_{\mathsf{i}_1}, \mathsf{N}_{\mathsf{j}_1}\}, \{\mathsf{N}_{\mathsf{i}_2}, \mathsf{N}_{\mathsf{j}_2}\}, \ldots\}$$

Similarly we expect the graph G_2 to be represented in the usual form:

$$\mathsf{Nodes2} = \{\mathsf{a}_1, \ldots, \mathsf{a}_k\} \wedge \mathsf{Edges1} = \{\{\mathsf{a}_{\mathsf{i}_1}, \mathsf{a}_{\mathsf{j}_1}\}, \{\mathsf{a}_{\mathsf{i}_2}, \mathsf{a}_{\mathsf{j}_2}\}, \ldots\}$$

Finally, the computation of the subgraphs of G_2 which are isomorphic to G_1 is expressed by the solutions of

$$\mathsf{Nodes1} \subseteq \mathsf{Nodes2} \wedge \mathsf{Edges1} \subseteq \mathsf{Edges2}$$

Let us now consider a simple example from planning. In planning, it is common to impose constraints (e.g., temporal and procedural constraints [35]) on the structure of a planning trajectory, e.g., ordering of actions or restrictions on how often an action can be performed (e.g., due to lack of resources). The constraints can be easily encoded in our framework by representing a plan as a set of actions (each with its execution time—e.g., using the term occ(Action,Time)) and imposing constraints on such set. For example:

- if we want the plan to contain at least one occurrence of action a then we can write

$$\{\mathsf{occ}(\mathsf{a}, \mathsf{T})\} \subseteq \mathsf{Plan}$$

- if we require action a to be executed at least twice, then we can write:

$$\{\mathsf{occ(a,T1)},\mathsf{occ(a,T2)}\} \subseteq \mathsf{Plan} \wedge \mathsf{T1} \neq \mathsf{T2}$$

- if we want at time t that not all actions a and b and c to be executed

$$\{\mathsf{occ(a,t)},\mathsf{occ(b,t)},\mathsf{occ(c,t)}\} \not\subseteq \mathsf{Plan}$$

- if we are not willing to accept plans that start with action a and end with action b we can write:

$$\{\{\mathsf{occ(a,1)}\} \not\subseteq \mathsf{Plan} \wedge \{\mathsf{occ(b,n)}\} \not\subseteq \mathsf{Plan}$$

- if the plan cannot contain more than k occurrences of an action a the we can write

$$\{\mathsf{T}_1,\ldots,\mathsf{T}_\mathsf{n}\} = \{1,\ldots,\mathsf{n}\} \wedge \bigwedge_{\mathsf{i}=\mathsf{k}+1}^{\mathsf{n}} \{\mathsf{occ(a,T_i)}\} \not\subseteq \mathsf{Plan}$$

  if we additionally require exactly k occurrences we need to also add:

$$\{\mathsf{occ(a,T_1)},\ldots,\mathsf{occ(a,T_k)}\} \subseteq \mathsf{Plan}$$

Consider a simple scheduling problem (e.g., preparing a time-table); we have two workers (a, b) who can be employed in six possible time slots $(1, \ldots, 6)$; worker a can work only in time slots $1, 3, 5, 6$ while worker b can only work in time slots $1, 2, 4$. To accomplish the required job, a needs to work for three time slots, while b needs to work for two time slots. In addition, during each time slots only one worker can be employed. This problem can be encoded as follows:

$$\begin{array}{l}\mathsf{Pref_a} = \{1,3,5\} \wedge \mathsf{Pref_b} = \{1,2,4\} \ \wedge \\ \mathsf{Work_a} = \{\mathsf{A_1},\mathsf{A_2},\mathsf{A_3}\} \wedge \mathsf{Work_b} = \{\mathsf{B_1},\mathsf{B_2}\} \ \wedge \\ \mathsf{Work_a} \subseteq \mathsf{Pref_a} \wedge \mathsf{Work_b} \subseteq \mathsf{Pref_b} \ \wedge \\ \mathsf{Work_a} \cap \mathsf{Work_b} = \emptyset\end{array}$$

The $\cap$ constraint can be simply written as:

$$\{\mathsf{B_1}\} \not\subseteq \mathsf{Work_a} \wedge \{\mathsf{B_2}\} \not\subseteq \mathsf{Work_a}$$

We conclude this subsection with an example on Web Databases. According to [1], semistructured data can be represented by directed graphs with labeled edges and labeled leaf nodes (see Figure 9). The graph can be directly encoded by the two following sets (node numbers are assigned arbitrarily):

$$\begin{array}{lcl}\mathsf{E} & = & \{(0,1,\mathsf{name}),(0,2,\mathsf{father}),(0,3,\mathsf{father}),(0,4,\mathsf{tel}), \\ & & (2,5,\mathsf{name}),(2,6,\mathsf{tel}),(2,7,\mathsf{email}),(3,8,\mathsf{name}),(3,9,\mathsf{email})\} \\ \mathsf{N} & = & \{(1,\mathsf{michael}),(4,2345),(5,\mathsf{john}),(6,2143),(7,\mathsf{jsmith@libero.it}), \\ & & (8,\mathsf{mary}),(9,\mathsf{msmith@cs.nmsu.edu})\}\end{array}$$

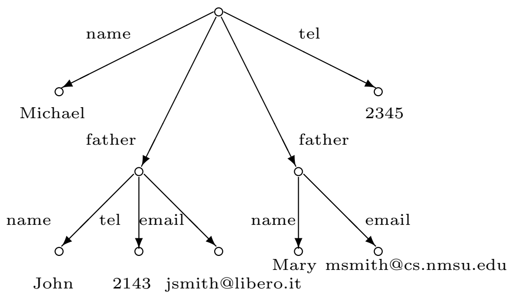

Figure 9: A semistructured database instance

The expressiveness of the $ACI1$ constraints allows us to specify queries directly in the constraint language. For instance, if we want to know the telephone number of john we can write:

$$\{(\mathsf{A},\mathsf{john}),(\mathsf{C},\mathsf{Telephone})\} \subseteq \mathsf{N} \wedge \{(\mathsf{B},\mathsf{name},\mathsf{A}),(\mathsf{B},\mathsf{tel},\mathsf{C})\} \subseteq \mathsf{E}$$

The answer will be collected in the variable Telephone. The possibility of expressing negative information allows us to write a query to find the name of the Siblings of john:

$$\{(\mathsf{A},\mathsf{john}),(\mathsf{E},\mathsf{Sibling})\} \subseteq \mathsf{N} \wedge \{(\mathsf{B},\mathsf{name},\mathsf{A}),(\mathsf{C},\mathsf{father},\mathsf{B}),(\mathsf{C},\mathsf{father},\mathsf{D}),(\mathsf{D},\mathsf{name},\mathsf{E})\} \subseteq \mathsf{E} \wedge \mathsf{B} \neq \mathsf{D}$$

In these last examples we relied exclusively on the power of the constraint solver. More expressive queries (e.g., those involving transitive closures) can be encoded through the use of recursion, e.g., in the context of a language such as the $CLP(ACI1)$ language described next.

6.2 Using the Language $CLP(ACI1)$

One of the objectives of the research presented in this paper is the development of the constraints solving algorithms for the development of a constraint logic programming framework [36] dealing with different representations of sets. We will refer to this framework as $CLP(ACI1)$.

Let us assume that a graph is represented in a CLP program by the predicates

- nodes($Name$, $Nodes$) where $Name$ is the name of the graph and $Nodes$ is the set of nodes in the graph;
- edges($Name$, $edge$) where $Name$ is the name of the graph and $edge$ is one of the edges of the graph.

This is a standard representation technique used in CLP [36]. To define the *join* $a_1 + a_2$ of two graphs a_1, a_2 we can use the following program:

```
nodes($Name_1 + Name_2, Nodes_1 \cup Nodes_2$) :−
    nodes($Name_1, Nodes_1$),
    nodes($Name_2, Nodes_2$).
edges($Name_1 + Name_2, \{V, W\}$) :−
    edges($Name_1, \{V, W\}$).
edges($Name_1 + Name_2, \{V, W\}$) :−
    edges($Name_2, \{V, W\}$).
edges($Name_1 + Name_2, \{V, W\}$) :−
    nodes($Name_1, Nodes_1$), nodes($Name_2, Nodes_2$),
    $\{V\} \subseteq Nodes_1$, $\{W\} \subseteq Nodes_2$.
```

For the next example, let us extend the considerations about planning from the previous section. The use of $CLP(ACI1)$ allows to express more powerful constraints on trajectories. For example, if we want to forbid action b to follow action a, we can execute the goal $\mathsf{not\_succ}(a, b, Plan, n)$, with respect to the program:

```
$\mathsf{not\_succ}(Xa, Xb, Plan, 0).$
$\mathsf{not\_succ}(Xa, Xb, Plan, N)$ :−
    $N > 0,$
    $\{\mathsf{occ}(Xa, N), \mathsf{occ}(Xb, N + 1)\} \not\subseteq Plan,$
    $\mathsf{not\_succ}(Xa, Xb, Plan, N - 1).$
```

Let us now consider the combinatorial problem of computing Schur Numbers [8]. Consider the presence of N bins and suppose that each bin can contain a set of numbers as long as the sum of no two numbers in the same bin produces another number in the bin. The problem is to determine for a given N what is the largest value of M such that all numbers between 1 and M can be partitioned in N bins without violating the stated condition.[6]

If we want to attempt to construct a placement of the numbers up to M in 4 bins, we can use the following goal

$$\leftarrow \mathsf{schur4}(\{1, 2, \ldots, \mathsf{M}\}, \mathsf{P}).$$

The predicate schur4 can be defined as follows:

```
schur4(Set,Partition) :−
    Set = $A_1 \cup A_2 \cup A_3 \cup A_4$,
    Partition = $\{A_1, A_2, A_3, A_4\}$,
    disj(Partition),
    check_sum(Partition).
disj($S$) :−
    $(\forall A \in S)(\forall B \in S)(\forall X \in A)(X \notin B).$
check_sum($S$) :−
    $(\forall A \in S)(\forall X \in B)(\forall Y \in B)(\{X + Y\} \not\subseteq B).$
```

In this program we have used restricted universal quantifications, that is very natural when programming with sets. We have shown in [18] how to encode this form of quantifications using membership and equality constraints over sets and recursion. Membership and equality constraints can be seen as special cases of the constraints discussed in this paper.

We conclude this section by presenting an example from [13], where $ACI1$-unification (equality constraints) is used for checking *type dependencies* during compile-time program analysis ([13] uses the $ACI1$ symbol $\oplus$ instead of $\cup$).

Consider the following PROLOG clauses:

```
nat(0).
nat(s(X)) :− nat(X).
append([ ], Y, Y).
append([A | X], Y, [A | Z]) :− append(X, Y, Z)
```

[6]There is currently no known solution to this problem for values of n greater than 4 [8].

In [13] types are assigned recursively to terms using the $ACI1$ operator. In the case above, we have that:

$$\begin{array}{rcl}\text{type}(\mathsf{0}) & = & \text{nat}\\ \text{type}(\mathsf{s}(\mathsf{t})) & = & \text{nat} \cup \text{type}(\mathsf{t})\\ \text{type}([\,]) & = & \text{list}\\ \text{type}([\mathsf{h} \mid \mathsf{t}]) & = & \text{list} \cup \text{type}(\mathsf{t})\end{array}$$

So, for instance,

$$\text{type}([\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0}))]) = \text{list} \cup \text{list} \cup \text{list} = \text{list}$$

If a term is non-ground, then its type can be partially undefined. Using this approach, the authors show how to replace a program with another set of clauses constructed with the type information. For instance, the clauses for append become:

$$\begin{array}{l}\mathsf{append}(\text{list}, \mathsf{Y}, \mathsf{Y})\\ \mathsf{append}(\text{list} \cup \mathsf{X}, \mathsf{Y}, \text{list} \cup \mathsf{Z}) :- \mathsf{append}(\mathsf{X}, \mathsf{Y}, \mathsf{Z})\end{array}$$

These clauses manipulate *types* as data, and allow to describe the type dependencies imposed by the append clauses. The program obtained is a $CLP(ACI1)$ program.

7 Discussion and Related Work

In this section we present some observations regarding how our results extend to the cases of nested sets (Section 7.1). Section 7.2 compares our results with similar proposals presented in the literature.

7.1 Nested Sets

The algorithms in this paper, as well as any other algorithm for dealing with $ACI1$-constraints when Σ contains at least a function symbol f of arity greater than 0, can be used to solve constraints on hereditarily finite and hybrid sets (where hybrid means that also terms built using "free" symbols—atoms—can be used as elements of sets). As a matter of fact, using a unary function symbols, say $\{\cdot\}$, as done in Example 4.6, sets with any level of nesting can be written. Thus, we can encode problems such as:

$$\{X, \{X, \emptyset, \{Y\}\} \cup Z \cup W\} \neq \{Z, \{\{\{X \cup Y\}, \emptyset\}\}\} \cup W$$

Set inclusion can be easily encoded by $X \subseteq Y \leftrightarrow Y = X \cup Y$. Moreover, with nested sets, membership becomes very important. This operation can be encoded as: $X \in Y \leftrightarrow Y = \{X\} \cup Y$.

In order to show that general signatures are needed to work with sets, let us consider a set equality problem of the type:

$$\{s_1, \ldots, s_m\} = \{t_1, \ldots, t_n\}$$

If $s_1, \ldots, s_m, t_1, \ldots, t_n$ are constants, then there is an equi-satisfiable $ACI1$-unification problem with constants. For instance, the two problems

$$\{a, b, b\} = \{b, a\}, \{a, b\} = \{b, b\}$$

can be encoded by:

$$a \cup b \cup b = b \cup a, a \cup b = b \cup b$$

If we are not in this simple case (e.g., when the s_i, t_j are variables or complex terms, possibly involving other sets) it is rather unnatural or impossible to express them in a non-general $ACI1$ framework. For instance, consider the matching problem

$$\{X_1, \dots, X_m\} = \{c_1, \dots, c_n\}$$

Its "natural" encoding

$$X_1 \cup \cdots \cup X_m = c_1 \cup \cdots \cup c_n$$

admits more solutions. Any solution of the former is a solution of the latter but not vice versa. For instance, $X_1 = \emptyset, \dots, X_{n-1} = \emptyset, X_n = c_1 \cup \cdots \cup c_n$ is a solution of the latter, but not of the former.

However, it is possible to automatically add constraints to the latter in order to prune undesired solutions. For instance, if $m = 3$ and $n = 2$ we need to add the constraints:

$$\bigwedge_{i=1}^{3} X_i \neq \emptyset \wedge X_i \neq c_1 \cup c_2$$

Thus, introducing a suitable number of constraint a simple set unification problem can be mapped into an exponential size $ACI1$-constraint with constants.

It is possible to prove [20] the existence of set-equality problems that cannot be encoded at all in a non-general setting without using universal quantifications. For example, consider the equality $X = \{Y\}$. In all its solutions X is mapped to a singleton set. In [20] it is proved that no equi-satisfiable formula of the type $\exists Z_1 \cdots Z_m(\varphi)$, where φ is a quantifier-free formula built with literals based on $\emptyset, \cup, =, \in$ (thus, a richer language), can be constructed. Constraints of this form are special cases of formulae of *multi-level syllogistic*, whose decision problem is discussed in [10].

7.2 Related Works

In [15] Comon studies the problem of determining adequate solved forms for disunification problems in the context of quotient algebras $T(\Sigma)/ =_E$ for various classes of equational theories E. Comon identifies a class of formulae, called *compact formulae.* All the four solved forms presented in this paper satisfy the requirement of being compact formulae. Additionally, Comon proves that *compact* equational theories, i.e., theories for which:

- E-unification is finitary and decidable
- each satisfiable equation $s = t$, such that $vars(s, t) = \{X\}$ and such that $s \neq_E t$, admits a finite number of solutions in $\mathcal{H}$

guarantee that every compact formula distinct from false is satisfiable in $T(\Sigma)/ =_E$ [15].

However, $ACI1$ is not compact, since equations of the type $X = X \cup a$ do not admit a finite set of solutions if $T(\Sigma)/ =_{ACI1}$ is infinite—which is always the case when Σ is general. Nevertheless, we have demonstrated in the previous sections how to reduce an arbitrary disunification problem to a compact form, as well as the fact that the specific compact forms considered in our context are always satisfiable (Corollary 5.1). Thus, $ACI1$ represents a good example to indicate that compactness of the equational theory is a sufficient but *not necessary* condition for the satisfiability of formulae in compact form.

Bürckert [9] introduces a general scheme for solving disunification problems in the context of an arbitrary equational theory E. Solutions of disunification problems are described through the use of *substitutions with exceptions*, i.e., entities of the form (σ, Ψ) where σ is a substitution and Ψ a set of substitutions. An actual solution to the disunification problem is represented by any instantiation of σ which is not an instantiation of any of the substitutions in Ψ. In the context of a theory E which is finitary with respect to unification, the set of all solutions of a disunification problem can be represented using a finite set of substitutions with exceptions. Additionally, the Ψ component of each of them is guaranteed to be also finite. Substitutions with exceptions can be obtained from the solutions of a set of unification problems.

Nevertheless, this approach is not suitable to be used in the context of a *CLP* language. Each substitution with exceptions is equivalent to a formula of the type:

$$\exists \bar{W} \forall \bar{Y} (X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge W_1 \neq s_1 \wedge \ldots \wedge W_m \neq s_m)$$

where $\mathit{vars}(s_1, \ldots, s_m) = \bar{Y}$, $\mathit{vars}(t_1, \ldots, t_n) \cap \bar{Y} = \emptyset$ and $W_1, \ldots, W_m \in \bar{W}$. This leads to the generation of formulae with arbitrary quantifications, that are inadequate to a *CLP* framework. Moreover, to guarantee the existence of solutions it is necessary to verify whether σ is an E-instance of any substitution in Ψ—as stems from the *Inconsistency Lemma* in [9]. This requires solving additional E-unification problems as well as having an *explicit* representation of Ψ.

In [5] Baader and Schulz develop a general technique capable of combining the satisfiability algorithms (based on substitutions with exceptions) for disjoint equational theories. The approach is general and can be applied to $ACI1$ as well. However, it provides a unique normal form that can be reached in exponential time. The solution of [5] introduces a great variety of new variables and opens a large number of alternatives. In particular, with this method one has to guess: a partition of the m variables present in the problem ($m \leq n$) into equivalence classes, a *linear ordering* over the variables (among the possible $m!$), and a *type information* for each variable, specifying to which theory E_0, E_1 the variable belongs to (2^m possible choices). This leads to an overall complexity—modulo the usual combinatorial approximations—of $\approx \sqrt{2}\left(n^{\frac{3n-2}{2}}\right)\left(\left(\frac{2}{e}\right)^{\frac{n-2}{2}}\right)$. Thus, in a particular case as that presented in this paper, it is reasonable to improve their constraint solver, developed for a universal framework. In the first two solved forms we do not introduce new variables and, using the implicit normal form we do not introduce disjunctions. Starting from a constraint made of disequations, the complexity of our approach seems to be more promising (e.g., $O(n^2)$ for the implicit normal form) and practical.

Example 7.1 *Let* $E_1 = ACI1$ *on* $\Sigma_1 = \{\emptyset, \cup\}$ *and* $E_2 = f(X) = f(X) \wedge c = c$ *on* $\Sigma_2 = \{c, f\}$, *and consider* $\mathcal{C}$: $X \cup f(Y) \neq V \cup f(Z)$. *To apply the algorithm of [5], it is necessary to transform* $\mathcal{C}$ *into the equivalent constraint* $\mathcal{C}'$: $A = f(Y) \wedge B = f(Z) \wedge X \cup A \neq V \cup B$, *where* A *and* B *are new variables. Then, all the partitions of the set* $\{A, B, X, Y, V, Z\}$ *of variables have to be taken into consideration. Variables in the same block of a partition are forced to be equal; however, not all the partitions lead to a solution of the problem. For instance, the partition*

$$\{\{X, V\}, \{Z, Y\}, \{A, B\}\}$$

cannot lead to a solution, independently from the other non-deterministic choices.[7]

Our proposal, on the contrary, first applies the function ρ *to the constraint* $\mathcal{C}$, *obtaining the same constraint. This constraint is already in implicit normal form, and thus it is satisfiable in*

[7]A solution derived from this partition would be a solution of: $A = f(X) \wedge A = f(Z) \wedge X \cup A \neq X \cup A$, which is clearly unsatisfiable.

every model of $ACI1F_1F_2F_3D_f$, thanks to Theorem 4.17. On demand, we can transform it in the explicit form (the first and the third disjuncts are also in the subset solved form):

$$\begin{array}{lll}(Y \neq Z \land V \neq V \cup f(Y)) & \lor & (X \cup V \neq V \land X \cup V \neq V \cup f(Z)) \quad \lor \\ (Z \neq Y \land X \neq X \cup f(Z)) & \lor & (V \cup X \neq X \land V \cup X \neq X \cup f(Y))\end{array}$$

in which the 4 *disjuncts are all satisfiable.*

In the context of *CLP* with sets, three major proposals have been presented in the literature. In [25], Gervet presents a language, called *Conjunto* which incorporates a constraint solver over boolean lattices built from (flat) set intervals. The constraints can be more complex (e.g., boolean constraints) than those considered in this context, but the domain is less general. In particular, the simulation of nested sets (see Section 7.1) is not possible—which prevents the direct encoding of many interesting problems. Conjunto has been embedded in the recent releases of the ECLiPSe system [42].

In CLPS [32] the authors use a normal form similar to the implicit one presented in this paper. On the other hand, their constraint solving mechanisms appear to be based on reducing the problem to standard forward-checking and look-ahead techniques. The limited literature on the topic prevents us from a deeper comparison with the capabilities of CLPS.

{log} [18, 19] is a constraint logic programming language over hybrid and hereditarily finite sets. Sets in {log} are represented using a more restricted construction—based on the use of the constant $\emptyset$ and the binary function symbol *with*. In particular, $\cup$ is not directly expressible but is derived through the use of recursive {log} definitions and from the axiomatization of *with*. An extension of this language to handle $\cup$ as a primitive constraint has also been explored [19]. Unification in this context is still NP-complete and can be seen as an instance of the cases analyzed in this paper. Disunification is relatively simpler: the constraint solvers developed for {log} [18] are capable of handling both equalities and disequations, leading to a solved form containing only primitive constraints of the form $X = t$ and $Y \neq s$, where X, Y are variables, X occurs only once in the resulting constraint, and Y does not appear in s.

We have investigated the possibility of transforming our constraints in explicit solved form (Section 5.3) in the same solved form as the one used by {log}. This task requires the ability of forcing variables to assume only atoms as values, something which cannot be explicitly expressed in the context of a possibly unbound signature. This can be achieved through the introduction of an additional type of constraint: `kernel` (see also [18]) which is satisfied by all the atoms of the lattice used for the interpretation of the $ACI1$ theory.

A considerable amount of research has also been performed in the area of set-constraints [3, 30]. Set-constraints are conjunctions of literals based on the predicate $\subseteq$ and the *constant symbols* $\underline{\mathbf{0}}, \underline{\mathbf{1}}$, and a non-empty set of first-order constant symbols (e.g., a, b); *function symbols:* $\cup, \cap, \complement$, where $\complement$ stands for the complement operator, and, possibly, an additional set of free function symbols (e.g., f, g). If a function symbol f is available, then also its projection f_i^{-1} on the ith argument can be used as term-constructors.

The domain on which satisfiability of set-constraints must be checked is the powerset of $T(\Sigma)$, with Σ containing all symbols above but $\underline{\mathbf{0}}, \underline{\mathbf{1}}, \cup, \cap, \complement, f_i^{-1}$. Moreover, the interpretation for the functions and their projection is fixed:

$$\begin{array}{rcll}(f(a_1, \dots, a_n))^{\mathcal{P}} & = & \{f(t_1, \dots, t_n) \,:\, \bigwedge_{i=1}^{n} t_i \in a_i^{\mathcal{P}}\} & \text{for } f \in \Sigma \\ (f_i^{-1}(a))^{\mathcal{P}} & = & \{t_i \,:\, \exists t_1, \dots, t_n \, f(t_1, \dots, t_n) \in (a)^{\mathcal{P}}\} & \text{for } f \in \Sigma\end{array}$$

$(\_)^{\mathcal{P}}$ is extended to set-based operators in the intuitive way. In particular, the interpretation is a model of $\cup$. Thus, the interpretation domain must be a join-semilattice with bottom. If we forbid the use of symbols $\cap, \underline{\mathbf{1}}, \complement$, and the projection symbols, the work in the area of set constraint can be seen as a work on solving $ACI1$ constraints on particular domains.

8 Complexity Analysis

In this section we prove that the satisfiability problem for $ACI1$ constraints in the *general* case (i.e., when free function symbols are allowed in the signature) in the domain $\mathcal{H}$ is NP-complete. *NP-hardness* follows from [28]. We here recall briefly a simple way to reduce 3-SAT to the "set" unification (as in [18]). Consider the instance of 3-SAT:

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

Introducing variables Y_1, Y_2, Y_3 for $\neg X_1, \neg X_2, \neg X_3$ respectively, it is immediate to see that solving the following $ACI1$ unification problem ($\{\cdot\}$ is the free singleton operator, false and true are two distinct constant symbols that can be replaced, for instance, by $\emptyset$ and $\{\emptyset\}$, $\{t_1, \dots, t_n\}$ is a short form for $\{t_1\} \cup \cdots \cup \{t_n\}$) is equivalent to checking satisfiability of φ:

$$\begin{array}{c} \{\{X_1, Y_1\}, \{X_2, Y_2\}, \{X_3, Y_3\}, \{X_1, Y_2, X_3, \mathsf{false}\}, \{X_2, Y_3, X_1, \mathsf{false}\}, \{X_3, Y_1, X_2, \mathsf{false}\}\} \quad = \\ \{\{\mathsf{false},\ \mathsf{true}\}\} \end{array}$$

To prove *NP-completeness* we can prove that each $ACI1$-constraint problem can be polynomially reduced to a decision problem over sets involving open formulae. The decision problem for the latter class of formula is NP-complete [10]. This is done in Appendix A. The algorithm we have presented in this paper is polynomial as far as pure disequation constraints are concerned. For a generic $ACI1$ constraint, since we first solve equality constraints, by making use of general $ACI1$ unification, and then apply a substitution obtained to the remaining part of the constraint, we can fall outside of NP because of the size explosion of terms due to application of substitution.

The combined effect of the results in Theorem 4.12 and in Corollary 4.18 allows us to decide the satisfiability of every constraint in the general theory in every model. This property is commonly called *satisfaction completeness* [27]. Satisfaction completeness represent a weak form of completeness for the formula we are interested in.

As far as we know, the decidability problem for arbitrary first-order formulas of the equational theory $ACI1$ is not yet studied. Since, having one unary function symbol, say $\{\cdot\}$, it is possible to encode the basics of set theory (empty set $\emptyset$, the union symbol $\cup$, and the membership predicate $x \in y \leftrightarrow Y = Y \cup \{X\}$) our feeling is that testing the satisfiability of a generic first-order $ACI1$-formula over the domain $\mathcal{H}$ is undecidable. But a careful proof should be done.

9 Conclusion

In this paper we have studied the problem of verifying the satisfiability of conjunctions of equations and disequations with respect to an $ACI1$ theory. The ability to efficiently verify the satisfiability of this class of formulae is vital to the development of more general and effective CLP languages embedding sets. Existing results in the area of E-disunification (e.g., [15, 9, 5, 14, 23, 29]) present general techniques to solve these problems, but such techniques are either inadequate to the needs of a CLP framework (e.g., [9, 5]) or unsuitable to the characteristics of $ACI1$ equational theories (e.g., [15]).

The contributions of this paper can be summarized as follows:

- We have characterized the class of structures which are suitable to model $ACI1$-like theories; in particular, we have related the form of the structure to the problem of verifying satisfiability of the disunification problem.
- We have provided complexity results for the problem of verifying satisfiability of elementary disunification and disunification with constants.
- In the case of disunification under a general signature (i.e., a signature with free non-constant function symbols), we have characterized the axiomatization which captures the desired properties, and which corresponds to the "standard" $T(\Sigma)/=_{ACI1}$ model.
- We have proposed four solved forms. Each solved form provides a different trade-off between computational complexity and explicit representation of the set of solutions. We have developed algorithms to compute the equivalent solved form for arbitrary conjunctions of disequations. Each solved form can be trivially tested for satisfiability. Furthermore two of the four normal forms can be efficiently computed and tested in polynomial time.

As future work, we will continue to explore the issue of solved forms by extending the discussion to embrace different $ACI1$ operators. E.g., we are interested in dealing with languages which include a second $ACI1$ operator $\cap$, related to $\cup$ in the same way as $\bigvee$ is related to $\bigwedge$ in lattice structures. This will allow us to extend our set-theoretic operations to more complex set representations, including both union and intersections. We will also explore the relationship between the solved form adopted in this work and those adopted in previous proposals [19].

Since our goal is to provide constraint solving algorithms which can be integrated in the context of a constraint logic programming framework, the issue of efficiency has to be taken into account. In spite of the general high complexity of the problem at hand, as discussed in Section 8, we have pointed out that there are solved forms for disequation constraints that can be computed very efficiently; the efficiency depends heavily on the choice of the correct data structures to represent terms and constraints (e.g., use of structure sharing and term factoring). In our future work we propose to continue this study and to investigate the practical issues leading to an effective implementation, along with its integration within a Constraint Logic Programming system.

Acknowledgments

This work would have not been possible without the constant help and support of Gianfranco Rossi. The authors would also like to thank Roberto Giacobazzi and Desh Ranjan for the help on this paper, and the anonymous referees that greatly helped in improving the presentation of the paper.

A. Dovier is partially supported by MIUR project: *Aggregate—and number—reasoning for computing*. E. Pontelli is partially supported by NSF grants CDA9729848, EIA9810732, CCR9900320, HRD9906130, and CCR9875279.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web. From Relations to Semistructured Data and XML* Morgan Kaufmann Publishers, 2000.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[3] A. Aiken. Set Constraints: Results, Applications and Future Directions. Technical report, University of California, Berkeley, 1994.

[4] F. Baader and W. Büttner. Unification in Commutative and Idempotent Monoids. *Theoretical Computer Science*, 56:345–352, 1988.

[5] F. Baader and K. U. Schulz. Combination Techniques and Decision Problems for Disunification. *Theoretical Computer Science*, 142:229–255, 1995.

[6] F. Baader and K. U. Schulz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. *Journal of Symbolic Computation*, 21:211–243, 1996.

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

[8] A. Beutelspacher and W. Brestovansky. Generalized Schur Numbers. In *Combinatorial Theory*, Springer Verlag, Lecture Notes in Mathematics, pp. 30-38, 1982.

[9] H.-J. Bürckert. Solving Disequations in Equational Theories. In E. L. Lusk and R. A. Overbeek, editors, *CADE 1988*, *Lecture Notes in Computer Science 310*, pages 517–526. Springer-Verlag, Berlin, 1988.

[10] D. Cantone, E. G. Omodeo, A. Policriti. *Set Theory for Computing.* Springer Verlag, 2001.

[11] W. Charatonik and L. Pacholski. Negative Set Constraints with Equality. In *Proc. 9th Symp. Logic in Computer Science.* IEEE, 1994.

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

[13] M. Codish and V. Lagoon. Type Dependencies for Logic Programs using ACI-unification. *Theoretical Computer Science* 238(1–2):131–159 (2000).

[14] A. Colmerauer. Equations and Inequations on Finite and Infinite Trees. In *International Conference on Fifth Generation Computer Systems*, ICOT, pp. 85–99, 1984.

[15] H. Comon. Disunification: a Survey. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson.* The MIT Press, Cambridge, Mass., 1991.

[16] H. Comon and C. Kirchner. Constraint Solving on Terms. In *Constraints in Computational Logics*, Springer Verlag, pp. 47–103, 1999.

[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Mass., 1990.

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

[19] A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. Sets and constraint logic programming. *ACM Transaction on Programming Language and Systems*, 22(5):861–931, 2000.

[20] A. Dovier, C. Piazza, and A. Policriti. Comparing expressiveness of set constructor symbols. In H. Kirchner and C. Ringeissen, editors, *Frontier of Combining Systems 2000.* Lecture Notes in Computer Science 1794, pp. 275–289, Springer-Verlag, 2000.

[21] A. Dovier, E. Pontelli, and G. Rossi. Set Unification Revisited. NMSU-CSTR-9817, Dept. of Computer Science, New Mexico State University, USA, October 1998.

[22] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press Inc., 1972.

[23] M. Fernandez. Narrowing based Procedures for Equational Disunification. In *Applicable Algebra in Engineering, Communications, and Computing*, Springer Verlag, pp. 1–26, 1992.

[24] C. Fidge, I. Hayes, A. Martin, and A. Wabenhorst. A Set-theoretic Model for Real-Time Specification and Reasoning. In *Mathematics of Program Construction*, Springer Verlag, 1998.

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

[26] G. Grätzer. *General Lattice Theory.* Birkhäuser Verlag Basel und Stuttgart, 1978.

[27] J. Jaffar, M. Maher, K. Marriot, and P. Stuckey. The Semantics of Constraint Logic Programs. *Journal of Logic Programming*, 37:1–46, 1998.

[28] D. Kapur and P. Narendran. NP-Completeness of the Set Unification and Matching Problems, In J. H. Siekmann ed., 8th International Conference on Automated Deduction, Lecture Notes in Computer Science 230, pp. 489–495, Springer-Verlag, 1986.

[29] C. Kirchner and P. Lescanne. Solving Disequations. In *Proceedings of the Annual IEEE Symposium on Logic in Computer Science*, Ithaca, NY, pp. 347–352, 1987.

[30] D. Kozen. Logical Aspects of Set Constraints. In *Procs. Conf. on Computer Science Logic*, Vol. 832 Lecture Notes in Computer Science, pp. 175–188. Springer-Verlag, 1993.

[31] G. M. Kuper. Logic Programming with Sets. *Journal of Computer and System Science 41*, 1, pages 66–75, 1990.

[32] B. Legeard and E. Legros. Short Overview of the CLPS System. In *Proc. Third Int'l Symp. on Programming Language Implementation and Logic Programming*, Lecture Notes in Computer Science 528, pp. 431–433. Springer-Verlag, 1991.

[33] M. J. Maher. Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees. In *Proceedings of 3rd Symposium Logic in Computer Science*, pages 349–357, 1980.

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

[35] . R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems.* MIT Press, 2001.

[36] K. Marriott and P. Stuckey. *Programming with Constraints.* MIT Press, 1998.

[37] A. Robinson. *Introduction to Model Theory and to the Metamathematics of Algebra.* North Holland, Amsterdam, 1963.

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

[39] T. Soininen and I. Niemelä. Developing a Declarative Rule Language for Applications in Product Configuration. In *Symposium on Practical Aspects of Declarative Languages*, Springer Verlag, pp. 305–319, 1999.

[40] J.M. Spivey. *The Z Notation: A reference Manual, 2^{nd} edition.* International Series in Computer Science. Prentice Hall, 1992.

[41] M. Thielscher. Reasoning about Actions with CHR and Finite Domain Constraints. In *International Conference on Logic Programming*, Springer Verlag, pp. 70–84, 2002.

[42] M. Wallace, S. Novello, and J. Schimpf. ECLiPSe: a Platform for Constraint Logic Programming. IC-PARC, Imperial College, 1997.

[43] D. H. D. Warren. An Abstract Prolog Instruction Set. Technical Report 309, SRI International, 1983.

A NP-completeness of general $ACI1$-constraints

To prove *NP-completeness* we will make use of the model $\mathbb{HF}$ of hereditarily finite well-founded sets whose domain $\mathcal{U}$ can be inductively defined as $\mathcal{U} = \bigcup_{i \geq 0} U_i$ with $U_0 = \emptyset, U_{i+1} = \wp(U_i)$ (where $\wp$ stands for the power-set operator) and whose interpretation for $\mathcal{L} = \{\emptyset, \cup, \in, =\}$ is the natural one. Any open formula built using $\emptyset, \{\cdot\}, \cup, =, \in$ is satisfiable in some model of set theory if and only if it is satisfiable over $\mathbb{HF}$ [10]. We also denote by $\overline{n}$ the singleton set of *rank* n, i.e., the set inductively defined as: $\overline{0} = \emptyset, \overline{n+1} = \{\overline{n}\}$.

Let $\Sigma \supseteq \{\cup, \emptyset\}$ be a signature. Consider a function $\hat{\_}$ which maps each symbol of $\Sigma \setminus \{\cup, \emptyset\}$ into a set of the form $\overline{n}$ with $n \geq 1$ and $\hat{f} \neq \hat{g}$ if $f \not\equiv g$. We use this function to define the function $\alpha : T(\Sigma, \mathcal{V}) \longrightarrow T(\mathcal{L}, \mathcal{V})$:

$$\begin{cases} \alpha(X) & = & X & \text{if } X \text{ is a variable} \\ \alpha(\emptyset) & = & \emptyset^{\mathbb{HF}} & \\ \alpha(s \cup t) & = & \alpha(s) \cup^{\mathbb{HF}} \alpha(t) & \\ \alpha(f(t_1, \dots, t_n)) & = & \{\langle \hat{f}, \alpha(t_1), \dots, \alpha(t_n) \rangle\} & f \not\equiv \emptyset, f \not\equiv \cup, ar(f) = n \end{cases}$$

where $\emptyset^{\mathbb{HF}}$ and $\cup^{\mathbb{HF}}$ are the interpretations of the empty set and of the union operator on $\mathbb{HF}$. When the context is clear we simply use $\emptyset$ and $\cup$ for them. $\langle a_1, \dots, a_n \rangle$ denotes the ordered tuple constructor that can be defined as usual in set theory (e.g., $\langle x, y \rangle = \{\{x\}, \{x, y\}\}$). The range of α is $\mathcal{U}$ when the term is ground. Otherwise, with a slight mixing of syntax and semantics, it returns a pure set term involving variables. We define also the function $\gamma : \mathcal{U} \longrightarrow T(\Sigma, \mathcal{V})$:

$$\begin{cases} \gamma(\emptyset) & = & \emptyset & \\ \gamma(\{t_1, \dots, t_m\}) & = & \gamma(\{t_1\}) \cup \cdots \cup \gamma(\{t_m\}) & \text{if } m > 1 \\ \gamma(\{\langle \hat{f}, r_1, \dots, r_n \rangle\}) & = & f(\gamma(r_1), \dots, \gamma(r_n)) & \\ \gamma(\{r\}) & = & R_r & \text{if } r \not\equiv \langle \hat{f}, r_1, \dots, r_n \rangle \end{cases}$$

where R_r is a new variable, different for each r.

Example A.1 *Assume* $\hat{\{\cdot\}} = \overline{1}$, $\hat{a} = \overline{2}$, $\hat{f} = \overline{3}$, $ar(f) = 2$.

- $\alpha(\{a\} \cup \{\emptyset\}) = \{\langle \overline{1}, \{\langle \overline{2} \rangle\} \rangle, \langle \overline{1}, \emptyset \rangle\}$
- $\alpha(\{f(a, \emptyset)\} \cup X) = \{\langle \overline{3}, \{\langle \overline{2} \rangle\}, \emptyset \rangle\} \cup X$
- $\gamma(\{\langle \overline{1}, \{\langle \overline{2} \rangle\} \rangle, \langle \overline{1}, \emptyset \rangle\}) = \{a\} \cup \{\emptyset\}$
- $\gamma(\{\langle \overline{1}, \emptyset \rangle, \{\emptyset, \{\emptyset\}\}\}) = \{a\} \cup N$, *with* N *a new variable.*

It is easy to verify by structural induction that for $t \in T(\Sigma)$ it holds that $\mathcal{H} \models t = \gamma(\alpha(t))$.

Since we are in a *general* signature, we know that there is $f \in \Sigma \setminus \{\cup, \emptyset\}$, $ar(f) > 0$: we denote by f^1 the term $f(\emptyset, \dots, \emptyset)$ and by f^{k+1} the term $f(f^k, \dots, f^k)$. A valuation σ over $\mathcal{H}$ is said to be a f^i valuation if all variables involved are assigned to a term of this form. From Lemma 4.16 we know that, for $s, t \in T(\Sigma)$

$$depth(t_1) \neq depth(t_2) \quad \rightarrow \quad \mathcal{H} \models t_1 \neq t_2 \tag{17}$$

If $\sigma : \mathcal{V} \longrightarrow \mathcal{H}$ is a valuation, then with $\alpha(\sigma) : \mathcal{V} \longrightarrow \mathbb{HF}$ we denote the valuation defined as: $\alpha(\sigma)(X) = \alpha(\sigma(X))$. It is immediate to prove, by structural induction on $|t|$, that for all terms t $\alpha(\sigma)(t) = \alpha(\sigma(t))$.

The following result will help us in proving the main theorem of this section:

Lemma A.2 *Let s and t be two different elements of $\mathcal{U}$, and let $h = max\{depth(\gamma(s)), depth(\gamma(t))\}$. If σ is a f^i-valuation such that:*

- *for all $X \in vars(\gamma(t)) \cup vars(\gamma(s))$ it holds that $\sigma(X) = f^{n_X}$ with $n_X > h$, and*
- *for all $X, Y \in vars(\gamma(t)) \cup vars(\gamma(s))$ such that $X \not\equiv Y$ it holds that $|n_X - n_Y| > h$,*

then $\mathcal{H} \models \sigma(\gamma(s)) \neq \sigma(\gamma(t))$.

Proof. 1. If $\gamma(s)$ and $\gamma(t)$ are ground terms, then s and t belongs to $\alpha(T(\Sigma))$. The result holds trivially for all the valuations σ, since γ is injective on $\alpha(T(\Sigma))$.

2. If $\gamma(s)$ is ground and $\gamma(t)$ is not ground (or vice versa), then the result holds by property (17), since $depth(\sigma(\gamma(t)) > h \geq depth(\sigma(\gamma(s)))$ (or vice versa).

3. If $\gamma(s)$ and $\gamma(t)$ are not ground and there is a variable X which occurs in $\gamma(s)$, but not in $\gamma(t)$, then consider a subtree $\sigma(X)$ in the term $\sigma(\gamma(s))$. It cannot be the case that this subtree is equal to a subtree $\sigma(r)$ of $\sigma(\gamma(t))$:

- if $\sigma(r) = r$, i.e. r was already ground in $\gamma(t)$, then $depth(r) < h < depth(\sigma(X))$, and hence $\sigma(r) \neq \sigma(X)$;
- if in r there is a variable Y such that $n_Y > n_X$, then $depth(\sigma(r)) \geq depth(\sigma(Y)) > depth(\sigma(X))$;
- if all the variables $Y_1, \ldots, Y_n$ in r are such that $n_{Y_i} < n_X$ (i.e. $n_X - n_{Y_i} > h$), then $depth(\sigma(r)) \leq h + max_{i=1,\ldots,n}\{depth(\sigma(Y_i))\} \leq h + (n_{Y_j} + 1) > n_X + 1 = depth(\sigma(X))$.

4. The same reasoning can be applied if $\gamma(s)$ and $\gamma(t)$ are not ground and there is a variable X which occurs in $\gamma(t)$ but not in $\gamma(s)$.

5. Assume $\gamma(s)$ and $\gamma(t)$ are not ground and $vars(\gamma(s)) = vars(\gamma(t))$. Without loss of generality, consider $\rho(\gamma(s))$ and $\rho(\gamma(t))$. Let us consider one of the minimal subtrees s' of $\rho(\gamma(s))$ such that the subtree t' of $\rho(\gamma(t))$ in the same position of s' is not syntactically equal to s'.

- If s' and t' have two different outermost functional symbols, then $\sigma(s') \neq \sigma(t')$ and hence $\sigma(\gamma(s)) \neq \sigma(\gamma(t))$.
- Otherwise, since s' is minimal, it must be the case that s' is a variable or t' is a variable.

  If $s' \equiv X$ and X occurs in t', then $depth(\sigma(s')) < depth(\sigma(t'))$.

  Otherwise,If $s' \equiv X$ and X does not occur in t', then we can apply the same reasoning applied in case 3. If $t' \equiv X$, then we can apply the same reasoning of the last two sub-cases.

□

Without loss of generality, we can assume to deal with flat form constraints, namely constraints C whose literals are of the form $X \neq Y, X = f(Y_1, \ldots, Y_n)$, with $f \in \Sigma$. Literals of the form $X = Y$ can be removed by application of substitution. Starting from a generic constraint it is immediate to obtain in polynomial time an equi-satisfiable flat form constraint. With $\alpha(\mathcal{C})$ we denote the set-based formula obtained replacing $s \; op \; t$ with $\alpha(s) \; op \; \alpha(t)$ where op is $=$ or $\neq$.

Theorem A.3 *Let C be a flat form constraint. Then $\mathcal{H} \models \tilde{\exists}\mathcal{C}$ if and only if $\mathbb{HF} \models \tilde{\exists}\alpha(\mathcal{C})$.*

Proof. Assume $\mathcal{H} \models \tilde{\exists}C$, and let $\sigma : vars(\mathcal{C}) \longrightarrow T(\Sigma, \mathcal{V})$ be a satisfying valuation. It is immediate to prove that $\mathbb{HF} \models \alpha(\sigma)(\alpha(\mathcal{C}))$.

Assume now that $\alpha(\mathcal{C})$ is satisfied by $\tau : vars(\mathcal{C}) \longrightarrow \mathcal{U}$ in $\mathbb{HF}$. We prove that there is a valuation ν of the variables in $\gamma(\tau(\mathcal{C}))$ such that the valuation $\mu : vars(\mathcal{C}) \longrightarrow \mathcal{H}$, defined as: $\mu(X) = \nu(\gamma(\tau(X))))$ is a satisfying valuation for C over $\mathcal{H}$. Let $vars(\mathcal{C}) = \{X_1, \ldots, X_m\}$ and $S = \{\tau(X_1), \ldots, \tau(X_m)\}$, consider $h = max_{i=1,\ldots,m}\{depth(\gamma(\tau(X_i))\}$. Let ν be a f^i-valuation such that:

- for all $W \in vars(\gamma(\tau(\mathcal{C})))$ it holds $\nu(W) = f^{n_W}$, with $n_W > h$, and
- for all $W, V \in vars(\gamma(\tau(\mathcal{C})))$ such that $W \not\equiv V$ it holds $|n_W - n_V| > h$

We analyze the various literals of C:

$X = \emptyset$: $X = \emptyset$ is in $\alpha(\mathcal{C})$ and $\tau(X) = \emptyset$. By definition, $\gamma(\tau(X)) = \emptyset$.

$X = f(Y_1, \ldots, Y_n)$: This means that $X = \{\langle \hat{f}, Y_1, \ldots, Y_n \rangle\}$ is in $\alpha(\mathcal{C})$. Since τ is a satisfying valuation, then $\tau(X) = \{\langle \hat{f}, \tau(Y_1), \ldots, \tau(Y_n) \rangle\}$. This implies that for any ν:

$$\begin{array}{rcl}\mu(X) & = & \nu(\gamma(\tau(X))) & = & \nu(\gamma(\{\langle \hat{f}, \tau(Y_1), \ldots, \tau(Y_n) \rangle\})) \\ & = & \nu(f(\gamma(\tau(Y_1)), \ldots, \gamma(\tau(Y_n)))) & = & f(\nu(\gamma(\tau(Y_1))), \ldots, \nu(\gamma(\tau(Y_n)))) \\ & = & f(\mu(Y_1), \ldots, \mu(Y_n)) & = & \mu(f(Y_1, \ldots, Y_n))\end{array}$$

$X = Y \cup Z$: $X = Y \cup Z$ is in $\alpha(\mathcal{C})$ and $\tau(X) = \tau(Y) \cup \tau(Z)$. Thus, for any ν:

$$\begin{array}{rcl}\mu(X) & = & \nu(\gamma(\tau(Y) \cup \tau(Z))) & = & \nu(\gamma(\tau(Y)) \cup \gamma(\tau(Z))) \\ & = & \nu(\gamma(\tau(Y))) \cup \nu(\gamma(\tau(Z))) & = & \mu(Y) \cup \mu(Z) \\ & = & \mu(Y \cup Z)\end{array}$$

$X \neq Y$: $X \neq Y$ is in $\alpha(\mathcal{C})$ and $\tau(X)$ and $\tau(Y)$ are two different sets. From, Lemma A.2, we have the thesis, since the ν we have defined satisfies the hypothesis of the Lemma.

□

Corollary A.4 *Testing the satisfiability of general $ACI1$ constraints over $\mathcal{H}$ is NP-complete.*

Proof. A general $ACI1$ constraint C can be transformed in polynomial time to a equi-satisfiable flat form constraint. The result follows from Theorem A.3 and [10]. □