Constraint Logic Programming over Unions of Constraint Theories

Cesare Tinelli and Mehdi Harandi

Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Ave.
Urbana, IL 61801 – USA
{tinelli,harandi}@cs.uiuc.edu
tel: +1(217)333-5821
fax: +1(217)333-3501

Abstract. In this paper, we propose an extension of the Jaffar-Lassez Constraint Logic Programming scheme that operates with unions of constraint theories with different signatures and decides the satisfiability of *mixed* constraints by appropriately combining the constraint solvers of the component theories. We describe the extended scheme and provide logical and operational semantics for it along the lines of those given for the original scheme. Then we show how the main soundness and completeness results of Constraint Logic Programming lift to our extension.

Keywords: Constraint Logic Programming, combination of satisfiability procedures.

1 Introduction

The Constraint Logic Programming scheme was originally developed in [8] by Jaffar and Lassez as a principled way to combine the two computational paradigms of Logic Programming and Constraint Solving. The scheme extends conventional Logic Programming by replacing the notion of *unifiability* with that of *constraint solvability* over an underlying constraint domain. As originally proposed, the CLP scheme extends immediately to the case of multiple constraint domains as long as they do not share function or predicate symbols. The scheme though does not account for the presence of *mixed* terms, terms built with functors from different signatures, and corresponding mixed constraints. The reason for this is that, although the CLP scheme allows in principle multiple, separate constraint theories, each with its own constraint solver, it is not designed to operate on their *combination*, to which mixed constraints belong.

This paper proposes an extension of the scheme to include constraint domains built as the combination of a number of independent domains, such as, for instance, the domains of finite trees and real numbers, the domains of lists, strings, and integers, and the like.

In principle, we can always instantiate the CLP scheme with a suitable constraint domain once we have a constraint solver for it, no matter whether the

domain is simple or "composite". For composite constraint domains however it is desirable not to have to build a solver from scratch if a constraint solver is already available for each component domain.

A lot of research has been done in recent years on domain combination (see, for instance [2, 3, 13, 16]) although most of the efforts have been concentrated on unification problems and equational theories (see [1, 4, 5, 6, 10, 14, 19], among others). The results of these investigations are still limited in scope and a deep understanding of many model- and proof-theoretic issues involved is still out of reach. In spite of that, we try to show the effectiveness of combination techniques by choosing one of the most general and adapting it so that it can be incorporated in the CLP scheme with few modification of the scheme itself.

1.1 Notation and Conventions

We adhere rather closely to the notation and definitions given in [15] for what concerns mathematical logic in general and [9] for what concerns constraint logic programming in particular. We report here the most notable notational conventions followed. Other notation which may appear in the sequel follows the common conventions of the two fields.

In this paper, we use v, x, y, z as meta-variables for the logical variables, s, t for first-order terms, p, q for predicate symbols, f, g for function symbols, a, b, h for atoms, A for a multi-set of atoms, c, d for constraints, C for a multi-set of constraints, φ, ψ for first order formulas, and ϑ for a value assignment, or valuation, to a set of variables. Some of these symbols may be subscripted or have an over-tilde which will represent a finite sequence. For instance, $\tilde{x}$ stands for a sequence of the form $(x_1, x_2, \ldots, x_n)$ for some natural number n. When convenient, we will use the tilde notation to denote sets of symbols (as opposed to sequences). Where $\tilde{s}$ and $\tilde{t}$ have both length n, the equation $\tilde{s} = \tilde{t}$ stands for the system of equations $\{s_1 = t_1 \wedge \cdots \wedge s_n = t_n\}$.

In general, $var(\varphi)$ is the set of φ's free variables. The shorthand $\exists_{-\tilde{x}} \varphi$ stands for the existential quantification of all the free variables of φ that are not contained in $\tilde{x}$, while $\tilde{\exists} \varphi$ stands for the existential closure of φ.

We will identify union of multi-sets of formulas with their logical conjunction. We will also identify first-order theories with their deductive closure. Where $\mathcal{S}, \mathcal{T}$ are sets of Σ-sentences, for some signature Σ, $Mod(\mathcal{T})$ is the set of all the Σ-models of $\mathcal{T}$. The notation $\mathcal{T} \models \varphi$ means that $\mathcal{T}$ logically entails the universal closure of φ, while $\mathcal{S}, \mathcal{T} \models \varphi$ stands for $\mathcal{S} \cup \mathcal{T} \models \varphi$.

We will say that a formula φ is *satisfiable in* a theory $\mathcal{T}$ iff there exists a model of $\mathcal{T}$ that satisfies $\tilde{\exists} \varphi$.

If P is a CLP program we will denote with P^* its Clark completion.

1.2 Organization of the Paper

In Section 2, we briefly describe the original CLP scheme and motivate the need for constraints with mixed terms, which the CLP scheme does not explicitly

accommodate. In Section 3, we mention a method for deriving a satisfiability procedure for a combination of theories admitting mixed terms from the satisfiability procedures of the single theories. In Section 4, we explain how one can use the main idea of that combination method to extend the CLP scheme and allow composite constraint domains and mixed terms over them. In Section 5 we prove some soundness and completeness results for the new scheme. In Section 6, we summarize the main contribution of this paper, outlining directions for further development.

2 The CLP Scheme

A thorough description of CLP($\mathcal{X}$), the CLP scheme, can be found in [9]. We recall here that, in essence, an instance of CLP($\mathcal{X}$) is obtained by assigning the parameter $\mathcal{X}$ with a quadruple that specifies the chosen constraint domain, its axiomatization, and the features of the constraint language. More specifically, $\mathcal{X} := \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$ where Σ is a signature, $\mathcal{D}$ is a Σ-structure representing the constraint domain over which computation is performed, $\mathcal{L}$ is the constraint language, that is, the class of Σ-formulas used to express the constraints, and $\mathcal{T}$ is a first-order Σ-theory (with equality) describing the relevant properties of the domain. A number of assumptions are generally made about $\mathcal{X}$. The most important are:

- Σ contains the equality symbol which $\mathcal{D}$ interprets as the identity in the underlying domain.
- $\mathcal{L}$ contains an identically true and an identically false predicate and a set $\mathcal{L}_P$ of *primitive constraints*.
- $\mathcal{L}$ is closed under variable renaming and logical conjunction.
- $\mathcal{D}$ and $\mathcal{T}$ *correspond* on $\mathcal{L}$, that is, $\mathcal{D}$ is a model of $\mathcal{T}$ and every formula of $\mathcal{L}$ that is satisfiable in $\mathcal{D}$ is satisfiable in every model of $\mathcal{T}$.

For some applications $\mathcal{T}$ is required to be *satisfaction complete* with respect to $\mathcal{L}$: for every $c \in \mathcal{L}$, either $\mathcal{T} \models \tilde{\exists}\, c$ or $\mathcal{T} \models \neg\tilde{\exists}\, c$.

Prolog itself can be seen as an instance of the CLP scheme, specifically as CLP($\mathcal{FT}$), where $\mathcal{FT}$ is the constraint domain of finite trees represented as first-order terms. Actually, all the CLP($\mathcal{X}$) systems in which $\mathcal{X}$ is not $\mathcal{FT}$ or an extension of it[1] still retain the possibility of building uninterpreted terms and so are at least CLP($\mathcal{FT}, \mathcal{X}$) systems. Furthermore, many systems support several constraint domains. They can be seen as CLP($\tilde{\mathcal{X}}$) systems, with $\tilde{\mathcal{X}} := \{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$, where the $\mathcal{X}_i$'s are built over disjoint signatures and their constraints are processed by different, specialized solvers. In these systems, predicate or function symbols in one signature are applicable, with few exceptions, only to (non-variable) terms entirely built with symbols from the same signature.

Thus, although in one way or another all CLP systems use more than one constraint domain, they do not freely allow *mixed* terms or predicates, that is,

[1] PrologII, for instance, works with rational trees instead of finite trees.

expressions built with symbols from different signatures. Meaningful constraints over such heterogeneous expression, however, arise naturally in many classes of applications. An example of a constraint with mixed terms in the theory of lists and natural numbers is

$$v = (x + 9) :: y \ \wedge \ head(y) > z + u$$

where :: is the list constructor and *head* returns the first element of a list. An example, adapted from [12], in the theory of finite trees and real numbers, is

$$f(f(x) - f(y)) \neq f(z) \ \wedge \ y + z \leq x \ .$$

Proper instances of CLP($\mathcal{X}$) cannot deal with these types of constraints simply because the CLP computational paradigm does not consider them. In the rest of this paper, we will show a method for extending the CLP scheme to a new scheme, MCLP($\bar{\mathcal{X}}$), that offers a systematic and consistent treatment of constraints with mixed terms. Specifically, we will show how to convert a CLP($\tilde{\mathcal{X}}$) system into a MCLP($\bar{\mathcal{X}}$) system which operates on the constraint structure generated by a suitable combination of the various $\mathcal{X}_i$'s.

3 Combining Satisfiability Procedures

The main idea of our extension is to adapt and use in CLP a well-known method, originally proposed by Nelson and Oppen [13], for combining first-order theories and their satisfiability procedures. Although the method applies to the combination of any finite number of theories, for simplicity we will consider the case of just two theories here.

Definition 1. We say that a formula is in *simple Conjunctive Normal Form* if it is a conjunction of literals. Given a Σ-theory $\mathcal{T}$, we will denote with $sCNF(\mathcal{T})$ the set of simple Conjunctive Normal Form Σ-formulas.

Definition 2. A consistent theory Σ-$\mathcal{T}$ is called *stably-infinite* iff any quantifier-free Σ-formula is satisfiable in $\mathcal{T}$ iff it is satisfiable in an infinite model of $\mathcal{T}$.

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably-infinite theories with respective signatures Σ_1, Σ_2 such that $\Sigma_1 \cap \Sigma_2 = \emptyset$.[2] The simplest combination of $\mathcal{T}_1$ and $\mathcal{T}_2$ is the $(\Sigma_1 \cup \Sigma_2)$-theory $\mathcal{T}_1 \cup \mathcal{T}_2$ defined as (the deductive closure of) the union $\mathcal{T}_1$ and $\mathcal{T}_2$.

If for each $i = 1, 2$ we have a procedure Sat_i that decides the satisfiability in $\mathcal{T}_i$ of the formulas of $sCNF(\mathcal{T}_i)$, we can generate a procedure that decides the satisfiability in $\mathcal{T}_1 \cup \mathcal{T}_2$ of any formula $\varphi \in sCNF(\mathcal{T}_1 \cup \mathcal{T}_2)$ by using Sat_1 and Sat_2 modularly. Clearly, because of the expanded signature, φ cannot in general be processed directly by either satisfiability procedure, unless it is of the form $\varphi_1 \wedge \varphi_2$—call it *separate form*—where φ_i is a (possibly empty) formula of $sCNF(\mathcal{T}_i)$.[3] In that case, each Sat_i will process φ_i separately.

[2] We consider the equality symbol "=" as a logical constant.

[3] Since φ_i does not contain symbols from the other signature, we say that it is *pure*.

If φ is not already in separate form, it is always possible to apply a procedure that, given φ, returns a formula $\ddot{\varphi}$ which is in separate form and is satisfied exactly by the same models and variable assignments that satisfy φ.[4] Some examples of formulas and their separate forms are given in Figure 1.

$$v = (x + 9) :: y \wedge head(y) > z + u \quad (1)$$
$$f(f(x) - f(y)) \neq f(z) \wedge y + z \leq x \quad (2)$$

$$(v = x_1 :: y \ \wedge \ head(y) = x_2) \wedge (x_1 = x + 9 \ \wedge \ x_2 > z + u) \quad (3)$$
$$(f(x_1) \neq f(z) \ \wedge \ x_2 = f(x) \ \wedge \ x_3 = f(y)) \wedge (x_1 = x_2 - x_3 \ \wedge \ y + z \leq x) \quad (4)$$

Fig. 1. Formulas (3) and (4) are separate forms of formulas (1) and (2), respectively.

A description of the Nelson-Oppen combination procedure is not necessary here. For our present purposes it is enough to say that global consistency between the separate satisfiability procedures is achieved by propagation, between the procedures, of the entailed equalities between the variables of the input formula. That this approach is correct and sufficient is justified by the model-theoretic result given in the following.

Definition 3. If P is any partition on a set of variables V and R is the corresponding equivalence relation, we call *arrangement of V (determined by P)* the set $ar(V)$ made of all the equations between any two equivalent words of V and all the disequations between any two non-equivalent words. Formally, $ar(V) := \{x = y \mid x, y \in V \text{ and } xRy\} \cup \{x \neq y \mid x, y \in V \text{ and not } xRy\}$.

Proposition 4 [18]. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be as above. Consider $\varphi_1 \in sCNF(\mathcal{T}_1)$, $\varphi_2 \in sCNF(\mathcal{T}_2)$ and let $\tilde{x} := var(\varphi_1) \cap var(\varphi_2)$. Then, $\varphi_1 \wedge \varphi_2$ is satisfiable in $\mathcal{T}_1 \cup \mathcal{T}_2$ iff there exists and arrangement $ar(\tilde{x})$ such that $\varphi_1 \wedge ar(\tilde{x})$ is satisfiable in $\mathcal{T}_1$ and $\varphi_2 \wedge ar(\tilde{x})$ is satisfiable in $\mathcal{T}_2$.*

4 Extending CLP($\mathcal{X}$)

We want to extend the CLP scheme so that we go from a language of type CLP($\tilde{\mathcal{X}}$), where $\mathcal{X} := \{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$ is a set of signature-disjoint constraint structures, to a language of type MCLP($\tilde{\mathcal{X}}$), where $\tilde{\mathcal{X}}$ is a combination of the previous

[4] The gist of the separation procedure is to repeatedly replace in φ each term t of the "wrong" signature with a new variable and add the equation $x = t$ to φ. More details can be found in [17].

structures in the sense that it allows any computation performed in CLP($\tilde{\mathcal{X}}$) and, furthermore, poses no signature restriction on term construction.

The reason why we are interested in combinations of satisfiability procedures is that CLP systems already utilize *separate* satisfaction procedures, the constraint solvers, to deal with the various constraint theories they support and so already have a main module to drive the goal reduction process and control the communication with the solvers.

Intuitively, if we rewrite MCLP($\bar{\mathcal{X}}$) statements in a separate form similar to that mentioned earlier, we may be able to use the various constraint solvers much the same way the Nelson-Oppen combination procedure uses the various satisfiability procedures. Moreover, the machinery we will need for a MCLP($\bar{\mathcal{X}}$) system will be essentially the same we would need for a corresponding CLP($\tilde{\mathcal{X}}$) system. The only necessary addition, to realize the solvers combination, will be a mechanism for generating equations and disequations between variables shared by the different solvers and propagating them to the solvers themselves. More precisely, we will need a procedure that, each time a new constraint is given to one solver, (a) identifies the variables that that constraint shares with those in the other solvers, (b) creates a backtrack point in the computation and chooses a (novel) arrangement of those variables, and (c) passes the arrangement to all the other solvers.

We clarify and formalize all this in the following subsections.

4.1 The Extended Scheme

The first issue we are confronted with in extending the CLP scheme is the impossibility of fixing a single domain of computation. Recall that the CLP scheme puts primacy on a particular structure which represents the intended constraint domain. The combination procedure we are considering, however, combines *theories*, not structures[5]: it succeeds when the input formula is satisfiable in *some* model of the combined theory. For this reason, our extension will use as "constraint domain" a whole class of structures instead of a single one. In this respect, our scheme is actually a restriction of the Höhfeld-Smolka constraint logic programming framework [7]. The restriction is achieved along two dimensions: the constraint language and the set of solution structures. We use only sCNF formulas as constraints and elementary classes[6] as the class of structures over which constraint satisfiability is tested. In particular, the class associated to a given MCLP($\bar{\mathcal{X}}$) language is the set of the models of the union of the component theories.

Formally, the constraint structure $\bar{\mathcal{X}}$ for the MCLP($\bar{\mathcal{X}}$) scheme is defined as the tuple

$$\bar{\mathcal{X}} := \langle \langle \Sigma_1, \mathcal{T}_1 \rangle, \ldots, \langle \Sigma_n, \mathcal{T}_n \rangle \rangle$$

[5] For some very recent work on the combination of structures, see [3].

[6] Recall that a class of structures is called *elementary* if it coincides with the set of models of some first-order theory.

where $\Sigma_1, \ldots, \Sigma_n$ are pairwise disjoint signatures and $\mathcal{T}_i$ is a stably-infinite Σ_i-theory, for each $i \in \{1, \ldots, n\}$. The constraint theory for MCLP($\bar{\mathcal{X}}$) is $\mathcal{T} := \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_n$, the constraint language is $sCNF(\mathcal{T})$, and the set of solution structures is $Mod(\mathcal{T})$.

Next, we describe a logical and a top-down operational model of MCLP($\bar{\mathcal{X}}$) where we assume that, for each $\mathcal{T}_i$, a solver Sat_i is available that decides satisfiability of sCNF formulas in $\mathcal{T}_i$.

4.2 Logical Semantics

The format of MCLP($\bar{\mathcal{X}}$) statements is identical to that of CLP statements except that mixed constraints are allowed with no restrictions. As a consequence, MCLP($\bar{\mathcal{X}}$) adopts CLP($\mathcal{X}$)'s logical semantics for both its programs and their completion. The only difference concerns the notation used to describe MCLP($\bar{\mathcal{X}}$) programs.

Since we want to apply the available solvers modularly, it is convenient to look at each MCLP($\bar{\mathcal{X}}$) statement as if it had first been converted into an appropriate separate form. After we define the computation transitions, the careful reader will observe that it is not necessary to actually write MCLP($\bar{\mathcal{X}}$) programs in separate form because a separation procedure can be applied on the fly during subgoal expansion. We use the separate form here simply for notational convenience. Specifically, instead of a standard CLP rule of the form, $p(\tilde{x}) \leftarrow B$, as defined in [9] for instance, we consider the *separate form*

$$p(\tilde{x}) \leftarrow \ddot{B}$$

obtained by applying the procedure mentioned in Section 3 to the body of the rule[7]. Analogously, instead of a goal G we consider the corresponding goal $\ddot{G}$.

It should be clear that, under the CLP logical semantics, a MCLP($\bar{\mathcal{X}}$) statement and its separate form are equivalent.

4.3 Operational Semantics

We will only consider the case of two component theories here, as the n-component case is an easy generalization.

As with CLP($\mathcal{X}$), computation in MCLP($\bar{\mathcal{X}}$) can be described as a sequence of state transitions. Each state in turn is described by either the symbol *fail* or a tuple of the form $\langle A, C_1, C_2 \rangle$ where A is a set of pure atoms and constraints, and for $i = 1, 2$ C_i, the constraint store, is a set of Σ_i-constraints.[8] We assume the presence of a function, *select*, that when applied to the first element A of

[7] This is always possible as the body of a MCLP($\bar{\mathcal{X}}$) rule is a sCNF formula.

[8] For simplicity, we have decided to ignore the issue of delayed constraints here. We would like to point out however that our extension could be easily applied with comparable results to an operational model including delayed constraints such as the one described in [9].

a transition returns a member of A non-deterministically. State transitions are defined as follows.

1. $$\langle A, C_1, C_2\rangle \rightarrow_r \langle A \cup B - a(\tilde{x}), C_1', C_2'\rangle$$

where $a(\tilde{x}) := \mathit{select}(A)$ is an atom, the program P contains the rule $a(\tilde{y}) \leftarrow B$, and $C_i' := C_i \cup \tilde{x} = \tilde{y}$ for $i = 1, 2$.

2. $$\langle A, C_1, C_2\rangle \rightarrow_r \mathit{fail}$$

where $a(\tilde{x}) := \mathit{select}(A)$ is an atom and no rule in P has a as the predicate symbol of its head.

3. $$\langle A, C_1, C_2\rangle \rightarrow_c \langle A - c, C_1', C_2'\rangle$$

where $c := \mathit{select}(A)$ is a constraint literal and, for $i = 1, 2$, $C_i' := C_i \cup c$ if c is a Σ_i-constraint, $C_i' = C_i$ otherwise.

4. $$\langle A, C_1, C_2\rangle \rightarrow_s \langle A, C_1', C_2'\rangle$$

where $ar(\tilde{v})$ is an arrangement of the variables shared by C_1 and C_2 and $C_i' := C_i \cup ar(\tilde{v})$ and $Sat_i(C_i)$ succeeds for $i = 1, 2$.

5. $$\langle A, C_1, C_2\rangle \rightarrow_s \mathit{fail}$$

where $ar(\tilde{v})$ is an arrangement of the variables shared by C_1 and C_2, $C_i' := C_i \cup ar(\tilde{v})$ for $i = 1, 2$, and either of $Sat_1(C_1)$ or $Sat_2(C_2)$ fails.

Similarly to CLP($\mathcal{X}$), transitions of type $\rightarrow_r$ are just goal reduction steps. The difference here is that the variable equalities produced by matching the selected predicate with the head of some rule go to both constraint solvers as, by definition, an equality predicate with variable arguments belongs to both $\mathcal{L}_1$ and $\mathcal{L}_2$.

Transitions of type $\rightarrow_c$ feed the constraint solvers with a new constraint, where each constraint goes to the relative solver (variable equalities going to both solvers).

Transitions of type $\rightarrow_s$ differ more significantly from the corresponding transitions in CLP($\mathcal{X}$) as they actually implement, in an incremental fashion, the combination procedure. They deserve a more detailed explanation then.

In terms of the procedure, for every $\rightarrow_s$ transition, we consider the constraint stores C_1 and C_2 as the i-pure halves of sCNF formulas whose satisfiability must be checked. For each constraint store, we use the constraint solver "as is" but we make sure that *global consistency* information is shared by the two solvers by choosing an arrangements of the variables they have in common. It goes without saying that, like the transitions of type $\rightarrow_r$, transitions of type $\rightarrow_s$ are non-deterministic. With the first type, the choice is among the possible reductions of the selected predicate, with the second, it is among the possible arrangements of the shared variables. In actual implementations of the scheme then, backtracking

mechanisms similar to those used for $\rightarrow_r$ transitions must be used. For space limitations we cannot include a more complete discussion on the implementation of $\rightarrow_r$ transitions and its various possible optimization. The interested reader is again referred to [17].

The concepts of derivation, final state, fair/failed/successful derivation, answer constraint, and computation tree can be defined analogously to the corresponding CLP($\mathcal{X}$) concepts (see [9]).

5 Computational Properties of MCLP($\bar{\mathcal{X}}$)

To discuss the main computational properties of MCLP($\bar{\mathcal{X}}$) it is necessary to specify a more detailed operational semantics than the one given in the previous section. Since any implementation of MCLP($\bar{\mathcal{X}}$) is a deterministic system, a particular computation rule has to be defined. For us, this amounts to specifying the behavior of the *select* function and the order in which the various types of transitions are applied. We will need to further restrict our attention to specific classes of MCLP($\bar{\mathcal{X}}$) systems to prove some of the properties of MCLP($\bar{\mathcal{X}}$).

Definition 5. Let $\rightarrow_{cs}$ be the two-transition sequence $\rightarrow_c\rightarrow_s$. We say that a MCLP($\bar{\mathcal{X}}$) system is *quick-checking* if all of its derivations are sequences of $\rightarrow_r$ and $\rightarrow_{cs}$ transitions only.

A quick-checking CLP($\mathcal{X}$) system verifies the consistency of the constraint store immediately after it modifies it. Analogously, a quick-checking MCLP($\bar{\mathcal{X}}$) system verifies the consistency of the union of all the constraint stores (by means of equality sharing among the solvers) immediately after it modifies at least one of them.

Definition 6. A MCLP($\bar{\mathcal{X}}$) system is *ideal* if it is quick-checking and uses a fair computation rule.[9]

In both schemes, we can define the concept of finite failure if we restrict ourselves to the class of ideal systems. We say that a goal G is *finitely failed* for a program P if, in any ideal system, every derivation of G in P is failed.

5.1 Comparing CLP($\mathcal{X}$) with MCLP($\bar{\mathcal{X}}$)

To show that the main soundness and correctness properties of the CLP schema lift to our extension, we will consider, together with the given MCLP($\bar{\mathcal{X}}$) system, a corresponding CLP($\mathcal{X}$) system that, while accepting the very same programs, supports the combined constraint theory directly (i.e., with a single solver) and show that the two systems have the same computational properties.

[9] This definition differs from that given in [9] because we adopt a slightly different definition of derivation (see [17] for more details), but it refers to the same class of systems.

Actually, MCLP($\bar{\mathcal{X}}$) systems cannot have a *corresponding* CLP($\mathcal{X}$) system since the original scheme and ours define constraint satisfiability in a different manner. In CLP($\mathcal{X}$), the satisfiability test on the constraint store is successful if the store is satisfiable in the *fixed* structure $\mathcal{D}$. In MCLP($\bar{\mathcal{X}}$) instead, the satisfiability test is successful if the (union of all) constraint store(s) is satisfiable in *any* structure among those modeling the constraint theory. However, correspondence becomes possible if we "relax" the CLP($\mathcal{X}$) system by testing satisfiability within the class of structures $Mod(\mathcal{T})$, where $\mathcal{T}$ is the chosen constraint theory, instead of a single structure.

The relaxed CLP scheme is more general than the original scheme but less general than that proposed by Höhfeld and Smolka in [7]. In fact, its soundness is derivable as a consequence of the soundness of Höhfeld and Smolka's. Its completeness, however, cannot be derived from their scheme because it is essentially a consequence of the choice of a *first-order* constraint language and theory, which Höhfeld and Smolka do not require.

In Section 5.2, we will see that not only is the relaxed CLP scheme sound and complete, but also, and more importantly, it has logical properties no weaker than those exhibited by the CLP scheme.

It should not be difficult to see now that, once we have shown that CLP($\mathcal{X}$) maintains its nice properties even with a satisfiability test over an elementary class of structures, soundness and completeness results of MCLP($\bar{\mathcal{X}}$) easily follow—the intuitive justification being that it is immaterial whether we check for satisfiability in the union constraint theory utilizing a combined procedure or a non-combined one.

5.2 Soundness and Completeness of Relaxed CLP

We will now consider CLP($\mathcal{X}$) systems that are instances of the relaxed CLP scheme mentioned earlier. For these systems, the tuple $\mathcal{X}$ is defined as in Section 2 with the difference that $\mathcal{D}$ is replaced by $Mod(\mathcal{T})$, and the satisfiability test succeeds if and only if the input constraint is satisfiable in some element of $Mod(\mathcal{T})$.

Assuming a relaxed CLP($\mathcal{X}$) system with constraint language $\mathcal{L}$ and theory $\mathcal{T}$, we have formulated the following results after those given in [9] for CLP($\mathcal{X}$). For lack of space we forgo their proofs here; they are very similar, however, to those given in [8] and [11] for the corresponding CLP results and can be found in [17].

Proposition 7 Soundness. *Given a program P and a goal G:*

1. *If G has a successful derivation with answer constraint c, then $P, \mathcal{T} \models c \rightarrow G$.*
2. *When $\mathcal{T}$ is satisfaction complete wrt to $\mathcal{L}$, if G has a finite computation tree with answer constraints $c_1, \ldots, c_n$, then $P^*, \mathcal{T} \models G \leftrightarrow c_1 \vee \cdots \vee c_n$.*

Proposition 8 Completeness. *Given a program P, a simple goal G and a constraint c:*

1. *If $P, \mathcal{T} \models c \rightarrow G$ and c is satisfiable in $\mathcal{T}$, then there are $n > 0$ derivations of G with respective answer constraint $c_1, \ldots, c_n$ such that $\mathcal{T} \models c \rightarrow c_1 \vee \cdots \vee c_n$.*
2. *When $\mathcal{T}$ is satisfaction complete wrt to $\mathcal{L}$, if $P^*, \mathcal{T} \models G \leftrightarrow c_1 \vee \cdots \vee c_n$ then G has a computation tree with answer constraints $c'_1, \ldots, c'_m$ such that $\mathcal{T} \models c_1 \vee \cdots \vee c_n \leftrightarrow c'_1 \vee \cdots \vee c'_m$.*

In [8], Jaffar and Lassez show that Negation-as-Failure can be used correctly in their scheme provided that the constraint theory is satisfaction complete with respect to the constraint language. We discovered that we can still use negation as failure properly in the relaxed CLP scheme and, in addition, we do not need satisfaction completeness of the component theories at all. As before, a sufficient condition for this result is that we use a first-order constraint language.

Proposition 9 Soundness and Completeness of Negation-as-Failure. *In an ideal system, a goal G is finitely failed for a program P iff $P^*, \mathcal{T} \models \neg G$.*

5.3 Main Results

In the following, we consider a MCLP($\bar{\mathcal{X}}$) system, where $\tilde{\mathcal{X}}$ is defined as in Section 4 but limited, again for simplicity, to the combination of only two stably-infinite theories. We will assume that, while the system satisfies the general implementation requirements given earlier, its computation rule is flexible enough with respect to the order in which the various transitions can be applied. Such assumption is not necessary for our results but makes their proofs easier and more intuitive.

We will also assume that programs and goals are all given in separate form. For notational ease, we will use $\rightarrow_{r/c}$ to denote either a $\rightarrow_r$ or a $\rightarrow_c$ transition. We start with some easy to prove lemmas.

Lemma 10. *If goal G has a successful derivation in a MCLP($\bar{\mathcal{X}}$) program P, then it has a successful derivation with the same answer constraint and such that all of its transitions are $\rightarrow_{r/c}$ transitions, except the last one which is a $\rightarrow_s$ transition.*

Essentially, the lemma states that a successful derivation can be always rearranged into a derivation of the form $\langle G, \emptyset, \emptyset \rangle \stackrel{*}{\rightarrow}_{r/c} \langle \emptyset, C_1, C_2 \rangle \rightarrow_s \langle \emptyset, C'_1, C'_2 \rangle$ by first reducing the goal to the empty set and then testing the consistency of the collected constraints.

Lemma 10 also entails that a successful derivation in MCLP($\bar{\mathcal{X}}$) is not just a finite derivation not ending with *fail*, but one whose answer constraint is satisfiable in the union theory. In fact, according to the MCLP($\bar{\mathcal{X}}$) operation model, a necessary condition for the above derivation to be successful is that C'_i be satisfiable in $\mathcal{T}_i$ for $i = 1, 2$. From Prop. 4 then, we can infer that $\exists_{-\,var(G)}(C'_1 \wedge C'_2)$, the answer constraint of the derivation, is satisfiable in $\mathcal{T}_1 \cup \mathcal{T}_2$.

Lemma 11. *Given a MCLP($\bar{\mathcal{X}}$) program P and a goal G, if a derivation of the form $\langle G, \emptyset, \emptyset \rangle \stackrel{*}{\rightarrow}_{r/c} \langle \emptyset, C_1, C_2 \rangle$ exists in P, then $\mathcal{T}, P \models \exists_{-\,var(G)}(C_1 \wedge C_2) \rightarrow G$.*

Proposition 12 Soundness of MCLP($\bar{\mathcal{X}}$). *Given a program P and a goal G,*

1. *If G has a successful derivation with answer constraint c, then $P, \mathcal{T} \models c \rightarrow G$.*
2. *When $\mathcal{T}$ is satisfaction complete wrt to $\mathcal{L}$, if G has a finite computation tree with answer constraints $c_1, \ldots, c_n$, then $P^*, \mathcal{T} \models G \leftrightarrow c_1 \vee \cdots \vee c_n$.*

Proof. Let $\tilde{x} := var(G)$.

1. By Lemma 10, we can assume without loss of generality that the derivation of G is of the form $\langle G, \emptyset, \emptyset \rangle \stackrel{*}{\rightarrow}_{r/c} \langle \emptyset, C_1, C_2 \rangle \rightarrow_s \langle \emptyset, C_1', C_2' \rangle$ where c is then $\exists_{-\tilde{x}} (C_1' \wedge C_2')$. Recalling the definition of $\rightarrow_s$ transitions, it is immediate that $\models c \rightarrow \exists_{-\tilde{x}} (C_1 \wedge C_2)$. The claim is then a direct consequence of Lemma 11.

2.[sketch] Let us call the MCLP($\bar{\mathcal{X}}$) system S and assume a corresponding relaxed CLP system S_{rel}. With no loss of generality we assume that, for each $i \in \{1, \ldots, n\}$, the derivation δ_i in S, having answer constraint c_i, is of the form $\langle G, \emptyset, \emptyset \rangle \stackrel{*}{\rightarrow}_{r/c} \langle \emptyset, C_{1i}, C_{2i} \rangle \rightarrow_s \langle \emptyset, C_{1i}', C_{2i}' \rangle$. Since S_{rel} has the same computational rule as S, for each δ_i there is a corresponding derivation $h(\delta_i)$ in S_{rel} of the form $\langle G, \emptyset, \emptyset \rangle \stackrel{*}{\rightarrow}_{r/c} \langle \emptyset, D_i \rangle \rightarrow_s \langle \emptyset, D_i \rangle$, where $D_i = C_{1i} \cup C_{2i}$.

Let $\sim$ be an equivalence relation over $\{1, \ldots, n\}$ such that $i \sim j$ iff δ_i and δ_j coincide up to the last transition. Notice that $h(\delta_i) = h(\delta_j)$ if $i \sim j$. Recalling the definition of $\rightarrow_s$ transitions, it should not be difficult to see that for each $j \in \{1, \ldots, n\}$, if $[j]$ is the equivalence class of j with respect to $\sim$, the following chain of logical equivalences holds,

$$\bigvee_{i \in [j]} c_i \leftrightarrow \bigvee_{i \in [j]} \exists_{-\tilde{x}} (C_{1i}' \wedge C_{2i}') \leftrightarrow \exists_{-\tilde{x}} (C_{1j} \wedge C_{2j}) \leftrightarrow \exists_{-\tilde{x}} D_j. \tag{5}$$

By an analogous of Lemma 10 for the relaxed CLP scheme, we can show that the tree made by all the $h(\delta_i)$'s above is indeed the finite computation tree of G in S_{rel}. By the soundness of the relaxed CLP scheme, we then have that

$$P^*, \mathcal{T} \models G \leftrightarrow \bigvee_{j \in \{1, \ldots, n\}} \exists_{-\tilde{x}} D_j. \tag{6}$$

The claim follows then immediately, combining (5) and (6) above. □

The following lemma is also easy to prove.

Lemma 13. *Consider a program P, a MCLP($\bar{\mathcal{X}}$) system S, and a corresponding relaxed CLP system S_{rel}. Then, for any transition t in S_{rel} of the form $\langle A, C \rangle \rightarrow_{r/c} \langle A', C' \rangle$ there is a transition t' in S of the form $\langle A, C_1, C_2 \rangle \rightarrow_{r/c} \langle A', C_1', C_2' \rangle$ such that $\mathcal{T} \models C \leftrightarrow C_1 \wedge C_2$ implies $\mathcal{T} \models C' \leftrightarrow C_1' \wedge C_2'$.*

Proposition 14 Completeness of MCLP($\bar{\mathcal{X}}$). *Given a MCLP($\bar{\mathcal{X}}$) system, a program P, a simple goal G and a constraint c,*

1. *If $P, \mathcal{T} \models c \rightarrow G$ and c is satisfiable in $\mathcal{T}$, then there are $n > 0$ derivations of G with respective answer constraint $c_1, \ldots, c_n$ such that $\mathcal{T} \models c \rightarrow c_1 \vee \cdots \vee c_n$.*

2. *When $\mathcal{T}$ is satisfaction complete wrt to $\mathcal{L}$, if $P^*, \mathcal{T} \models G \leftrightarrow c_1 \vee \cdots \vee c_n$, then G has a computation tree with answer constraints $c'_1, \ldots, c'_m$ such that $\mathcal{T} \models c_1 \vee \cdots \vee c_n \leftrightarrow c'_1 \vee \cdots \vee c'_m$.*

Proof: 1. Let us call the MCLP($\bar{\mathcal{X}}$) system S and assume a corresponding relaxed CLP system S_{rel}. Let $\tilde{x} := var(G)$. To simplify the notation, if δ is a successful derivation, we will denote its answer constraint by $ans(\delta)$.

Now, by the completeness of the relaxed CLP scheme, there exists a set D of successful derivations of G in S_{rel} such that $\mathcal{T} \models c \rightarrow \bigvee_{\delta \in D} ans(\delta)$. We show that, for each $\delta \in D$, there is a set D_δ of successful derivations of G in S such that $\mathcal{T} \models ans(\delta) \leftrightarrow \bigvee_{\gamma \in D_\delta} ans(\gamma)$. Then, the claim follows immediately by taking $c_1 \vee \cdots \vee c_n$ as $\bigvee_{\delta \in D}(\bigvee_{\gamma \in D_\delta} ans(\gamma))$.

Consider any $\delta \in D$. We generate a derivation δ' in S with initial state $\langle G, \emptyset, \emptyset \rangle$ such that δ' has a $\rightarrow_{\tau/c}$ transition for each $\rightarrow_{\tau/c}$ transition of δ in the way given in Lemma 13, and an *empty* transition for each $\rightarrow_s$ transition of δ. Using Lemma 13 and the fact that $\rightarrow_s$ transitions in S_{rel} preserve equivalence of the constraint stores, it is easy to show that if $\langle \emptyset, C \rangle$ is the final state of δ, then the last state of δ' has the form $\langle \emptyset, C_1, C_2 \rangle$ with $\mathcal{T} \models C \leftrightarrow C_1 \wedge C_2$.

We obtain the set D_δ mentioned above by completing δ' with one $\rightarrow_s$ transition from $\langle \emptyset, C_1, C_2 \rangle$ for each possible arrangement of the shared variables $\tilde{v}$ between C_1 and C_2 that is consistent with both stores. Observe that, since $C_1 \cup C_2$ is satisfiable, for being equivalent to the final constraint store of a successful derivation, we are guaranteed by Prop. 4 that at least one arrangement of $\tilde{v}$ is consistent with both C_1 and C_2 and, consequently, that D_δ is non-empty. It follows that, for every $\gamma \in D_\delta$, $\mathcal{T} \models ans(\gamma) \leftrightarrow \exists_{-\tilde{x}} (C_1 \wedge C_2 \wedge ar(\tilde{v}))$, for some arrangement $ar(\tilde{v})$.

Observing that the disjunction of all the arrangements of $\tilde{v}$ is a valid formula, it is then easy to deduce the following chain of logical equivalences in $\mathcal{T}$

$$\begin{aligned} \exists_{-\tilde{x}} C \leftrightarrow \exists_{-\tilde{x}} (C_1 \wedge C_2) &\leftrightarrow \exists_{-\tilde{x}} (C_1 \wedge C_2 \wedge \textstyle\bigvee_{ar(\tilde{v})} ar(\tilde{v})) \\ \leftrightarrow \textstyle\bigvee_{ar(\tilde{v})} \exists_{-\tilde{x}} (C_1 \wedge C_2 \wedge ar(\tilde{v})) &\leftrightarrow \textstyle\bigvee_{\gamma \in D_\delta} ans(\gamma) \end{aligned}$$

which concludes our proof.

2. The result follows as a consequence of the corresponding result for relaxed CLP and the construction in the proof of case 1 above. □

Observe that the necessity to consider multiple derivations to show the completeness of the system is not generated is already present in the CLP scheme itself. Our extension, however, may increase the number of necessary derivations because $\rightarrow_s$ transitions can generate multiple successful derivations, instead of just one, whenever more than one arrangement of variables is consistent with both the constraint stores.

By essentially the same arguments given for the relaxed CLP scheme, we can also prove soundness and completeness of Negation-As-Failure in MCLP($\bar{\mathcal{X}}$).

Proposition 15 Negation-as-Failure. *In an ideal MCLP($\bar{\mathcal{X}}$) system, a goal G is finitely failed for a program P iff $P^*, \mathcal{T} \models \neg G$.*

6 Conclusions

In this paper, we have described a way of extending the CLP($\mathcal{X}$) scheme to admit constraint theories generated as the union of several stably-infinite theories with pairwise disjoint signatures. The main idea of the extension is to incorporate in the scheme a well-known method of obtaining a satisfiability procedure for a union theory as the combination, by means of variable equality sharing, of the satisfiability procedures of each component theory.

By adopting a non-deterministic equality sharing mechanism, we have been able to prove that the main properties of our extension directly compare to those of the original scheme, provided that the CLP($\mathcal{X}$) consistency test on the constraint store is relaxed from satisfiability in a single structure to satisfiability in an elementary class thereof.

Specifically, we have first claimed that the relaxation of the satisfiability test (which gives rise to what we called a *relaxed CLP scheme*) does not modify the original soundness and completeness properties, even in the case of the negation-as-failure inference rule. Then, we have shown how the results given for the relaxed CLP scheme lift to our extension.

Finally, we would like to point out the advantages of adopting a non deterministic version of the original equality-sharing mechanism by Nelson and Oppen. On the theoretical side, our version fits rather nicely into the CLP scheme as it simply adds another level of "don't-know" non-determinism (corresponding to the choice of a variable arrangement) into the computational paradigm. On the practical side, where incremental solvers are already available for each constraint theory, not only does this scheme preserve their incrementality, a key computational feature for the implementation of any CLP system, but also allows one to use them as they are, with no modification whatsoever to their code or interface.

7 Acknowledgments

We would like to thank Michele Zito and Joshua Caplan for a number of discussions on some model-theoretic issues related to this work. This work is partially supported by grant DACA88-94-0014 from the US Army Construction Engineering Laboratories.

References

1. Franz Baader and Klaus U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. In *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 50–65. Springer-Verlag, 1992.
2. Franz Baader and Klaus U. Schulz. Combination of constraint solving techniques: An algebraic point of view. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, volume 914 of *Lecture Notes in Computer Science*, pages 50–65. Springer-Verlag, 1995.

3. Franz Baader and Klaus U. Schulz. On the combination of symbolic constraints, solution domains, and constraint solvers. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming, Cassis (France)*, September 1995.
4. Alexandre Boudet. Combining unification algorithms. *Journal of Symbolic Computation*, 16(6):597–626, December 1993.
5. E. Domenjoud, F. Klay, and C. Ringeissen. Combination techniques for non-disjoint equational theories. In A. Bundy, editor, *Proceedings 12th International Conference on Automated Deduction, Nancy (France)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 267–281. Springer-Verlag, 1994.
6. A. Herold. Combination of unification algorithms. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Artificial Intelligence*, pages 450–469. Springer-Verlag, 1986.
7. Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. *Journal of Logic Programming*, 1991.
8. Joxan Jaffar and Jean-Louis Lassez. Constraint Logic Programming. Technical Report 86/74, Monash University, Victoria, Australia, June 1986.
9. Joxan Jaffar and Michael Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
10. Hélène Kirchner and Christophe Ringeissen. A constraint solver in finite algebras and its combination with unification algorithms. In K. Apt, editor, *Proc. Joint International Conference and Symposium on Logic Programming*, pages 225–239. MIT Press, 1992.
11. Michael Maher. Logic semantics for a class of committed-choice programs. In Jean-Louis Lassez, editor, *ICLP'87: Proceedings 4th International Conference on Logic Programming*, pages 858–876, Melbourne, May 1987. MIT.
12. Greg Nelson. Combining satisfiability procedures by equality-sharing. *Contemporary Mathematics*, 29:201–211, 1984.
13. Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, October 1979.
14. Manfred Schmidt-Schauß. Combination of unification algorithms. *Journal of Symbolic Computation*, 8(1–2):51–100, 1989.
15. Joseph. R. Shoenfield. *Mathematical Logic*. Addison-Wesley, Reading, MA, 1967.
16. Robert E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31:1–12, 1984.
17. Cesare Tinelli. Extending the CLP scheme to unions of constraint theories. Master's thesis, Department of Computer Science, University of Illinois, Urbana-Champaign, Illinois, October 1995.
18. Cesare Tinelli and Mehdi Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In *(to appear in) Proceedings of the 1st International Workshop "Frontiers of Combining Systems", Munich (Germany)*, Applied Logic. Kluwer, 1996.
19. K. Yelik. Unification in combinations of collapse-free regular theories. *Journal of Symbolic Computation*, 3(1–2):153–182, April 1987.

This article was processed using the LaTeX macro package with LLNCS style