The Journal of Functional and Logic Programming

The MIT Press

Volume 1998, Article 6

10 December, 1998

ISSN 1080–5230. MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142-1493, USA; (617)253-2889; *journals-orders@mit.edu, journals-info @mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://www.cs.tu-berlin.de/journal/jflp/*
- *http://mitpress.mit.edu/JFLP/*
- *gopher.mit.edu*
- *ftp://mitpress.mit.edu/pub/JFLP*

Constraint Logic Programming over Unions of Constraint Theories

Cesare Tinelli Mehdi T. Harandi

10 December, 1998

Abstract

In this paper, we present 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. We then show how the main soundness and completeness results of constraint logic programming lift to our extension.

1 Introduction

The constraint logic programming (CLP) scheme was originally developed by Jaffar and Lassez [JL86] as a principled way to combine the 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 these domains do not share function or predicate symbols. The scheme, however, does not deal with *mixed* terms (i.e., terms built with function symbols from different signatures) and corresponding mixed constraints. The reason for this limitation is that although the CLP scheme in principle allows multiple constraint theories, each with its own constraint solver, it is not designed to operate on their *combination*, to which mixed constraints belong.

In general, we can always instantiate the CLP scheme with a suitable constraint domain once we have a constraint solver for that domain, 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. Ideally, we would like to build more complex solvers by combining simpler ones, much the same way we build conventional programs by combining smaller modules (see [GPT96] for a generalized approach). In recent years, considerable research has focused on both domain and solver combinations (see, for instance, [BS95a, BS95b, KS96, NO79, Sho84]) although most of the efforts have been concentrated on unification problems and equational theories ([BS92, Bou93, DKR94, Her86, KR92, SS89, Yel87], among others).

The current results of these investigations are limited in scope, and a deep understanding of many model- and proof-theoretic issues involved is still out of reach. Despite that, in this paper we attempt to show the effectiveness of combination techniques by adapting an existing combination method and incorporating it into the CLP scheme, with few modifications to the scheme itself. We present an extension of the scheme that can include constraint domains built as the combination of a number of independent domains, such as, for instance, the domains of finite trees, real numbers, lists, strings, partial orders, and so on. An earlier, less detailed version of our extension appeared in [TH96a].

1.1 Notation and Conventions

We adhere rather closely to the notation and definitions given in [Sho67] for what concerns mathematical logic in general, and [JM94] for what concerns constraint logic programming in particular. We report here the most notable notational conventions followed. Further notation, which may appear in the sequel, follows the common conventions of the two fields.

The letters v, x, y, z denote logical variables. Although these are metasymbols ranging over the set of logical variables, by a slight abuse of notation, we will occasionally use them in our examples as the actual logical variables of the given object language. The letters s, t denote first-order terms, p, q predicate symbols, f, g function symbols, a, b, h atoms, A a multiset of atoms, c, d constraints, C, D multisets of constraints, φ, ψ first-order formulas, and

ϑ a value assignment, or valuation, to a set of variables. As usual, the letter ω denotes the first infinite ordinal number.

Some of the above 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 just sets of symbols (as opposed to sequences). When $\tilde{s}$ and $\tilde{t}$ both have length n, the equation $\tilde{s} = \tilde{t}$ stands for the system of equations $\{s_1 = t_1 \wedge \ldots \wedge s_n = t_n\}$.

The notation $\varphi(\tilde{x})$ is used to indicate that the free variables of φ are *exactly* the ones in $\tilde{x}$. In general, $var(\varphi)$ is the set of all the free variables of φ. Analogously, if S is a set of formulas, $var(S)$ denotes the set $\bigcup_{\varphi \in S} var(\varphi)$. 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 $\tilde{\exists}\, \varphi$ and $\tilde{\forall}\, \varphi$ stand for the existential, respectively universal, closure of φ.

Where $\mathcal{M}$ is a structure in the sense of model theory and φ is a sentence, that is, a closed formula, the notation $\mathcal{M} \models \varphi$ means that $\mathcal{M}$ satisfies φ or, equivalently, that φ is true in $\mathcal{M}$. For brevity, where $\tilde{x}$ is a sequence of variables, we will call *$\mathcal{M}$-valuation* of $\tilde{x}$ a valuation of $\tilde{x}$ into the universe of $\mathcal{M}$. If φ is a formula and ϑ is an $\mathcal{M}$-valuation of φ's free variables, the notation $\mathcal{M} \models \varphi\vartheta$ means that ϑ satisfies φ in $\mathcal{M}$. Notice that, in analogy with substitutions, we write a valuation application in postfix form.

Where $\mathcal{S}, \mathcal{T}$ are sets of sentences, $Mod(\mathcal{T})$ denotes the set of all models of $\mathcal{T}$, $\mathcal{T} \models \varphi$ means that $\mathcal{T}$ logically entails $\tilde{\forall}\, \varphi$, while $\mathcal{S}, \mathcal{T} \models \varphi$ stands for $\mathcal{S} \cup \mathcal{T} \models \varphi$. We will often identify first-order theories with their deductive closure. We will also identify the union of a finite multiset of formulas with their logical conjunctions.

1.2 Organization of the Paper

In Section 2, we briefly describe the Jaffar-Lassez scheme, which we will simply refer to as the *CLP scheme*, and motivate the need for mixed constraints, which the CLP scheme does not explicitly consider. 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 this combination method to extend the CLP scheme and allow composite constraint domains and mixed terms over them. In Section 5, we describe our proposed extension more formally and provide logical and operational semantics for it. In Section 6, we prove some soundness and completeness properties of the new scheme. In Section 7, we summarize the main contribution of this paper, outlining directions for further development.

2 The Problem

In essence, constraint logic programming is logic programming over a computation domain other than the Herbrand domain. This is often achieved by retaining logic programming's computational paradigm, goal reduction by SLD resolution, and replacing term unification with constraint solving over the new background domain, the *constraint domain.*

The CLP scheme formalizes this idea essentially by assuming (1) a computation structure corresponding to the constraint domain; (2) a first-order axiomatization of the main properties of the domain; and (3) a constraint language for expressing constraints over the domain objects. The various CLP languages are then seen as instances of the scheme CLP($\mathcal{X}$), with the parameter $\mathcal{X}$ standing for the quadruple

$$\mathcal{X} := \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$$

where Σ is a signature containing the constraint predicate and function symbols along with their arities, $\mathcal{D}$ is a Σ-structure representing the constraint domain over which computation is performed, $\mathcal{L}$ is the constraint language, that is, a 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 relation;
- $\mathcal{L}$ contains an identically true predicate, an identically false predicate, and at least all the *primitive constraints* (see later);
- $\mathcal{L}$ is closed under variable renaming, conjunction, and existential quantification; and
- $\mathcal{D}$ and $\mathcal{T}$ *correspond* on $\mathcal{L}$, that is, $\mathcal{D}$ is a model of $\mathcal{T}$ and every formula of $\mathcal{L}$ satisfiable in $\mathcal{D}$ is satisfiable in every model of $\mathcal{T}$.

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

The number of instances of CLP($\mathcal{X}$) has grown so much in the last years that it would be impractical to cite them all. Classical CLP($\mathcal{X}$) systems, so to speak, are CLP($\mathcal{R}$) [JMSY92], which computes over the constraint domain of linear arithmetic over the real numbers, CHIP [DVS$^+$88], which also computes over the domain of Boolean values and functions, and Prolog III [Col87], which has a host of constraint domains. Prolog itself can be seen as CLP($\mathcal{FT}$) where $\mathcal{FT}$ is the constraint domain of finite trees represented as terms in the Herbrand universe.

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, as mentioned above. They can be seen as CLP($\mathcal{X}_1, \ldots, \mathcal{X}_n$) systems where the various domains 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 (nonvariable) terms entirely built with symbols from the same signature. Variables are considered to be in all the signatures; in programming language terminology, they are *untyped*, at least until the first constraint containing them is processed. From that point on, they can only be bound to terms of a particular domain.

Thus, although in one way or another all CLP systems use more than one constraint domain, they do not freely allow mixed terms or constraints, that is, constraint expressions containing nonvariable symbols from different signatures. Notable exceptions are uninterpreted function symbols, which can generally be applied to any term. In CLP($\mathcal{R}$), for instance, $\mathsf{f}(\mathsf{X} + \mathsf{1}, \mathsf{g}(\mathsf{3}))$ is a legal term, while $\mathsf{f}(\mathsf{X}) + \mathsf{1}$ is not, because the first argument of the constraint function symbol $+$ is not an arithmetic term. Model-theoretically, this is justified by assuming that CLP($\mathcal{R}$)'s constraint domain is actually the (sorted) domain of finite trees of real numbers, where uninterpreted symbols are tree constructors and arithmetic terms are leaf names. A similar, many-sorted approach is also used in Prolog III and its successors [Col90]. In particular, Prolog III does allow more complex mixed constraints; for instance, it allows constraints of the form $\mathit{length}(s) = t$, where s is a term of the list sort, t is a term of the integer sort, and *length* denotes the length function. However,

[1]Prolog II, for instance [JLM87], works with infinite trees instead of finite trees.

such constraints can be processed only if s and t are ground terms or, more generally, are equivalent in the respective constraint stores to some ground term. Otherwise, they are *suspended* until both terms become equivalent to ground terms thanks to the addition of further constraints to the stores.

In general, meaningful, and not necessarily ground, mixed constraints arise naturally in many classes of applications such as program verification, deductive program synthesis, completion procedures, theorem proving, artificial intelligence, and so on. In these applications, input problems do not range simply over one constraint domain, but over some *combination* of a number of them. For instance, in a domain that combines lists with natural numbers, the constraint

$$v = \mathit{cons}(x + 9, y) \ \wedge\ \mathit{first}(y) > z + u$$

(where *cons* is the list constructor and *first* returns the first element of a list) is a very natural one. Similar claims can be made about constraints such as

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

(where $+, -,$ and $\leq$ are arithmetic symbols, and f is an uninterpreted symbol), which often appear in program verification [Nel84]. These types of constraints cannot be dealt with by the present CLP systems, simply because their computational paradigm does not consider them.

For a perhaps more poignant example of this shortcoming of the CLP scheme, consider a CLP($\mathcal{X}_1, \ldots, \mathcal{X}_n$) solver with two constraint domains, $\mathcal{X}_1$ and $\mathcal{X}_2$ and respective solvers S_1 and S_2. Suppose that S_2 is a solver for systems of equations and disequations over the real numbers, while S_1 is a simple scheduler which, given a number of activities classified by type, returns one or more possible orderings, if any, of these activities. The language of S_1 includes atomic constraints such as $x : y$, stating that the activity x is of type y; $x \prec y$, stating that the time point x strictly precedes the time point y; constants such as $a_1, a_2, a_3, \ldots$, denoting activities; constants such as $t_1, t_2, t_3, \ldots$, denoting activity types; and terms such as $\mathit{begin}(x)$ and $\mathit{end}(x)$, denoting the start and end times of activity x, respectively. Depending on their type, some activities can overlap, or must end before some others, or can overlap with others but must start before, and so on. In particular, suppose that activities have nonzero duration (in formulas, $\forall x\ \mathit{begin}(x) \neq \mathit{end}(x)$), an activity of type t_2 can start only *after* an activity of type t_1 has ended (that is, $\forall x, y\ x : t_1 \wedge y : t_2 \rightarrow \mathit{end}(x) \prec \mathit{begin}(y)$), and an activity of type t_3

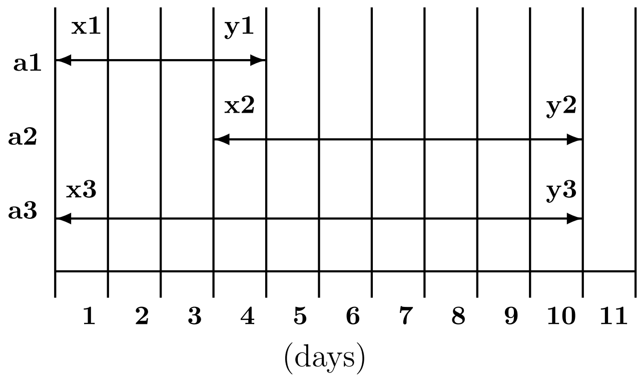

Figure 1: A simple scheduling problem

can overlap with activities of type t_1 or t_2. A typical conjunctive constraint for this solver could be something like:

$$\begin{cases} a_1 : t_1 \\ a_2 : t_2 \\ a_3 : t_2 \\ \mathit{begin}(a_1) \prec \mathit{begin}(a_2) \\ \mathit{begin}(a_1) = \mathit{begin}(a_3) \\ \dots \end{cases}$$

Notice that it is entirely possible to have a solver like this that possesses no arithmetic notion whatsoever. The constraint domain is essentially a set of (typed) intervals that can be partially ordered, and input problems are solved using standard arc-consistency techniques.

Now assume that we need to schedule three activities, $a_1, a_2,$ and a_3, of type $t_1, t_2,$ and t_3, respectively, subject to these requirements: a_1 starts the day a_3 starts and lasts 4 days; a_2 ends the day a_3 ends and lasts 7 days; and a_3 lasts 10 days. As one can immediately verify by looking at Figure 1, there is no possible schedule for these activities, because the problem requirements force a_2 to start the very day that a_1 ends, which we saw is not allowed in the given scheduling domain. Can we use our CLP($\mathcal{X}_1, \dots, \mathcal{X}_n$) system to detect that our scheduling problem is overconstrained? First observe that despite its simplicity, this problem cannot be expressed in the language of S_1 alone, because S_1 does not know about numbers that we have used to define the duration of each activity. But S_2 does, so we can think of formalizing the problem by using both the languages of S_1 and S_2. A simple and intuitive formalization of the problem is given by the following conjunctive constraint:

$$c := \begin{cases} a_1 : t_1 \\ a_2 : t_2 \\ a_3 : t_3 \\ \mathit{begin}(a_1) = \mathit{begin}(a_3) \\ \mathit{end}(a_2) = \mathit{end}(a_3) \\ \mathit{end}(a_1) - \mathit{begin}(a_1) = 3 \\ \mathit{end}(a_2) - \mathit{begin}(a_2) = 6 \\ \mathit{end}(a_3) - \mathit{begin}(a_3) = 9 \end{cases}$$

Since the constraint above combines the languages of the two solvers by using the mixed terms $\mathit{end}(a_i) - \mathit{begin}(a_i)$ for $i = 1, 2, 3$, it cannot be processed by either of them. However, as we will see later, it is easily transformable into an equivalent constraint without mixed terms. As a matter of fact, it is possible to assume that the system itself is able to carry this transformation and convert c into the conjunction of the constraints c_1 and c_2 below:

$$c_1 := \begin{cases} a_1 : t_1 \\ a_2 : t_2 \\ a_3 : t_3 \\ \mathit{begin}(a_1) = \mathit{begin}(a_3) \\ \mathit{end}(a_2) = \mathit{end}(a_3) \\ x_1 = \mathit{begin}(a_1) \\ x_2 = \mathit{begin}(a_2) \\ x_3 = \mathit{begin}(a_3) \\ y_1 = \mathit{end}(a_1) \\ y_2 = \mathit{end}(a_2) \\ y_3 = \mathit{end}(a_3) \end{cases} \qquad c_2 := \begin{cases} y_1 - x_1 = 3 \\ y_2 - x_2 = 6 \\ y_3 - x_3 = 9 \end{cases}$$

where $x_1, x_2, x_3, y_1, y_2,$ and y_3 are free variables. Now each c_i is a constraint entirely in the language of S_i, and so it can be readily processed by S_i. Furthermore, both c_1 and c_2 are solvable in their respective domains. A closer look at likely solved forms for c_1 and c_2, as given below, will convince the reader that this is indeed so:

$$c'_1 := \begin{cases} \mathit{end}(a_1) \prec \mathit{begin}(a_2) \\ \mathit{begin}(a_1) = \mathit{begin}(a_3) \\ \mathit{end}(a_2) = \mathit{end}(a_3) \\ x_1 = \mathit{begin}(a_1) \\ \dots \end{cases} \qquad c'_2 := \begin{cases} y_1 = x_1 + 3 \\ y_2 = x_2 + 6 \\ y_3 = x_3 + 9 \end{cases}$$

The problem is that our CLP($\mathcal{X}_1, \dots, \mathcal{X}_n$) system has no way to realize that the original constraint is nevertheless unsolvable. This is because S_2's store is actually underconstrained. It is not difficult to see that all the solutions ϑ of c_1 are such that $x_1\vartheta = x_3\vartheta$, $y_2\vartheta = y_3\vartheta$, and $y_1\vartheta \neq x_2\vartheta$,[2] but this information is not known to S_2. If we passed it to S_2 in the form of the constraint $x_1 = x_3 \wedge y_2 = y_3 \wedge y_1 \neq x_2$, which is indeed a legal input constraint for S_2, its store would become equivalent to something like:

$$\begin{cases} x_1 = x_3 \\ y_2 = y_3 \\ x_2 \neq y_1 \\ \\ x_2 = x_3 + 3 \\ y_1 = x_3 + 3 \\ y_3 = x_3 + 9 \end{cases}$$

which is clearly unsatisfiable. Now, although $x_1 = x_3 \wedge y_2 = y_3 \wedge y_1 \neq x_2$ is a constraint that S_2 understands, it is unknown to S_2 because it is only entailed by S_1's constraint store and the system has no way of communicating it to S_2. What is missing in the system, therefore, is a mechanism for propagating constraints of this kind from one solver to another. Ideally, such a mechanism would propagate just enough information for each solver to return compatible—that is, globally consistent—solutions. In the example above, this mechanism would effectively provide the CLP($\mathcal{X}_1, \dots, \mathcal{X}_n$) system with a virtual solver for a combination of its scheduling and arithmetic domains.

When attempting to achieve intersolver constraint propagation, one is faced with serious model-theoretic questions about the nature of the constraint domains at hand and the type of information that can and must be propagated. Again, recall that a mixed constraint is not a query over any

[2]The last one because $\mathit{end}(a_1) \prec \mathit{begin}(a_2)$.

particular domain of a CLP($\mathcal{X}_1, \ldots, \mathcal{X}_n$) system, but over some *combination* of them. Defining an adequate (class of) combined constraint domain(s) out of a number of component domains is already a hard task [BS95a, KS96]; but even when that is done, finding sound and complete solver-combination algorithms can still prove extremely difficult. As a matter of fact, most combination problems have no possible solutions. It is relatively easy to find domains or theories such that a certain satisfiability problem is decidable in each of them and undecidable in their combination; for instance, see [DKR94, BT97]. As a result, all the existing combination methods pose more or less serious restrictions on the classes of constraint domains and languages they can combine. Furthermore, essentially all of them require the component constraint languages to have pairwise-disjoint signatures.[3]

A relatively general method has been proposed by Nelson and Oppen in [NO79] for combining decision procedures for the satifiability of quantifier-free formulas with respect to first-order theories with equality. In this paper, we will show how the main idea of the Nelson-Oppen method can be incorporated into the CLP scheme to provide a systematic and consistent treatment of a large class of mixed constraints and combined domains. In essence, we will show how to convert a system of type CLP($\mathcal{X}_1, \ldots, \mathcal{X}_n$) into a system of type CLP($\bar{\mathcal{X}}$), where $\bar{\mathcal{X}}$ is the constraint structure generated by a suitable combination of all the $\mathcal{X}_i$s.

3 Combining Satisfiability Procedures

We start with a brief review of the model-theoretic properties of the Nelson-Oppen combination method. For a description of the original combination procedure, refer to [NO79] or [Nel84]. These properties have been used to prove that the Nelson-Oppen method is sound and complete for a certain class of component theories.

A formula is in *simple conjunctive normal form* if it is a conjunction of *literals*, that is, a conjunction of atomic and negated atomic formulas. Given a signature Σ, we denote by *sCNF*(Σ) the set of all the Σ-formulas in simple conjunctive normal form. We consider the following notion of satisfiability.

[3]But see [Rin96] for a first attempt to lift this restriction. A more general approach is proposed in [TR98].

Definition 1 *A formula φ is* satisfiable in *a theory $\mathcal{T}$ if and only if it is satisfiable in some model of $\mathcal{T}$, that is, if and only if there exists a model $\mathcal{M} \in Mod(\mathcal{T})$ such that $\mathcal{M} \models \tilde{\exists}\, \varphi$.*

Now, let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two theories with equality and respective signatures Σ_1, Σ_2 such that $\Sigma_1 \cap \Sigma_2 = \emptyset$.[4] 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 of $\mathcal{T}_1$ 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 $sCNF(\Sigma_i)$, we can generate a procedure that decides the satisfiability in $\mathcal{T}_1 \cup \mathcal{T}_2$ of any formula $\varphi \in sCNF(\Sigma_1 \cup \Sigma_2)$ 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 $sCNF(\Sigma_i)$. If φ is not already in separate form, we can apply a conversion procedure that, given φ, returns an equivalent separate form of φ. To describe such a procedure, we need to introduce some definitions and notation that we have adapted from those in [BS92], among others.

Consider the signatures introduced above. For $i = 1, 2$, a member of Σ_i is an *i-symbol*. A term t is an *i-term* if it is a variable or if its root symbol is an i-symbol. An i-atomic formula (*i-atom* for short) is defined analogously. A subterm of an i-term t is an *alien* subterm of t if it is a j-term, with $j \neq i$, and all of its superterms in t are i-terms. An i-term is *pure* (*i-pure* for short) if it only contains i-symbols. Alien predicate arguments are defined analogously. An i-atom is pure if all of its arguments are i-pure. Pure formulas are thus defined in the obvious way. Observe that, given our assumption on the various signatures, a variable is an i-term for any i and an equation is always a pure atom if at least one of its arguments is a variable. Furthermore, an equation between two variables is both a 1-atom and a 2-atom.

3.1 The Separation Procedure

Let φ be a formula of $sCNF(\Sigma_1 \cup \Sigma_2)$, seen as a multiset of literals:

- We first perform *variable abstraction* on φ by recursively replacing each alien subterm t with a newly generated variable z and adding the equation $z = t$ to φ. In case of equations between nonvariable terms, the choice of the term to replace with a new variable is made arbitrarily.

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

$$\{f(f(x)-f(y)) \neq f(z),\ y+z \leq x\}$$

$$\{\ \{f(x_1) \neq f(z),\ x_2 = f(x),\ x_3 = f(y)\},\ \ \{x_1 = x_2 - x_3,\ y+z \leq x\}\ \}$$

Figure 2: A multiset of mixed constraints and its separate form

- We then partition the new multiset in $m \leq 2$ sets containing only i-pure literals, respectively. In analogy with the step above, equations between variables are partitioned arbitrarily.

The resulting partition can be seen as an *sCNF* formula of the form $\varphi_1 \wedge \varphi_2$, where each φ_i is an i-pure *sCNF* formula. An example of the result of the separation procedure, where $\Sigma_1 := \{f\}$ and $\Sigma_2 := \{+,-,\leq\}$, is given in Figure 2. Although a formula may have many separate forms, these forms are all equivalent modulo variable renaming and the logical properties of conjunction and equality. Hence, it is appropriate to speak of *the* separate form of a formula. We indicate the separate form of a formula $\varphi \in sCNF(\Sigma)$ with $\ddot{\varphi}$. For notational convenience, we can always think of $\ddot{\varphi}$ as a conjunction of the form $\varphi_1 \wedge \varphi_2$, where each φ_i is an i-pure *sCNF* formula, even if φ does not contain any i-symbol for some $i = 1, 2$. In that case, φ_i is defined as the identically true formula—which can be thought of as belonging to all $sCNF(\Sigma_i)$s.

It is easy to show that any $\varphi \in sCNF(\Sigma)$ is logically equivalent to $\exists \tilde{z}\, \ddot{\varphi}$, where $\tilde{z}$ is the set of fresh variables introduced by the separation procedure. According to our notion of satisfiability, this entails the following:

Proposition 1 *An sCNF formula is satisfiable in a theory $\mathcal{T}$ if and only if its separate form is satisfiable in $\mathcal{T}$.*

As we saw in the previous section, the problem with deciding the satisfiability of a formula φ by analyzing its separate form is that, in general, each subformula φ_i of $\ddot{\varphi}$ could be singly satisfiable without its conjunction being satisfiable. Hence, to be able to apply distinct satisfiability procedures to each φ_i and correctly decide the satisfiability of φ, we need to establish some sort of communication between the various procedures. In Nelson and Oppen's method, such communication is achieved by propagating from one procedure to the other any implied equalities between the variables of $\ddot{\varphi}$. The correctness of this approach is ensured by Theorem 1 below [TH96b], provided that the component theories are stably infinite [Opp80].

Definition 2 *A consistent, universal theory $\mathcal{T}$ of signature Σ is called* stably infinite *if every quantifier-free Σ-formula satisfiable in $\mathcal{T}$ is satisfiable in an infinite model of $\mathcal{T}$.*

Definition 3 *If P is any partition on a set of variables V, and R is the corresponding equivalence relation, we call the* arrangement of V (determined by P) *the set:*

$$ar(V) \;:=\; \{x = y \mid x, y \in V, xRy\} \cup \{x \neq y \mid x, y \in V,\ \textit{not } xRy\}$$

In practice, $ar(V)$ is made of all the equations between any two equivalent variables of V and all the disequations between any two nonequivalent variables.

Theorem 1 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably infinite theories with disjoint signatures Σ_1 and Σ_2. Let $\varphi \in sCNF(\Sigma_1 \cup \Sigma_2)$ and assume that $\ddot{\varphi}$ is $\varphi_1 \wedge \varphi_2$ with $\varphi_i \in sCNF(\Sigma_i)$ for $i = 1, 2$. Then, where $\tilde{x} := var(\varphi_1) \cap var(\varphi_2)$, the following are equivalent:*

1. *φ is satisfiable in $\mathcal{T}_1 \cup \mathcal{T}_2$.*
2. *There is an arrangement $ar(\tilde{x})$ such that $\varphi_i \wedge ar(\tilde{x})$ is satisfiable in $\mathcal{T}_i$, for $i = 1, 2$.*

The essence of this result is that given the right conditions on the component theories, we can test the satisfiability of φ above in the combined theory by testing the satisfiability of each φ_i in the corresponding component theory, provided that we add a *global consistency* restriction to both φ_1 and φ_2. The original contribution by Nelson and Oppen was to show that this restriction must only be a certain (dis)equality constraint on the variables shared by φ_1 and φ_2.

The theorem above can be actually lifted to the combination of $n \geq 2$ stably infinite theories with pairwise disjoint signatures by extending the definition of the separate form of a formula as obvious, and letting $\tilde{x}$ be the union of all the *shared* variables.[5] An immediate proof of the general case can

[5]Where by "shared variable" we now mean a variable occurring in at least two distinct pure formulas φ_i and φ_j of the separate form $\varphi_1 \wedge \ldots \wedge \varphi_n$.

be given by using the following modularity result and applying Theorem 1 iteratively.

Proposition 2 ([Rin96]) *The union of two stably infinite theories with disjoint signatures is stably infinite.*

To better understand the scope of the combination results above, recall that they require the component theories to be stably infinite. Now, although several interesting constraint theories are indeed stably infinite [Nel84], many others are not; as an example, just consider the universal theory of a finite structure, all of whose models are finite.[6] Luckily, the class of stably infinite theories can be considerably expanded without compromising any of the combination results: it is enough to simply lift the restriction in Definition 2 that the theory be universal. With the new definition, then every complete theory admitting an infinite model, for instance, becomes stably infinite. Since a proof of the correctness of such lifting is beyond the scope of this paper, we refer the interested reader to [TR98], where the claim is actually a mere consequence of more general combination results. There, one can also see that *stable infiniteness* is a sufficient but not a necessary condition for Theorem 1 to hold, which means that in principle the combination results above may apply even to some nonstably infinite theories.

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

Recall that our main goal is to extend the CLP scheme to go from a language of type CLP($\mathcal{X}_1, \ldots, \mathcal{X}_n$), where $\{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$ is a set of signature-disjoint constraint structures, to a language of type CLP($\bar{\mathcal{X}}$), where $\bar{\mathcal{X}}$ is a combination of these structures in the sense that it allows any computation performed in CLP($\mathcal{X}_1, \ldots, \mathcal{X}_n$) and, furthermore, poses no signature restriction on term construction. To avoid confusion, we will refer to the extended language and scheme as MCLP($\bar{\mathcal{X}}$).

The reason we are interested in combinations of satisfiability procedures is that CLP systems already utilize separate satisfiability 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. For instance, CLP($\mathcal{R}$) uses a

[6]It is rather easy to write a universal formula stating that there exist at most n individuals [CK90].

standard unification algorithm for equalities between uninterpreted terms (*unification constraints*) and a constraint solver, based on linear programming techniques, for arithmetic constraints. A main module, which is a Prolog-like SLD-resolution engine, takes care of the goal-reduction process and the communication between the unification module and the constraint solver. Observe that communication is needed because CLP($\mathcal{R}$) admits unification constraints such as $\mathsf{f(X + Z) = f(3)}$, which generate arithmetic constraints as well, in this case $\mathsf{X + Z = 3}$. Other systems are analogous and possibly more complex, because they support many constraint domains, as mentioned earlier.

Intuitively, if we rewrite MCLP($\bar{\mathcal{X}}$) statements into a separate form similar to the one mentioned in Section 3, we may be able to use the various constraint solvers much the same way the Nelson-Oppen method uses the various satisfiability procedures. Moreover, the machinery we will need for an MCLP($\bar{\mathcal{X}}$) system will be essentially the same we would need for a corresponding CLP($\mathcal{X}_1, \ldots, \mathcal{X}_n$) 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, (1) identifies the variables that the constraint shares with those in the other solvers, (2) creates a backtrack point in the computation and chooses a (novel) arrangement of those variables, and (3) passes the arrangement to all the solvers.

We formalize this idea in the following sections, starting with a brief review of the logical and operational model of the original CLP scheme.

4.1 The Semantics of CLP($\mathcal{X}$)

Two of the main semantics of CLP are the logical semantics and the top-down operational semantics. In describing them below, we follow almost literally [JM94]'s more recent notation and definitions.

4.1.1 Logical Semantics

Where $\mathcal{X} := \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$ is defined as in Section 2, we will call an *atom* an expression of the form $p(\tilde{t})$, where p is a user-defined predicate and $\tilde{t}$ is a sequence of Σ-terms. We will call a *primitive constraint* any atomic Σ-

formula,[7] and simply a *constraint* any formula in $\mathcal{L}$. The standard format for a statement, or rule, of a CLP($\mathcal{X}$) program is:

$$p(\tilde{t}) \quad \leftarrow \quad b_1(\tilde{t}_1), \ldots, b_n(\tilde{t}_n)$$

while that of a query, or goal, to the program is

$$b_1(\tilde{t}_1), \ldots, b_n(\tilde{t}_n)$$

where $n \geq 0$, $p(\tilde{t})$ is an atom, and each $b_i(\tilde{t}_i)$ is either an atom or a constraint. A rule whose body is composed only of (zero or more) constraints is called a *fact*. A goal must contain at least one atom or constraint. *Simple goals* are goals that contain exactly one atom or constraint.

Analogously to the logic programming paradigm, the logical semantics of CLP interprets a rule and a goal of the above form as the universal formula

$$\tilde{\forall}\,(b_1(\tilde{t}_1) \wedge \ldots \wedge b_n(\tilde{t}_n) \rightarrow p(\tilde{t}))$$

and the existential formula

$$\tilde{\exists}\,(b_1(\tilde{t}_1) \wedge \ldots \wedge b_n(\tilde{t}_n))$$

If P is a CLP program, when it is convenient we will also denote by P the set of sentences of the above kind that are associated with the rules of P. The logical semantics above allows us to rewrite every CLP rule into the equivalent *normal form*:

$$p(\tilde{x}) \quad \leftarrow \quad \tilde{x} = \tilde{t}, \tilde{x}_1 = \tilde{t}_1, \ldots, \tilde{x}_n = \tilde{t}_n,\ b_1(\tilde{x}_1), \ldots, b_n(\tilde{x}_n)$$

Similarly, we can rewrite goals into an equivalent normal form as well. For simplicity, we will assume such normal form from now on. Moreover, since the constraint language is assumed to be closed under conjunction, with no loss of generality, we will sometimes represent a rule concisely as $p(\tilde{x}) \leftarrow c, A$ where c is a possibly composite constraint and A is a possibly empty multiset of atoms. Sometimes we will regard the body of a rule generically as a multiset of atoms and constraints, and therefore represent the rule simply as $p(\tilde{x}) \leftarrow B$. Similarly, we will represent a goal as the multiset of atoms and constraints G.

[7]This also includes the equation between two Σ-terms.

A more appropriate logical semantics for CLP programs takes into account the fact that CLP systems implement predicate completion. In that case, if the set of all the rules in the program that have the same predicate symbol as their head is (possibly, after some variable renaming)

$$\{p(\tilde{x}) \leftarrow B_1, \ldots,\, p(\tilde{x}) \leftarrow B_m\}$$

then that set is associated with the formula:

$$\forall \tilde{x}\ (p(\tilde{x}) \leftrightarrow \exists_{-\tilde{x}}\, B_1 \vee \ldots \vee \exists_{-\tilde{x}}\, B_m) \tag{1}$$

Moreover, each of the infinitely many atoms $p(\tilde{x})$ not occurring in the program as the head of a rule is associated with the formula:

$$\forall \tilde{x}\ \neg p(\tilde{x}) \tag{2}$$

We will denote by P^* the *completion of* P, that is, the set of all formulas of the forms (1) and (2) that are associated with a program P.

4.1.2 Operational Semantics

Here we consider only a top-down model of execution. Although this is not the only way to describe the functioning of a CLP system, we can adopt this goal-reduction-based view of computation to formalize the operational semantics of most CLP systems. We also assume that the system is composed of a main module that performs goal reduction and a specialized module, the constraint solver, that processes the constraints.

With these assumptions, the computation of a CLP($\mathcal{X}$) system can be illustrated formally as a sequence of state transitions. Each state of the computation is completely described by either the symbol *fail* or the pair $\langle A, C\rangle$, where A is a multiset of atoms and constraints and C is a multiset of constraints. Intuitively, A is the current set of subgoals yet to be considered, and C, the *constraint store*, is the set of constraints already passed to the constraint solver.[8]

[8]Mainly to allow *incomplete* solvers into the CLP scheme, [JM94] defines a transition as the triple $\langle A, C, S\rangle$ where S is a multiset of *delayed* constraints. For the sake of simplicity, we have decided to ignore the issue of delayed constraints in this paper. We would like to point out, however, that our extension could be easily applied to an operational model including delayed constraints with comparable results.

Transitions between states are defined by three different operators, and are represented as $\rightarrow_r$, $\rightarrow_c$, and $\rightarrow_s$. The transition system utilizes a selection function, *select*, defined over multisets of atoms and constraints, and a total Boolean function, *Sat*, defined over multisets of constraints. The operators are defined as follows where P is the CLP program in execution:[9]

r_1: $\langle A, C\rangle \rightarrow_r \langle A \cup B - p(\tilde{x}),\ C \cup \tilde{x} = \tilde{y}\rangle$

where $p(\tilde{x}) = \mathit{select}(A)$ is an atom in A and $p(\tilde{y}) \leftarrow B$ is a renaming to fresh variables of a rule of P;

r_2: $\langle A, C\rangle \rightarrow_r \mathit{fail}$

where $p(\tilde{x}) = \mathit{select}(A)$ is an atom, and P contains no rule whose head's predicate symbol is p;

c: $\langle A, C\rangle \rightarrow_c \langle A - c, C \cup c\rangle$

where $c = \mathit{select}(A)$ is a constraint in A;

s_1: $\langle A, C\rangle \rightarrow_s \langle A, C\rangle$

when $\mathit{Sat}(C)$ succeeds; and

s_2: $\langle A, C\rangle \rightarrow_s \mathit{fail}$

when $\mathit{Sat}(C)$ fails.

The function *Sat* is implemented by means of a constraint solver for $\mathcal{L}$ with respect to the theory $\mathcal{T}$. *Sat* decides the *$\mathcal{T}$-satisfiability* of the conjunction of the constraints in the constraint store: where $C = \{c_1, \ldots, c_p\}$, $\mathit{Sat}(C)$ succeeds if and only if $\mathcal{T} \models \tilde{\exists}(c_1 \wedge \ldots \wedge c_p)$. Alternatively and more commonly, *Sat* performs a satisfiability test just in the particular structure $\mathcal{D}$, and so succeeds on C if and only if $\mathcal{D} \models \tilde{\exists}\, C$. Observe that the two approaches are equivalent whenever $\mathcal{T}$ and $\mathcal{D}$ correspond on $\mathcal{L}$.

An actual CLP system also has to implement an appropriate computation rule that resolves the nondeterminism of the above transition system by choosing at each state the next-state operator and, in case of $\rightarrow_r$ transitions, by also choosing one of the possible matching rules.

[9]To make the transitions more readable, we will identify a singleton set with its unique element whenever this does not generate confusion.

In CLP($\mathcal{X}$), a *derivation* (in a given program P) of a set A of atoms and constraints is any sequence of applicable transitions that starts with a state of the form $\langle A, C\rangle$ and contains no consecutive $\rightarrow_s$ transitions.[10]

The last state of a finite derivation is a *final* state if no transitions apply to it. A derivation with a final state is *failed* if the final state is *fail*; it is *successful* if the final state has the form $\langle \emptyset, C\rangle$. Observe that by this definition, the final constraint store of a successful derivation is satisfiable in $\mathcal{D}$. If an initial goal G with variables $\tilde{x}$ has a successful derivation with final state $\langle \emptyset, C\rangle$, the formula $\exists_{-\tilde{x}}\, C$ is called the *answer constraint* of the derivation. Observe that by definition, all the free variables of an answer constraint are also free variables of the corresponding initial goal.

A derivation is *fair* if it is failed or such that, for every state $\langle A, C\rangle$ in the derivation, every $a \in A$ is processed in a later transition. A computation rule is *fair* if it only generates fair derivations. We will call derivations $\mathcal{D}$-*derivations* whenever we want to stress the fact that constraint satisfiability is tested on the distinguished structure $\mathcal{D}$.

We can see a derivation as an oriented path in which nodes are represented by the computation states and edges are represented by the state transitions. Then, a *computation tree* of a state s in a program P is the tree rooted at s and containing all the possible derivations from s in P. The computation tree of a goal G is defined as the computation tree of the state $\langle G, \emptyset\rangle$. Observe that while computation trees can be infinite, they are always finitely branching.

5 MCLP($\bar{\mathcal{X}}$): The Extended Scheme

The first issue to deal 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 that represents the intended constraint domain. The combination method we are considering, however, combines *theories*, not structures: it succeeds when the input formula is satisfiable in *some* model of the combined theory. For this reason, our extension will use as its "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 [HS88] (see also [Smo89]). The re-

[10]This last condition is not present in [JM94], but is necessary to guarantee the existence of final states (see later). It also eliminates the need for the concept of a *progressive* system, that is, a system that never infinitely avoids $\rightarrow_r$ and $\rightarrow_c$ transitions in a derivation.

striction is achieved along two dimensions: the constraint language and the set of solution structures. We use *sCNF* formulas as constraints, and axiomatizable classes[11] as the class of structures over which constraint satisfiability is tested. In particular, the class associated with a given MCLP($\bar{\mathcal{X}}$) language is the set of models of the union of the component theories.

Formally, the parameter $\bar{\mathcal{X}}$ in the MCLP($\bar{\mathcal{X}}$) scheme is defined as the tuple:

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

where

- $\Sigma_1, \ldots, \Sigma_n$ are pairwise-disjoint signatures,
- $\mathcal{L}_i := sCNF(\Sigma_i)$ for all $i \in \{1, \ldots, n\}$, and
- $\mathcal{T}_i$ is a stably infinite Σ_i-theory for all $i \in \{1, \ldots, n\}$.

The combined constraint theory for MCLP($\bar{\mathcal{X}}$) is $\mathcal{T} := \mathcal{T}_1 \cup \ldots \cup \mathcal{T}_n$, the combined constraint language is $\mathcal{L} := sCNF(\Sigma)$ with $\Sigma := \Sigma_1 \cup \ldots \cup \Sigma_n$, and the set of solution structures is $Mod(\mathcal{T})$.

We can now describe the logical and operational models of MCLP($\bar{\mathcal{X}}$). Again, we will consider a top-down model of execution.

5.1 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. That is, instead of MCLP($\bar{\mathcal{X}}$), a rule of the form:

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

[11] A class of structures is *axiomatizable* if it coincides with the set of models of some first-order theory.

Assumptions: p, q user-defined, $\{g, f\} \subseteq \Sigma_1$, $\{1, +, <\} \subseteq \Sigma_2$.

$$\begin{array}{lcl} p(g(x)+g(y), z) & \leftarrow & q(f(x+1), z),\ f(x)+f(y) < z \\ \\ p(v_1, z) & \leftarrow & v_1 = v_2 + v_3,\ v_2 = g(x),\ v_3 = g(y), \\ & & q(v_4, z),\ v_4 = f(v_5),\ v_5 = v + 1, \\ & & v_6 + v_7 < z,\ v_6 = f(x),\ v_7 = f(y) \end{array}$$

Figure 3: An MCLP($\bar{\mathcal{X}}$) rule and its separate form

as defined in Section 4.1.1, we consider its *separate form*:

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

obtained by applying to the body of the rule a separation procedure similar to the one described in Section 3.[12] It should be clear that, under the CLP logical semantics, an MCLP($\bar{\mathcal{X}}$) rule and its separate form are equivalent. An example of an MCLP($\bar{\mathcal{X}}$) rule and a possible separate form for it is given in Figure 3.

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.

5.2 Operational Semantics

We assume that for each component theory $\mathcal{T}_i$ and corresponding constraint language $\mathcal{L}_i := sCNF(\Sigma_i)$, we have a decision procedure, Sat_i, for the satisfiability in $\mathcal{T}_i$ of formulas of $\mathcal{L}_i$. We will only consider the case of two component theories here, as the n-component case is a straightforward 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, C_1 is a set of Σ_1-constraints, and C_2 is a set of Σ_2-constraints. A represents the current set of subgoals yet to be considered,

[12]Observe that the body of a rule is in fact an *sCNF* formula.

while each C_i represents the constraint store of the solver implementing Sat_i. State transitions are defined as follows, where P is the MCLP($\bar{\mathcal{X}}$) program in execution:

r_1: $\langle A, C_1, C_2 \rangle \rightarrow_r \langle A \cup B - p(\tilde{x}), C_1', C_2' \rangle$

where $p(\tilde{x}) := \mathit{select}(A)$ is an atom, $p(\tilde{y}) \leftarrow B$ is a renaming to fresh variables of a rule of P, and $C_i' := C_i \cup \tilde{x} = \tilde{y}$ for $i = 1, 2$;

r_2: $\langle A, C_1, C_2 \rangle \rightarrow_r \mathit{fail}$

where $p(\tilde{x}) = \mathit{select}(A)$ is an atom, and P contains no rule whose head's predicate symbol is p;

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

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, and $C_i' := C_i$ otherwise;

s_1: $\langle A, C_1, C_2 \rangle \rightarrow_s \langle A, C_1', C_2' \rangle$

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

s_2: $\langle A, C_1, C_2 \rangle \rightarrow_s \mathit{fail}$

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.

The concepts of derivation, final state, failed derivation, successful derivation, and computation tree in MCLP($\bar{\mathcal{X}}$) can be defined the same way they are defined in CLP($\mathcal{X}$).

Similarly to CLP($\mathcal{X}$), transitions of type $\rightarrow_r$ are just goal-reduction steps. The difference 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 corresponding solver (with 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 method mentioned in Section 3. In the method's terms, for every $\rightarrow_s$ transition, we consider the constraint stores C_1 and C_2 as the 1-pure and 2-pure halves of an *sCNF* formula 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 Theorem 1, all we need to do to enforce the global consistency of the stores is guess an arrangement of their shared variables and then add it to each of them.

For a better feeling of how an MCLP($\bar{\mathcal{X}}$) system works, let us go back to the scenario presented in Section 2, where we had a CLP system with a scheduler S_1 and an arithmetic solver S_2. Assume an MCLP($\bar{\mathcal{X}}$) system with the same solvers, and consider the following simple MCLP($\bar{\mathcal{X}}$) program in separate form:

$$P \quad := \quad \{d(x,y) \leftarrow z_e = e(x), z_b = b(x), z_e - z_b = y - 1\}$$

which defines the duration of an activity.[13] Now consider as input goal the constraint c seen in Section 2, rewritten to use the user-defined predicate d:

$$G \quad := \quad \left\{ \begin{array}{l} a_1 : t_1, a_2 : t_2, a_3 : t_3 \\ b(a_1) = b(a_3), e(a_2) = e(a_3) \\ d(a_1, 4), d(a_2, 7), d(a_3, 10) \end{array} \right\}$$

For simplicity, also assume that the system represents goals internally as lists, that the selection function always chooses the first element of the current goal, and that goal expansion is done in place. Modulo some unimportant details, a possible derivation of G is then the one given below, where we have omitted the results of $\rightarrow_r$ transitions, which are essentially the same as in CLP, and renamed the variables in the constraint stores for better readability:

$$\left(\left\{ \begin{array}{l} a_1 : t_1, a_2 : t_2, a_3 : t_3 \\ b(a_1) = b(a_3), e(a_2) = e(a_3) \\ d(a_1, 4), d(a_2, 7), d(a_3, 10) \end{array} \right\}, \emptyset, \emptyset \right)$$

$$\downarrow_c$$
$$\vdots$$
$$\downarrow_c$$

[13]Where x, y, z_e, and z_b are variables, $e(x)$ abbreviates *end*(x), and $b(x)$ abbreviates *begin*(x).

$$\left(\left\{\ d(a_1,4), d(a_2,7), d(a_3,10)\ \right\}, \left\{\begin{array}{l} a_1 : t_1, a_2 : t_2, a_3 : t_3 \\ b(a_1) = b(a_3), e(a_2) = e(a_3) \end{array}\right\}, \emptyset\right)$$

$$\downarrow_r$$
$$\vdots$$
$$\downarrow_c$$

$$\left(\left\{\ d(a_3,10)\ \right\}, \left\{\begin{array}{l} a_1 : t_1, a_2 : t_2, a_3 : t_3 \\ b(a_1) = b(a_3), e(a_2) = e(a_3) \\ x_1 = b(a_1), x_2 = b(a_2) \\ y_1 = e(a_1), y_2 = e(a_2) \end{array}\right\}, \left\{\begin{array}{l} y_1 - x_1 = 3 \\ y_2 - x_2 = 6 \end{array}\right\}\right)$$

$$\downarrow_s$$

$$\left(\left\{\ d(a_3,10)\ \right\}, \left\{\begin{array}{l} a_1 : t_1, a_2 : t_2, a_3 : t_3 \\ b(a_1) = b(a_3), e(a_2) = e(a_3) \\ x_1 = b(a_1), x_2 = b(a_2) \\ y_1 = e(a_1), y_2 = e(a_2) \\ y_1 \neq x_1, y_1 \neq x_2 \\ y_2 \neq x_1, y_2 \neq x_2 \end{array}\right\}, \left\{\begin{array}{l} y_1 - x_1 = 3 \\ y_2 - x_2 = 6 \\ y_1 \neq x_1, y_1 \neq x_2 \\ y_2 \neq x_1, y_2 \neq x_2 \end{array}\right\}\right)$$

$$\downarrow_r$$
$$\vdots$$
$$\downarrow_c$$

$$\left(\emptyset, \left\{\begin{array}{l} a_1 : t_1, a_2 : t_2, a_3 : t_3 \\ b(a_1) = b(a_3), e(a_2) = e(a_3) \\ x_1 = b(a_1), x_2 = b(a_2), x_3 = b(a_3) \\ y_1 = e(a_1), y_2 = e(a_2), y_3 = e(a_3) \\ y_1 \neq x_1, y_1 \neq x_2 \\ y_2 \neq x_1, y_2 \neq x_2 \end{array}\right\}, \left\{\begin{array}{l} y_1 - x_1 = 3 \\ y_2 - x_2 = 6 \\ y_3 - x_3 = 9 \\ y_1 \neq x_1, y_1 \neq x_2 \\ y_2 \neq x_1, y_2 \neq x_2 \end{array}\right\}\right)$$

$$\downarrow_s$$

fail

The last transition moves to the *fail* state by generating an arrangement of $x_1, \ldots, x_3, y_1, \ldots, y_3$ that in addition to including the old one,

$$\{y_1 \neq x_1, y_1 \neq x_2, y_2 \neq x_1, y_2 \neq x_2\}$$

also contains the equations $x_1 = x_3$ and $y_2 = y_3$. All the other derivations of G differ in the order and number of $\rightarrow_s$ transitions and in the arrangement they choose for the shared variables. We leave it to the reader to verify that they too are failed, which shows that, contrary to the CLP system we considered in Section 2, the MCLP($\bar{\mathcal{X}}$) system above does succeed in determining the (un)solvability of G.

Looking back at Figure 1, it is obvious that our scheduling problem would be solvable in the union of S_1's and S_2's theories if a_3 lasted 11 days instead of 10, because then a_2's theory could be scheduled to start on day 5. A goal expressing the new problem by replacing the subgoal $d(a_3, 10)$ with $d(a_3, 11)$ in G has, as expected, a successful derivation in the MCLP($\bar{\mathcal{X}}$) system. In fact, modulo the replacement of $d(a_3, 10)$ by $d(a_3, 11)$, one such derivation is identical to the one shown above, except for the final state, which is instead of the (simplified) form

$$\left(\emptyset, \left\{ \begin{array}{l} a_1 : t_1, a_2 : t_2, a_3 : t_3, \\ b(a_3) = b(a_1), e(a_1) \prec b(e_2), e(a_2) = e(a_3), \\ x_1 = b(a_1), x_2 = b(a_2), x_3 = b(a_3), \\ y_1 = e(a_1), y_2 = e(a_2), y_3 = e(a_3) \end{array} \right\}, \left\{ \begin{array}{l} x_1 = x_3 \\ y_1 = x_3 + 3 \\ x_2 = x_3 + 4 \\ y_2 = x_3 + 9 \\ y_3 = x_3 + 9 \end{array} \right\} \right)$$

5.3 On the Combined Constraint Language

As mentioned, the constraint language for MCLP($\bar{\mathcal{X}}$) is *sCNF*(Σ). Although we will not show it here, it is possible to modify our framework slightly to allow a constraint language at least as large as the class of quantifier-free Σ-formulas. Sometimes, however, we may need to actually *restrict* the constraint language to a subclass of *sCNF*(Σ). This will happen any time the constraint solver at our disposal for a certain component theory $\mathcal{T}_i$ is able to decide satisfiability in $\mathcal{T}_i$ only for a proper subset of *sCNF*(Σ_i). Many constraint solvers, for instance, do not accept negative literals, because of efficiency or decidability concerns. In such cases, *sCNF*(Σ) is too powerful a combined language, as it includes the whole *sCNF*(Σ_i).[14] The question then is: how much can we restrict the combined constraint language of MCLP($\bar{\mathcal{X}}$) without compromising its operations? Looking at the definitions of $\rightarrow_c$ and $\rightarrow_s$ transitions, it should be easy to see that, in general, any subset of *sCNF*(Σ) can be chosen as a combined constraint language as long as

[14]This is because $\Sigma_i \subseteq \Sigma$ for all i.

this subset includes equations and disequations of variables—and continues to satisfy the closure conditions mentioned in Section 2. In constraint-solver terms, this means that a CLP solver can be *plugged into* an MCLP($\bar{\mathcal{X}}$) system only if it also accepts disequations of variables. Alternatively, the solver must be able to tell for any two variables x and y whether or not its store entails $x = y$ in the solver's constraint theory. Notice that the two alternatives are basically the same, because a constraint of the form $c \wedge x \neq y$ is satisfiable in a theory $\mathcal{T}$ if and only if $\mathcal{T} \not\models c \rightarrow x = y$.

5.4 Implementation Issues

Like the transitions of type $\rightarrow_r$, transitions of type $\rightarrow_s$ are nondeterministic. With $\rightarrow_r$ transitions, the choice is among the possible reductions of the selected subgoal; with $\rightarrow_s$ transitions, it is among the possible arrangements of the shared variables. This means that in actual implementations of the MCLP($\bar{\mathcal{X}}$) scheme, backtracking mechanisms similar to those used for $\rightarrow_r$ transitions must be used.

It should be noted, however, that the kind of *don't know* nondeterminism introduced in $\rightarrow_s$ transitions poses greater computational complexity concerns than in the original CLP scheme. In fact, the number of possible arrangements of a set V of variables, and hence the number of choice points in correspondence of a $\rightarrow_s$ transition, grows extremely quickly in the cardinality of V. To contain the number of generated arrangements, several optimization techniques can be used when implementing $\rightarrow_s$ transitions in an actual MCLP($\bar{\mathcal{X}}$) system.

The most obvious optimization is suggested by the observation that, since the constraint stores are built incrementally, it is not necessary to recompute a totally new variable arrangement for each $\rightarrow_s$ transition in a derivation. To exemplify, consider a given MCLP($\bar{\mathcal{X}}$) implementation which, again just for simplicity, has only two solvers, say S_1 and S_2. Assume that the main module of the system, the engine, is provided with data structures of its own containing, for $i = 1, 2$, the set V_i of variables currently occurring in the store of S_i and a partition P of $V_1 \cap V_2$, which corresponds to the current arrangement of shared variables in the stores. After a new constraint is added to one of the stores because of a $\rightarrow_c$ transition, the set $V_1 \cap V_2$ may increase, and so a new arrangement will eventually have to be generated and passed to the solvers. It is clear, however, that the only meaningful new arrangements will be those that *extend* the current one to include the new shared variables.

Selecting only among these arrangements drastically reduces the number of choice points for each $\rightarrow_s$ transition. In performing a $\rightarrow_s$ transition, all the engine has to do to extend the current arrangement is guess in which block of the partition P to insert each new shared variable. For example, if $P = \{\{x_1, x_4\}, \{x_2\}, \{x_3, x_5\}\}$ and x_6 is a new shared variable, the engine guesses whether to insert x_6 in x_1's or x_2's or x_3's block, or in none of them and add the singleton block $\{x_6\}$ to P. After that, it updates the arrangement in the solvers' stores just by adding the minimal set of (dis)equalities that completely identifies the partition block of the newly inserted variable. In this example, the set will be $\{x_6 \neq x_1, x_6 \neq x_2, x_6 \neq x_2\}$ if the engine inserts x_6 in its own block, $\{x_6 = x_1, x_6 \neq x_2, x_6 \neq x_2\}$ if the engine inserts x_6 in x_1's block, and so on.

Another obvious optimization comes from the consideration that after a $\rightarrow_c$ transition, one of the constraint stores can already become locally unsatisfiable. In that case, testing for *global* consistency through variable equality propagation is pointless, because the stores are already globally inconsistent. When performing a $\rightarrow_s$ transition then, the engine can first check the consistency of each store that has been modified by a $\rightarrow_c$ transition since the last check. If the stores are fine, it can then go ahead and guess a new variable arrangement as seen above; otherwise, it can fail immediately and backtrack instead of hopelessly generating all the possible variable arrangements at that point.

Yet another optimization can be achieved by modifying the current variable partition P according to more informed guesses. For instance, each time a $\rightarrow_c$ transition adds an equality constraint in the variables x, y to (necessarily both of) the stores, the engine can insert them into P taking their equality relation into account. If the constraint is $x = y$ and x is already in P, y must be inserted in x's block. If neither x or y is in P, they can be inserted in any block, but this block must be the same for both. The situation is analogous when the constraint is $x \neq y$. More informed guesses on how to insert new shared variables in P can be made, depending on the particular constraint theories and solvers in the system. Sometimes, asking a solver what equalities between a given number of variables are entailed by its constraint store is a rather inexpensive operation.[15] In that case, before

[15]This typically happens when equality constraints are kept in the store in a solved form such as $x = t$, where x is a variable and t is a term, and the data structures representing these constraints share common subexpressions.

inserting a variable x in the partition, the engine can ask one or more solvers whether x must be equal to any of the variables already there. If so, the engine will deterministically insert x in P where appropriate; otherwise, it will need to make a guess, as already discussed.

Further improvements can be obtained by providing the engine itself with some meta-knowledge on the various constraint theories. Using this knowledge, the engine can perform simple, inexpensive tests on the constraints chosen during $\rightarrow_c$ transitions, and acquire more information on their variables. For instance, if the engine successfully adds a constraint of the form $x = t$ to a solver for the theory of finite trees, it can conclude immediately that x cannot be equated to any variable occurring in t.[16] Similar conclusions can be drawn for other simple constraints, such as $x < y$, where $<$ denotes a strict order in the given constraint theory; $x = f(\tilde{y})$, where f denotes a projection function; $g(\tilde{x}) = g(\tilde{y})$, where g denotes an injective function; and so on. It is clear, however, that the frequency of such constraints in the application domain in question will mostly determine the real computational impact and usefulness of these tests. An approach of this sort is followed in [Ric96] to improve the performance of an algorithm implementing the Baader-Schulz method [BS92] for combining unification procedures.

6 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 properties of MCLP($\bar{\mathcal{X}}$).

Definition 4 *Let $\rightarrow_{cs}$ denote the two-transition sequence $\rightarrow_c\rightarrow_s$. We say that an 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 modifying it. Analogously, a quick-checking

[16]Because this would violate the *occurs check*.

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 5 *An MCLP($\bar{\mathcal{X}}$) system is* ideal *if it is quick checking and uses a fair computation rule.*[17]

In both CLP($\mathcal{X}$) and MCLP($\bar{\mathcal{X}}$), 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.

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

To show that the main soundness and completeness properties of the CLP scheme 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.

Actually, MCLP($\bar{\mathcal{X}}$) systems cannot have a *corresponding* CLP($\mathcal{X}$) system, since the original scheme defines constraint satisfiability in a different manner from our scheme. In CLP($\mathcal{X}$), the satisfiability test for the constraint store is successful if the store is satisfiable in the *fixed* structure $\mathcal{D}$. Instead, in MCLP($\bar{\mathcal{X}}$) 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, so to speak, the CLP($\mathcal{X}$) system by testing satisfiability within the class of structures $\mathit{Mod}(\mathcal{T})$, where $\mathcal{T}$ is the chosen constraint theory, instead of the single structure $\mathcal{D}$.

The impact of going from the CLP scheme, which is based on a distinguished structure $\mathcal{D}$, to a *relaxed* CLP scheme, which is based on a distinguished class $\mathcal{K}$ of structures, is perhaps best understood with the following observation. In the relaxed case, we can associate with each state of a derivation the set of all the structures of $\mathcal{K}$ that satisfy the constraint store of that state. The set associated with the initial state is obviously the whole $\mathcal{K}$,

[17]This definition differs from that given in [JM94], because we adopt a slightly different definition of "derivation," but it refers to the same class of systems.

since the constraint store is empty and therefore satisfiable in any structure. As new constraints are added to the store, the set is restricted to only those structures that also satisfy the new constraints. That is to say, for each non-initial state i of the derivation, the set $\mathcal{K}_i$ of structures associated to i is a subset of the set $\mathcal{K}_{i-1}$ associated to the previous state. The structures left at the end, if any, are all structures in which the initial goal is satisfiable, as we will see. In the original case, that of the CLP scheme, the situation is analogous, but each $\mathcal{K}_i$ contains at most one element: the distinguished structure $\mathcal{D}$. An immediate but important consequence of the above argument is that every successful derivation in a relaxed CLP system is also a successful $\mathcal{M}$-derivation, in the sense given in Section 4.1.2, for any $\mathcal{M} \in \mathit{Mod}(\mathcal{T})$ satisfying the answer constraint.

Now, it so happens that the main CLP soundness and completeness results hold, in analogous form, even when we move to relaxed CLP. As far as we know, this point seems to have been overlooked in the CLP literature, perhaps because of the initial interest in a single interpretation for the constraints, which paralleled the exclusive interest of logic programming in the Herbrand interpretation of unification constraints and, more importantly, permitted the development of a solid algebraic semantics for the CLP scheme.

With [HS88], Höhfeld and Smolka were probably the first to recognize that the focus on a single structure was too restrictive for constraint logic programming, and unnecessary. That led them to the development of a more general scheme, which included the original scheme as a special case. With the increase in generality, however, they obtained somewhat weaker results. Generality-wise, the relaxed CLP scheme is between the two schemes, and in fact its soundness is derivable as a consequence of the soundness of Höhfeld and Smolka's scheme. Its completeness, however, cannot be derived from that 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 the following, we prove that, for axiomatizable classes of structures, not only is the relaxed CLP scheme correct and complete, but also and more importantly, it exhibits logical properties no weaker than those of the CLP scheme. From that, similar soundness and completeness results for MCLP($\bar{\mathcal{X}}$) will easily follow, as we show in Section 6.3.

6.2 Relaxed CLP($\mathcal{X}$)

Some notation and definitions are necessary before we go further. If $\mathcal{M}$ is a structure for a constraint language $\mathcal{L}$, we will implicitly expand $\mathcal{L}$, and hence $\mathcal{M}$, to include a constant for each individual of $\mathcal{M}$'s universe, which we will call the *name* of the individual. By a common abuse of notation then, when a is either an atom or a primitive constraint and ϑ is an $\mathcal{M}$-valuation of a's variables, we will use $a\vartheta$ to univocally denote the ground predicate obtained by applying to a the substitution that assigns to each variable x of a the name of $x\vartheta$.[18]

In the following, $\mathcal{L}$ will be a constraint language, $\mathcal{T}$ the associated constraint theory, $\mathcal{M}$ a model of $\mathcal{T}$, and P a CLP program with constraint from $\mathcal{L}$.

Definition 6 ($\mathcal{M}$-Solution) *Let $c \in \mathcal{L}$. We call the $\mathcal{M}$-solution of c any $\mathcal{M}$-valuation ϑ such that $\mathcal{M} \models c\vartheta$. We will denote the set of $\mathcal{M}$-solutions of c with $Sol_{\mathcal{M}}(c)$.*

Definition 7 ($\mathcal{M}$-Interpretations, $\mathcal{M}$-Models) *An* $\mathcal{M}$-interpretation *of P is a structure that expands $\mathcal{M}$ to include an interpretation of the set* Π *of predicate symbols occurring in P. An $\mathcal{M}$-interpretation I of P is an $\mathcal{M}$-*model *of P (P^*) if every sentence of P (P^*) is true in I. The set*

$$\mathcal{B}^P_{\mathcal{M}} \;:=\; \{p(\tilde{x})\vartheta \mid p \in \Pi,\ \vartheta\ \mathcal{M}\text{-valuation of } \tilde{x}\}$$

is called the $\mathcal{M}$-base *of P.*

Where Q is either P or P^*, we will denote by $(Q, \mathcal{M})$ the set of all the $\mathcal{M}$-models of Q. Observe that $\bigcup_{\mathcal{M} \in \mathcal{T}} (Q, \mathcal{M}) = Mod(Q \cup \mathcal{T})$.

It is easy to see that there is a bijection h from $\mathcal{M}$-interpretations onto the power set of the $\mathcal{M}$-base of P where $h(I) := \{a\vartheta \in \mathcal{B}^P_{\mathcal{M}} \mid I \models a\vartheta\}$. As is customary in the field, we will often identify $\mathcal{M}$-interpretations with their images under h. Under such identification and in analogy with the set of all Herbrand models of a logic program, $(Q, \mathcal{M})$ is a poset with respect to set inclusion, and has a minimal element, the *least $\mathcal{M}$-model of Q*, which we will denote by $lm(Q, \mathcal{M})$. It is possible to show (see [JM94]) that

$$lm(P, \mathcal{M}) = lm(P^*, \mathcal{M}) = \{a\vartheta \mid P^*, \mathcal{M} \models c \to a, \mathcal{M} \models c\vartheta\}$$

[18] In other words, given a structure, we will identify valuations with substitution from variables into names.

Finally, observe that if A is a finite set of atoms, an $\mathcal{M}$-interpretation I is a model of $\tilde{\exists}\, A$ if and only if there is an $\mathcal{M}$-valuation ϑ such that $A\vartheta \subset I$.

6.2.1 Soundness and Completeness

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

We have formulated the following results after those given in [JM94] for the CLP scheme.

Proposition 3 (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 with respect to $\mathcal{L}$, if G has a finite computation tree with answer constraints $c_1, \ldots, c_n$, then $P^*, \mathcal{T} \models G \leftrightarrow c_1 \vee \ldots \vee c_n$.*

Proof of Proposition 3

1. The proof is essentially identical to that for the original CLP scheme. First, we show that in general, if $\langle A, C\rangle$ is a state in a derivation and $\langle A', C'\rangle$ is its successor state, then

$$P, \mathcal{T} \models A' \cup C' \rightarrow A \cup C$$

When $\langle A', C'\rangle$ is generated by a $\rightarrow_c$ or $\rightarrow_s$ transition, the claim is trivial because $A' \cup C' = A \cup C$ (see Section 4.1.2). When $\langle A', C'\rangle$ is generated by an $\rightarrow_r$ transition, it has the form $\langle A \cup B - p(\tilde{x}), C \cup \tilde{x} = \tilde{y}\rangle$, where $p(\tilde{x})$ is an atom of A, and $p(\tilde{y}) \leftarrow B$ is a renaming of a rule of P. By elementary logical reasoning, one can see that if $A \cup B - p(\tilde{x}) \cup C \cup \tilde{x} = \tilde{y}$ is satisfied in $P \cup \mathcal{T}$ by some valuation ϑ, then $A \cup B \cup C \cup \tilde{x} = \tilde{y}$ is also satisfied by ϑ. In particular, $A \cup C$ is also satisfied by ϑ, from which the claim follows immediately.

Now, let $\langle G_0, C_0\rangle, \langle G_1, C_1\rangle, \ldots \langle G_n, C_n\rangle$ be the sequence of all the states in a successful derivation of G with answer constraint c. We then know that $\langle G_0, C_0\rangle = \langle G, \emptyset\rangle$, $\langle G_n, C_n\rangle = \langle \emptyset, C_n\rangle$, and $c = \exists_{-var(G)}\, C_n$. By a repeated application of the argument above, we can conclude that $P, \mathcal{T} \models G_n \cup C_n \rightarrow G_0 \cup C_0$; that is, $P, \mathcal{T} \models C_n \rightarrow G$, which then entails that $P, \mathcal{T} \models \exists_{-var(G)}\, C_n \rightarrow G$.

2. Immediately follows by the corresponding CLP result in [JM94] since, if $\mathcal{T}$ is satisfaction complete with respect to $\mathcal{L}$, a constraint is satisfiable in $\mathcal{T}$ if and only if it is satisfiable in every model of $\mathcal{T}$; and so the given system behaves exactly as a CLP system in the original sense.[19]

Proof of Proposition 3 □

Lemma 1 *Given a program P and a simple goal G, let $\mathcal{M} \in Mod(\mathcal{T})$ and ϑ be an $\mathcal{M}$-valuation of $var(G)$. If $G\vartheta$ is true in every $\mathcal{M}$-model of P, then G has a successful $\mathcal{M}$-derivation with an answer constraint d such that $\mathcal{M} \models d\vartheta$.*

Proof of Lemma 1 If $G\vartheta$ is true in every $\mathcal{M}$-model of P, then $G\vartheta \in lm(P, \mathcal{M})$. It follows from an earlier observation that there exists a constraint $c \in \mathcal{L}$ such that $\mathcal{M} \models c\vartheta$ and $(P^*, \mathcal{M}) \models c \rightarrow a$. By Theorem 6.1(1) of [JM94], then, G has a successful $\mathcal{M}$-derivation with answer constraint d such that $\mathcal{M} \models c \leftrightarrow d$, from which the claim follows immediately by logical reasoning.

Proof of Lemma 1 □

Given $G, \mathcal{M}, \vartheta$ such that $G\vartheta$ is true in every $\mathcal{M}$-model of P, we will denote by $Ans(G, \mathcal{M}, \vartheta)$ the set of all the answer constraints d that satisfy the lemma above.

Theorem 2 (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 $\mathcal{T} \models c \rightarrow c_1 \vee \ldots \vee c_n$;*

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

[19]Incidentally, observe that any model of $\mathcal{T}$ can be seen as the intended constraint domain $\mathcal{D}$, as they all correspond to $\mathcal{T}$ on $\mathcal{L}$.

Proof of Theorem 2 What follows is essentially Maher's argument (see [Mah87]) with minor variations. We reproduce it here to stress the fact that satisfiability in a set of structures suffices for this result.

1. Let c be a constraint satisfiable in $\mathcal{T}$ such that $P, \mathcal{T} \models c \to G$. Let $\tilde{v} := var(G)$, and assume for now that $var(c) \subseteq \tilde{v}$. Then let $N_G := \{\neg d \mid d \in C_G\}$, where C_G is the union of all the $Ans(G, \mathcal{M}, \vartheta)$ such that $\mathcal{M}$ is model of $\mathcal{T}$ and ϑ is an $\mathcal{M}$-valuation of G's variables that makes $G\vartheta$ true in every $\mathcal{M}$-model of P. Observe that $var(N_G \cup \{c\})$ is exactly $\tilde{v}$, because every $d \in C_G$ is an answer constraint for G and $var(c) \subseteq \tilde{v}$ by assumption. We prove by contradiction that $N_G \cup \{c\}$ is not satisfiable in $\mathcal{T}$; that is, for no $\mathcal{M} \in Mod(\mathcal{T})$ is there a valuation of $\tilde{v}$ that is an $\mathcal{M}$-solution for each element of the set.

Suppose there is an $\mathcal{M} \in Mod(\mathcal{T})$ and an $\mathcal{M}$-valuation ϑ of $\tilde{v}$ that satisfy $N_G \cup \{c\}$. Then, $c\vartheta$ is true in $\mathcal{M}$, and hence, since $P, \mathcal{T} \models c \to G$, $G\vartheta$ is true in every $\mathcal{M}$-model of P. But then, by Lemma 1, there is a constraint $d \in C_G$ such that $d\vartheta$ is true, against the fact that ϑ must falsify every element of C_G as, by assumption, it satisfies their negations.

Then, by the compactness of first-order logic and the assumption that c is satisfiable in $\mathcal{T}$, we can conclude that there is a finite, nonempty set $\{\neg c_1, \ldots, \neg c_n\} \subseteq N_G$ of negated answer constraints such that $\{c, \neg c_1, \ldots, \neg c_n\}$ is unsatisfiable in $\mathcal{T}$. It follows that the formula $c \wedge \neg c1 \wedge \ldots \wedge \neg c_n$ is unsatisfiable in $\mathcal{T}$ or, equivalently, that $\mathcal{T} \models c \to c_1 \vee \ldots \vee c_n$.

To complete the proof, we must consider the case in which $var(c) \not\subseteq \tilde{v}$. Now, since $P, \mathcal{T} \models \exists_{-\tilde{v}}\, c \to G$ as well, we know by the above that G has $n > 0$ answer constraints such that $\mathcal{T} \models \exists_{-\tilde{v}}\, c \to c_1 \vee \ldots \vee c_n$. That $\mathcal{T} \models c \to c_1 \vee \ldots \vee c_n$ then follows from the fact that $var(c_1 \vee \ldots \vee c_n) \subseteq \tilde{v}$.

2. By the same argument at point 2 of Proposition 3.

Proof of Theorem 2 □

6.2.2 Negation as Failure

We have seen some of the nice properties of the CLP scheme that lift to its relaxed version. A perhaps more surprising lifting, however, concerns the properties of the negation-as-failure rule of computation.

Negation as failure is widely used in logic programming, and is a provably correct and complete inference rule. Jaffar and Lassez have showed that it can also be used in their scheme, provided that the constraint theory is strong enough: it must be satisfaction complete with respect to the constraint language. Despite the fact that many typical constraint theories are indeed satisfaction complete, this requirement is a potential problem for our MCLP($\bar{\mathcal{X}}$) systems, because the union of two satisfaction-complete theories is not necessarily satisfaction complete. Here is a simple counterexample.

Example 1 Consider the first-order theory $\mathcal{T}_1$ whose signature Σ_1 is composed of a countably infinite set $\{a_1, a_2, \dots\}$ of constant symbols and a countably infinite set $\{f_1, f_2, \dots\}$ of unary function symbols. $\mathcal{T}_1$ is axiomatized by the set consisting of all the following (implicitly universally quantified) formulas:

(1.1)	for all $i < \omega$,	$f_i(x) = f_i(y) \to x = y$
(1.2)	for all $i < j < \omega$,	$f_i(x) \neq f_j(y)$
(1.3)	for all nonvariable Σ_1-terms t,	$x \neq t$

where x, and y are variables. Then consider the first-order theory $\mathcal{T}_2$ whose signature Σ_2 is composed of a countably infinite set $\{b_1, b_2, \dots\}$ of constant symbols and a countably infinite set $\{g_1, g_2, \dots\}$ of unary function symbols. $\mathcal{T}_2$ is axiomatized by the set consisting of all the following (implicitly universally quantified) formulas:

(2.1)	for all $i < \omega$,	$g_i(x) = g_i(y) \to x = y$
(2.2)	for all $i < j < \omega$,	$g_i(x) \neq g_j(y)$
(2.3)	for all nonvariable Σ_2-terms t,	$x \neq t$

where x, and y are variables. Then assume that $\Sigma_1 \cap \Sigma_2 = \emptyset$.

Essentially, $\mathcal{T}_1$ and $\mathcal{T}_2$ are two signature-disjoint versions of the theory of finite unary trees. Each $\mathcal{T}_i$ is satisfaction complete with respect to the class of equational problems over Σ_i-terms (see [Mah88]). Now, let $\mathcal{T} := \mathcal{T}_1 \cup \mathcal{T}_2$, which is provably consistent, and $\mathcal{L}$ be the class of equational problems over $(\Sigma_1 \cup \Sigma_2)$-terms. Then, consider a formula of the form:

$$f_i(x) = g_j(x) \tag{3}$$

for some $i, j < \omega$. The existential closure of Equation 3 is not entailed by $\mathcal{T}$, nor is its negation: it is straightforward to construct models of $\mathcal{T}$ in which Equation 3 is satisfiable, or even valid, and others where Equation 3 is unsatisfiable.[20] This entails that $\mathcal{T}$ is not satisfaction complete with respect to $\mathcal{L}$.[21]

Given the nonmodularity of satisfaction completeness with respect to theory union, it is not clear whether negation as failure can be used correctly in the MCLP($\bar{\mathcal{X}}$) scheme, which works exactly with union of theories. Höhfeld and Smolka's work does not help us here, since they decide to ignore the whole issue after concluding, maybe a little too hastily, that negation by failure is unnecessary in their framework because negation is supported directly by the constraint language.[22]

What we discovered, however, is that not only can we still use negation as failure properly in MCLP($\bar{\mathcal{X}}$), but we do not need satisfaction completeness of the component theories at all. As before, a sufficient condition for our results is that we use a first-order language for the constraints. In light of this, it seems that such results can be easily extended to all the instances of the Höhfeld-Smolka framework that use a first-order constraint language. A proof of that, however, is out of the scope of this work.

To prove our claims, we will first show that negation-as-failure is sound and complete for the relaxed CLP scheme. We will start with a lemma that characterizes the models of complete programs in terms of the standard one-step consequence function $T_P^{\mathcal{M}}$ (see [JM94]), whose definition is given below.

Definition 8 *Given a structure $\mathcal{M} \in Mod(\mathcal{T})$, and a program P, the function $T_P^{\mathcal{M}}$ mapping from and into the $\mathcal{M}$-base of P is defined as*

$$T_P^{\mathcal{M}}(I) \;\; := \;\; \{p(\tilde{x})\vartheta \mid (p(\tilde{x}) \leftarrow c, A) \in P,\ \vartheta \in Sol_{\mathcal{M}}(c),\ A\vartheta \subseteq I\}$$

It is not difficult to prove that $T_P^{\mathcal{M}}$ is monotonic. The following lemma is often cited without proof in the CLP literature with the implicit claim that it is an easy lifting of a corresponding lemma for logic programming. We provide a proof here for completeness.

[20] We stress that Equation 3 is not consistent with either points (1.2) or (2.2).

[21] Observe that since the theories chosen above are also complete in the standard sense, the very same example shows—when we consider the universal closure of Equation 3—that completeness as well is not modular with respect to the union of theories. But this is not all. The example shows that in general the union of complete theories is not even satisfaction complete—which is a weaker property than being complete.

[22] Apart from the fact that the CLP scheme itself does not exclude this possibility, it is not clear how negation of *atoms* would be dealt with in the Höhfeld-Smolka framework.

Lemma 2 (Fix-Point Lemma [JL86]) *Let P be a program, and I an $\mathcal{M}$-interpretation of P for some $\mathcal{M} \in Mod(\mathcal{T})$. Then, I is a model of P^* if and only if it is a fix-point of $T_P^{\mathcal{M}}(I)$.*

Proof of Lemma 2 ($\Rightarrow$) Assume that $I \in Mod(P)$, and let p be the predicate symbol of an atom occurring in P^*. Then we know that P^* contains the universal closure of exactly one of the following formulas:

$$\tilde{\forall}\,(p(\tilde{x}) \leftrightarrow \exists_{-\tilde{x}}\,(c_1 \wedge A_1) \vee \ldots \vee \exists_{-\tilde{x}}\,(c_n \wedge A_n)) \tag{4}$$

$$\tilde{\forall}\,\neg p(\tilde{x}) \tag{5}$$

where each c_i is a constraint and each A_i is a set of atoms. Assume that Equation 4 is in P^*, and let ϑ be an $\mathcal{M}$-valuation of $\tilde{x}$. Then, $p(\tilde{x})\vartheta \in I$ if and only if there is an $i \in \{1, \ldots, n\}$ such that $I \models \exists_{-\tilde{x}}\,(c_i \wedge A_i)\vartheta$ if and only if there is an extension ϑ' of ϑ, and an $i \in \{1, \ldots, n\}$ such that $\vartheta' \in Sol_{\mathcal{M}}(c_i)$ and $A_i\vartheta' \subseteq I$. Since, by construction of P^*, there is a rule in P of the form $p(\tilde{x}) \leftarrow c_i, A_i$ for every disjunct of Equation 4, we can conclude that for every $\mathcal{M}$-valuation ϑ, $p(\tilde{x})\vartheta \in I$ if and only if $p(\tilde{x})\vartheta \in T_P^{\mathcal{M}}(I)$. Now assume that Equation 5 is in P^*. Then, I is a model of $\neg p(\tilde{x})$ if and only if for no $\mathcal{M}$-valuation ϑ, $p(\tilde{x})\vartheta \in I$. Since by construction of P^* there is no rule or fact headed by $p(\tilde{x})$ in P, it is immediate from the definition of $T_P^{\mathcal{M}}$ that $p(\tilde{x})\vartheta \in T_P^{\mathcal{M}}(I)$ for no $\mathcal{M}$-valuation ϑ. In conclusion, we have shown that for all atoms $p(\tilde{x})$ and valuations ϑ, $p(\tilde{x})\vartheta \in I$ if and only if $p(\tilde{x})\vartheta \in T_P^{\mathcal{M}}(I)$, which entails that $I = T_P^{\mathcal{M}}(I)$.

($\Leftarrow$) Suppose that $I \notin Mod(P^*)$. Then, there is a sentence $\tilde{\forall}\,\varphi \in P^*$ and an $\mathcal{M}$-valuation ϑ such that:

$$I \;\models\; \neg\varphi\vartheta. \tag{6}$$

If φ is an equivalence like Equation 4 above, we can conclude by Equation 6 that I and ϑ satisfy one side of the equivalence and falsify the other. Assume that $I \models \exists_{-\tilde{x}}\,(c_i \wedge A_i)\vartheta$ for some $i \in \{1, \ldots, n\}$. Then, $I \not\models p(\tilde{x})\vartheta$, that is, $p(\tilde{x})\vartheta \notin I$. As before, however, we can show that $p(\tilde{x})\vartheta \in T_P^{\mathcal{M}}(I)$. Now assume that $I \models p(\tilde{x})\vartheta$, and so $p(\tilde{x})\vartheta \in I$. Then $I \models \exists_{-\tilde{x}}\,(c_i \wedge A_i)\vartheta$ for no $i \in \{1, \ldots, n\}$, which entails that $p(\tilde{x})\vartheta \notin T_P^{\mathcal{M}}(I)$. If φ has the form of Equation 5, it is easy to show that, again, $p(\tilde{x})\vartheta \in I$ but $p(\tilde{x})\vartheta \notin T_P^{\mathcal{M}}(I)$. In conclusion, in all cases we obtain that $I \neq T_P^{\mathcal{M}}(I)$.

Proof of Lemma 2 □

The proofs of the propositions below are closely modeled after those given in [JL86]. As in the CLP scheme, we only consider ideal systems here.

Proposition 4 *If the goal G is finitely failed for a program P, then $P^*, \mathcal{T} \models \neg G$.*

Proof of Proposition 4 We prove the claim by contradiction, showing that if $P^*, \mathcal{T} \not\models \neg G$, the computation tree of G contains either a successful derivation or an infinite one. Therefore, assume that $P^*, \mathcal{T} \not\models \neg G$. We first build by induction a family $\{\langle G_i, C_i \rangle \mid i < \omega\}$ of computation states such that, for every $i < \omega$, $G_i \cup C_i$ is finite and satisfiable in $P^* \cup \mathcal{T}$.

For $(i = 0)$, let $\langle G_0, C_0 \rangle := \langle G, \emptyset \rangle$. Clearly, $G_0 \cup C_0$ is satisfiable in $P^* \cup \mathcal{T}$ by the assumption that $P^*, \mathcal{T} \not\models \neg G$.

For $(i > 0)$, if G_i is empty, simply let $\langle G_{i+1}, C_{i+1} \rangle := \langle G_i, C_i \rangle$. Otherwise, apply *select* to G_i. If *select*(G_i) is a constraint c, it is easy to see that, since $G_i \cup C_i$ is satisfiable in $P^* \cup \mathcal{T}$ by assumption, $C_i \cup c$ is satisfiable in $\mathcal{T}$. It follows that applying a $\rightarrow_{cs}$ transition to $\langle G_i, C_i \rangle$ leads to the nonfail state $\langle G_i - c, C_i \cup c \rangle$. Then, let $\langle G_{i+1}, C_{i+1} \rangle$ be such a state. In both cases above, then the set $G_{i+1} \cup C_{i+1}$ is finite and satisfiable in $P^* \cup \mathcal{T}$ as it coincides with $G_i \cup C_i$. If *select*(G_i) is a predicate $p(\tilde{x})$, we know that P^* must contain a sentence of the form:

$$\tilde{\forall}\, (p(\tilde{y}) \quad \leftrightarrow \quad \exists_{-\tilde{y}}\, B_1 \vee \ldots \vee \exists_{-\tilde{y}}\, B_m) \tag{7}$$

otherwise, it would contain a sentence of the form $\tilde{\forall} \neg p(\tilde{y})$, which is impossible because $p(\tilde{x})$ is satisfiable in $P^* \cup \mathcal{T}$ for being a subgoal of G. Let $G_i = p(\tilde{x}) \cup G_i'$, and choose an $I \in Mod(P^* \cup \mathcal{T})$ in which $G_i \cup C_i$ is satisfiable. Assuming with no loss of generality that no variable in Equation 7 is also a variable of G_i, we can conclude by logical reasoning that there is an $i \in \{1, \ldots, m\}$ such that

$$B_i \cup (\tilde{x} = \tilde{y}) \cup G_i' \cup C_i$$

is satisfiable in I. By construction of P^*, there is a rule of the form $p(\tilde{y}) \leftarrow B_i$ in P; therefore, define $\langle G_{i+1}, C_{i+1} \rangle$ to be the result of applying to $\langle G_i, C_i \rangle$ the $\rightarrow_r$ transition that picks that rule to expand $p(\tilde{x})$. It is easy to see then that $\langle G_{i+1}, C_{i+1} \rangle$ is finite and satisfiable in $P^* \cup \mathcal{T}$.

Now, let $\delta := \{\langle G_i, C_i \rangle \mid i < n + 1\}$ if there is a smallest n such that $G_n = \emptyset$, and let $\delta := \{\langle G_i, C_i \rangle \mid i < \omega\}$ otherwise. In each case, δ describes a possible derivation of G in the system. In the first case, the derivation is finite and clearly successful; in the second case, the derivation is infinite.

Proof of Proposition 4 $\square$

Proposition 5 *If $P^*, \mathcal{T} \models \neg G$ for some goal G and program P, then G is finitely failed.*

Proof of Proposition 5 Assume G is not finitely failed. We prove that G is satisfiable in $P^* \cup \mathcal{T}$, against the hypothesis.

If G has a successful derivation, we know from Proposition 3 that $P^*, \mathcal{T} \models c \rightarrow G$, where c is the answer constraint of the derivation. By definition, c is satisfiable in $\mathcal{T}$ and so G is satisfiable in $P^* \cup \mathcal{T}$.

If G only has infinite derivations, we choose any of them and indicate with $\langle G_i, C_i \rangle$ its i^{th} state, where $\langle G_0, C_0 \rangle := \langle G, \emptyset \rangle$ is the initial state. In addition, we denote by c_i the union of all the constraints contained in G_i. Given that the system is quick checking, it is easy to see that C_i must be satisfiable in $\mathcal{T}$ for all $i < \omega$; otherwise, the derivation would be finite. Now let

$$C := \bigcup_{i<\omega} c_i \cup C_i$$

and consider any finite subset C' of C. Since every derivation in the system is fair by assumption, we know that each constraint of each G_i in the derivation chosen gets added to the constraint store eventually. This means that C' is a subset of C_n for some $n < \omega$, and so is satisfiable in $\mathcal{T}$. By the compactness of first-order logic, then, we can conclude that C itself is satisfiable in $\mathcal{T}$. Therefore, let $\mathcal{M}$ be a model of $\mathcal{T}$ that satisfies C, and consider the set:

$$I \;:=\; \{p(\tilde{x})\vartheta \mid p(\tilde{x}) \text{ atom of } G_i \text{ for } i < \omega, \vartheta \in \mathit{Sol}_{\mathcal{M}}(C)\} \qquad (8)$$

Observe that I is well defined as $\tilde{x} \subseteq \mathit{var}(C)$ for every $i < \omega$ and atom $p(\tilde{x})$ of G_i. In fact, because the derivation is both fair and not failed, every such $p(\tilde{x})$ will be certainly selected by an $\rightarrow_r$ transition at some state $n \geq i$ and replaced by the body of a rule of the form $p(\tilde{y}) \leftarrow B$.[23] At the same time, the constraint $\tilde{x} = \tilde{y}$ will be added to store. But this means that C_{n+1} will include $\tilde{x}$ among its variables, and so will C.

[23]Such a rule exists because otherwise the derivation would fail.

The set I, seen as an $\mathcal{M}$-interpretation, is a model of $\tilde{\exists} G$, because $G_0 = G$, $C_0 \subseteq C$, and $a\vartheta \in I$ for every atom a of G_0 and $\mathcal{M}$-solution ϑ of C. We prove below that $I \subseteq T_P^{\mathcal{M}}(I)$, which implies by the Knaster-Tarski fixed-point lemma (see [Tar55]) that there is an $\mathcal{M}$-interpretation J such that $I \subseteq J$ and $J = T_P^{\mathcal{M}}(J)$. Now, J is an $\mathcal{M}$-model of $\tilde{\exists} G$, because it includes I, and of P^*, because of Lemma 2. This means that G is satisfiable in $P^* \cup \mathcal{T}$, against the hypothesis.

To see that $I \subseteq T_P^{\mathcal{M}}(I)$, consider any $a \in I$. By definition of I, there are a $\vartheta \in Sol_{\mathcal{M}}(C)$, an $i < \omega$, and a $p(\tilde{x}) \in G_i$ such that $a = p(\tilde{x})\vartheta$. If $\langle c_n \cup p(\tilde{x}) \cup A_n,\ C_n \rangle$ is the state at which $p(\tilde{x})$ is selected for expansion, we know that:

$$\langle G_{n+1}, C_{n+1} \rangle \quad = \quad \langle C_n \cup c \cup B \cup A_n,\ C_n \cup \tilde{x} = \tilde{y} \rangle$$

where $(p(\tilde{y}) \leftarrow c, B)$ is the renaming of some rule in P. Since $c \in c_{n+1}$, $\tilde{x} = \tilde{y} \subseteq C_{n+1}$, and $c_{n+1} \cup C_{n+1} \subseteq C$, we can immediately conclude that ϑ is also an $\mathcal{M}$-solution of c and $a = p(\tilde{x})\vartheta = p(\tilde{y})\vartheta$. Moreover, $b\vartheta \in I$ for each atom $b \in B$, because $b \in G_{n+1}$ and $\vartheta \in Sol_{\mathcal{M}}(C)$. It follows that there exists a rule $(p(\tilde{y}) \leftarrow c, B) \in P$ and a $\vartheta \in Sol_{\mathcal{M}}(c)$ such that $B\vartheta \subseteq I$. By definition of $T_P^{\mathcal{M}}$, then, $p(\tilde{x})\vartheta \in T_P^{\mathcal{M}}(I)$; that is, $a \in T_P^{\mathcal{M}}(I)$.

Proof of Proposition 5 □

In conclusion, we obtain the following soundness and completeness result for the negation-as-failure rule in the relaxed CLP scheme.

Corollary 1 *In an ideal system, a goal G is finitely failed for a program P if and only if $P^*, \mathcal{T} \models \neg G$.*

6.3 Main Results

We are now ready to prove the main computational properties of MCLP($\bar{\mathcal{X}}$) using the results given above for the relaxed CLP scheme. We will consider an MCLP($\bar{\mathcal{X}}$) system with $\bar{\mathcal{X}} := \langle\langle\Sigma_1, \mathcal{T}_1\rangle, \ldots, \langle\Sigma_n, \mathcal{T}_n\rangle\rangle$, as in Section 5. For simplicity, we will assume that $n = 2$ and that, while the system satisfies the general implementation requirements given earlier, its computation rule is flexible enough with respect to the order of application of the various transitions. Such assumptions are not necessary for our results, but make their proofs easier and more intuitive.

First, we prove that MCLP($\bar{\mathcal{X}}$) is sound. In the following, $\rightarrow_{r/c}$ will denote either an $\rightarrow_r$ or a $\rightarrow_c$ transition.

Lemma 3 *If goal G has a successful derivation in an 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.*

A similar version of this easily proven lemma actually holds for every CLP scheme we have considered so far. Essentially, the lemma states that a successful derivation can be always rearranged into a derivation of the form $\langle G, \emptyset, \emptyset\rangle \overset{*}{\rightarrow}_{r/c} \langle \emptyset, C_1, C_2\rangle \rightarrow_s \langle \emptyset, C_1', C_2'\rangle$ by first reducing the goal to the empty set and then testing the consistency of the collected constraints. The lemma 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 $\mathcal{T}_1 \cup \mathcal{T}_2$. In fact, according to the MCLP($\bar{\mathcal{X}}$) operational 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 Theorem 1 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 4 *Let P be an MCLP($\bar{\mathcal{X}}$) program, and G be a goal. Then, for all paths $\langle G, \emptyset, \emptyset\rangle \overset{*}{\rightarrow}_{r/c} \langle \emptyset, C_1, C_2\rangle$ in the derivation tree of G,*

$$P, \mathcal{T} \models \exists_{-var(G)}\,(C_1 \wedge C_2) \rightarrow G$$

Proof of Lemma 4 Assume that P is in separate form. In a relaxed CLP system supporting $\bar{\mathcal{X}}$ directly with a single solver and having the same computation rule, we would have the mirror derivation $\langle G, \emptyset, \emptyset\rangle \overset{*}{\rightarrow}_{r/c} \langle \emptyset, C\rangle$ with C equal to $C_1 \cup C_2$. The claim follows then immediately by the soundness of the relaxed CLP scheme.

Proof of Lemma 4 □

Proposition 6 (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 $P, \mathcal{T} \models c \rightarrow G$; and*

2. *when $\mathcal{T}$ is satisfaction complete with respect to sCNF$(\Sigma_1 \cup \Sigma_2)$, if G has a finite computation tree with answer constraints $c_1, \ldots, c_n$, then $P^*, \mathcal{T} \models G \leftrightarrow c_1 \vee \ldots \vee c_n$.*

Proof of Proposition 6 Let $\tilde{x} := var(G)$.

1. By Lemma 3, we can assume that the derivation of G is of the form $\langle G, \emptyset, \emptyset\rangle \xrightarrow{*}_{r/c} \langle \emptyset, C_1, C_2\rangle \rightarrow_s \langle \emptyset, C_1', C_2'\rangle$, 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 4.

2. 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 $\langle G, \emptyset, \emptyset\rangle \xrightarrow{*}_{r/c} \langle \emptyset, C_{1i}, C_{2i}\rangle \rightarrow_s \langle \emptyset, C_{1i}', C_{2i}'\rangle$. 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 $\langle G, \emptyset, \emptyset\rangle \xrightarrow{*}_{r/c} \langle \emptyset, D_i\rangle \rightarrow_s \langle \emptyset, D_i\rangle$, where $D_i = C_{1i} \cup C_{2i}$.

Let $\sim$ be an equivalence relation over $\{1, \ldots, n\}$ such that $i \sim j$ if and only if 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:

$$\bigvee_{i \in [j]} c_i \;\leftrightarrow\; \bigvee_{i \in [j]} \exists_{-\tilde{x}}\,(C_{1i}' \wedge C_{2i}') \leftrightarrow \exists_{-\tilde{x}}\,(C_{1j} \wedge C_{2j}) \leftrightarrow \exists_{-\tilde{x}}\,D_j \qquad (9)$$

By an analog of Lemma 3 for the relaxed CLP scheme, it is possible to 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:

$$P^*, \mathcal{T} \models G \leftrightarrow \bigvee_{j \in \{1, \ldots, n\}} \exists_{-\tilde{x}}\,D_j \qquad (10)$$

The claim follows then immediately, combining Equations 9 and 10 above.

Proof of Proposition 6 □

We now prove that MCLP($\bar{\mathcal{X}}$) is complete.

Lemma 5 *Consider a program P, an 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 $\mathcal{T} \models C' \leftrightarrow C_1' \wedge C_2'$.*

Proof of Lemma 5 If t is a $\rightarrow_c$ transition, t' is the $\rightarrow_c$ transition that chooses from A the same constraint chosen by t and adds it to the appropriate constraint store.[24] If t is an $\rightarrow_r$ transition, t' is the $\rightarrow_r$ transition that chooses from A the same atom and from P the same rule chosen by t and adds the generated unification constraints to both the constraint stores.

Proof of Lemma 5 $\square$

Proposition 7 (Completeness of MCLP($\bar{\mathcal{X}}$)) *Consider an 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 $\mathcal{T} \models c \rightarrow c_1 \vee \ldots \vee c_n$;*

2. *when $\mathcal{T}$ is satisfaction complete with respect to sCNF$(\Sigma_1 \cup \Sigma_2)$, if $P^*, \mathcal{T} \models G \leftrightarrow c_1 \vee \ldots \vee c_n$, then G has a computation tree with answer constraints $c'_1, \ldots, c'_m$ such that $\mathcal{T} \models c_1 \vee \ldots \vee c_n \leftrightarrow c'_1 \vee \ldots \vee c'_m$.*

Proof of Proposition 7
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 $ans(\delta)$.

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 $c_1 \vee \ldots \vee c_n$ 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 an $\rightarrow_{r/c}$ transition for each $\rightarrow_{r/c}$ transition of δ in the way given in Lemma 5, and an empty transition for each $\rightarrow_s$ transition of δ. Using Lemma 5 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 $\tilde{v} := var(C_1) \cap var(C_2)$ that is consistent with both stores. Notice that since $C_1 \cup C_2$ is

[24]Recall that all constraints are pure, as we assume both the program and the goal in separate form.

satisfiable, for being equivalent to the final constraint store of a successful derivation, we are guaranteed by Theorem 1 that at least one arrangement of $\tilde{v}$ is consistent with both C_1 and C_2 and, consequently, that D_δ is nonempty. It follows that for every $\gamma \in D_\delta$ there is an arrangement $ar(\tilde{v})$ such that $\mathcal{T} \models ans(\gamma) \leftrightarrow \exists_{-\tilde{x}}\,(C_1 \wedge C_2 \wedge ar(\tilde{v}))$. 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 equivalences in $\mathcal{T}$:

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

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.

Proof of Proposition 7 □

Notice that considering multiple derivations of an input goal to achieve completeness is already necessary in the CLP scheme itself—and in its relaxed version. Our scheme, however, may increase the number of derivations to consider, because $\rightarrow_s$ transitions can generate multiple successful derivations, instead of just one, whenever more than one arrangement of variables is consistent with both constraint stores.

By essentially the same arguments given for the relaxed CLP scheme, it is also possible to prove the soundness and completeness of negation as failure in MCLP($\bar{\mathcal{X}}$).

Proposition 8 *In an ideal MCLP($\bar{\mathcal{X}}$) system, a goal G is finitely failed for a program P if and only if $P^*, \mathcal{T} \models \neg G$.*

7 Conclusions and Further Developments

In this paper, we 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 for 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 nondeterministic 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 axiomatizable class of structures.

Specifically, we have shown 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. Such a result, which is important in its own right, seems to have been overlooked in the CLP literature so far. Then, we have shown how the properties of the relaxed CLP scheme lift to our extension.

We would like to point out the advantages of adopting a nondeterministic version of the original equality-sharing mechanism by Nelson and Oppen [NO79]. On the theoretical side, our version fits rather nicely into the CLP scheme, as it simply adds another level of *don't know* nondeterminism (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.

There are two issues, among others, that we believe are very significant and deserve further investigation and development. Both of them involve the combination method used in our extension.

The first issue concerns the requirements on the signatures of the component theories. The combination results we appeal to do not allow theories that share function or predicate symbols. An extension of the results to cases in which the theories share (a finite number of) constant symbols is almost trivial and could be used, in principle, to further extend MCLP($\bar{\mathcal{X}}$). For the more interesting case of component theories sharing predicate and function symbols of nonzero arity, some initial combination results have been recently published by Christophe Ringeissen in [Rin96]. Although promising, these results are somewhat limited on model-theoretic and computational grounds. In [TR98], Tinelli (first author of this paper) and Ringeissen reconsider the problem with a different model-theoretic approach, trying to get more satisfactory results.

The second issue has a much wider scope and is, in fact, not addressed by most of the methods found in the literature related to combination methods. The main assumption of MCLP($\bar{\mathcal{X}}$) is that the combined constraint structure $\bar{\mathcal{X}}$ uses exclusively the symbols and the axioms of the component theories. Composite-domain theories, however, often contain functions that are not definable within the theory of any component domains. Immediate examples can be found in several abstract-data-type theories with standard functions, such as $\mathit{length}, \mathit{size}, \ldots$, whose inductive definitions can be given only if the theory already includes that of the natural numbers. This means, for instance, that if we have a constraint solver for the theory of lists, say, and another for the theory of natural numbers under addition, our scheme will still be unable to reason about a length function over lists because neither of the individual theories defines it. An interesting approach for enriching constraint domains with new function or predicate symbols and extending their solvers accordingly is given in [MR96]. The focus of that work is on a single domain and solver. It would be interesting to see how the approach described there applies to the combination of multiple domains, so that it can fit into our extension.

Acknowledgment of support: We would like to thank Mahmood-Reza Ziaei for his careful proofreading of the paper, and Joshua Caplan and Michele Zito for a number of discussions on some model-theoretic issues related to this work. We would also like to thank the paper's anonymous referees for their invaluable comments and corrections. This work was partially supported by grant DACA88-94-0014 from the US Army Construction Engineering Laboratories.

References

[Bou93] Alexandre Boudet. Combining unification algorithms. *Journal of Symbolic Computation*, 16(6):597–626, December 1993.

[BS92] 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, Berlin, 1992. Springer-Verlag.

[BS95a] 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, Berlin, 1995. Springer-Verlag.

[BS95b] 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)*, volume 976 of *Lecture Notes in Artificial Intelligence*, Berlin, September 1995. Springer-Verlag.

[BT97] Franz Baader and Cesare Tinelli. A new approach for combining decision procedures for the word problem, and its connection to the Nelson-Oppen combination method. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction (Townsville, Australia)*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 19–33, Berlin, 1997. Springer-Verlag.

[CK90] C. C. Chang and H. Jerome Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Foundations of Mathematics.* North-Holland. New York, 1990.

[Col87] Alain Colmerauer. Opening the Prolog III universe. *Byte Magazine*, 12(9):177–182, August 1987.

[Col90] Alain Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, July 1990.

[DKR94] E. Domenjoud, F. Klay, and C. Ringeissen. Combination techniques for non-disjoint equational theories. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction (Nancy, France)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 267–281, Berlin, 1994. Springer-Verlag.

[DVS+88] Mehmet Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *FGCS-88: Proceedings of the International Con-*

ference on Fifth Generation Computer Systems, pages 693–702, Tokyo, December 1988. ICOT.

[GPT96] Fausto Giunchiglia, Paolo Pecchiari, and Carolyn Talcott. Reasoning theories—towards an architecture for open mechanized reasoning systems. In F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop*, Applied Logic, pages 157–174, Norwell, MA, March 1996. Kluwer.

[Her86] A. Herold. Combination of unification algorithms. In J. Siekmann, editor, *Proceedings of the 8th International Conference on Automated Deduction (Oxford, UK)*, volume 230 of *Lecture Notes in Artificial Intelligence*, pages 450–469, Berlin, 1986. Springer-Verlag.

[HS88] Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. LILOG Report 53, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, October 1988.

[JL86] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. Technical Report 86/74, Monash University, Victoria, Australia, June 1986.

[JLM87] Joxan Jaffar, Jean-Louis Lassez, and Michael Maher. Prolog II as an instance of the logic programming language scheme. In M. Wirsing, editor, *Formal Description of Programming Concepts III.* August 1987. North-Holland.

[JM94] Joxan Jaffar and Michael Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[JMSY92] Joxan Jaffar, Spiro Michayov, Peter Stuckey, and Roland Yap. The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, July 1992.

[KR92] Hélène Kirchner and Christophe Ringeissen. A constraint solver in finite algebras and its combination with unification algorithms. In K. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 225–239, Cambridge, MA, 1992. MIT Press.

[KS96] Stephan Kepser and Klaus U. Schulz. Combination of constraint systems II: Rational amalgamation. In E. C. Freuder, editor, *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming (Cambridge, MA)*, volume 1118 of *Lecture Notes in Computer Science*, pages 282–296, Berlin, August 1996. Springer-Verlag.

[Mah87] Michael Maher. Logic semantics for a class of committed-choice programs. In Jean-Louis Lassez, editor, *ICLP'87: Proceedings of the 4th International Conference on Logic Programming*, pages 858–876, Cambridge, MA, May 1987. MIT Press.

[Mah88] M. J. Maher. Complete axiomatizations of finite, rational, and infinite trees. In *Proceedings of the 3rd Symposium on Logic in Computer Science*, pages 348–357, Washington, DC, June 1988. IEEE Computer Society Press.

[MR96] Eric Monfroy and Christophe Ringeissen. Domain-independent constraint solver extension. Technical Report 96-R-043, Centre de Recherche en Informatique de Nancy, 1996.

[Nel84] Greg Nelson. Combining satisfiability procedures by equality sharing. *Contemporary Mathematics*, 29:201–211, 1984.

[NO79] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, October 1979.

[Opp80] Derek C. Oppen. Complexity, convexity, and combinations of theories. *Theoretical Computer Science*, 12, 1980.

[Ric96] Jörn Richts. Optimization for combining unification algorithms. In K. U. Schulz and S. Kepser, editors, *Proceedings of the 10th International Workshop on Unification, UNIF'96*. CIS-Report 96-9, CIS, Universität München, June 1996. (Extended abstract).

[Rin96] Christophe Ringeissen. Cooperation of decision procedures for the satisfiability problem. In F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop*, Applied Logic, pages 121–140, Norwell, MA, March 1996. Kluwer.

[Sho67] Joseph. R. Shoenfield. *Mathematical Logic.* Addison-Wesley. Reading, MA, 1967.

[Sho84] Robert E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31:1–12, 1984.

[Smo89] Gert Smolka. Logic programming over polymorphically order-sorted types. PhD Thesis, Fachbereich Informatik, Universität Kaiserslautern, Germany, 1989.

[SS89] Manfred Schmidt-Schauß. Combination of unification algorithms. *Journal of Symbolic Computation*, 8(1–2):51–100, 1989.

[Tar55] Alfred Tarski. A lattice-theoretic fix-point theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[TH96a] Cesare Tinelli and Mehdi Harandi. Constraint logic programming over unions of constraint theories. In E. C. Freuder, editor, *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming (Cambridge, MA)*, volume 1118 of *Lecture Notes in Computer Science*, pages 436–450, Berlin, August 1996. Springer-Verlag.

[TH96b] Cesare Tinelli and Mehdi Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop*, Applied Logic, pages 103–120, Norwell, MA, March 1996. Kluwer.

[TR98] Cesare Tinelli and Christophe Ringeissen. Non-disjoint unions of theories and combinations of satisfiability procedures: First results. Technical Report UIUCDCS-R-98-2044, Department of Computer Science, University of Illinois at Urbana-Champaign, April 1998. (Also available as INRIA research report no. RR-3402.)

[Yel87] K. Yelik. Unification in combinations of collapse-free regular theories. *Journal of Symbolic Computation*, 3(1–2):153–182, April 1987.