Efficient Structural Information Analysis for Real CLP Languages*

Roberto Bagnara[1], Patricia M. Hill[2], and Enea Zaffanella[1]

[1] Department of Mathematics, University of Parma, Italy.
{bagnara,zaffanella}@cs.unipr.it
[2] School of Computer Studies, University of Leeds, U. K.
hill@scs.leeds.ac.uk

Abstract. We present the rational construction of a generic domain for structural information analysis of real CLP languages called Pattern($\mathcal{D}^\sharp$), where the parameter $\mathcal{D}^\sharp$ is an abstract domain satisfying certain properties. Our domain builds on the parameterized domain for the analysis of logic programs `Pat(`$\Re$`)`, which is due to Cortesi et al. However, the formalization of our CLP abstract domain is independent from specific implementation techniques: `Pat(`$\Re$`)` (suitably extended in order to deal with CLP systems omitting the occurs-check) is one of the possible implementations. Reasoning at a higher level of abstraction we are able to appeal to familiar notions of unification theory. This higher level of abstraction also gives considerable more latitude for the implementer. Indeed, as demonstrated by the results summarized here, an analyzer that incorporates structural information analysis based on our approach can be highly competitive both from the precision *and*, contrary to popular belief, from the efficiency point of view.

1 Introduction

Most interesting CLP languages [16] offer a constraint domain that is an amalgamation of a domain of syntactic trees — like the classical domain of finite trees (a.k.a. Herbrand domain) or the domain of rational trees [8] — with a set of "non-syntactic" domains, like finite domains, the domain of rational numbers and so forth. The inclusion of uninterpreted functors is essential for preserving Prolog programming techniques. Moreover, the availability of syntactic constraints greatly contributes to the expressive power of the overall language. When syntactic structures can be used to build aggregates of interpreted terms one can express, for instance, "records" or "unbounded containers" of numerical quantities.

From the experience gained with the first prototype version of the China data-flow analyzer [2] it was clear that, in order to attain a significant precision

* This work has been partly supported by MURST project "Certificazione automatica di programmi mediante interpretazione astratta." Some of this work was done during a visit of the first and third authors to Leeds, funded by EPSRC under grant M05645.

in the analysis of numerical constraints in CLP languages, one must keep at least part of the uninterpreted terms in concrete form. Note that almost any analysis is more precise when this kind of structural information is retained to some extent: in the case mentioned here the precision loss was just particularly acute. Of course, structural information is very valuable in itself. When exploited for optimized compilation it allows for enhanced clause indexing and simplified unification. Moreover, several program verification techniques are highly dependent on this kind of information.

Cortesi et al. [9, 10], after the work of Musumbu [21], put forward a very nice proposal for dealing with structural information in the analysis of logic programs. Using their terminology, they defined a generic abstract domain $\mathtt{Pat}(\Re)$ that automatically upgrades a domain $\Re$ (which must support a certain set of elementary operations) with structural information.

As far as the overall approach is concerned, we extend the work described in [10] by allowing for the analysis of any CLP language [16]. Most importantly, we do *not* assume that the analyzed language performs the *occurs-check* in the unification procedure. This is an important contribution, since the vast majority of implemented CLP languages (in particular, almost all Prolog systems) do omit the occurs-check, either as a mere efficiency measure or because they are based upon a theory of extended rational trees [8]. We describe a generic construction for structural analysis of CLP languages. Given an abstract domain $\mathcal{D}^\sharp$ satisfying a small set of very reasonable and weak properties, the structural abstract domain Pattern($\mathcal{D}^\sharp$) is obtained automatically by means of this construction. In contrast to [10], where the authors define a specific implementation of the generic structural domain (e.g., of the representation of term-tuples), the formalization of Pattern($\cdot$) is implementation-independent: $\mathtt{Pat}(\Re)$ (suitably extended in order to deal with CLP languages and with the occurs-check problem) is a possible base for the implementation. Reasoning at a higher level of abstraction we are able to appeal to familiar notions of unification theory. One advantage is that we can identify an important parameter (a common anti-instance function) that gives some control over the precision and computational cost of the resulting structural domain. In addition, we believe our implementation-independent treatment can be more easily adapted to different analysis frameworks/systems.

One of the merits of $\mathtt{Pat}(\Re)$ is to define a generic implementation that works on any domain $\Re$ that provides a certain set of elementary, fine-grained operations. Because of the simplicity of these operations it is particularly easy to extend an existing domain in order to accommodate them. However, this simplicity has a high cost in terms of efficiency: the execution of many isolated small operations over the underlying domain is much more expensive than performing few macro-operations where global effects can be taken into account. The operations that the underlying domain must provide are thus more complicated in our approach. However, this extra complication and the higher level of abstraction give considerable more latitude for the implementer. Indeed, as demonstrated by the results summarized here, an analyzer that incorporates structural information analysis based on our approach can be highly competitive both from the precision and the efficiency point of view. One of the contributions of this paper is that it disproves the common belief (now reinforced by [7]) whereby abstract domains enhanced with structural information are inherently inefficient.

The paper is structured as follows: Section 2 introduces some basic concepts and the notation that will be used in the paper; Section 3 presents the main ideas behind the tracking of explicit structural information for the analysis of CLP languages; Section 4 introduces the $\mathcal{D}^\sharp$ and Pattern($\mathcal{D}^\sharp$) domains and explains how an abstract semantics based on $\mathcal{D}^\sharp$ can systematically be upgraded to one on Pattern($\mathcal{D}^\sharp$); Section 5 summarizes the extensive experimental evaluation that has been conducted to validate the ideas presented in this paper; Section 6 discusses recent related work and, finally, Section 7 concludes with some final remarks.

2 Preliminaries

Let U be a set. The cardinality of U is denoted by $|U|$. We will denote by U^n the set of n-tuples of elements drawn from U, whereas U^* denotes $\bigcup_{n\in\mathbb{N}} U^n$. Elements of U^* will be referred to as *tuples* or as *sequences*. The *empty sequence*, i.e., the only element of U^0, is denoted by ε. Throughout the paper all variables denoting sequences will be written with a “bar accent” like in $\bar{s}$. For $\bar{s} \in U^*$, the *length* of $\bar{s}$ will be denoted by $|\bar{s}|$. The concatenation of the sequences $\bar{s}_1, \bar{s}_2 \in U^*$ is denoted by $\bar{s}_1 :: \bar{s}_2$. We define the operation $\cdot \setminus \cdot \colon U^\star \times \wp_\mathrm{f}(U) \to U^\star$ as follows. For each $\bar{s} \in U^\star$ and each set $X \in \wp_\mathrm{f}(U)$, the sequence $\bar{s} \setminus X$ is obtained by removing from $\bar{s}$ all the elements that appear in X. Formally,

$$\varepsilon \setminus X \stackrel{\mathrm{def}}{=} \varepsilon;$$

$$\bigl((x) :: \bar{s}\bigr) \setminus X \stackrel{\mathrm{def}}{=} \begin{cases} \bar{s} \setminus X, & \text{if } x \in X; \\ (x) :: (\bar{s} \setminus X), & \text{if } x \notin X. \end{cases}$$

The *projection mappings* $\pi_i \colon U^n \to U$ are defined, for $i = 1, \ldots, n$, by

$$\pi_i\bigl((e_1, \ldots, e_n)\bigr) \stackrel{\mathrm{def}}{=} e_i.$$

We will also use the liftings $\pi_i \colon \wp(U^n) \to \wp(U)$ given by

$$\pi_i(S) \stackrel{\mathrm{def}}{=} \bigl\{\, \pi_i(\bar{s}) \bigm| \bar{s} \in S \,\bigr\}.$$

If a sequence $\bar{s}$ is such that $|\bar{s}| \geq i$, we let $\mathrm{prefix}_i(\bar{s})$ denote the sequence of the first i elements of $\bar{s}$.

Let *Vars* denote a denumerable and totally ordered set of variable symbols. We assume that *Vars* contains (among others) two infinite, disjoint subsets: $\mathbf{z}$ and $\mathbf{z}'$. Since *Vars* is totally ordered, $\mathbf{z}$ and $\mathbf{z}'$ are as well. Thus we assume $\mathbf{z} \stackrel{\mathrm{def}}{=} (Z_1, Z_2, Z_3, \ldots$ and $\mathbf{z}' \stackrel{\mathrm{def}}{=} (Z'_1, Z'_2, Z'_3, \ldots$. If $W \subseteq$ *Vars* we will denote by $\mathcal{T}_W$ the set of terms with variables in W. For any term or a tuple of terms t we will denote the set of variables occurring in t by $\mathit{vars}(t)$. We will also denote by $\mathit{vseq}(t)$ the sequence of first occurrences of variables that are found on a depth-first, left-to-right traversal of t. For instance, $\mathit{vseq}\bigl(\bigl(f(g(X), Y), h(X)\bigr)\bigr) = (X, Y)$.

We implement the "renaming apart" mechanism by making use of two strong normal forms for tuples of terms. Specifically, the set of *n-tuples in* $\mathbf{z}$*-form* is given by

$$\mathbf{T}^n_{\mathbf{z}} \stackrel{\mathrm{def}}{=} \Bigl\{\, \bar{t} \in \mathcal{T}^n_{\mathit{Vars}} \Bigm| \mathit{vseq}(\bar{t}) = \bigl(Z_1, Z_2, \ldots, Z_{|\mathit{vars}(\bar{t})|}\bigr) \,\Bigr\}.$$

The set of all the tuples in $\mathbf{z}$-form is denoted by $\mathbf{T}^*_{\mathbf{z}}$. The definitions for $\mathbf{T}^n_{\mathbf{z}'}$ and $\mathbf{T}^*_{\mathbf{z}'}$ are obtained in a similar way, by replacing $\mathbf{z}$ with $\mathbf{z}'$.

There is a useful device for toggling between $\mathbf{z}$- and $\mathbf{z}'$-forms. Let $\bar{t} \in \mathbf{T}^n_{\mathbf{z}} \cup \mathbf{T}^n_{\mathbf{z}'}$ and $\bigl|\mathit{vars}(\bar{t})\bigr| = m$. Then

$$\bar{t}' \stackrel{\mathrm{def}}{=} \begin{cases} \bar{t}[Z'_1/Z_1, \ldots, Z'_m/Z_m], & \text{if } \bar{t} \in \mathbf{T}^n_{\mathbf{z}}; \\ \bar{t}[Z_1/Z'_1, \ldots, Z_m/Z'_m], & \text{if } \bar{t} \in \mathbf{T}^n_{\mathbf{z}'}. \end{cases}$$

Notice that $\bar{t}'' \stackrel{\mathrm{def}}{=} \bigl(\bar{t}'\bigr)' = \bar{t}$.

We will make use of a *normalization function* $\eta \colon \mathcal{T}^*_{\mathit{Vars}} \to \mathbf{T}^*_{\mathbf{z}}$ such that, for each $\bar{t} \in \mathcal{T}^*_{\mathit{Vars}}$, the resulting tuple $\eta(\bar{t}) \in \mathbf{T}^*_{\mathbf{z}}$ is a variant of $\bar{t}$.

For each $\bar{s} \in \mathcal{T}^*_{\mathit{Vars}}$ and each syntactic object o such that $FV(o) \subset \mathbf{z}$, we write $\varrho_{\bar{s}}(o)$ (read "rename o away from $\bar{s}$") to denote $o[Z_{n+i_1}/Z_{i_1}, \ldots, Z_{n+i_m}/Z_{i_m}]$, where $n = \bigl|\mathit{vars}(\bar{s})\bigr|$ and $\{Z_{i_1}, \ldots, Z_{i_m}\} = \mathit{vars}(o)$. This device will be useful for concatenating normalized term-tuples, still obtaining a normalized term-tuple. In fact, for each $\bar{s}_1, \bar{s}_2 \in \mathbf{T}^*_{\mathbf{z}}$ we have $\bar{s}_1 :: \varrho_{\bar{s}_1}(\bar{s}_2) \in \mathbf{T}^*_{\mathbf{z}}$.

When $\bar{V} \in \mathit{Vars}^m$ is a finite sequence of distinct variables and $\bar{t} \in \mathcal{T}^m_{\mathit{Vars}}$ we use $[\bar{t}/\bar{V}]$ as a shorthand for the substitution

$$\bigl[\pi_1(\bar{t})/\pi_1(\bar{V}), \ldots, \pi_m(\bar{t})/\pi_m(\bar{V})\bigr],$$

if $m > 0$, and to denote the empty substitution if $m = 0$. The substitution $[\bar{t}/\bar{V}]$ is said *idempotent* if $\mathit{vars}(\bar{t}) \cap \bar{V} = \varnothing$. Suppose that $\bar{s} = (s_1, \ldots, s_m) \in \mathcal{T}^m_{\mathit{Vars}}$ and $\bar{t} = (t_1, \ldots, t_m) \in \mathcal{T}^m_{\mathit{Vars}}$, then, $\bar{s} = \bar{t}$ denotes $\{\bar{s}_1 = \bar{t}_1, \ldots, \bar{s}_m = \bar{t}_m\}$. It is also useful to sometimes regard a substitution $[\bar{t}/\bar{V}]$ as the finite set of equations $\bar{V} = \bar{t}$. A couple of observations are useful for what follows. If $\bar{s} \in \mathbf{T}^*_{\mathbf{z}}$ and $\bar{u} \in \mathbf{T}^{|\mathit{vars}(\bar{s})|}_{\mathbf{z}}$, then $\bar{s}'\bigl[\bar{u}/\mathit{vseq}(\bar{s}')\bigr] \in \mathbf{T}^*_{\mathbf{z}}$. Moreover $\mathit{vseq}\bigl(\bar{s}'\bigl[\bar{u}/\mathit{vseq}(\bar{s}')\bigr]\bigr) = \mathit{vseq}(\bar{u})$.

The logical theory underlying a CLP constraint system [16] is denoted by $\mathfrak{T}$. To simplify the notation, we drop the outermost universal quantifiers from (closed) formulas so that if F is a formula with free variables $\bar{Z}$, then we write $\mathfrak{T} \models F$ to denote the expression $\mathfrak{T} \models \forall \bar{Z} : F$.

3 Making the Herbrand Information Explicit

A quite general picture for the analysis of a CLP language is as follows. We want to describe a (possibly infinite) set of constraint stores over a tuple of distinct *variables of interest* $(V_1, \dots, V_k)$. Each constraint store can be represented, at some level of abstraction, by a formula of the kind

$$\exists_{\Delta} \,.\, \bigl(\{V_1 = t_1, \dots, V_k = t_k\} \wedge C\bigr), \tag{1}$$

such that

$$\{V_1 = t_1, \dots, V_k = t_k\}, \quad \text{with } t_1, \dots, t_k \in \mathcal{T}_{\mathit{Vars}}, \tag{2}$$

is a system of Herbrand equations in solved form, $C \in \mathcal{C}^\flat$ is a constraint, and $\Delta \overset{\mathrm{def}}{=} \mathit{vars}(C) \cup \mathit{vars}(t_1) \cup \cdots \cup \mathit{vars}(t_k)$ is such that $\Delta \cap \{V_1, \dots, V_k\} = \varnothing$. Roughly speaking, C limits the values that the quantified variables occurring in $t_1, \dots, t_k$ can take. Notice that this treatment does not exclude the possibility of dealing with domains of rational trees: the non-Herbrand constraints will simply live in the constraint component. For example, the constraint store resulting from execution of the SICStus goal '`?- X = f(a, X)`' may be captured by

$$\exists X \,.\, \bigl(\{V_1 = X\} \wedge X = f(a, X)\bigr)$$

but also by

$$\exists X \,.\, \bigl(\bigl\{V_1 = f(a, X)\bigr\} \wedge X = f(a, X)\bigr).$$

Once variables $V_1, \dots, V_k$ have been fixed, the Herbrand part of the constraint store (1), the system of equations (2), can be represented as a k-tuple of terms. We are thus assuming a concrete domain where the Herbrand information is explicit and other kinds of information are captured by some given constraint domain $\mathcal{C}^\flat$. For instance, if the target language of the analysis is CLP($\mathcal{R}$) [17], $\mathcal{C}^\flat$ may encode, in addition to the cyclic bindings due to the omission of the occurs-check, conjunctions of equations and inequations over arithmetic expressions, the mechanisms for delaying non-linear constraints, and other peculiarities of the arithmetic part of the language. We assume constraints are modeled by logical formulas, so that it makes sense to talk about the *free variables* of $C^\flat \in \mathcal{C}^\flat$, denoted by $FV(C^\flat)$. These are the variables that the constraint solver makes visible to the Herbrand engine, all the other variables being restricted in scope to the solver itself. Since we want to characterize any set of constraint stores, our concrete domain is

$$\mathcal{D}^\flat \overset{\mathrm{def}}{=} \bigcup_{n \in \mathbb{N}} \wp\Bigl(\bigl\{\, (\bar{s}, C^\flat) \bigm| \bar{s} \in \mathbf{T}^n_{\mathbf{z}}, C^\flat \in \mathcal{C}^\flat, FV(C^\flat) \subseteq \mathit{vars}(\bar{s}) \,\bigr\}\Bigr)$$

partially ordered by subset inclusion.

An abstract interpretation [11] of $\mathcal{D}^\flat$ can be specified by choosing an abstract domain $\mathcal{D}^\sharp$ and a suitable abstraction function $\alpha \colon \mathcal{D}^\flat \to \mathcal{D}^\sharp$. If $\mathcal{D}^\sharp$ is not able to encode enough structural information from $\mathcal{C}^\flat$ so as to achieve the desired precision, it is possible to improve the situation by keeping some Herbrand information explicit. One way of doing that is to perform a change of representation for $\mathcal{D}^\flat$ and use the new representation as the basis for abstraction. The new representation is obtained by factoring out some common Herbrand information. The meaning of 'some' is encoded by a function.

Definition 1. (Common anti-instance function.) *For each $n \in \mathbb{N}$, a function $\phi \colon \wp(\mathbf{T}_{\mathbf{z}}^{n}) \to \mathbf{T}_{\mathbf{z}'}^{n}$ is called a* common anti-instance function *if and only if the following holds: whenever $T \in \wp(\mathbf{T}_{\mathbf{z}}^{n})$, if $\phi(T) = \bar{r}'$ and $\bigl|vars(\bar{r}')\bigr| = m$ with $m \geq 0$, then*

$$\forall \bar{t} \in T : \exists \bar{u} \in \mathbf{T}_{\mathbf{z}}^{m} \mathrel{.} \bar{r}'\bigl[\bar{u}/vseq(\bar{r}')\bigr] = \bar{t}.$$

In words, $\phi(T)$ is an anti-instance [18], *in $\mathbf{z}'$-form, of each $\bar{t} \in T$.*

Any choice of ϕ induces a function $\varPhi_{\phi} \colon \mathcal{D}^{\flat} \to \mathbf{T}_{\mathbf{z}}^{*} \times \mathcal{D}^{\flat}$, which is given, for each $E^{\flat} \in \mathcal{D}^{\flat}$, by

$$\varPhi_{\phi}(E^{\flat}) \stackrel{\mathrm{def}}{=} \left(\bar{s}, \left\{ (\bar{u}, G^{\flat}) \;\middle|\; (\bar{t}, G^{\flat}) \in E^{\flat}, \bar{s}'\bigl[\bar{u}/vseq(\bar{s}')\bigr] = \bar{t} \right\}\right),$$

where $\bar{s}' \stackrel{\mathrm{def}}{=} \phi\bigl(\pi_1(E^{\flat})\bigr)$. The corestriction to the image of $\varPhi_{\phi}$, that is the function $\varPhi_{\phi} \colon \mathcal{D}^{\flat} \to \varPhi_{\phi}\bigl(\mathcal{D}^{\flat}\bigr)$, is an isomorphism, the inverse being given, for each $F^{\flat} \in \mathcal{D}^{\flat}$, by

$$\varPhi_{\phi}^{-1}\bigl((\bar{s}, F^{\flat})\bigr) \stackrel{\mathrm{def}}{=} \left\{ \left(\bar{s}'\bigl[\bar{u}/vseq(\bar{s}')\bigr], G^{\flat}\right) \;\middle|\; (\bar{u}, G^{\flat}) \in F^{\flat} \right\}.$$

So far, we have just chosen a different representation for $\mathcal{D}^{\flat}$, that is $\varPhi_{\phi}\bigl(\mathcal{D}^{\flat}\bigr)$.

$$\begin{tikzcd}
\wp\bigl(\mathbf{T}_{\mathbf{z}}^{*} \times \mathcal{C}^{\flat}\bigr) \arrow[r, "\alpha"] \arrow[d, shift right, "\varPhi_{\phi}"'] & \mathcal{D}^{\sharp} \\
\mathbf{T}_{\mathbf{z}}^{*} \times \wp\bigl(\mathbf{T}_{\mathbf{z}}^{*} \times \mathcal{C}^{\flat}\bigr) \arrow[r, "(\mathrm{id},\alpha)"'] \arrow[u, shift right, "\varPhi_{\phi}^{-1}"'] & \mathbf{T}_{\mathbf{z}}^{*} \times \mathcal{D}^{\sharp} \arrow[u, dotted, "\alpha'"']
\end{tikzcd}$$

Fig. 1. Upgrading a domain with structural information.

The idea behind structural information analysis is to leave the first component of the new representation (the *pattern component*) untouched, while abstracting the second component by means of α, as illustrated in Figure 1. The dotted arrow indicates a *residual abstraction function* α'. As we will see in Section 4.2, such a function is implicitly required in order to define an important operation over the new abstract domain $\mathbf{T}_{\mathbf{z}}^{*} \times \mathcal{D}^{\sharp}$. Notice that, in general, α' does not make the diagram of Figure 1 commute.

This approach has several advantages. First, factoring out common structural information improves the analysis precision, since part of the approximated k-tuples of terms is recorded, *in concrete form*, into the first component of $\mathbf{T}_{\mathbf{z}}^{*} \times \mathcal{D}^{\sharp}$. Secondly, the above construction is adjustable by means of the parameter ϕ. The most precise choice consists in taking ϕ to be a *least common anti-instance* (lca) function. For example, the set

$$E^{\flat} \stackrel{\mathrm{def}}{=} \Bigl\{ \bigl\langle \bigl(s(0), c(Z_1, nil)\bigr), C_1 \bigr\rangle, \bigl\langle \bigl(s(s(0)), c(Z_1, c(Z_2, nil))\bigr), C_2 \bigr\rangle \Bigr\},$$

where $C_1, C_2 \in \mathcal{C}^\flat$, is mapped by the Φ_{lca} function onto

$$\Phi_{\mathrm{lca}}(E^\flat) = \Bigl(\bigl(s(Z_1), c(Z_2, Z_3)\bigr), \\ \Bigl\{ \bigl\langle (0, Z_1, \mathit{nil}), C_1 \bigr\rangle, \bigl\langle \bigl(s(0), Z_1, c(Z_2, \mathit{nil})\bigr), C_2 \bigr\rangle \Bigr\} \Bigr).$$

At the other end of the spectrum is the possibility of choosing ϕ so that it returns a k-tuple of distinct variables for each set of k-tuples of terms. This corresponds to a framework where structural information is simply discarded. With this choice, $E^\flat$ would be mapped onto $\bigl((Z_1, Z_2), E^\flat\bigr)$. In-between these two extremes there are a number of possibilities that help to manage the complexity/precision tradeoff. The tuples returned by ϕ can be limited in *depth* [20, 22], for instance. Another possibility is to limit them in *size*, that is, limiting the number of occurrences of symbols or the number of variables. This flexibility enables the analysis' domains to be designed without considering the structural information: the problem for the domain designers is always to approximate the elements of $\wp\bigl(\mathbf{T}_{\mathbf{Z}}^k \times \mathcal{C}^\flat\bigr)$ with respect to the property of interest. It does not really matter whether k is fixed by the arity of a predicate or k is the number of variables occurring in a pattern.

It must be stressed that the abstract interpretation framework we are outlining — the concrete semantics in particular — while providing an adequate basis for most data-flow analyses, is "too abstract" when the properties of interest concern the internal workings of the Herbrand constraint solver. An example is *structure-sharing analysis* [23], whose aim is to determine those *structure cells* (elementary objects used to represent terms) that are possibly shared by more than one term representation. For example, the system

$$\{V_1 = f(a), V_2 = f(a)\} \tag{3}$$

does not say anything about structure sharing: we might have a shared $f/1$ cell, or two distinct ones. In the second case we might have a shared $a/0$ cell, or two distinct ones. Thus, there are a total of 3 cases that cannot be distinguished by looking at (3), the obvious consequence being that we cannot base structure-sharing analysis on a concrete domain made up of representations of the form (1). We thus assume that we are dealing with properties that are insensible to the internal representation of terms.

4 Parametric Structural Information Analysis

In this section we describe how a complete abstract semantics — which includes an abstract domain plus all the operations needed to approximate the concrete semantics — can be turned into one keeping track of structural information.

We first need some assumptions on the domain $\mathcal{C}^\flat$, which represents the non-Herbrand part of constraint stores. Following [13], it is not at all restrictive to assume that, in order to define the concrete semantics of programs, four operations over $\mathcal{C}^\flat$ need to be characterized. These model the constraint accumulation process, parameter passing, projection, and renaming apart (see also [2, 3] on this subject).

Constraint accumulation is modeled by the binary operator '$\otimes$': $\mathcal{C}^\flat \times \mathcal{C}^\flat \to \mathcal{C}^\flat$ and the unsatisfiability condition in the constraint solver is modeled by the special value $\bot^\flat \in \mathcal{C}^\flat$. Notice that, while '$\otimes$' may be reasonably expected to satisfy certain properties, such as $\forall C^\flat \in \mathcal{C}^\flat : \bot^\flat \otimes C^\flat = \bot^\flat$, these are not really required for what follows. The same applies to all the other operators we will introduce: only properties that are actually used will be singled out.

Parameter passing requires, roughly speaking, the ability of adding equality constraints to a constraint store. Notice that we assume $\mathcal{C}^\flat$ and its operations encode both the proper *constraint solver* and the so called *interface* between the *Herbrand engine* and the solver [16]. In particular, the interface is responsible for *type-checking* of the equations it receives. For example in CLP($\mathcal{R}$) the interface is responsible for the fact that $X = a$ cannot be consistently added to a constraint store where X was previously classified as numeric.

Another ingredient for defining the concrete semantics of any CLP system is the projection of a satisfiable constraint store onto a set of variables. This is modeled by the family of operators $\bigl\{\, \P^\flat_\varDelta : \mathcal{C}^\flat \to \mathcal{C}^\flat \bigm| \varDelta \in \wp_\mathrm{f}(\mathit{Vars}) \,\bigr\}$. If $\varDelta$ is a finite set of variables and $C^\flat \in \mathcal{C}^\flat$ represents a satisfiable constraint store (i.e., $C^\flat \neq \bot^\flat$), then $\P^\flat_\varDelta\, C^\flat$ represents the projection of $C^\flat$ onto the variables in $\varDelta$.

Let $\bar{s}, \bar{t}, \bar{u} \in \mathbf{T}^*_\mathbf{z}$ be normalized term tuples and $C^\flat, G^\flat \in \mathcal{C}^\flat$ such that $FV(C^\flat) \subseteq \mathit{vars}(\bar{t})$. We use the notation $(\bar{u}, G^\flat) = \varrho_{\bar{s}}\bigl((\bar{t}, C^\flat)\bigr)$ meaning that $\bar{u} = \varrho_{\bar{s}}(\bar{t})$ and that $G^\flat$ has been obtained from $C^\flat$ by applying the same renaming applied to $\bar{t}$ in order to obtain $\bar{u}$. This device is needed in order to ensure renaming apart. Similarly, we will write $(\bar{u}, G^\flat) = \eta\bigl((\bar{t}, C^\flat)\bigr)$ meaning that $\bar{u} = \eta(\bar{t})$ and that $G^\flat$ has been obtained from $C^\flat$ by applying the same renaming applied to $\bar{t}$ in order to obtain $\bar{u}$.

It is often helpful to think of $\mathcal{C}^\flat$ as made up of first-order formulas [3]: in this view, '$\cdot \otimes \cdot$' is logical conjunction, parameter passing amounts to conjunction of a formula with an equality formula, and '$\P^\flat_\varDelta \cdot$' corresponds to existential quantification on the variables in $\mathit{Vars} \setminus \varDelta$. While this is exactly how the "intended semantics" of CLP languages is captured, more concrete, *non-standard* semantics are often useful for the purpose of data-flow analysis. For instance, if the analysis must be sensible to the order in which constraints are posted to the solver, '$\cdot \otimes \cdot$' cannot be interpreted as logical conjunction (it will not even be commutative, for that matter) [13].

We will now show how any abstract domain can be upgraded so as to capture structural information by means of the Pattern($\cdot$) construction. Then we will focus our attention on the abstract semantic operators.

4.1 From $\mathcal{D}^\sharp$ to Pattern($\mathcal{D}^\sharp$)

Since one of the driving aims of this work is maximum generality, we refer to a very weak abstract interpretation framework [11]. To start with, we assume very little on abstract domains.

Definition 2. (Abstract domain for $\mathcal{D}^\flat$.) *An* abstract domain for $\mathcal{D}^\flat$ *is a set $\mathcal{P}^\sharp$ equipped with a preorder relation '$\preceq$' $\subseteq \mathcal{P}^\sharp \times \mathcal{P}^\sharp$, an order preserving function $\gamma \colon \mathcal{P}^\sharp \to \mathcal{D}^\flat$, and a least element $\bot^\sharp$ such that $\gamma(\bot^\sharp) = \varnothing$. Moreover, γ is such that if $(\bar{p}_1, C^\flat) \in \gamma(E^\sharp)$, and $\mathfrak{T} \models C^\flat \to \bar{p}_1 = \bar{p}_2$, then $\eta\bigl((\bar{p}_2, C^\flat)\bigr) \in \gamma(E^\sharp)$.*

Informally, $\mathcal{P}^\sharp$ is a set of abstract properties on which the notion of "relative precision" is captured by the preorder '$\preceq$'. Moreover, $\mathcal{P}^\sharp$ is related to the concrete domain $\mathcal{D}^\flat$ by means of a *concretization function* γ that specifies the soundness correspondence between $\mathcal{D}^\flat$ and $\mathcal{P}^\sharp$. The distinguished element $\bot^\sharp$ models an impossible state of affairs.[1] In this framework, $d^\sharp \in \mathcal{P}^\sharp$ is a safe approximation of $d^\flat \in \mathcal{D}^\flat$ if and only if $d^\flat \subseteq \gamma(d^\sharp)$.

Suppose we are given an abstract domain complying with Definition 2. Here is how it can be upgraded with explicit structural information.

Definition 3. (The Pattern($\cdot$) **construction.)** *Let $\mathcal{D}^\sharp$ be an abstract domain for $\mathcal{D}^\flat$ and let γ be its concretization function. Then*

$$\mathrm{Pattern}(\mathcal{D}^\sharp) \stackrel{\mathrm{def}}{=} \{\bot_p^\sharp\} \cup \left\{ (\bar{s}, E^\sharp) \in \mathbf{T}_{\mathbf{z}}^* \times \mathcal{D}^\sharp \;\middle|\; \gamma(E^\sharp) \subseteq \mathbf{T}_{\mathbf{z}}^{|vars(\bar{s})|} \times \mathcal{C}^\flat \right\}.$$

The meaning of each element $(\bar{s}, E^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ is given by the concretization function $\gamma_p \colon \mathrm{Pattern}(\mathcal{D}^\sharp) \to \mathcal{D}^\flat$ such that $\gamma_p(\bot_p^\sharp) \stackrel{\mathrm{def}}{=} \varnothing$ and

$$\gamma_p\bigl((\bar{s}, E^\sharp)\bigr) \stackrel{\mathrm{def}}{=} \left\{ \eta\bigl((\bar{r}, C^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{u}, C^\flat) \in \gamma(E^\sharp) \\ \mathfrak{T} \models C^\flat \to \bar{r} = \bar{s}'\bigl[\bar{u}/vseq(\bar{s}')\bigr] \end{array} \right\}.$$

The binary relation '$\preceq_p$' $\subseteq \mathrm{Pattern}(\mathcal{D}^\sharp) \times \mathrm{Pattern}(\mathcal{D}^\sharp)$ is given, for each $d_1^\sharp, d_2^\sharp \in \mathrm{Pattern}(\mathcal{D}^\sharp)$, by

$$d_1^\sharp \preceq_p d_2^\sharp \quad \stackrel{\mathrm{def}}{\iff} \quad \gamma_p(d_1^\sharp) \subseteq \gamma_p(d_2^\sharp),$$

It is easily seen that Pattern($\mathcal{D}^\sharp$) is an abstract domain in the sense of Definition 2 provided $\mathcal{D}^\sharp$ is so. The ordering on the underlying domain is preserved by the Pattern($\mathcal{D}^\sharp$) domain.

Proposition 1. *If $(\bar{s}, E_1^\sharp), (\bar{s}, E_2^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$, then*

$$\gamma(E_1^\sharp) \subseteq \gamma(E_2^\sharp) \quad \Longrightarrow \quad \gamma_p\bigl((\bar{s}, E_1^\sharp)\bigr) \subseteq \gamma_p\bigl((\bar{s}, E_2^\sharp)\bigr).$$

[1] At this stage of the presentation, $\bot^\sharp$ is not really required and could be dispensed with. However, it is simplest to postulate its existence and properties now.

Proof. Suppose that $\gamma(E_1^\sharp) \subseteq \gamma(E_2^\sharp)$. Then, by Definition 3, we have

$$\begin{aligned}
\gamma_p\bigl((\bar{s}, E_1^\sharp)\bigr) &= \left\{ \eta\bigl((\bar{t}, C^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{u}, C^\flat) \in \gamma(E_1^\sharp) \\ \mathfrak{T} \models C^\flat \rightarrow \bar{t} = \bar{s}'\bigl[\bar{u}/vseq(\bar{s}')\bigr] \end{array} \right\} \\
&\subseteq \left\{ \eta\bigl((\bar{t}, C^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{u}, C^\flat) \in \gamma(E_2^\sharp) \\ \mathfrak{T} \models C^\flat \rightarrow \bar{t} = \bar{s}'\bigl[\bar{u}/vseq(\bar{s}')\bigr] \end{array} \right\} \\
&= \gamma_p\bigl((\bar{s}, E_2^\sharp)\bigr).
\end{aligned}$$

□

Up to now we have obtained from $\mathcal{D}^\sharp$ a new abstract domain Pattern($\mathcal{D}^\sharp$) that can constitute the basis for designing an abstract semantics for CLP. This will usually require selecting an abstract semantic function on Pattern($\mathcal{D}^\sharp$), an effective convergence criterion for the abstract iteration sequence (notice that the '$\preceq$' and '$\preceq_p$' relations are not required to be computable), and perhaps a convergence acceleration method ensuring rapid termination of the abstract interpreter [11]. The last ingredient to complete the recipe is a computable way to associate an abstract description $d^\sharp \in$ Pattern($\mathcal{D}^\sharp$) to each concrete property $d^\flat \in \mathcal{D}^\flat$.[2] For this purpose, we assume the existence of a computable function $\alpha_p \colon \mathcal{D}^\flat \rightarrow$ Pattern($\mathcal{D}^\sharp$) such that, for each $d^\flat \in \mathcal{D}^\flat$, we have $d^\flat \subseteq \gamma_p\bigl(\alpha_p(d^\flat)\bigr)$.

While one option is to design from scratch an abstract semantics based on Pattern($\mathcal{D}^\sharp$), it is more interesting to start with an abstract semantics centered around $\mathcal{D}^\sharp$. In this case, it is possible to systematically lift the semantic construction to Pattern($\mathcal{D}^\sharp$).

4.2 Operations over $\mathcal{D}^\sharp$ and Pattern($\mathcal{D}^\sharp$)

We now present the abstract operations we assume on $\mathcal{D}^\sharp$ and the derived operations over Pattern($\mathcal{D}^\sharp$). Each operator on $\mathcal{D}^\sharp$ is introduced by means of safety conditions that ensure the soundness of the derived operators over Pattern($\mathcal{D}^\sharp$).

Given the abstract domain, there are still many degrees of freedom for the design of a constructive abstract semantics. Thus, choices have to be made in order to give a precise characterization. In what follows we continue to strive for maximum generality. Where this is not possible we detail the design choices we have made in the development of the China analyzer [2]. While some things may need adjustments for other analysis frameworks, the general principles should be clear enough for anyone to make the necessary changes.

Meet with Renaming Apart We call *meet with renaming apart* (denoted by '$\triangleright$') the operation of taking two descriptions in $\mathcal{D}^\sharp$ and, roughly speaking,

[2] Strictly speaking, one could require computability of the method to hold only for a strict subset of $\mathcal{D}^\flat$ (e.g., depending on the kind of abstract iteration sequence employed, only the elements of $\mathcal{D}^\flat$ describing initial and/or final states need to be effectively associated with an abstract counterpart).

juxtaposing them. This is needed when "solving" a clause body with respect to the current interpretation and corresponds, at the concrete level, to a renaming followed by an application of the '$\otimes$' operator. Its counterpart on $\mathrm{Pattern}(\mathcal{D}^\sharp)$ is denoted by 'rmeet' and defined as follows.

Definition 4. ('$\rhd$' and 'rmeet') *Let '$\rhd$': $\mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ be such that, for each $E_1^\sharp, E_2^\sharp \in \mathcal{D}^\sharp$,*

$$\gamma(E_1^\sharp \rhd E_2^\sharp) = \left\{ \eta\bigl((\bar{r}, C_1^\flat \otimes G_2^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{r}_1, C_1^\flat) \in \gamma(E_1^\sharp) \\ (\bar{r}_2, C_2^\flat) \in \gamma(E_2^\sharp) \\ (\bar{w}_2, G_2^\flat) = \varrho_{\bar{r}_1}\bigl((\bar{r}_2, C_2^\flat)\bigr) \\ \mathfrak{T} \models (C_1^\flat \otimes G_2^\flat) \to \bar{r} = \bar{r}_1 :: \bar{w}_2 \end{array} \right\}.$$

Then, for each $(\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$,

$$\mathrm{rmeet}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr) \stackrel{\mathrm{def}}{=} \bigl(\bar{s}_1 :: \varrho_{\bar{s}_1}(\bar{s}_2),\, E_1^\sharp \rhd E_2^\sharp\bigr).$$

A consequence of this definition is that there is no precision loss in 'rmeet'.

Theorem 1. *For each $(\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$,*

$$\begin{aligned} &\gamma_p\Bigl(\mathrm{rmeet}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr)\Bigr) \\ &\qquad = \left\{ \eta\bigl((\bar{t}, C_1^\flat \otimes G_2^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{t}_1, C_1^\flat) \in \gamma_p\bigl((\bar{s}_1, E_1^\sharp)\bigr) \\ (\bar{t}_2, C_2^\flat) \in \gamma_p\bigl((\bar{s}_2, E_2^\sharp)\bigr) \\ (\bar{u}_2, G_2^\flat) = \varrho_{\bar{t}_1}\bigl((\bar{t}_2, C_2^\flat)\bigr) \\ \mathfrak{T} \models C_1^\flat \otimes G_2^\flat \to \bar{t} = \bar{t}_1 :: \bar{u}_2 \end{array} \right\}. \end{aligned}$$

Proof. To simplify the notation, let $\bar{q}_2 = \varrho_{\bar{s}_1}(\bar{s}_2)$. Then

$$\begin{aligned} &\gamma_p\Bigl(\mathrm{rmeet}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr)\Bigr) \\ &\qquad = \gamma_p\Bigl(\bigl(\bar{s}_1 :: \bar{q}_2, E_1^\sharp \rhd E_2^\sharp\bigr)\Bigr) \\ &\qquad \text{[by Definition 4]} \\ &\qquad = \left\{ \eta\bigl((\bar{t}, C^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{u}, C^\flat) \in \gamma(E_1^\sharp \rhd E_2^\sharp) \\ \mathfrak{T} \models C^\flat \to \bar{t} = (\bar{s}'_1 :: \bar{q}'_2)\bigl[\bar{u}/vseq(\bar{s}'_1 :: \bar{q}'_2)\bigr] \end{array} \right\} \qquad (4) \\ &\qquad \text{[by Definition 3]} \end{aligned}$$

Now, by the definition of $\rhd$, we have $(\bar{u}, C^\flat) \in \gamma(E_1^\sharp \rhd E_2^\sharp)$ if and only if all the following hold:

$$\begin{array}{ll} (\bar{r}_1, C_1^\flat) \in \gamma(E_1^\sharp), & (\bar{r}_2, C_2^\flat) \in \gamma(E_2^\sharp), \\ (\bar{w}_2, G_2^\flat) = \varrho_{\bar{r}_1}\bigl((\bar{r}_2, C_2^\flat)\bigr), & C^\flat = C_1^\flat \otimes G_2^\flat, \\ \mathfrak{T} \models C^\flat \to \bar{u} = \bar{r}_1 :: \bar{w}_2. & \end{array}$$

By the definition of Pattern($\mathcal{D}^\sharp$) $|\bar{r}_1| = |vars(\bar{s}_1)|$ and $|\bar{w}_2| = |vars(\bar{q}_2)| = |vars(\bar{s}_2)|$. Hence, the expression (4) can be rewritten as

$$\left\{ \eta\big((\bar{t}, C_1^\flat \otimes G_2^\flat)\big) \;\middle|\; \begin{array}{l} (\bar{r}_1, C_1^\flat) \in \gamma(E_1^\sharp) \\ (\bar{r}_2, C_2^\flat) \in \gamma(E_2^\sharp) \\ (\bar{w}_2, G_2^\flat) = \varrho_{\bar{r}_1}\big((\bar{r}_2, C_2^\flat)\big) \\ \mathfrak{T} \models C_1^\flat \otimes G_2^\flat \rightarrow \bar{t} = \bar{s}_1'\big[\bar{r}_1/vseq(\bar{s}_1')\big] :: \bar{q}_2'\big[\bar{w}_2/vseq(\bar{q}_2')\big] \end{array} \right\}. \tag{5}$$

Suppose that, for some $\bar{t}_1 \in \mathcal{T}_{vars(\bar{r}_1)}, \bar{u}_2 \in \mathcal{T}_{vars(\bar{w}_2)}$,

$$\begin{aligned} \mathfrak{T} &\models C_1^\flat \rightarrow \bar{t}_1 = \bar{s}_1'\big[\bar{r}_1/vseq(\bar{s}_1')\big] \\ \mathfrak{T} &\models G_2^\flat \rightarrow \bar{u}_2 = \bar{q}_2'\big[\bar{w}_2/vseq(\bar{q}_2')\big]. \end{aligned}$$

We note that, by the definition of the renaming function ϱ,

$$\bar{q}_2'\big[\bar{w}_2/vseq(\bar{q}_2')\big] = \bar{s}_2'\big[\bar{w}_2/vseq(\bar{s}_2')\big]$$

so that

$$\mathfrak{T} \models G_2^\flat \rightarrow \bar{u}_2 = \bar{s}_2'\big[\bar{w}_2/vseq(\bar{s}_2')\big].$$

It therefore follows from the definition of $(\bar{w}_2, G_2^\flat)$ that as $vars(\bar{u}_2) = vars(\bar{w}_2)$ and $vars(\bar{t}_1) = vars(\bar{r}_1)$, there exists $\bar{t}_2 \in \mathcal{T}_{vars(\bar{r}_2)}$ such that

$$(\bar{u}_2, G_2^\flat) = \varrho_{\bar{t}_1}\big((\bar{t}_2, C_2^\flat)\big)$$

and

$$\mathfrak{T} \models C_2^\flat \rightarrow \bar{t}_2 = \bar{s}_2'\big[\bar{r}_2/vseq(\bar{s}_2')\big].$$

Then, by Definition 3, the expression (5) is equal to

$$\left\{ \eta\big((\bar{t}, C_1^\flat \otimes G_2^\flat)\big) \;\middle|\; \begin{array}{l} (\bar{t}_1, C_1^\flat) \in \gamma_p\big((\bar{s}_1, E_1^\sharp)\big) \\ (\bar{t}_2, C_2^\flat) \in \gamma_p\big((\bar{s}_2, E_2^\sharp)\big) \\ (\bar{u}_2, G_2^\flat) = \varrho_{\bar{t}_1}\big((\bar{t}_2, C_2^\flat)\big) \\ \mathfrak{T} \models C_1^\flat \otimes G_2^\flat \rightarrow \bar{t} = \bar{t}_1 :: \bar{u}_2 \end{array} \right\}.$$

□

Parameter Passing Concrete parameter passing is realized by an extended unification procedure. Unification is *extended* because it must involve the constraint solver(s). Remember that our notion of "constraint solver" includes also the interface between the Herbrand engine and the proper solver [16]. The interface needs to be notified about all the bindings performed by the Herbrand engine in order to maintain consistency between the solver and the Herbrand part. We also assume that CLP programs are normalized in such a way that interpreted function symbols only occur in explicit constraints. Note that this is not a restriction, since this kind of normalization is either required by the language syntax itself (for instance, this is the case of the clp(Q, R) libraries of SICStus Prolog) or is performed automatically by the CLP system (see, e.g., [19] for a description of how normalization can be achieved).

At the abstract level we do not prescribe the use of any particular algorithm. This is to keep our approach as general as possible. For instance, an implementor is not forced to use any particular representation for term-tuples (as in [10]). Similarly, one can choose any sound unification procedure that works well with the selected representation. Of particular interest is the possibility of choosing a representation and procedure that closely match the ones employed in the concrete language being analyzed. In the current version of the China analyzer we adopted a representation of terms similar to the one used in the Warren's Abstract Machine and its variants [1] and a unification procedure derived from [14]. In this case, all the easy steps typical of any unification procedure (functor name/arity checks, peeling, and so on) are handled, at the abstract level, exactly as they are at the concrete level.[3] The only crucial operation in abstract parameter passing over Pattern($\mathcal{D}^\sharp$) is the binding of an abstract variable to an abstract term. This is performed by first applying a non-cyclic approximation of the binding to the pattern component and then notifying the original (possibly cyclic) binding to the abstract constraint component. The correctness of this approach, proved below, assumes the existence of a bind operator on the underlying abstract constraint system satisfying the following condition.

Definition 5. (bind) *Let $E^\sharp \in \mathcal{D}^\sharp$ be a description such that $\gamma(E^\sharp) \subseteq \mathbf{T}_{\mathbf{Z}}^m \times \mathcal{C}^\flat$. Let $\bar{Z} = (Z_1, \ldots, Z_m)$, $u \in \mathcal{T}_{\bar{Z}}$, $vseq(u) = (Z_{j_1}, \ldots, Z_{j_l})$ and let $1 \leq h \leq m$. Then, define*

$$\begin{aligned}(k_1, \ldots, k_{m_1}) \stackrel{\mathrm{def}}{=} \Bigl((1, \ldots, h-1) \\ :: \bigl((j_1, \ldots, j_l) \setminus \{1, \ldots, h-1\}\bigr) \\ :: \bigl((h+1, \ldots, m) \setminus \{j_1, \ldots, j_l\}\bigr)\Bigr).\end{aligned}$$

[3] Thus greatly reducing the proof obligations. For instance, termination of the abstract unification procedure will be a consequence of termination of the concrete one.

If $E_1^\sharp \overset{\text{def}}{=} \operatorname{bind}(E^\sharp, u, Z_h)$*, then,*

$$\gamma(E_1^\sharp) \supseteq \left\{ \eta\bigl((\bar{q}\theta, C_1^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{p}, C^\flat) \in \gamma(E^\sharp) \\ \bar{p} = (p_1, \ldots, p_m) \\ \bar{q} = (p_{k_1}, \ldots, p_{k_{m_1}}) \\ \theta \textit{ is an idempotent substitution} \\ FV(C_1^\flat) \subseteq \mathit{vars}(\bar{q}\theta) \\ \mathfrak{T} \models \theta \leftarrow (Z_h' = u')[\bar{p}/\bar{Z}'] \\ \mathfrak{T} \models C_1^\flat \leftrightarrow \bigl((Z_h' = u')[\bar{p}/\bar{Z}'] \wedge C^\flat\bigr)\theta \end{array} \right\}.$$

Note that $m_1 = m - 1$ *if* $Z_h \notin \mathit{vars}(u)$*, and* $m_1 = m$*, otherwise.*

Suppose we take any sound unification algorithm and modify it as follows:

1. add a new parameter consisting of a description in $\mathcal{D}^\sharp$;
2. replace the step that performs the binding $Z_h \mapsto u$, by one that performs the binding $Z_h \mapsto u[Z_{\text{new}}/Z_h]$ (where Z_{new} is a fresh variable that avoids the creation of cyclic terms in the pattern component), and updates the description $E^\sharp$ to $\operatorname{bind}(E^\sharp, u, Z_h)$;
3. add an η-normalization step so that the final result is still a description in $\operatorname{Pattern}(\mathcal{D}^\sharp)$.

With the above definition for bind the only difficult part of the proof of correctness of the abstract (modified) unification algorithm with respect to the concrete (original) algorithm is a corollary of the following result.

Theorem 2. *Let* $(\bar{s}, E^\sharp) \in \operatorname{Pattern}(\mathcal{D}^\sharp)$, $(\bar{p}, C^\flat) \in \gamma(E^\sharp)$, $\bar{Z} = (Z_1, \ldots, Z_m) = \mathit{vseq}(\bar{s})$, $Z_h \in \bar{Z}$, $u \in \mathcal{T}_{\bar{Z}}$, $\mathit{vseq}(u) = (Z_{j_1}, \ldots, Z_{j_l})$*, and*

$$\begin{aligned}(k_1, \ldots, k_{m_1}) = \Bigl((1, \ldots, h-1) \\ :: \bigl((j_1, \ldots, j_l) \setminus \{1, \ldots, h-1\}\bigr) \\ :: \bigl((h+1, \ldots, m) \setminus \{j_1, \ldots, j_l\}\bigr)\Bigr).\end{aligned}$$

We define

$$\begin{aligned}\bar{p} &\overset{\text{def}}{=} (p_1, \ldots, p_m), \\ \bar{q} &\overset{\text{def}}{=} (p_{k_1}, \ldots, p_{k_{m_1}}).\end{aligned}$$

Suppose that, for some idempotent substitution θ *with variables in* $\mathbf{z}$ *and constraint* $C_1^\flat \in \mathcal{C}^\flat$ *such that* $FV(C_1^\flat) \subseteq \mathit{vars}(\bar{q}\theta)$,

$$\begin{aligned}E_1^\sharp &= \operatorname{bind}(E^\sharp, Z_h, u) \\ \mathfrak{T} &\models \theta \leftarrow (Z_h' = u')[\bar{p}/\bar{Z}'] \\ \mathfrak{T} &\models C_1^\flat \leftrightarrow \bigl((Z_h' = u')[\bar{p}/\bar{Z}'] \wedge C^\flat\bigr)\theta \\ w &= u[Z_{m+1}/Z_h]\end{aligned}$$

Then $\big(\eta\big(\bar{s}[w/Z_h]\big), E_1^\sharp\big) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ *and*

$$\eta\Big(\big(\bar{s}'[\bar{p}\theta/\bar{Z}'], C_1^\flat\big)\Big) \in \gamma_p\Big(\big(\eta\big(\bar{s}[w/Z_h]\big), E_1^\sharp\big)\Big).$$

Proof. The following terminology is used in the proof. A substitution $[\bar{W}/\bar{V}]$ is called a *renaming substitution in* $\mathbf{z}$ *for* $\bar{V}$ if $\bar{W}$ is a sequence of distinct variables in $\mathbf{z}$ (though not necessarily disjoint from $\bar{V}$).

Note that, since $Z_{m+1} \notin \mathit{vars}(\bar{s}) \cup \mathit{vars}(u)$, w is a variant of u, and, hence, $\bar{s}[w/Z_h]$ is a variant of $\bar{s}[u/Z_h]$ so that $\eta\big(\bar{s}[w/Z_h]\big) = \eta\big(\bar{s}[u/Z_h]\big)$. Therefore, we just need to show that

$$\Big(\eta\big(\bar{s}[u/Z_h]\big), E_1^\sharp\Big) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$$

and

$$\eta\Big(\big(\bar{s}'[\bar{p}\theta/\bar{Z}'], C_1^\flat\big)\Big) \in \gamma_p\Big(\big(\eta\big(\bar{s}[u/Z_h]\big), E_1^\sharp\big)\Big).$$

Note that $m_1 = m - 1$ if $Z_h \notin \mathit{vars}(u)$, and $m_1 = m$, otherwise. Note also that $Z_h[\bar{p}/\bar{Z}] = p_h$.

Let ν be a renaming substitution in $\mathbf{z}$ for $\bar{s}[u/Z_h]$ such that

$$\bar{s}[u/Z_h]\nu = \eta\big(\bar{s}[u/Z_h]\big). \tag{6}$$

Then, using the definition of $(k_1, \dots, k_{m_1})$, we have

$$(Z_1, \dots, Z_{m_1}) = (Z_{k_1}, \dots, Z_{k_{m_1}})\nu.$$

However, by definition of $(k_1, \dots, k_{m_1})$,

$$(Z_{k_1}, \dots, Z_{k_{m_1}}) = \mathit{vseq}\big(\bar{s}[u/Z_h]\big) \tag{7}$$

so that

$$(Z_1, \dots, Z_{m_1}) = \mathit{vseq}\Big(\eta\big(\bar{s}[u/Z_h]\big)\Big). \tag{8}$$

Thus, we have the first of the required results that

$$\Big(\eta\big(\bar{s}[u/Z_h]\big), E_1^\sharp\Big) \in \mathrm{Pattern}(\mathcal{D}^\sharp).$$

Let μ be a renaming substitution in $\mathbf{z}$ for $\mathit{vars}(\bar{p}) \cup \mathit{vars}(\theta)$ such that

$$\bar{q}\theta\mu = \eta(\bar{q}\theta). \tag{9}$$

Then, by the hypothesis, Eq. 9, and Definition 5,

$$(\bar{q}\theta\mu, C_1^\flat\mu) = \eta\big((\bar{q}\theta, C_1^\flat)\big) \in \gamma(E_1^\sharp). \tag{10}$$

Also, as $vseq(\bar{s}') = \bar{Z}'$ and, by the hypothesis, $FV(C_1^\flat) \subseteq vars(\bar{p}\theta)$, we have

$$\eta\Big(\big(\bar{s}'[\bar{p}\theta/\bar{Z}']\mu, C_1^\flat\mu\big)\Big) = \eta\Big(\big(\bar{s}'[\bar{p}\theta/\bar{Z}'], C_1^\flat\big)\Big). \tag{11}$$

Observe that, by the hypothesis,

$$\begin{aligned}
&\mathfrak{T} \models C_1^\flat \rightarrow \big((Z'_h = u')[\bar{p}/\bar{Z}']\big)\theta,\\
&\mathfrak{T} \models C_1^\flat \rightarrow \big(p_h = u'[\bar{p}/\bar{Z}']\big)\theta,\\
&\mathfrak{T} \models C_1^\flat \rightarrow p_h\theta = u'[\bar{p}/\bar{Z}']\theta,
\end{aligned}$$

so that, as $\bar{Z}' \supseteq vars(u')$,

$$\mathfrak{T} \models C_1^\flat \rightarrow p_h\theta = u'[\bar{p}\theta/\bar{Z}']. \tag{12}$$

Now, since $[\bar{p}\theta/\bar{Z}']$ is idempotent, we have

$$\bar{s}'\big[u'[\bar{p}\theta/\bar{Z}']/Z'_h\big][\bar{p}\theta/\bar{Z}'] = \bar{s}'[u'/Z'_h][\bar{p}\theta/\bar{Z}'] \tag{13}$$

and also, as $Z'_h \mapsto p_h\theta$ is a binding in $[\bar{p}\theta/\bar{Z}']$,

$$\bar{s}'[\bar{p}\theta/\bar{Z}'] = \bar{s}'[p_h\theta/Z'_h][\bar{p}\theta/\bar{Z}']. \tag{14}$$

Hence,

$$\begin{aligned}
&\mathfrak{T} \models C_1^\flat \rightarrow \bar{s}'[\bar{p}\theta/\bar{Z}'] = \bar{s}'[p_h\theta/Z'_h][\bar{p}\theta/\bar{Z}']\\
&\qquad\qquad \text{[by (14)]}\\
&\mathfrak{T} \models C_1^\flat \rightarrow \bar{s}'[\bar{p}\theta/\bar{Z}'] = \bar{s}'\big[u'[\bar{p}\theta/\bar{Z}']/Z'_h\big][\bar{p}\theta/\bar{Z}']\\
&\qquad\qquad \text{[by (12)]}\\
&\mathfrak{T} \models C_1^\flat \rightarrow \bar{s}'[\bar{p}\theta/\bar{Z}'] = \bar{s}'[u'/Z'_h][\bar{p}\theta/\bar{Z}']\\
&\qquad\qquad \text{[by (13)]}\\
&\mathfrak{T} \models C_1^\flat\mu \rightarrow \bar{s}'[\bar{p}\theta/\bar{Z}']\mu = \big(\bar{s}[u/Z_h]\big)'[\bar{p}\theta/\bar{Z}']\mu \qquad (15)\\
&\qquad\qquad \text{[by applying } \mu\text{]}.
\end{aligned}$$

However,

$$\begin{aligned}
\big(\bar{s}[u/Z_h]\big)'\big[\bar{p}\theta/\bar{Z}'\big]\mu &= \big(\bar{s}[u/Z_h]\big)'\big[\bar{p}\theta\mu/\bar{Z}'\big]\\
&\quad \text{[as } \mu \text{ is a renaming for } vars(\bar{p}) \cup vars(\theta) \subseteq \mathbf{z}\text{]}\\
&= \big(\bar{s}[u/Z_h]\big)'\big[\bar{q}\theta\mu/(Z_{k_1}, \ldots, Z_{k_{m_1}})'\big]\\
&\quad \text{[by the definition of } \bar{q} \text{ and (7)]}\\
&= \big(\bar{s}[u/Z_h]\nu\big)'\big[\bar{q}\theta\mu/(Z_1, \ldots, Z_{m_1})'\big].\\
&\quad \text{[by definition of } \nu\text{]}
\end{aligned}$$

Therefore, by (15),

$$\mathfrak{T} \models C_1^\flat\mu \rightarrow \bar{s}'[\bar{p}\theta/\bar{Z}']\mu = \bigl(\bar{s}[u/Z_h]\nu\bigr)'\bigl[\bar{q}\theta\mu/(Z_1,\ldots,Z_{m_1})'\bigr]. \tag{16}$$

It therefore follows from (8), (10), and (16) that we can apply Definition 3 and hence obtain

$$\eta\Bigl(\bigl(\bar{s}'[\bar{p}\theta/\bar{Z}']\mu, C_1^\flat\mu\bigr)\Bigr) \in \gamma_p\Biggl(\Bigl(\eta\bigl(\bar{s}[u/Z_h]\nu\bigr), E_1^\sharp\Bigr)\Biggr).$$

Therefore, by (11) and (6), we have the required result

$$\eta\Bigl(\bigl(\bar{s}'[\bar{p}\theta/\bar{Z}'], C_1^\flat\bigr)\Bigr) \in \gamma_p\Biggl(\Bigl(\eta\bigl(\bar{s}[u/Z_h]\bigr), E_1^\sharp\Bigr)\Biggr).$$

□

Projection When all the goals in a clause body have been solved, projection is used to restrict the abstract description to the tuple of arguments of the clause's head. The projection operations on $\mathcal{D}^\flat$ consist simply in dropping a suffix of the term-tuple component, with the consequent projection on the underlying constraint domain.

Definition 6. ('$\mathrm{project}_k^\flat$') $\bigl\{\, \mathrm{project}_k^\flat \colon \mathcal{D}^\flat \rightarrow \mathcal{D}^\flat \bigm| k \in \mathbb{N} \,\bigr\}$ *is a family of operations such that, for each $k \in \mathbb{N}$ and each $(\bar{u}, C^\flat) \in \mathcal{D}^\flat$ with $|\bar{u}| \geq k$,*

$$\mathrm{project}_k^\flat\bigl((\bar{u}, C^\flat)\bigr) \stackrel{\mathrm{def}}{=} \bigl(\mathrm{prefix}_k(\bar{u}), \P_\Delta^\flat\, C^\flat\bigr),$$

where $\Delta \stackrel{\mathrm{def}}{=} \mathit{vars}\bigl(\mathrm{prefix}_k(\bar{u})\bigr)$.

We now introduce the corresponding projection operations on $\mathrm{Pattern}(\mathcal{D}^\sharp)$ and, in order to establish their correctness, we impose a safety condition on the projection operations of $\mathcal{D}^\sharp$.

Definition 7. ($\P_k^\sharp$ and $\mathrm{project}_k^\sharp$) *Assume we are given a family of operations $\bigl\{\, \P_k^\sharp \colon \mathcal{D}^\sharp \rightarrow \mathcal{D}^\sharp \bigm| k \in \mathbb{N} \,\bigr\}$ such that, for each $E^\sharp \in \mathcal{D}^\sharp$ with $\gamma(E^\sharp) \subseteq \mathbf{T}_{\mathbf{z}}^m \times \mathcal{C}^\flat$ and each $k \leq m$,*

$$\gamma\bigl(\P_k^\sharp\, E^\sharp\bigr) \supseteq \Bigl\{\, \mathrm{project}_k^\flat\bigl((\bar{u}, C^\flat)\bigr) \Bigm| (\bar{u}, C^\flat) \in \gamma(E^\sharp) \,\Bigr\}.$$

Then, for each $(\bar{s}, E^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ such that $\bar{s} \in \mathbf{T}_{\mathbf{z}}^m$ and each $k \leq m$, we define

$$\mathrm{project}_k^\sharp\bigl((\bar{s}, E^\sharp)\bigr) \stackrel{\mathrm{def}}{=} \bigl(\mathrm{prefix}_k(\bar{s}), \P_j^\sharp\, E^\sharp\bigr),$$

where $j \stackrel{\mathrm{def}}{=} \bigl|\mathit{vars}\bigl(\mathrm{prefix}_k(\bar{s})\bigr)\bigr|$.

With these definitions 'project$^\sharp_k$' is correct with respect to 'project$^\flat_k$'.

Theorem 3. *For each* $(\bar{s}, E^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ *such that* $\bar{s} \in \mathbf{T}^m_{\mathbf{z}}$ *and each* $k \leq m$,

$$\gamma_p\Bigl(\mathrm{project}^\sharp_k\bigl((\bar{s}, E^\sharp)\bigr)\Bigr) \supseteq \Bigl\{\, \mathrm{project}^\flat_k\bigl((\bar{p}, C^\flat)\bigr) \Bigm| (\bar{p}, C^\flat) \in \gamma_p\bigl((\bar{s}, E^\sharp)\bigr) \,\Bigr\}.$$

Proof. In the following we let $\bar{s}_k \stackrel{\mathrm{def}}{=} \mathrm{prefix}_k(\bar{s})$, $j \stackrel{\mathrm{def}}{=} \bigl|\mathit{vars}(\bar{s}_k)\bigr|$, $\bar{r}_j \stackrel{\mathrm{def}}{=} \mathrm{prefix}_j(\bar{r})$. Then

$$\begin{aligned}
&\gamma_p\Bigl(\mathrm{project}^\sharp_k\bigl((\bar{s}, E^\sharp)\bigr)\Bigr)\\
&\qquad = \gamma_p\bigl((\bar{s}_k, \P^\sharp_j\, E^\sharp)\bigr)\\
&\qquad \text{[by Definition 7]}\\
&\qquad = \left\{\, \eta\bigl((\bar{q}, G^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{u}, G^\flat) \in \gamma(\P^\sharp_j\, E^\sharp) \\ \mathfrak{T} \models G^\flat \to \bar{q} = \bar{s}'_k\bigl[\bar{u}/\mathit{vseq}(\bar{s}'_k)\bigr] \end{array} \right\}\\
&\qquad \text{[by Definition 3]}\\
&\qquad \supseteq \left\{\, \eta\bigl((\bar{q}, G^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{u}, G^\flat) = \mathrm{project}^\flat_j\bigl((\bar{r}, C^\flat)\bigr) \\ (\bar{r}, C^\flat) \in \gamma(E^\sharp) \\ \mathfrak{T} \models G^\flat \to \bar{q} = \bar{s}'_k\bigl[\bar{u}/\mathit{vseq}(\bar{s}'_k)\bigr] \end{array} \right\}\\
&\qquad \text{[by Definition 7]}\\
&\qquad = \left\{\, \eta\bigl((\bar{q}, \P^\flat_{\mathit{vars}(\bar{r}_j)}\, C^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{r}, C^\flat) \in \gamma(E^\sharp) \\ \mathfrak{T} \models \P^\flat_{\mathit{vars}(\bar{r}_j)}\, C^\flat \to \bar{q} = \bar{s}'_k\bigl[\bar{r}_j/\mathit{vseq}(\bar{s}'_k)\bigr] \end{array} \right\}\\
&\qquad \text{[by Definition 6]}\\
&\qquad = \left\{\, \eta\bigl((\mathrm{prefix}_k(\bar{p}), \P^\flat_{\mathit{vars}(\bar{r}_j)}\, C^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{r}, C^\flat) \in \gamma(E^\sharp) \\ \mathfrak{T} \models C^\flat \to \bar{p} = \bar{s}'\bigl[\bar{r}/\mathit{vseq}(\bar{s}')\bigr] \end{array} \right\}\\
&\qquad \text{[by Definition 2]}\\
&\qquad = \Bigl\{\, \bigl(\mathrm{prefix}_k(\bar{p}), \P^\flat_{\mathit{vars}(\bar{r}_j)}\, C^\flat\bigr) \Bigm| (\bar{p}, C^\flat) \in \gamma_p\bigl((\bar{s}, E^\sharp)\bigr) \,\Bigr\}\\
&\qquad \text{[by Definition 3]}\\
&\qquad = \Bigl\{\, \mathrm{project}^\flat_k\bigl((\bar{p}, C^\flat)\bigr) \Bigm| (\bar{p}, C^\flat) \in \gamma_p\bigl((\bar{s}, E^\sharp)\bigr) \,\Bigr\}\\
&\qquad \text{[by Definition 6].}
\end{aligned}$$

□

Remapping The operation of *remapping* is used to adapt a description in $\mathrm{Pattern}(\mathcal{D}^\sharp)$ to a different, less precise, pattern component. Remapping is essential to the definition of various *join* and *widening* operators. Consider a description $(\bar{s}, E^\sharp_{\bar{s}}) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ and a pattern $\bar{r}' \in \mathbf{T}^*_{\mathbf{z}'}$ such that $\bar{r}'$ is an anti-instance of $\bar{s}$. We want to obtain $E^\sharp_{\bar{r}} \in \mathcal{D}^\sharp$ such that

$$\gamma_p\bigl((\bar{r}, E^\sharp_{\bar{r}})\bigr) \supseteq \gamma_p\bigl((\bar{s}, E^\sharp_{\bar{s}})\bigr). \tag{17}$$

This is what we call *remapping* $(\bar{s}, E^\sharp_{\bar{s}})$ *to* $\bar{r}'$.

Definition 8. **('remap')** *Let* $(\bar{s}, E^\sharp_{\bar{s}}) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ *be a description with* $\bar{s} \in \mathbf{T}^k_{\mathbf{z}}$ *and let* $\bar{r}' \in \mathbf{T}^k_{\mathbf{z}'}$ *be an anti-instance of* $\bar{s}$. *Assume also* $\left|vars(\bar{r}')\right| = m$ *and let* $\bar{u} \in \mathbf{T}^m_{\mathbf{z}}$ *be the unique tuple such that* $\bar{r}'\left[\bar{u}/vseq(\bar{r}')\right] = \bar{s}$. *Then the operation* $\mathrm{remap}(\bar{s}, E^\sharp_{\bar{s}}, \bar{r}')$ *yields* $E^\sharp_{\bar{r}}$ *such that* $\gamma(E^\sharp_{\bar{r}}) \supseteq \gamma_p\bigl((\bar{u}, E^\sharp_{\bar{s}})\bigr)$.

Observe that the remap function is closely related to the residual abstraction function α' of Figure 1.[4] With this definition, the specification of 'remap' meets our original requirement given by (17).

Theorem 4. *Let* $(\bar{s}, E^\sharp_{\bar{s}})$ *be a description with* $\bar{s} \in \mathbf{T}^k_{\mathbf{z}}$. *Let also* $\bar{r}' \in \mathbf{T}^k_{\mathbf{z}'}$ *be an anti-instance of* $\bar{s}$. *If* $E^\sharp_{\bar{r}} = \mathrm{remap}(\bar{s}, E^\sharp_{\bar{s}}, \bar{r}')$ *then* $\gamma_p\bigl((\bar{r}, E^\sharp_{\bar{r}})\bigr) \supseteq \gamma_p\bigl((\bar{s}, E^\sharp_{\bar{s}})\bigr)$.

Proof. Assume $\left|vars(\bar{r}')\right| = m$ and let $\bar{u} \in \mathbf{T}^m_{\mathbf{z}}$ be the unique tuple such that

$$\bar{r}'\left[\bar{u}/vseq(\bar{r}')\right] = \bar{s}. \tag{18}$$

Then,

$$\begin{aligned}
\gamma_p\bigl((\bar{r}, E^\sharp_{\bar{r}})\bigr) &= \left\{ \eta\bigl((\bar{q}, C^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{w}, C^\flat) \in \gamma(E^\sharp_{\bar{r}}) \\ \mathfrak{T} \models C^\flat \to \bar{q} = \bar{r}'\left[\bar{w}/vseq(\bar{r}')\right] \end{array} \right\} \\
&\quad \text{[by Definition 3]} \\
&\supseteq \left\{ \eta\bigl((\bar{q}, C^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{w}, C^\flat) \in \gamma_p\bigl((\bar{u}, E^\sharp_{\bar{s}})\bigr) \\ \mathfrak{T} \models C^\flat \to \bar{q} = \bar{r}'\left[\bar{w}/vseq(\bar{r}')\right] \end{array} \right\} \\
&\quad \bigl[\text{since, by Definition 8, we have } \gamma(E^\sharp_{\bar{r}}) \supseteq \gamma_p\bigl((\bar{u}, E^\sharp_{\bar{s}})\bigr)\bigr] \\
&= \left\{ \eta\bigl((\bar{q}, C^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{t}, C^\flat) \in \gamma(E^\sharp_{\bar{s}}) \\ \mathfrak{T} \models C^\flat \to \bar{q} = \Bigl(\bar{r}'\left[\bar{u}/vseq(\bar{r}')\right]\Bigr)'\left[\bar{t}/vseq(\bar{u}')\right] \end{array} \right\} \\
&\quad \text{[by Definition 3]} \\
&= \left\{ \eta\bigl((\bar{q}, C^\flat)\bigr) \;\middle|\; \begin{array}{l} (\bar{t}, C^\flat) \in \gamma(E^\sharp_{\bar{s}}) \\ \mathfrak{T} \models C^\flat \to \bar{q} = \bar{s}'\left[\bar{t}/vseq(\bar{s}')\right] \end{array} \right\} \\
&\quad \text{[by (18), since } \bar{s} = \bar{r}'\left[\bar{u}/vseq(\bar{r}')\right] \text{ and hence, } vseq(\bar{u}) = vseq(\bar{s})\text{]} \\
&= \gamma_p\bigl((\bar{s}, E^\sharp_{\bar{s}})\bigr) \\
&\quad \text{[by Definition 3].}
\end{aligned}$$

□

[4] Indeed, one can define $\alpha' \stackrel{\mathrm{def}}{=} \lambda(\bar{s}, E^\sharp) \in \mathbf{T}^k_{\mathbf{z}} \times \mathcal{D}^\sharp \,.\, \mathrm{remap}\bigl(\bar{s}, E^\sharp, (Z'_1, \ldots, Z'_k)\bigr)$.

Upper Bound Operators A concrete (collecting) semantics for CLP will typically use set union to gather results coming from different computation paths. We assume that our base domain $\mathcal{D}^\sharp$ captures this operation by means of an upper bound operator '$\oplus$'.

Definition 9. *A partial function $\oplus \colon \mathcal{D}^\sharp \times \mathcal{D}^\sharp \rightarrowtail \mathcal{D}^\sharp$ is an* upper bound operator *over $\mathcal{D}^\sharp$ if and only if, for each $E_1^\sharp, E_2^\sharp \in \mathcal{D}^\sharp$ such that $E_1^\sharp \oplus E_2^\sharp$ is defined, we have $E_1^\sharp \preceq E_1^\sharp \oplus E_2^\sharp$ and $E_2^\sharp \preceq E_1^\sharp \oplus E_2^\sharp$.*

The operation of merging two descriptions in $\mathrm{Pattern}(\mathcal{D}^\sharp)$ is defined in terms of 'remap'. Let $(\bar{s}_1, E_1^\sharp)$ and $(\bar{s}_2, E_2^\sharp)$ be two descriptions with $\bar{s}_1, \bar{s}_2 \in \mathbf{T}_{\mathbf{z}}^k$. The resulting description is $\bigl(\bar{r}, E_1^\sharp \oplus E_2^\sharp\bigr)$, where $\bar{r}' \in \mathbf{T}_{\mathbf{z}'}^k$ is an anti-instance of both $\bar{s}_1$ and $\bar{s}_2$, and $E_i^\sharp = \mathrm{remap}(\bar{s}_i, E_i^\sharp, \bar{r}')$, for $i = 1, 2$. We note again that $\bar{r}'$ might be the least common anti-instance of $\bar{s}_1$ and $\bar{s}_2$, or it can be a further approximation of $\mathrm{lca}(\bar{s}_1, \bar{s}_2)$: this is one of the degrees of freedom of the framework. Thus, the family of operations we are about to present is parameterized with respect to a common anti-instance function and the analyzer may dynamically choose which anti-instance function is used at each step.

Definition 10. ('join_ϕ') *Let ϕ be any common anti-instance function and '$\oplus$' an upper bound operator over $\mathcal{D}^\sharp$. The operation (partial function)*

$$\mathrm{join}_\phi \colon \wp_\mathrm{f}\bigl(\mathrm{Pattern}(\mathcal{D}^\sharp)\bigr) \rightarrowtail \mathrm{Pattern}(\mathcal{D}^\sharp)$$

is defined as follows. For each $k, I \in \mathbb{N}$ and each finite family

$$F \stackrel{\mathrm{def}}{=} \bigl\{ (\bar{s}_i, E_i^\sharp) \bigm| i \in I, \bar{s}_i \in \mathbf{T}_{\mathbf{z}}^k \bigr\}$$

of elements of $\mathrm{Pattern}(\mathcal{D}^\sharp)$*, if $\bigoplus_{i \in I} \mathrm{remap}(\bar{s}_i, E_i^\sharp, \bar{r}')$ is defined, then*

$$\mathrm{join}_\phi(F) \stackrel{\mathrm{def}}{=} (\bar{r}, E^\sharp),$$

where

$$\begin{aligned} \bar{r}' &\stackrel{\mathrm{def}}{=} \phi\bigl(\{\, \bar{s}_i \mid i \in I \,\}\bigr), \\ E^\sharp &\stackrel{\mathrm{def}}{=} \bigoplus_{i \in I} \mathrm{remap}(\bar{s}_i, E_i^\sharp, \bar{r}'). \end{aligned}$$

If ϕ is any common anti-instance function then 'join_ϕ' is an upper bound operator on the domain $\mathrm{Pattern}(\mathcal{D}^\sharp)$.

Theorem 5. *For each $k \in \mathbb{N}$, let $F \stackrel{\mathrm{def}}{=} \bigl\{(\bar{s}_i, E_i^\sharp)\bigr\}_{i \in I}$ be a finite family of elements of* $\mathrm{Pattern}(\mathcal{D}^\sharp)$ *such that $\bar{s}_i \in \mathbf{T}_{\mathbf{z}}^k$, for each $i \in I$. For each common anti-instance function ϕ and each $j \in I$ we have $\gamma_p\bigl(\mathrm{join}_\phi(F)\bigr) \supseteq \gamma_p\bigl((\bar{s}_j, E_j^\sharp)\bigr)$.*

Proof. Let $j \in I$, $\bar{r}' \stackrel{\mathrm{def}}{=} \phi\bigl(\{\bar{s}_i\}_{i\in I}\bigr)$ and $E^\sharp \stackrel{\mathrm{def}}{=} \bigoplus_{i \in I} \mathrm{remap}(\bar{s}_i, E_i^\sharp, \bar{r}')$. Then $\bar{r}'$ is an anti-instance of $\bar{s}_j$ and, by Definition 9, $E_j^\sharp \preceq E^\sharp$. Thus

$$\begin{aligned}
\gamma_p\bigl(\mathrm{join}_\phi(F)\bigr) &\supseteq \gamma_p\bigl((\bar{r}, E^\sharp)\bigr) \\
&\quad \text{[by Definition 10]} \\
&\supseteq \gamma_p\Bigl(\bigl(\bar{r}, \mathrm{remap}(\bar{s}_j, E_j^\sharp, \bar{r}')\bigr)\Bigr) \\
&\quad \text{[by Proposition 1]} \\
&\supseteq \gamma_p\bigl((\bar{s}_j, E_j^\sharp)\bigr) \\
&\quad \text{[by Theorem 4]}
\end{aligned}$$

$\square$

Widenings It is possible to devise a (completely unnatural) abstract domain $\mathcal{D}^\sharp$ that enjoys the *ascending chain condition*[5] still preventing $\mathrm{Pattern}(\mathcal{D}^\sharp)$ from possessing the same property. This despite the fact that any element of $\mathbf{T}_{\mathbf{z}}^n$ has a finite number of distinct anti-instances in $\mathbf{T}_{\mathbf{z}}^n$. However, this problem is of no practical interest if the analysis applies 'join_ϕ' at each step of the iteration sequence. In this case, if we denote by $(\bar{s}_j, E_j^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ the description at step $j \in \mathbb{N}$, we have $(\bar{s}_{i+1}, E_{i+1}^\sharp) = \mathrm{join}_\phi\bigl(\bigl\{(\bar{s}_i, E_i^\sharp), \dots\bigr\}\bigr)$, assuming no widening is employed. This implies that $\bar{s}'_{i+1}$ is an anti-instance of $\bar{s}_i$. As any ascending chain in $\mathbf{T}_{\mathbf{z}}^n$ is finite, the iteration sequence will eventually stabilize if $\mathcal{D}^\sharp$ enjoys the ascending chain condition.

In some cases, however, rapid termination of the analysis on $\mathcal{D}^\sharp$ can only be ensured by using one or more widening operators $\nabla \colon \mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ [12]. These can be lifted to work on $\mathrm{Pattern}(\mathcal{D}^\sharp)$. As an example, we show the default lifting used by the China analyzer:

$$\mathrm{widen}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr) \stackrel{\mathrm{def}}{=} \begin{cases} (\bar{s}_2, E_2^\sharp), & \text{if } \bar{s}_1 \neq \bar{s}_2; \\ (\bar{s}_2, E_1^\sharp \nabla E_2^\sharp), & \text{if } \bar{s}_1 = \bar{s}_2. \end{cases} \tag{19}$$

This operator refrains from widening unless the pattern component has stabilized. A more drastic choice for a widening is given by

$$\mathrm{Widen}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr) \stackrel{\mathrm{def}}{=} \bigl(\bar{s}_2, \mathrm{remap}(\bar{s}_1, E_1^\sharp, \bar{s}'_2) \nabla E_2^\sharp\bigr). \tag{20}$$

Widenings only need to be evaluated over $(\bar{s}_1, E_1^\sharp)$ and $(\bar{s}_2, E_2^\sharp)$ when $\bar{s}'_2$ is an anti-instance of $\bar{s}_1$. Thus, as $\mathbf{T}_{\mathbf{z}}^n$ satisfies the ascending chain condition, 'widen' and 'Widen' are well-defined widening operators on $\mathrm{Pattern}(\mathcal{D}^\sharp)$.

Theorem 6. *For $i = 1, 2$, suppose $(\bar{s}_i, E_i^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ where $\bar{s}'_2$ is an anti-instance of $\bar{s}_1$ and $\gamma_p\bigl((\bar{s}_1, E_1^\sharp)\bigr) \subseteq \gamma_p\bigl((\bar{s}_2, E_2^\sharp)\bigr)$ holds. Then*

[5] Namely, each strictly increasing chain is finite.

1. $\gamma_p\bigl((\bar{s}_2, E_2^\sharp)\bigr) \subseteq \gamma_p\Bigl(\mathrm{widen}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr)\Bigr)$;
2. $\gamma_p\bigl((\bar{s}_2, E_2^\sharp)\bigr) \subseteq \gamma_p\Bigl(\mathrm{Widen}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr)\Bigr)$.

Proof. (1) Suppose $(\bar{s}, E^\sharp) = \mathrm{widen}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr)$. By the definition of widen, if $\bar{s}_1 \neq \bar{s}_2$, then $(\bar{s}, E^\sharp) = (\bar{s}_2, E_2^\sharp)$ and there is nothing left to prove. If $\bar{s}_1 = \bar{s}_2$, then $(\bar{s}, E^\sharp) = (\bar{s}_2, E_1^\sharp \nabla E_2^\sharp)$. As ∇ is a widening on the $\mathcal{D}^\sharp$ domain, we have $E_2^\sharp \preceq E_1^\sharp \nabla E_2^\sharp$. By γ monotonicity, this implies $\gamma(E_2^\sharp) \subseteq \gamma(E_1^\sharp \nabla E_2^\sharp)$ and the result follows from Proposition 1.

(2) Suppose $(\bar{s}, E^\sharp) = \mathrm{Widen}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr)$, so that we have $\bar{s} = \bar{s}_2$ and

$$E^\sharp = \mathrm{remap}(\bar{s}_1, E_1^\sharp, \bar{s}_2') \nabla E_2^\sharp.$$

As ∇ is a widening on the $\mathcal{D}^\sharp$ domain, we have $E_2^\sharp \preceq E^\sharp$. Again, the result follows from γ monotonicity and Proposition 1. □

Besides ensuring termination, widening operators are also used to accelerate convergence of the analysis. It is therefore important to be able to define widening operators on $\mathrm{Pattern}(\mathcal{D}^\sharp)$ without relying on the existence of corresponding widenings on $\mathcal{D}^\sharp$. There are many possibilities in this direction and some of them are currently under experimental evaluation. Just note that any upper bound operator 'join_ϕ' can be regarded as a widening as soon as the common anti-instance function ϕ is different from lca. In order to ensure the convergence of the abstract computation, we will only consider widening operators on $\mathrm{Pattern}(\mathcal{D}^\sharp)$ satisfying the following (very reasonable) condition: if $(\bar{s}, E^\sharp)$ is the result of the widening applied to $(\bar{s}_1, E_1^\sharp)$ and $(\bar{s}_2, E_2^\sharp)$, where $\bar{s}_2'$ is an anti-instance of $\bar{s}_1$, then $\bar{s}'$ is an anti-instance of $\bar{s}_2$. Both widen and Widen comply with this restriction.

Comparing Descriptions The *comparison* operation on $\mathrm{Pattern}(\mathcal{D}^\sharp)$ is used by the analyzer in order to check whether a local fixpoint has been reached.

Definition 11. **('compare')** *Let '$\precsim$' $\subseteq \mathcal{D}^\sharp \times \mathcal{D}^\sharp$ be a computable preorder that correctly approximates '$\preceq$', that is, for each $E_1^\sharp, E_2^\sharp \in \mathcal{D}^\sharp$, we have $E_1^\sharp \preceq E_2^\sharp$ whenever $E_1^\sharp \precsim E_2^\sharp$. The* approximated ordering relation over $\mathrm{Pattern}(\mathcal{D}^\sharp)$, *denoted by* 'compare' $\subseteq \mathrm{Pattern}(\mathcal{D}^\sharp) \times \mathrm{Pattern}(\mathcal{D}^\sharp)$, *is defined, for each* $(\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$, *by*

$$\mathrm{compare}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr) \quad \overset{\mathrm{def}}{\iff} \quad \bigl(\bar{s}_1 = \bar{s}_2 \wedge E_1^\sharp \precsim E_2^\sharp\bigr).$$

It must be stressed that the above ordering is "approximate" since it does not take into account the peculiarities of $\mathcal{D}^\sharp$.[6] More refined orderings can be obtained in a domain-dependent way, namely, when $\mathcal{D}^\sharp$ has been fixed. It is easy to show that compare is a preorder over $\mathrm{Pattern}(\mathcal{D}^\sharp)$ that correctly approximates the approximation ordering '$\preceq_p$'

The following is a trivial consequence of Definition 2 and Proposition 1.

[6] It is also important not to confuse this *approximate* ordering with the *approximation* ordering of $\mathrm{Pattern}(\mathcal{D}^\sharp)$, denoted by '$\preceq_p$', given in Definition 3.

Theorem 7. *If* $\mathrm{compare}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr)$ *holds, then*

$$\gamma_p\bigl((\bar{s}_1, E_1^\sharp)\bigr) \subseteq \gamma_p\bigl((\bar{s}_2, E_2^\sharp)\bigr).$$

Moreover, compare *is a preorder over* $\mathrm{Pattern}(\mathcal{D}^\sharp)$*.*

Proof. If $\mathrm{compare}\bigl((\bar{s}_1, E_1^\sharp), (\bar{s}_2, E_2^\sharp)\bigr)$ holds, then, by Definition 11, $\bar{s}_1 = \bar{s}_2$ and $E_1^\sharp \precsim E_2^\sharp$. Hence, since '$\precsim$' is a correct approximation of '$\preceq$', we have $E_1^\sharp \preceq E_2^\sharp$ and, by Proposition 1, $\gamma_p\bigl((\bar{s}_1, E_1^\sharp)\bigr) \subseteq \gamma_p\bigl((\bar{s}_2, E_2^\sharp)\bigr)$. Furthermore, as $\precsim$ is defined as a preorder, compare is a preorder. $\square$

Observe that the ability of comparing descriptions only when they have the same pattern is not restrictive in our setting. The analyzer, in fact, will only need to compare the descriptions arising from the iteration sequence at two consecutive steps. By the definition of join_ϕ and the condition we imposed on widenings, if $(\bar{s}_i, E_i^\sharp)$ and $(\bar{s}_{i+1}, E_{i+1}^\sharp)$ are the descriptions at steps i and $i+1$, then $\bar{s}'_{i+1}$ is an anti-instance of $\bar{s}_i$. Moreover, since 'join_ϕ' and the widening operators are all upper-bound operators on $\mathrm{Pattern}(\mathcal{D}^\sharp)$, we have

$$\gamma_p\bigl((\bar{s}_i, E_i^\sharp)\bigr) \subseteq \gamma_p\bigl((\bar{s}_{i+1}, E_{i+1}^\sharp)\bigr). \tag{21}$$

If also the reverse inclusion holds in (21) then we have reached a local fixpoint. The analyzer uses the approximate ordering to check for this possibility. Namely, it asks whether

$$\mathrm{compare}\bigl((\bar{s}_{i+1}, E_{i+1}^\sharp), (\bar{s}_i, E_i^\sharp)\bigr)$$

holds. The approximate test, of course, can fail even when equality does hold in (21). But this will be a fault of the pattern component only a finite number of times, since $\bar{s}'_{i+1}$ is an anti-instance of $\bar{s}_i$ and $\mathbf{T}_{\mathbf{z}}^n$, ordered by the anti-instance relation, has finite height. Thus, there exists $\ell \in \mathbb{N}$ such that, for each $i \geq \ell$, $\bar{s}_i = \bar{s}_\ell$. After the ℓ-th step the accuracy of the approximate ordering is in the hands of $\mathcal{D}^\sharp$.

5 Experimental Evaluation

We have conducted an extensive experimentation on the analysis using the $\mathrm{Pattern}(\cdot)$ construction: this allowed to tune the implementation and to gain insight on the implications of keeping track of explicit structural information. In order to put ourselves in a realistic situation, we have assessed the impact of the $\mathrm{Pattern}(\cdot)$ construction on a very precise and complex domain for mode analysis that we call *Modes*.[7] The *Modes* domain captures information on simple types, groundness, boundedness, pair-sharing, freeness, and linearity. It is a combination of, among other things, two copies of the GER representation for *Pos* [5] —

[7] This is an important point: if one starts from an imprecise domain it is rather easy to show big precision improvements.

one for groundness, the other for boundedness — and the non-redundant pair-sharing domain *PSD* [4] with widenings as described in [24]. Some details on how these domain are combined can be found in [6]. All the domains used have been suitably extended in order to ensure correctness and precision also for the analysis of real systems omitting the occurs-check [2, 15].

The benchmark suite used for the development and tuning of the China analyzer is probably the largest one ever employed for this purpose. The suite comprises all the programs we have access to (i.e., everything we could find by systematically dredging the Internet): 286 programs, 15 MB of code, 500.000 lines, the largest program containing 10063 clauses in 45658 lines of code.

The comparison between *Modes* and Pattern(*Modes*) involves the two usual things: *precision* and *efficiency*. However, how are we going to compare the precision of the domain tracking explicit structural information with a domain ignoring it? That is something that should be established in advance. Let us consider a simple but not trivial Prolog program: `mastermind.pl`.[8] Consider also the only direct query for which it has been written, '`?- play.`', and focus the attention on the procedure `extend_code/1`. A standard goal-dependent analysis of the program with the *Modes* domain is only able to tell something like

```
extend_code(A) :-
  list(A).
```

This means: "during any execution of the program, whenever `extend_code/1` succeeds it will have its argument bound to a *list cell* (i.e., a term whose principal functor is either `'.'/2` or `[]/0`)". Not much indeed. Especially because this can be established instantly by visual inspection: `extend_code/1` is *always* called with a list argument and this completes the proof. If we perform the analysis with Pattern(*Modes*) the situation changes radically. Here is what such a domain allows China to derive:[9]

```
extend_code([([A|B],C,D)|E]) :-
  list(B),
  (functor(C,_,1);integer(C)),
  (functor(D,_,1);integer(D)),
  list(E),
  ground([C,D]),
  may_share([[A,B,E]]).
```

Under the circumstances mentioned above, this means that: "the argument of `extend_code/1` will be bound to a term of the form `[([A|B],C,D)|E]`, where

[8] A program implementing the game "Mastermind", rewritten by H. Koenig and T. Hoppe after code by M. H. van Emden. Also in H. Coelho and J. C. Cotta, "Prolog by Example", *Symbolic Computation*, Springer-Verlag, Berlin, 1988. Available at `http://www.cs.unipr.it/China/Benchmarks/Prolog/mastermind.pl`.

[9] Some extra groundness information obtained by the analysis has been omitted for the sake of simplicity: this says that, if `A` and `B` turn out to be ground, then `E` will also be ground.

`B` and `E` are bound to list cells; `C` is either bound to a functor of arity 1 or to an integer, and likewise for `D`; both `C` and `D` are ground, and (consequently) pair-sharing may only occur between `A`, `B`, and `E`".

It is clear that the analysis with Pattern($Modes$) yields much more information. The value of this extra precision can only be measured from the point of view of the target application of the analysis (e.g., optimized compilation, abstract debugging and verification, etc.). In other words, it is not clear at all how to define a fair measure for the precision gain independently from the intended application. The approach we have chosen is simple though unsatisfactory: throw away all the structural information at the end of the analysis and compare the usual numbers (i.e., number of ground variables, number of free variables and so on). With reference to the above example, this metric pretends that explicit structural information gives no precision improvements on the analysis of `extend_code/1` in `mastermind.pl`. In fact, once all the structural information has been discarded, the analysis with Pattern(*Modes*) only specifies that, upon success, the argument of `extend_code/1` will be a list cell. We are thus measuring how the explicit structural information present in Pattern(*Modes*) improves the precision on *Modes* itself, which is only a tiny part of the real gain in accuracy.

It is important to note that the experimental results we are about to report have been obtained without using any widening on the pattern component. The widening operations are only propagated to the underlying *Modes* domain by means of the 'widen' operator given in Eq. (19). Moreover, the merge operation employed is always '$\mathrm{join}_{\mathrm{lca}}$'. This is a deliberate choice: as we are currently tuning the implementations of the Pattern($\cdot$) construction and of the *Modes* domains (including its widenings), we feel that the inclusion of *ad-hoc* widenings for the pattern component should be postponed. One of the contributions of this paper is to show that explicit structural information analysis is feasible even without such widenings.

Here we only summarize the results of the experimentation. The interested reader can find all the relevant details at the URI `http://www.cs.unipr.it/China`. The precision comparison is performed by measuring five different quantities:

indep: the total number of *independent* argument pairs;
ground: the total number of *ground* argument positions;
linear: the total number of *linear* argument positions;
free: the total number of *free* argument positions;
bound: the total number of *bound* (or *nonvar*) argument positions.

Note that our results can be considered as a lower bound on the overall precision improvement. First of all, as already noted, we are completely disregarding the precision gains coming from structural information in itself. Moreover, we are not reporting some relational information computed by the analyses, like groundness dependencies and sharing dependencies; often considered as a mere by-product of the analysis, sometimes this information can be suitably exploited to improve precision, e.g., when performing modular analyses.

$x = $%inc.	indep		ground		linear		free		bound	
	GI	GD	GI	GD	GI	GD	GI	GD	GI	GD
$x < 0$	0	1	0	0	0	1	0	0	0	0
$x = 0$	212	201	219	215	204	197	237	235	219	211
$0 < x \leq 2$	33	31	20	21	40	38	22	18	46	41
$2 < x \leq 5$	20	25	15	16	18	18	9	10	10	16
$5 < x \leq 10$	7	9	9	9	6	8	9	9	6	6
$x \geq 10$	14	19	23	25	18	24	9	14	5	12

Table 1. A summary of the *Modes* precision gained using structural information.

The results are summarized by partitioning the benchmark suite into six classes of programs, identified by the per cent increase in precision due to the Pattern($\cdot$) construction. Table 1 gives the cardinalities of these classes for both goal-independent (GI) and goal-dependent (GD) analyses. A precision increase (in at least one of the measured quantities) is observed on more than one third of the benchmarks. The only precision decrease is due to the interaction between the Pattern($\cdot$) construction and the widenings used in the *Modes* domain. It is also worth observing that, on average, goal-dependent analysis is more likely to benefit from the addition of structural information.

Structural information has the potential of pruning some computation paths on the grounds that they cannot be followed by the program being analyzed. In some cases the analysis is able to prove that a certain predicate can never succeed and/or never be called. This phenomenon actually happens for a dozen of programs in our benchmark suite: the analysis with Pattern(*Modes*) declares some procedures *dead*, but the same procedures are fine for the analysis with *Modes*. This means that systematic failure and/or the impossibility of calling these procedures is not due to undefined predicates or explicit failures caused by `fail/0` and similar built-in predicates, but rather because of unification failures. At a closer examination, the problem seems to be caused by bugs in the involved benchmark programs.[10] We have allowed for this in the results in Figure 1 by normalizing the precision results for the Pattern(*Modes*) domain. That is, we added to the results for Pattern(*Modes*) the number of variables or pair sharings, as appropriate, that were lost due to the pruning, before calculating the percentages. This amounts to the logically sound view whereby a procedure that always fails will have, in the impossible case of success, all its variables independent, ground, linear, free, and bound *at the same time*.

In order to evaluate the impact on efficiency of the Pattern transformation we computed the fixpoint computation time for all the programs, both with the *Modes* and with the Pattern(*Modes*) domains. The benchmark suite is thus

[10] Indeed, China has proved very valuable as a debugging tool in several occasions, even if it has not yet been integrated into a suitable programming environment.

Time diff. in seconds	programs	
	GI	GD
degradation ≥ 1	10	20
degradation < 1	105	120
same time	98	75
improvement < 1	51	48
improvement ≥ 1	22	23

Table 2. A summary of the impact of structural information on analysis time.

partitioned into five classes based on the absolute differences observed[11] and Table 2 gives the cardinality of the classes, again distinguishing between GI and GD analyses. The numbers show that the full range of possible behaviors is indeed observable. In particular, quite surprisingly, it is not uncommon the case when the Pattern construction, even being inherently more precise and complex, does allow significant time improvements. This is only partly due to the enhanced ability of pruning failed computation paths. Most importantly, the description of a set of tuples of terms in Pattern(*Modes*) is often much more efficient than the corresponding description in *Modes*.

In order to control the computational cost even in the few badly-behaving cases, which turn out to be much less than the well-behaving ones, it is necessary to provide the Pattern($\cdot$) construction with one or more specific widening operators, whose definitions do not depend on a corresponding widening on the underlying abstract domain. As noted in the previous section, simple solutions are readily available and currently under experimental evaluation.

6 Related Work

In [7], an alternative technique is proposed for augmenting a data-flow analysis with structural information. Instead of upgrading the analysis domain with structural information, this technique relies on program transformations. In this approach, the data-flow analysis of a given program is performed in four steps:

1. A pure structural information analysis is performed on the original program so as to collect an approximation of the common patterns in its success-set.
2. This common information is used to produce a specialized version of the original program where the common structures identified in the first phase are exposed. Roughly speaking, if all the success patterns for a predicate `p/n` in the original program are instances of a given structure, the specialized program is such that all of its syntactic calls to `p/n` are instances of that

[11] As the benchmark suite comprises several real programs of very respectable size, we believe absolute time is what matters to assess the feasibility of the approach. The experiments were conducted on a PC equipped with an AMD Athlon clocked at 700 MHz, 256 MB of RAM, and running Linux 2.2.14. Note, once again, that no widening was employed on the Pattern($\cdot$) construction.

structure. If the original program is a *pure logic program*, then the specialization phase presented in [7] preserves its success-patterns but not necessarily its call-patterns and termination behavior.[12]

3. The specialized program is then further transformed by *untupling*. The untupling transformation produces a new program where the sub-structures of common structures are represented by different argument positions. For example, if the specialized program computed in step 2 is such that all the calls to `p/1` are instances of $p(f(X, Y, h(Z)))$, the untupling transformation will introduce a new predicate `p'/3`, so that an instance $p(f(t_1, t_2, h(t_3)))$ in the specialized program becomes $p'(t_1, t_2, t_3)$ in the untupled program. The untupling transformation also adds a new clause for each predicate in the specialized program in order to maintain the connection between the specialized program itself and its untupled version. Following our example, the untupled version of the program contains the clause

   ```
   p(f(X, Y, h(Z))) :- p_prime(X, Y, Z).
   ```

   In [7] the untupling transformation is reported to preserve both success-patterns (of the predicates in the specialized program) and termination behavior.
4. Analysis is finally performed on the untupled program with the abstract domain of choice.

This new analysis technique is, as pointed out in [7], less precise, in general, than the one of $\mathtt{Pat}(\Re)$ or Pattern$(\cdot)$. This happens because the transformational approach may miss pruning information and thus lead to the analysis of computation branches that cannot lead to success (this phenomenon also has an impact on the cost of the analysis). The technique is advocated in [7] for its simplicity and efficiency.

As far as efficiency is concerned, [7] assumes that structural information analysis with $\mathtt{Pat}(\Re)$ (and, by extension, with Pattern$(\mathcal{D}^\sharp)$) is inherently inefficient. This claim is supported by the comparison of their performance evaluation to the one reported in [9] for the domain $\mathtt{Pat}(\mathit{Pos})$. However, the pioneering work described in [9] refers to implementations of *Pos* and of the generic structural domain that are no longer representative of the current state-of-the-art. Indeed, the experimental results we obtain for the Pattern$(\cdot)$ construction show that in most cases the slow-down is very limited and that even consistent *speed-ups* (up to a factor 5) can be achieved on the analysis of real programs. Moreover, in [7] the comparison with $\mathtt{Pat}(\mathit{Pos})$ is only conducted on the 11 (mainly small) programs evaluated in [9]. Our experience tells us that such a small benchmark suite is far too small to warrant any generalization of the results. It also seems that, for the comparison, in [7] only the cost of the analysis of the untupled program is computed. The computational costs of the other phases of the proposed

[12] This property does not carry through to, say, Prolog programs, as it can be easily established by considering a program constituted by only two clauses: `p(X) :- q(X)` and `q(X) :- var(X), X = a`.

method (pattern analysis, specialization and untupling) are disregarded, even though specialization and untupling are said to have linear complexity in the size of the program.

As far as simplicity is concerned, one advantage of the approach described in [7] is that one can reuse existing data-flow analyzers without having to develop a generic structural domain. Therefore, despite the need to implement the transformers used to obtain the specialized and the untupled versions of the original program, this new proposal may be simpler to implement. However, an implementer willing to apply the `Pat(`$\Re$`)` or the Pattern($\cdot$) constructions to a different abstract domain does not need to start *from scratch*. Since both these constructions are *generic*, they can be (and have been) implemented only once and for all. To be more precise, the Pattern($\cdot$) construction is implemented as a C++ template. If one has a class `Base_Domain` implementing the domain $\mathcal{D}^\sharp$, an implementation of the Pattern($\mathcal{D}^\sharp$) domain is provided by the class `Pattern<Base_Domain>`.[13] So, while it is true that a good deal of work was invested in designing, implementing, testing and tuning the Pattern($\cdot$) construction, this work needs not to be repeated and its use as a template is one of the most simple things one can imagine.

To complete the comparison, we observe that the transformational approach may lack the wide applicability of the Pattern($\cdot$) construction. For instance, it is not clear from the definitions given in [7] if their technique can be extended so as to correctly deal with any implemented (constraint) logic programming system (which may omit the occurs-check and provide built-ins such as `var/1`). This ability is, in turn, one of the strongest points of our proposal.

7 Conclusion

We have presented the rational construction of a generic domain for structural analysis of real CLP languages: Pattern($\mathcal{D}^\sharp$), where the parameter $\mathcal{D}^\sharp$ is an abstract domain satisfying certain properties. We build on the parameterized `Pat(`$\Re$`)` domain of Cortesi et al. [9, 10], which is restricted to logic programs and requires the occurs-check to be performed. However, while `Pat(`$\Re$`)` is presented as a *specific implementation* of a generic structural domain, our formalization is implementation-independent. Reasoning at a higher level of abstraction we are able to appeal to familiar notions of unification theory, while leaving considerable more latitude for the implementer. Indeed our results show that, contrary to popular belief, an analyzer incorporating structural information analysis based on our approach can be highly competitive even from the efficiency point of view.

[13] The `Pattern` template has additional arguments, each with its own default value, that allow the user to select the parameters of the construction such as the widening functions.

References

1. H. Aït-Kaci. *Warren's Abstract Machine. A Tutorial Reconstruction.* The MIT Press, 1991.
2. R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages.* PhD thesis, Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56125 Pisa, Italy, March 1997. Printed as Report TD-1/97.
3. R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. *Science of Computer Programming*, 30(1–2):119–155, 1998.
4. R. Bagnara, P. M. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. *Theoretical Computer Science*, 2000. To appear.
5. R. Bagnara and P. Schachte. Factorizing equivalent variable pairs in ROBDD-based implementations of *Pos*. In A. M. Haeberer, editor, *Proceedings of the "Seventh International Conference on Algebraic Methodology and Software Technology (AMAST'98)"*, volume 1548 of *Lecture Notes in Computer Science*, pages 471–485, Amazonia, Brazil, 1999. Springer-Verlag, Berlin.
6. R. Bagnara, E. Zaffanella, and P. M. Hill. Enhanced sharing analysis techniques: A comprehensive evaluation. In M. Gabbrielli and F. Pfenning, editors, *Proceedings of the ACM SIGPLAN 2nd International Conference on Principles and Practice of Declarative Programming*, Lecture Notes in Computer Science, Montreal, Canada, 2000. Springer-Verlag, Berlin. To appear.
7. M. Codish, K. Marriott, and C. Taboch. Improving program analyses by structure untupling. *Journal of Logic Programming*, 43(3):251–263, 2000.
8. A. Colmerauer. Prolog and infinite trees. In K. L. Clark and S. Å. Tärnlund, editors, *Logic Programming, APIC Studies in Data Processing*, volume 16, pages 231–251. Academic Press, New York, 1982.
9. A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Conceptual and software support for abstract domain design: Generic structural domain and open product. Technical Report CS-93-13, Brown University, Providence, RI, 1993.
10. A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Combinations of abstract domains for logic programming: Open product and generic pattern construction. *Science of Computer Programming*, 2000. To appear.
11. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.
12. P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295, Leuven, Belgium, 1992. Springer-Verlag, Berlin.
13. R. Giacobazzi, S. K. Debray, and G. Levi. Generalized semantics and abstract interpretation for constraint logic programs. *Journal of Logic Programming*, 25(3):191–247, 1995.
14. S. Haridi and D. Sahlin. Efficient implementation of unification of cyclic structures. In J. A. Campbell, editor, *Implementations of Prolog*, pages 234–249. Ellis Horwood/Halsted Press/Wiley, 1984.
15. P. M. Hill, R. Bagnara, and E. Zaffanella. The correctness of set-sharing. In G. Levi, editor, *Static Analysis: Proceedings of the 5th International Symposium*, volume 1503 of *Lecture Notes in Computer Science*, pages 99–114, Pisa, Italy, 1998. Springer-Verlag, Berlin.

16. J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19&20:503–582, 1994.
17. J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.
18. J.-L. Lassez, M. J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988.
19. P. Lim and J. Schimpf. A conservative approach to meta-programming in constraint logic programming. In M. Bruynooghe and J. Penjam, editors, *Proceedings of the 5th International Symposium on Programming Language Implementation and Logic Programming*, volume 714 of *Lecture Notes in Computer Science*, pages 44–59, Tallinn, Estonia, 1993. Springer-Verlag, Berlin. Also available as Technical Report ECRC-94-31, ECRC 1994.
20. K. Marriott and H. Søndergaard. On describing success patterns of logic programs. Technical Report 12, The University of Melbourne, 1988.
21. K. Musumbu. *Interprétation Abstraite des Programmes Prolog.* PhD thesis, Facultés Universitaires Notre-Dame de la Paix – Namur Institut d'Informatique, Belgium, September 1990.
22. T. Sato and H. Tamaki. Enumeration of success patterns in logic programs. *Theoretical Computer Science*, 34:227–240, 1984.
23. G. Weyer and W. Winsborough. Annotated structure shape graphs for abstract analysis of Prolog. In H. Kuchen and S. D. Swierstra, editors, *Programming Languages: Implementations, Logics and Programs, Proceedings of the Eighth International Symposium*, volume 1140 of *Lecture Notes in Computer Science*, pages 92–106, Aachen, Germany, 1996. Springer-Verlag, Berlin.
24. E. Zaffanella, R. Bagnara, and P. M. Hill. Widening Sharing. In G. Nadathur, editor, *Principles and Practice of Declarative Programming*, volume 1702 of *Lecture Notes in Computer Science*, pages 414–431, Paris, France, 1999. Springer-Verlag, Berlin.