Structural Information Analysis for CLP Languages

Roberto Bagnara*
School of Computer Studies
University of Leeds
Leeds, LS2 9JT, United Kingdom
bagnara@scs.leeds.ac.uk

Abstract

We present the rational construction of a generic domain for structural analysis of CLP languages: 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 Prolog programs $\mathtt{Pat}(\Re)$, which is due to Cortesi *et al.* [6, 7]. However, the formalization of our CLP abstract domain is independent from specific implementation techniques: $\mathtt{Pat}(\Re)$ (suitably extended) is one of the possible implementations. 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, missing in [6]) that gives some control over the precision and computational cost of the resulting generic structural domain.

1 Introduction

It is important to make a clear distinction between the language CLP($\mathcal{H}$, $\mathcal{X}$), where the Herbrand component $\mathcal{H}$ is completely separated from the domain $\mathcal{X}$, and the language CLP($\mathcal{H}_\mathcal{X}$), where $\mathcal{H}$ and $\mathcal{X}$ are somewhat amalgamated. Of course, CLP($\mathcal{H}$, $\mathcal{X}$) languages are simpler, though still useful. The simplicity come from the fact that interpreted terms are not allowed to occur as leaves of Herbrand terms. When interpreted terms are allowed to occur in Herbrand structures more complex programs can be built. For instance, one can express unbounded lists where interpreted terms occur. Independently of the need for "unbounded containers", it is a common CLP idiom to embed interpreted terms into Herbrand terms.

From the experience gained with the first prototype version of China [2] it was clear that, in order to attain a significant precision in the analysis of numerical constraints in CLP($\mathcal{H}_\mathcal{N}$) languages, one must keep at least part of the uninterpreted terms in concrete form. Note that almost any analysis is much more precise when this kind of structural information is retained to some extent: in our case the precision loss was just particularly acute.

Cortesi *et al.* [6, 7] have a nice proposal in this respect. 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. Their approach is limited to the analysis of logic programs. Most importantly, the

*This work has been partly supported by EPSRC under grant GR/L19515.

presentation in [6] has several drawbacks. First of all, the authors define a *specific implementation* of the generic structural domain. The implementation is forcedly cluttered with details that make the general principles difficult to understand. Moreover, they describe an implementation of the *pattern component* (taking care of representing the terms in concrete form) that appears to be unnecessarily complicated. Their representation of terms and subterms, while responsible for some of the intricacies in the description, does not seem to have any advantage, from the implementation point of view, with respect to more standard representations of terms (such as those employed in the Warren's Abstract Machine and its variants [1]). As a consequence, standard notions from unification theory, such as *instance*, *anti-instance*, and *(least) common anti-instance* [10], are never mentioned in [6], while being implicitly present.

In this paper we present the rational construction of a generic domain for structural analysis of CLP($\mathcal{H}_\mathcal{X}$) languages: Pattern($\mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$), where the parameter $\mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$ is an abstract domain satisfying certain properties. The formalization of the structural domain is independent from specific implementation techniques: `Pat(`$\Re$`)` (suitably extended) is a possible implementation of the domain. 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, missing in [6]) that gives some control over the precision and computational cost of the resulting generic structural domain.

It must be stressed that the merit of `Pat(`$\Re$`)` is to define a generic implementation that works on any domain $\Re$ providing a certain set of *elementary*, low-level operations. It is particularly easy to extend an existing domain in order to support the simple operations required. 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. This is not a limitation, if one considers that in the actual implementation even more complex operations are used. For instance, all the *abstract bindings* arising from a bunch of unifications are executed in one shot, instead of one-at-a-time [3].

2 Preliminaries

Let S be a set. We will denote by S^n the set of n-tuples of elements drawn from S, whereas S^* denotes $\bigcup_{n\in\mathbb{N}} S^n$. Let *Vars* be a denumerable set of variable symbols. We will denote by $\mathcal{T}_{Vars}$ the set of terms with variables in *Vars*. 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 $\mathbf{z} \stackrel{\text{def}}{=} (Z_1, Z_2, Z_3, \ldots$ and $\mathbf{z}' \stackrel{\text{def}}{=} (Z'_1, Z'_2, Z'_3, \ldots$. For any syntactic object o (a term or a tuple of terms) we will denote by $\mathit{vseq}(o)$ the sequence of first occurrences of variables which are found on a depth-first, left-to-right traversal[1] of o. For instance, $\mathit{vseq}\Bigl(\bigl(f(g(X),Y),h(X)\bigr)\Bigr) = (X,Y)$.

In order to avoid the burden of talking "modulo renaming" we will make use of two strong normal forms for tuples of terms. Specifically, the set of *n-tuples in $\mathbf{z}$-form* is given by

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

[1] Any other *fixed* ordering would be as good for our purposes.

All the tuples in $\mathbf{z}$-form are contained in $\mathbf{T}_{\mathbf{z}}^{\star}$. The definitions for $\mathbf{T}_{\mathbf{z}'}^{n}$ and $\mathbf{T}_{\mathbf{z}'}^{\star}$ 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 $\bar{t} \in \mathbf{T}_{\mathbf{z}}^{n} \cup \mathbf{T}_{\mathbf{z}'}^{n}$ and $\bigl|\mathit{vars}(\bar{t}\,)\bigr| = m$. Then

$$\bar{t}' \stackrel{\mathrm{def}}{=} \begin{cases} \bar{t}[Z_1'/Z_1, \ldots, Z_m'/Z_m], & \text{if } \bar{t} \in \mathbf{T}_{\mathbf{z}}^{n}; \\ \bar{t}[Z_1/Z_1', \ldots, Z_m/Z_m'], & \text{if } \bar{t} \in \mathbf{T}_{\mathbf{z}'}^{n}. \end{cases} \tag{1}$$

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

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

Another renaming we will use is the following: for each $\bar{s} \in \mathcal{T}_{\mathit{Vars}}^{\star}$ and each other syntactic object o such that $FV(o) \subset \mathbf{z}$, we write $\varrho_{\bar{s}}(o)$ 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 $\bar{s}_1, \bar{s}_2 \in \mathbf{T}_{\mathbf{z}}^{\star}$ we have $\bar{s}_1 :: \varrho_{\bar{s}_1}(\bar{s}_2) \in \mathbf{T}_{\mathbf{z}}^{\star}$.

When $\bar{V} \in \mathit{Vars}^m$ and $\bar{t} \in \mathcal{T}_{\mathit{Vars}}^{m}$ we use $[\bar{t}/\bar{V}]$ as a shorthand for the substitution $\bigl[\pi_1(\bar{t}\,)/\pi_1(\bar{V}), \ldots, \pi_m(\bar{t}\,)/\pi_m(\bar{V})\bigr]$. A couple of observations are useful for what follows. If $\bar{s} \in \mathbf{T}_{\mathbf{z}}^{\star}$ and $\bar{u} \in \mathbf{T}_{\mathbf{z}}^{|\mathit{vars}(\bar{s})|}$ then $\bar{s}'\bigl[\bar{u}/\,\mathit{vseq}(\bar{s}')\bigr] \in \mathbf{T}_{\mathbf{z}}^{\star}$. Moreover

$$\mathit{vseq}\bigl(\bar{s}'\bigl[\bar{u}/\,\mathit{vseq}(\bar{s}')\bigr]\bigr) = \mathit{vseq}(\bar{u}).$$

3 Factoring Out Structural Information

A quite general picture for the analysis of a CLP($\mathcal{H}_{\mathcal{X}}$) language is as follows. We want to describe a (possibly infinite) set of constraint stores over a tuple of *variables of interest* $V_1, \ldots, V_k$. These variables represent the arguments of some program predicate. Each constraint store σ can be represented (at some level of abstraction) by a formula of the kind

$$\exists_{\Delta} \,.\, \bigl(\{V_1 = t_1, \ldots, V_k = t_k\} \wedge C\bigr), \tag{2}$$

such that

$$\{V_1 = t_1, \ldots, V_k = t_k\}, \quad \text{with } t_1, \ldots, t_k \in \mathcal{T}_{\mathit{Vars}}, \tag{3}$$

is a system of Herbrand equations in solved form, $C \in \mathcal{D}_{\mathcal{X}}^{\flat}$ is a constraint on the domain $\mathcal{X}$, and $\Delta \stackrel{\mathrm{def}}{=} \mathit{vars}(C) \cup \mathit{vars}(t_1) \cup \cdots \cup \mathit{vars}(t_k)$ is such that $\Delta \cap \{V_1, \ldots, V_k\} = \varnothing$. Roughly speaking, the purpose of C is to limit the values that the (quantified) variables occurring in $t_1, \ldots, t_k$ can take.

Once variables $V_1, \ldots, V_k$ have been fixed, the Herbrand part of the constraint store (2), the system of equations (3), can be represented as a k-tuple of terms. Since we want to characterize any set of constraint stores, our concrete domain is

$$\mathcal{D}_{\mathcal{H}_{\mathcal{X}}}^{\flat} \stackrel{\mathrm{def}}{=} \bigcup_{n \in \mathbb{N}} \wp\bigl(\mathbf{T}_{\mathbf{z}}^{n} \times \mathcal{D}_{\mathcal{X}}^{\flat}\bigr). \tag{4}$$

$$\begin{tikzcd}
\wp\big(\mathbf{T}^\star_{\mathbf{z}} \times \mathcal{D}^\flat_{\mathcal{X}}\big) \arrow[r, "\alpha"] \arrow[d, shift right, "\Phi_\phi"'] & \mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}} \\
\mathbf{T}^\star_{\mathbf{z}} \times \wp\big(\mathbf{T}^\star_{\mathbf{z}} \times \mathcal{D}^\flat_{\mathcal{X}}\big) \arrow[r, "(\mathrm{id},\alpha)"'] \arrow[u, shift right, "\Phi_\phi^{-1}"'] & \mathbf{T}^\star_{\mathbf{z}} \times \mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}} \arrow[u, dotted, "\alpha'"']
\end{tikzcd}$$

Figure 1: Upgrading a domain with structural information.

An abstract interpretation of $\mathcal{D}^\flat_{\mathcal{H}_\mathcal{X}}$ can be specified by choosing an abstract domain $\mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$ and a suitable abstraction function

$$\alpha \colon \mathcal{D}^\flat_{\mathcal{H}_\mathcal{X}} \to \mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}. \tag{5}$$

If $\mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$ is able to encode enough structural (Herbrand) information from $\mathcal{D}^\flat_{\mathcal{H}_\mathcal{X}}$ so as to achieve the desired precision, fine. If this is not the case, it is possible to improve the situation by keeping some structural information explicit.

One way of doing that is to perform a change of representation for $\mathcal{D}^\flat_{\mathcal{H}_\mathcal{X}}$, which is the basis for further abstractions. 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.) *A function*

$$\phi \colon \bigcup_{n \in \mathbb{N}} \wp(\mathbf{T}^n_{\mathbf{z}}) \to \mathbf{T}^\star_{\mathbf{z}'}$$

is called a common anti-instance function *if, for each $n \in \mathbb{N}$ and each $\hat{T} \in \wp(\mathbf{T}^n_{\mathbf{z}})$:*

1. *$\phi(\hat{T}) \in \mathbf{T}^n_{\mathbf{z}'}$;*
2. *if $\phi(\hat{T}) = \bar{r}$ and $\big|vars(\bar{r})\big| = m$ with $m \geq 0$, then*

$$\forall \bar{t} \in \hat{T} : \exists \bar{u} \in \mathbf{T}^m_{\mathbf{z}} \,.\, \bar{r}\big[\bar{u}/\, vseq(\bar{r})\big] = \bar{t}.$$

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

Any choice of ϕ induces a function

$$\Phi_\phi \colon \mathcal{D}^\flat_{\mathcal{H}_\mathcal{X}} \to \mathbf{T}^\star_{\mathbf{z}} \times \wp\big(\mathbf{T}^\star_{\mathbf{z}} \times \mathcal{D}^\flat_{\mathcal{X}}\big), \tag{6}$$

which is given, for each $\hat{E}^\flat \in \mathcal{D}^\flat_{\mathcal{H}_\mathcal{X}}$ by

$$\Phi_\phi(\hat{E}^\flat) \stackrel{\mathrm{def}}{=} \left(\bar{s}', \left\{ (\bar{u}, D^\flat) \;\middle|\; (\bar{t}, D^\flat) \in \hat{E}^\flat, \quad \bar{s}\big[\bar{u}/vseq(\bar{s})\big] = \bar{t} \right\}\right), \tag{7}$$

where $\bar{s} \stackrel{\mathrm{def}}{=} \phi\big(\pi_1(\hat{E}^\flat)\big)$. The corestriction to the image of Φ_ϕ, that is $\Phi_\phi \colon \mathcal{D}^\flat_{\mathcal{H}_\mathcal{X}} \to \Phi_\phi\big(\mathcal{D}^\flat_{\mathcal{H}_\mathcal{X}}\big)$, is an isomorphism. So far, we have just chosen a different representation for $\mathcal{D}^\flat_{\mathcal{H}_\mathcal{X}}$, that is $\Phi_\phi\big(\mathcal{D}^\flat_{\mathcal{H}_\mathcal{X}}\big)$. 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 5.4, such a function is implicitly required in order to define an important operation over the new abstract domain $\mathbf{T}^\star_\mathbf{z} \times \mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$.[2]

This approach has several advantages. First of all, 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 $\mathbf{T}^\star_\mathbf{z} \times \mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$. 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

$$\hat{S} \stackrel{\text{def}}{=} \Bigl\{ \bigl\langle \bigl(s(0), c(Z_1, nil)\bigr), C_1 \bigr\rangle, \bigl\langle \bigl(s(s(0)), c(Z_1, c(Z_2, nil))\bigr), C_2 \bigr\rangle \Bigr\},$$

where $C_1, C_2 \in \mathcal{D}^\flat_\mathcal{X}$, is mapped by the Φ_{lca} function onto

$$\Phi_{\text{lca}}(\hat{S}) = \biggl(\bigl(s(Z_1), c(Z_2, Z_3)\bigr), \Bigl\{ \bigl\langle (0, Z_1, nil), C_1 \bigr\rangle, \bigl\langle \bigl(s(0), Z_1, c(Z_2, nil)\bigr), C_2 \bigr\rangle \Bigr\} \biggr).$$

At the other side of the spectrum is the possibility of choosing ϕ so that it returns a k-tuple of distinct, new variables for each set of k-tuples of terms. This correspond to a framework where structural information is just discarded. With this choice, $\hat{S}$ would be mapped onto $\bigl((Z_1, Z_2), \hat{S}\bigr)$. In-between these two extremes there are a number of possibilities that help managing the complexity/precision tradeoff. The k-tuples returned by ϕ can be limited in *depth* [13, 11], for instance. More useful is to limit them in *width*, that is, limiting the number of symbols' occurrences. This flexibility allows to design the analysis' domains without caring about structural information: the problem is always to approximate elements of $\wp\bigl(\mathbf{T}^a_\mathbf{z} \times \mathcal{D}^\flat_\mathcal{X}\bigr)$. Whether a is fixed by the arity of a predicate or a is the number of variables occurring in some pattern does not really matter.

4 Parametric Structural Analysis

In order to specify the abstract domain for the analysis, we need some assumptions on the concrete $\mathcal{X}$ domain $\mathcal{D}^\flat_\mathcal{X}$, which represents the $\mathcal{X}$-part of *consistent* constraint stores. One can think about $\mathcal{D}^\flat_\mathcal{X}$ as made up of first-order sentences [5]. In this view, the operator $\otimes \colon \mathcal{D}^\flat_\mathcal{X} \times \mathcal{D}^\flat_\mathcal{X} \to \mathcal{D}^\flat_\mathcal{X}$ corresponds to logical conjunction. Moreover, we assume that it makes sense to talk about the *free variables* of $D^\flat \in \mathcal{D}^\flat_\mathcal{X}$, denoted by $FV(D^\flat)$. Let $\bar{s}, \bar{t}, \bar{u} \in \mathbf{T}^\star_\mathbf{z}$ and $D^\flat, E^\flat \in \mathcal{D}^\flat_\mathcal{X}$ such that $FV(D^\flat) \subseteq vars(\bar{t}\,)$. When we write $(\bar{u}, E^\flat) = \varrho_{\bar{s}}\bigl((\bar{t}, D^\flat)\bigr)$, we mean that $\bar{u} = \varrho_{\bar{s}}(\bar{t}\,)$ and that $E^\flat$ has been obtained from $D^\flat$ by applying the same renaming applied to $\bar{t}$ in order to obtain $\bar{u}$.

Another natural thing to do is projecting a satisfiable store: thus $\exists_{\bar{V}} D^\flat$, where $\bar{V}$ is a tuple (or set) of variables, is assumed to be as defined.

The last thing we need is the ability of adding an equality constraint to a constraint store. Thus $D^\flat[t_1 = t_2]$ is the store obtained from $D^\flat$ by injecting the equation $t_1 = t_2$, *provided that the resulting store is consistent*, otherwise the operation is undefined. Notice that we assume $\mathcal{D}^\flat_\mathcal{X}$ and its operations encode both the proper $\mathcal{X}$*-solver* and the so called *interface* between the *Herbrand engine* and the $\mathcal{X}$-solver [8]. In particular, the interface is responsible for *type-checking* of the equations it receives. For example in CLP($\mathcal{R}$) [9]

[2]Notice that α' may or may not make the diagram of Figure 1 commute (although often α' turns out to have this property).

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.

We now turn our attention to the abstract domain which is the parameter of the generic structural domain. We will denote it simply by $\mathcal{D}^\sharp$, instead of $\mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$. Thus, assuming that $\mathcal{X}$ has been fixed, $\mathcal{D}^\flat_\mathcal{X}$ is indicated just by $\mathcal{D}^\flat$.

Since the aim here is maximum generality, we refer to a very weak abstract interpretation framework.

Definition 2 (Abstract domain.) *An* abstract domain for $\mathcal{H}_\mathcal{X}$ *is a set* $\mathcal{D}^\sharp$ *equipped with a preorder relation* $\sqsubseteq$, *an order preserving function* $\gamma\colon \mathcal{D}^\sharp \to \mathcal{D}^\flat$, *an upper-bound operator* $\oplus\colon \mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \mathcal{D}^\sharp$, *and a least element* $\bot^\sharp$ *such that* $\gamma(\bot^\sharp) = \varnothing$.

Thus, for each $D_1^\sharp, D_2^\sharp \in \mathcal{D}^\sharp$, we have both that $D_1^\sharp \sqsubseteq D_2^\sharp$ implies $\gamma(D_1^\sharp) \subseteq \gamma(D_2^\sharp)$ and $\gamma(D_1^\sharp \oplus D_2^\sharp) \supseteq \gamma(D_1^\sharp) \cup \gamma(D_2^\sharp)$.

The structural information construction upgrades any given abstract domain $\mathcal{D}^\sharp$ as follows.

Definition 3 (The $\mathrm{Pattern}(\cdot)$ **construction.)** *Let* $\mathcal{D}^\sharp$ *be an abstract domain. Then*

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

The meaning of each element $(\bar{s}, D) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ *is given by the concretization function* $\gamma\colon \mathrm{Pattern}(\mathcal{D}^\sharp) \to \wp(\mathbf{T}_\mathbf{z}^\star \times \mathcal{D}^\flat)$*:*

$$\gamma\big((\bar{s}, D^\sharp)\big) \stackrel{\mathrm{def}}{=} \left\{ \Big(\bar{s}'\big[\bar{u}/\, vseq(\bar{s}')\big],\, D^\flat\Big) \;\middle|\; (\bar{u}, D^\flat) \in \gamma(D^\sharp) \right\}.$$

5 Operations for the Analysis

In this section we define the operations over $\mathrm{Pattern}(\mathcal{D}^\sharp)$ that are needed for the analysis in a bottom-up framework. In order of appearance into the analysis process:

- we need an operation that takes two descriptions and, roughly speaking, juxtaposes them. This operation, which we call *meet with renaming apart*, is needed when "solving" a clause body with respect to the current interpretation.
- Unification, that realizes "parameter passing". The descriptions that were simply juxtaposed are thus made to communicate with each other.
- 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 operation of *remapping* is used to adapt a description to a different, less precise, pattern component. It is used in order to realize various *join* and *widening* operations.
- The *join* operation is parameterized with respect to a common anti-instance function. It is used to merge descriptions arising from the different sets of computation paths explored during the analysis.
- The *comparison* operation is employed by the analyzer in order to check whether a local (to a program clause or predicate) fixpoint has been reached.

The above operations over $\mathrm{Pattern}(\mathcal{D}^\sharp)$ induce the need for other operations on the underlying domain $\mathcal{D}^\sharp$. The latter are specified in the next sections so that the correctness of the analysis can be ensured.

5.1 Meet with Renaming Apart

This operation is very simple.

Definition 4 (The rmeet **operation.)** *Let* $\triangleright \colon \mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ *be such that, for each* $D_1^\sharp, D_2^\sharp \in \mathcal{D}^\sharp$,

$$\gamma(D_1^\sharp \triangleright D_2^\sharp) = \left\{ (\bar{r}_1 :: \bar{w}_2, D_1^\flat \otimes E_2^\flat) \;\middle|\; \begin{array}{l} (\bar{r}_1, D_1^\flat) \in \gamma(D_1^\sharp) \\ (\bar{r}_2, D_2^\flat) \in \gamma(D_2^\sharp) \\ (\bar{w}_2, E_2^\flat) = \varrho_{\bar{r}_1}\bigl((\bar{r}_2, D_2^\flat)\bigr) \end{array} \right\}.$$

Then, for each $(\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$.

$$\mathrm{rmeet}\bigl((\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp)\bigr) \stackrel{\mathrm{def}}{=} \bigl(\bar{s}_1 :: \varrho_{\bar{s}_1}(\bar{s}_2),\ D_1^\sharp \triangleright D_2^\sharp\bigr).$$

The following result is a direct consequence of the definition: there is no precision loss in rmeet.

Theorem 5 *For each* $(\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$.

$$\gamma\Bigl(\mathrm{rmeet}\bigl((\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp)\bigr)\Bigr) = \left\{ (\bar{t}_1 :: \bar{u}_2, D_1^\flat \otimes E_2^\flat) \;\middle|\; \begin{array}{l} (\bar{t}_1, D_1^\flat) \in \gamma\bigl((\bar{s}_1, D_1^\sharp)\bigr) \\ (\bar{t}_2, D_2^\flat) \in \gamma\bigl((\bar{s}_2, D_2^\sharp)\bigr) \\ (\bar{u}_2, E_2^\flat) = \varrho_{\bar{t}_1}\bigl((\bar{t}_2, D_2^\flat)\bigr) \end{array} \right\}.$$

5.2 Unification with Occur-Check

In this section we assume that the execution mechanism of the language being analyzed performs unifications without omitting the *occur-check*. With this hypothesis (which, unfortunately, is seldom verified) we can easily complete the unification algorithm given in [6]. When the *occur-check* fails in the abstract unification we know that the computation path being analyzed can be safely pruned, because the concrete unification would have failed at this point. Notice that, for the purpose of the present discussion, the *occur-check* need not be implemented explicitly, that is by making the unification *fail* in the logic programming sense. Since our data-flow analyses provide information of the kind

if control gets to this point, *then* that will hold there,

a more drastic handling of the *occur-check* is acceptable. If we are guaranteed that the concrete system enters either an error state or an infinite loop whenever a cyclic binding is attempted, then the abstract unification procedure presented in this section can safely be used. See [4] for a discussion about what can be done for those systems where the *occur-check* is, by any means, omitted.

We start with a description $(\bar{s}, D^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ and two terms to be unified, t and u, such that $\mathit{vars}\bigl((t,u)\bigr) \subseteq \mathit{vars}(\bar{s})$. We then apply the procedure unify, given as Algorithm 1, to $\bar{s}$, $D^\sharp$, t and u. In the macro-operation $\mathrm{bind}(\bar{s}, D^\sharp, u, Z_h)$, $\bar{s}$ is passed only in order to maintain the connection between the variables in (u, Z_h) and the description $D^\sharp$. We assume, without loss of generality, that whenever $\mathrm{bind}(\bar{s}, D^\sharp, u, Z_h)$ is invoked we have $\bigl|\mathit{vars}(\bar{s})\bigr| = m$, with $m \geq 0$. The result of the operation will be a description $D_1^\sharp$ such that $\gamma(D_1^\sharp) \subseteq \mathbf{T}_{\mathbf{z}}^{m-1} \times \mathcal{D}^\flat$. This is because, after the binding, Z_h will not be referenced anymore. What remains to be described is the operation of reflecting the binding of Z_h to u into $D^\sharp$ so to obtain $D_1^\sharp$. We will denote this operation by $D^\sharp[u/Z_h]$, and present its variants (depending on whether u is a constant or a number or a variable or a compound term) in the next sections.

procedure unify$(\bar{s}, D^\sharp, t, u)$
if $t \neq u$ **then**
  if $t = f(t_1, \ldots, t_n)$ and $u = f(u_1, \ldots, u_n)$ **then**
    for all $i = 1, \ldots, n$ **do**
      unify$(\bar{s}, D^\sharp, t_i, u_i)$
  else if $t = Z_h$ **then**
    if Z_h does not occur in u **then**
      $D^\sharp :=$ bind$(\bar{s}, D^\sharp, u, Z_h)$ {invokes underlying domain}
      $Z_h := u$ {instantiates *all* the occurrences of Z_h}
      $\bar{s} := \eta(\bar{s})$ {normalization}
    else
      $D^\sharp := \perp^\sharp$
  else if $u = Z_k$ **then**
    unify$(\bar{s}, D^\sharp, u, t)$
  else
    $D^\sharp := \perp^\sharp$

Algorithm 1: Unification for the parametric structural domain.

5.2.1 Binding to a Constant or a Number

The result of $D^\sharp[k/Z_h]$, where k is a symbolic constant or a number and $h \in \{1, \ldots, m\}$ is any $D_1^\sharp \in \mathcal{D}^\sharp$ such that

$$\gamma(D_1^\sharp) \supseteq \left\{ \left((t_1, \ldots, t_{h-1}, t_{h+1}, \ldots, t_m), D^\flat[t_h = k]\right) \,\middle|\, \left((t_1, \ldots, t_m), D^\flat\right) \in \gamma(D^\sharp) \right\}.$$

Notice that $vseq\big(\bar{s}[k/Z_h]\big) = (Z_1, \ldots, Z_{h-1}, Z_{h+1}, \ldots, Z_m)$. Similar comments apply also to what follows.

5.2.2 Binding to an Alias

Here we must specify an admissible result, $D_1^\sharp$, for the operation $D^\sharp[Z_i/Z_h]$ with $i, h \in \{1, \ldots, m\}$ and $i \neq h$. In order to reduce the complexity of the definition we need some special notation for sequences. Let U be a set. Then $\wp_{\mathrm{f}}(U)$ denotes the set of all the *finite* subsets of U. We define the operation $\cdot \setminus \cdot \colon U^\star \times \wp_{\mathrm{f}}(U) \to U^\star$ as follows. For each sequence $L \in U^\star$ and each set $S \in \wp_{\mathrm{f}}(U)$, the sequence $L \setminus U$ is obtained by removing from L all the elements that appear in U. Let us define $\tau(k) \colon \{1, \ldots, m-1\} \to \{1, \ldots, m\}$ as

$$\tau(k) \stackrel{\mathrm{def}}{=} \pi_k\Big((1, \ldots, h-1) :: \big((i) \setminus \{1, \ldots, h-1\}\big) :: \big((h+1, \ldots, m) \setminus \{i\}\big)\Big).$$

The transformation τ is such that, for each $k = 1, \ldots, m-1$,

$$\pi_k\Big(vseq\big(\bar{s}[Z_i/Z_h]\big)\Big) = Z_{\tau(k)}.$$

Now, $D_1^\sharp$ must satisfy

$$\gamma(D_1^\sharp) \supseteq \left\{ \Big(\big(\pi_{\tau(1)}(\bar{t}\,), \ldots, \pi_{\tau(m-1)}(\bar{t}\,)\big), D^\flat\big[\pi_h(\bar{t}\,) = \pi_i(\bar{t}\,)\big]\Big) \,\middle|\, (\bar{t}, D^\flat) \in \gamma(D^\sharp) \right\}.$$

5.2.3 Binding to a Compound

Specifying the result of the operation $D[u/Z_h]$ is only slightly more complicated. Suppose that $vseq(u) = \bigl(Z_{j_1}, \ldots, Z_{j_l}\bigr)$, so that $Z_h \notin \bigl\{Z_{j_1}, \ldots, Z_{j_l}\bigr\}$ and the transformation τ is given by

$$\tau(k) \stackrel{\text{def}}{=} \pi_k\Bigl((1, \ldots, h-1) :: \bigl((j_1, \ldots, j_l) \setminus \{1, \ldots, h-1\}\bigr) \\ :: \bigl((h+1, \ldots, m) \setminus \{j_1, \ldots, j_l\}\bigr)\Bigr). \quad (8)$$

Then $D^\sharp[u/Z_h]$ is allowed to return any $D_1^\sharp$ such that

$$\gamma(D_1^\sharp) \supseteq \left\{ \Bigl(\bigl(\pi_{\tau(k)}(\bar{t}\,)\bigr)_{k=1}^{m-1}, D_1^\flat\Bigr) \;\middle|\; \begin{array}{l} (\bar{t}, D^\flat) \in \gamma(D^\sharp) \\ \theta = \bigl[u_{j_1}/Z_{j_1}, \ldots, u_{j_l}/Z_{j_l}\bigr] \\ \pi_h(\bar{t}\,) = u\theta \\ D_1^\flat = D^\flat\bigl[\pi_{j_1}(\bar{t}\,) = u_{j_1}, \\ \qquad\qquad \ldots, \pi_{j_l}(\bar{t}\,) = u_{j_l}\bigr] \end{array} \right\}.$$

As observed in [6], the proof of the overall correctness of Algorithm 1 can be obtained by systematic generalization of the proof given in Musumbu's PhD thesis [12].

5.3 Projection

This operation consists simply in dropping a suffix of the pattern component, with the consequent projection on the underlying domain.

Definition 6 (The project **operation.)** *Let $\exists_m \colon \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ be a family of operations such that, for each $D^\sharp \in \mathcal{D}^\sharp$ with $\gamma(D^\sharp) \subseteq \mathbf{T}_{\mathbf{z}}^m \times \mathcal{D}^\flat$ and each $j < m$,*

$$\gamma\bigl(\bar{\exists}_j\, D^\sharp\bigr) \supseteq \left\{ (\bar{r}, E^\flat) \;\middle|\; \begin{array}{l} (\bar{u}, D^\flat) \in \gamma(D^\sharp) \\ \bar{r} = \bigl(\pi_1(\bar{u}), \ldots, \pi_j(\bar{u})\bigr) \\ \Delta = \overline{vars(\bar{r})} \\ E^\flat = \exists_\Delta D^\flat \end{array} \right\}.$$

Then, for each $(\bar{s}, D^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ such that $\bar{s} \in \mathbf{T}_{\mathbf{z}}^n$ and each $k < n$,

$$\mathrm{project}_k\bigl((\bar{s}, D^\sharp)\bigr) \stackrel{\text{def}}{=} \Bigl(\underbrace{\bigl(\pi_1(\bar{s}), \ldots, \pi_k(\bar{s})\bigr)}_{\bar{t}}, \bar{\exists}_j\, D^\sharp\Bigr),$$

where $j \stackrel{\text{def}}{=} \bigl|vars(\bar{t}\,)\bigr|$.

It is easy to show that project is indeed correct with respect to the obvious concrete operation.

5.4 Remapping

Consider a description $(\bar{s}, D_{\bar{s}}^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ and a pattern $\bar{r} \in \mathbf{T}_{\mathbf{z}'}^\star$ such that $\bar{r}$ is an anti-instance of $\bar{s}$. We want to obtain $D_{\bar{r}}^\sharp \in \mathcal{D}^\sharp$ such that

$$\gamma\bigl((\bar{r}', D_{\bar{r}}^\sharp)\bigr) \supseteq \gamma\bigl((\bar{s}, D_{\bar{s}}^\sharp)\bigr). \quad (9)$$

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

Definition 7 (The remap **operation.)** *Let* $(\bar{s}, D^\sharp_{\bar{s}})$ *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* $\left|vars(\bar{r})\right| = m$ *and let* $\bar{u} \in \mathbf{T}^m_{\mathbf{z}}$ *be the unique tuple such that*

$$\bar{r}\big[\bar{u}/\, vseq(\bar{r})\big] = \bar{s}. \tag{10}$$

Then the operation $\mathrm{remap}(\bar{s}, D^\sharp_{\bar{s}}, \bar{r})$ *yields* $D^\sharp_{\bar{r}}$ *such that*

$$\gamma(D^\sharp_{\bar{r}}) \supseteq \left\{ \left(\bar{u}'\big[\bar{t}/\, vseq(\bar{u}')\big],\, D^\flat\right) \;\middle|\; (\bar{t}, D^\flat) \in \gamma(D^\sharp_{\bar{s}}) \right\}. \tag{11}$$

Observe that the remap function is closely related to the residual abstraction function α' of Figure 1. It can be proven [4] that the specification of remap meets our original requirement.

Theorem 8 *Let* $(\bar{s}, D^\sharp_{\bar{s}})$ *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* $D^\sharp_{\bar{r}} = \mathrm{remap}(\bar{s}, D^\sharp_{\bar{s}}, \bar{r})$ *then* $\gamma\big((\bar{r}', D^\sharp_{\bar{r}})\big) \supseteq \gamma\big((\bar{s}, D^\sharp_{\bar{s}})\big)$.

5.5 Join and Widenings

The operation of merging two descriptions turns out to be an easy one, once remapping has been defined.

Definition 9 (The join_ϕ **operations.)** *Let* ϕ *be any common anti-instance function. The operation (partial function)*

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

is defined as follows. For each $k \in \mathbb{N}$ *and each finite family* $F \stackrel{\mathrm{def}}{=} \big\{(\bar{s}_i, D^\sharp_i)\big\}_{i\in I}$ *of elements of* $\mathrm{Pattern}(\mathcal{D}^\sharp)$ *such that* $\bar{s}_i \in \mathbf{T}^k_{\mathbf{z}}$ *for each* $i \in I$*, we have* $\bar{r} \stackrel{\mathrm{def}}{=} \phi\big(\{\bar{s}_i\}_{i\in I}\big)$ *and*

$$\mathrm{join}_\phi(F) \stackrel{\mathrm{def}}{=} \left(\bar{r}',\, \bigoplus_{i\in I} \mathrm{remap}(\bar{s}_i, D^\sharp_i, \bar{r})\right).$$

We note again that ϕ might be the least common anti-instance function or an approximation of it: this is one of the degrees of freedom of the framework.

Theorem 10 *Let* F *be as in* Definition 9*. For each common anti-instance function* ϕ *and each* $(\bar{s}_j, D^\sharp_j) \in F$ *we have* $\gamma\big(\mathrm{join}_\phi(F)\big) \supseteq \gamma\big((\bar{s}_j, D^\sharp_j)\big)$.

As far as widening operators are concerned, there are several possibilities. First of all, we might want to distinguish between widening in the pattern component and widening on the underlying domain. The former can be defined as any join operation join_ϕ with ϕ different from lca. The latter consists in propagating the widening to the underlying domain. For instance, the following widening operator is the default one applied by China:

$$\mathrm{widen}\big((\bar{s}_1, D^\sharp_1), (\bar{s}_2, D^\sharp_2)\big) \stackrel{\mathrm{def}}{=} \begin{cases} (\bar{s}_2, D^\sharp_2), & \text{if } \bar{s}_1 \neq \bar{s}_2; \\ (\bar{s}_2, D^\sharp_1 \mathbin{\nabla} D^\sharp_2), & \text{if } \bar{s}_1 = \bar{s}_2. \end{cases} \tag{12}$$

This operator refrains from widening unless the pattern component is stabilized (see the next section to see why it works). Other operators can be defined by using joins and remappings (see [4]).

5.6 Comparing Descriptions

The last operation that is needed in order to put $\mathrm{Pattern}(\mathcal{D}^\sharp)$ at work is for comparing descriptions.

Definition 11 (Approximation ordering.) *The* approximation ordering *of the domain* $\mathrm{Pattern}(\mathcal{D}^\sharp)$*, denoted by* $\sqsubseteq$*, is defined as follows, for each* $(\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$*:*

$$(\bar{s}_1, D_1^\sharp) \sqsubseteq (\bar{s}_2, D_2^\sharp) \quad \overset{\text{def}}{\Longleftrightarrow} \quad \bar{s}_1 = \bar{s}_2 \wedge D_1^\sharp \sqsubseteq D_2^\sharp.$$

It must be stressed that the above approximation ordering is also "approximate", since it does not take into account the peculiarities of $\mathcal{D}^\sharp$. More refined orderings can be obtained in a domain-dependent way, namely, when $\mathcal{D}^\sharp$ has been fixed.

The following result is a trivial consequence of Definition 2.

Theorem 12 *If* $(\bar{s}_1, D_1^\sharp) \sqsubseteq (\bar{s}_2, D_2^\sharp)$ *then* $\gamma\big((\bar{s}_1, D_1^\sharp)\big) \subseteq \big((\bar{s}_2, D_2^\sharp)\big)$*. Moreover,* $\sqsubseteq$ *is a preorder over* $\mathrm{Pattern}(\mathcal{D}^\sharp)$.

Observe that the ability of comparing descriptions only when they have the same pattern is not restrictive in a data-flow analysis setting. The analyzer, in fact, will only need to compare the descriptions arising from the iteration sequence at two consecutive steps. Moreover, if we denote by $(\bar{s}_n, D_n^\sharp)$ the description at step n, we have

$$(\bar{s}_{i+1}, D_{i+1}^\sharp) = \mathrm{widen}\Big((\bar{s}_i, D_i^\sharp), \mathrm{join}_\phi\Big(\big\{(\bar{s}_i, D_i^\sharp), \dots\big\}\Big)\Big), \tag{13}$$

where the widening is possibly omitted. Whether or not the widening has been applied, this implies that $\bar{s}'_{i+1}$ is an anti-instance of $\bar{s}_i$ and

$$\gamma\big((\bar{s}_i, D_i^\sharp)\big) \subseteq \gamma\big((\bar{s}_{i+1}, D_{i+1}^\sharp)\big). \tag{14}$$

If also the reverse inclusion holds in (14) then we have reached a local fixpoint. The analyzer uses the approximate ordering in order to check for this possibility. Namely, it asks whether $(\bar{s}_{i+1}, D_{i+1}^\sharp) \sqsubseteq (\bar{s}_i, D_i^\sharp)$. The approximate test, of course, can fail even when equality does hold in (14). 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 $\mathbf{T}_{\mathbf{z}}^k$, 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 $\mathcal{D}^\sharp$.

6 Conclusion

We have presented the rational construction of a generic domain for structural analysis of $\mathrm{CLP}(\mathcal{H}_\mathcal{X})$ languages: $\mathrm{Pattern}(\mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}})$, where the parameter $\mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$ is an abstract domain satisfying certain properties, We build on the parameterized $\mathtt{Pat}(\Re)$ domain of Cortesi *et al.* [6, 7], which is restricted to logic programs. However, while $\mathtt{Pat}(\Re)$ is presented as a *specific implementation* of a generic structural domain, our formalization is independent from specific implementation techniques. Reasoning at a higher level of abstraction we have been able to fully justify the ideas behind the structural domain. In particular, appealing to familiar notions of unification theory, we have identified an important parameter (a common anti-instance function, missing in [6]) that gives some control over the precision and computational cost of the resulting generic structural domain.

References

[1] H. Aït-Kaci. *Warren's Abstract Machine. A Tutorial Reconstruction.* The MIT Press, 1991.

[2] R. Bagnara. On the detection of implicit and redundant numeric constraints in CLP programs. In M. Alpuente, R. Barbuti, and I. Ramos, editors, *Proceedings of the "1994 Joint Conference on Declarative Programming (GULP-PRODE '94)"*, pages 312–326, Peñíscola, Spain, September 1994.

[3] R. Bagnara. A reactive implementation of *Pos* using ROBDDs. 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 107–121, Aachen, Germany, 1996. Springer-Verlag, Berlin.

[4] 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.

[5] R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. To appear in *Science of Computer Programming*. Extended version available as TR-96-10, Dipartimento di Informatica, Università di Pisa, 1997.

[6] 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.

[7] A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Combinations of abstract domains for logic programming. In *Conference Record of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 227–239, Portland, Oregon, 1994.

[8] J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19&20:503–582, 1994.

[9] 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.

[10] 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.

[11] K. Marriott and H. Søndergaard. On describing success patterns of logic programs. Technical Report 12, The University of Melbourne, 1988.

[12] K. Musumbu. *Interprétation Abstraite des Programmes Prolog.* PhD thesis, Facultés Universitaires Notre-Dame de la Paix – Namur Institut d'Informatique, Belgium, September 1990.

[13] T. Sato and H. Tamaki. Enumeration of success patterns in logic programs. *Theoretical Computer Science*, 34:227–240, 1984.