Analysis of Downward Closed Properties of Logic Programs*

Patricia M. Hill[1] and Fausto Spoto[2]**

[1] School of Computer Studies, University of Leeds, Leeds, LS2 9JT, UK
hill@scs.leeds.ac.uk
[2] Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56100 Pisa, Italy
spoto@di.unipi.it
Ph.: +39-(0)50887248 Fax: +39-(0)50887226

Abstract We study the analysis of downward closed properties of logic programs, which are a very abstract presentation of types. We generalise to a very large class of downward closed properties the construction of the traditional domains for groundness analysis in such a way that the results enjoy the good properties of that domain. Namely, we obtain abstract domains with a clear representation made of logical formulas and with optimal and well-known abstract operations. Moreover, they can be built using the linear refinement technique, and, therefore, are provably *optimal* and enjoy the condensing property, which is very important for a goal-independent analysis.

Keywords: *Abstract interpretation, domain theory, linear refinement, type theory, type analysis, logic programming.*

1 Introduction

A downward (instantiation) closed set of terms represents a property which is maintained during the computation. Therefore, it is a very abstract definition of a type. The analysis of downward closed properties is therefore a general way of doing type analysis, i.e., the upward approximation of the success set of a program through types. Type analysis of logic programs is important for verification as well as optimisation of unification. Note that other interesting properties of logic programs, like freeness and sharing, are not types since they are not downward closed.

An important and simple type, which distinguishes whether a term contains variables or not, is groundness [1, 5, 6]. The usual domain for groundness analysis, *Pos*, features some desirable properties: simplicity, human readability, effectivity, usefulness. Moreover, it has been shown [13, 15] that *Pos* is condensing and is the most precise domain for groundness analysis that does not consider the name

* Part of this work was supported by EPSRC grant GR/M05645.

** Part of this work was done while Fausto Spoto was visiting the School of Computer Studies of the University of Leeds, UK.

of the functors in a term. All these good properties of the *Pos* domain should have encouraged a generalisation of *Pos* to a general type domain. Instead, type domains have been developed, up to now, in a way totally independent from the domain *Pos* for groundness analysis. In [16] the authors define generic type domains with dependencies but, again, this is not a real generalisation of *Pos*, though it can be used for abstract compilation. The only paper which generalises the definition of the domain *Pos* to generic types is [3]. However, in this paper it is assumed that the usual properties of *Pos* hold for all the type domains. For instance, logical conjunction between formulas is used as conjunction operator and Schröder elimination as cylindrification operator. It is not obvious at all that these operators, which are optimal in the case of groundness analysis [6], are even correct in the general case of type analysis and no proof is given. Although in [4] a domain with properties similar to those of *Pos* is built, the way it is constructed is not a generalisation of that used for *Pos* [15]. Indeed, the domain defined in [4] is not made of logical formulas and cannot be obtained by linear refinement of a basic domain for the type properties.

In [12] it is shown that a hierarchy of domains for type analysis can be defined in the same way as it has been done in the case of groundness analysis [15]. Transfinite formulas, which are needed to deal with polymorphism, are used to represent the type domain and generalise the finite formulas of *Pos*. It is shown how a finite representation of these formulas can be achieved through the use of type variables. Two operators, logical conjunction and Schröder elimination, are used as abstract operators over transfinite formulas and are the generalisation of the same operators used for the *Pos* domain for groundness analysis.

In this paper we use the abstract interpretation framework [7, 8] to generalise the correctness result of [6] and [12] to a much larger class of type domains. Namely, we show that the abstract operators defined in these two papers are correct for every type domain, including then groundness and non-freeness, which were not covered in [12]. Moreover, we present a very weak condition, sufficient to entail the optimality of the abstract operators for a large class of type systems, including groundness, non-freeness and the type systems considered in [12] when speaking of correctness. Note that no optimality result were given in [12], while [6] proves optimality for the case of groundness only. The optimality result is important since it shows that, when our sufficient condition is satisfied, the use of transfinite formulas as a representation of type domains does not introduce any loss of precision for the computation of the abstract operators.

The correctness and optimality result proved in [6] for the two operations over the formulas of *Pos* cannot easily be lifted to a generic type domain. Given a different type domain, every proof should be rewritten. This is because a direct (Galois) connection between the concrete domain of existential Herbrand constraints and the abstract domain for groundness analysis is defined, and this connection is not parametrised with respect to a generic type. Instead, we use a two-step generic abstraction from existential Herbrand constraints into a generic type domain. In the first step we show that the collecting domain of existential Herbrand constraints can be abstracted into the set of downward closed sets of substitutions, which we call *Down*. A downward closed set of substitutions represents the union of the solutions of a set of existential Herbrand constraints. We provide the optimal abstract operators on *Down* induced by the corresponding operators on existential Herbrand constraints. Note that this first step is independent from the type analysis at hand. We consider the particular analysis only in the second step when we abstract *Down* into a type domain of transfinite formulas. Since the distance between type domains and *Down* is very much shorter than the distance between type domains and existential Herbrand constraints, the definition of a Galois connection between *Down* and the type domain is rather obvious. Moreover, it is quite easy to provide the optimal operators on the type domains induced by the optimal operators on *Down*.

The paper is organised as follows. Section 3 introduces the domain *Down*, whose constraints are downward closed sets of substitutions and shows a subset *Sol* of *Down* which is isomorphic to the set H of existential Herbrand constraints. Section 4 defines the domain $\wp(H)$ used for a collecting version of the semantics on H and its isomorphic counterpart $\wp(Sol)$, and shows that it can be abstracted into our *concrete* domain *Down*. This time, the whole domain *Down* will be used, rather than its subset *Sol*. From now on, the abstraction will be type-specific. Namely, Section 5 defines a generic type system and its induced type domain T formed by transfinite formulas. Moreover, it provides correct operators on T corresponding to the operators on *Down*, and shows sufficient conditions for their optimality. Section 6 shows that our work generalises some old results and provides logical domains for type analysis of logic programs. Section 7 shows the application of the linear refinement technique for developing abstract domains able to model conjunction in a very precise way. Some concluding remarks are discussed in Section 8. The picture below synthesises the various domains considered in this paper, and their relationships as abstraction (represented by horizontal arrows) or lifting to the powerset (represented by vertical arrows).

collecting	$\wp(H) \leftrightarrow$	$\wp(Sol) \rightarrow$	$Down$	$\rightarrow$	T
	$\uparrow$	$\uparrow$			
non-collecting	$H \leftrightarrow$	Sol			type dependent
	type independent abstractions				abstraction.

For reasons of space, proofs are omitted.

2 Preliminaries

2.1 Terms and substitutions

The *powerset* of a set S is $\wp(S) = \{S' \mid S' \subseteq S\}$. We denote by $\wp_f(S)$ the set of all finite subsets of S. If S is partially ordered w.r.t. $\leq$, we denote by $\wp\!\downarrow\!(S)$ the set of all the downward closed sets of S, i.e., $\wp\!\downarrow\!(S) = \{s \in \wp(S) \mid$ if $x \in$

s and $y \leq x$ then $y \in s\}$. A sequence is an ordered collection of elements. If $\tilde{x}$ is a sequence of elements of S, we write $\tilde{x} \subseteq S$.

Given a set of variables V and a set of function symbols Σ with associated arity, we define $\mathsf{terms}(\Sigma, V)$ as the set of terms built from V and Σ. We denote by $\mathsf{vars}(t)$ the set of variables which occur in a term t. When $\mathsf{vars}(t) = \emptyset$ we say that the term t is ground. Given a set of variables V and a variable x, $V \cup x$ means $V \cup \{x\}$ and $V \setminus x$ means $V \setminus \{x\}$.

We define $\Theta^Z_{V,W}$, with $Z \subseteq V \cap W$, as the set of substitutions θ such that $\mathsf{dom}(\theta) \subseteq V$, $\theta(x) \in \mathsf{terms}(\Sigma, W)$ for every $x \in V$ and $\mathsf{dom}(\theta) \cap \mathsf{rng}(\theta) \subseteq Z$. If $Z = \emptyset$ we omit the superscript. The elements of $\Theta_{V,W}$ are called idempotent substitutions. We write Θ_V for $\Theta_{V,V}$ and Θ^Z_V for $\Theta^Z_{V,V}$. Given θ and a set of variables R, we define $\theta|_R(x) = \theta(x)$ if $x \in R$ and $\theta|_R(x) = x$ otherwise. Given a term $t \in \mathsf{terms}(\Sigma, V)$ and $\theta \in \Theta^Z_{V,W}$, $t\theta \in \mathsf{terms}(\Sigma, W)$ is the term obtained with parallel substitution of every variable x in t with $\theta(x)$. Composition of substitutions $\theta \in \Theta_{V,W}$ and $\sigma \in \Theta_{W,Z}$ is defined as $(\theta\sigma)(x) = \theta(x)\sigma$ for every $x \in V \cup W$. If $f : \wp(\mathsf{terms}(\Sigma, V)) \to \wp(\mathsf{terms}(\Sigma, V))$ then $\mu t.f(t)$ denotes the least fixpoint of f.

For every set of variables V, a preorder is defined on Θ_V as $\theta' \leq_V \theta$ if there exists a substitution $\sigma \in \Theta^V_V$ such that $\theta' = \theta\sigma$. When V is clear from the context, we write $\leq$ instead of $\leq_V$. A preorder, called subsumption, is defined on $\mathsf{terms}(\Sigma, V)$ as $t_1 \leq_V t_2$ (t_1 is an instance of t_2) if $t_1 = t_2\theta$ for a suitable $\theta \in \Theta^V_V$. As usual, the subscript is omitted in $\leq_V$ when it is clear from the context. By $\equiv$ we denote the associated equivalence relation (*variance*).

2.2 Existential Herbrand constraints

Let Σ be a set of function symbols with associated arity and V a finite set of variables. We define the set of finite sets of Herbrand equations C_V as

$$C_V = \wp_f(\{t^1 = t^2 \mid t^1, t^2 \in \mathsf{terms}(\Sigma, V)\}) \ .$$

Suppose $c \in C_V$. Then we say that $c\theta$ *is true* if $t^1\theta$ is syntactically equal to $t^2\theta$ for every $(t^1 = t^2) \in c$. We know [14] that if there exists $\theta \in \Theta_V$ such that $c\theta$ is true, then c can be put in *normal form* $\mathsf{mgu}(c) = \{{t'}^1_j = {t'}^2_j \mid j \in J\}$, where ${t'}^1_j \in V$ are distinct variables which do not occur in ${t'}^2_k$ for every $j, k \in J$ and the variables in $\mathsf{mgu}(c)$ are all contained in the variables of c. Moreover, we have $c\theta$ is true if and only if $\mathsf{mgu}(c)\theta$ is true. If no θ exists such that $c\theta$ is true, then $\mathsf{mgu}(c)$ and hence the normal form of c are undefined. If c is in normal form, c can be seen as a substitution, and we use the notation $c(v)$ meaning the term t on the right of an equality $v = t \in c$ if such an equality exists, v itself otherwise. Moreover, every substitution can be seen as an existential Herbrand constraint without existential variable. Namely, we define $\mathsf{Eq}(\theta) = \exists_\emptyset\{v = \theta(v) \mid v \in \mathsf{dom}(\theta)\}$.

Let $\mathcal{V}$ and $\mathcal{W}$ be disjoint infinite sets of variables. For each $V \in \wp_f(\mathcal{V})$, we have a set of constraints, called existential Herbrand constraints, given by

$$H_V = \left\{ \exists_W c \,\middle|\, \begin{array}{l} W \in \wp_f(\mathcal{W}),\ c \in C_{V \cup W} \\ \text{there exists } \theta \in \Theta_{V \cup W} \text{ such that } \mathsf{rng}(\theta) \subseteq V \text{ and } c\theta \text{ is true} \end{array} \right\} \ .$$

Here, $\mathcal{V}$ are called the *program variables* and $\mathcal{W}$ the *existential variables*. The set of solutions of an existential Herbrand constraint is defined as

$$\mathsf{sol}_V(\exists_W c) = \{\theta|_V \mid \theta \in \Theta_{V\cup W},\ \mathsf{rng}(\theta) \subseteq V \text{ and } c\theta \text{ is true}\} \ .$$

Hence $\mathsf{sol}_V(\exists_W c) = \mathsf{sol}_V(\exists_W \mathsf{mgu}(c))$. A constraint $\exists_W c$ is said to be in *normal form* if c is in normal form. Note that $\mathsf{sol}_V(h)$ is a downward closed set.

A preorder is defined on H_V as $h_1 \leq h_2$ if and only if $\mathsf{sol}_V(h_1) \subseteq \mathsf{sol}_V(h_2)$. This preorder becomes a partial order if we consider equivalence classes of constraints, where $h_1, h_2 \in H_V$ are called *equivalent* if and only if $\mathsf{sol}_V(h_1) = \mathsf{sol}_V(h_2)$. In the following, a constraint will stand for its equivalence class. Since, as shown above, every constraint can be put in an equivalent normal form, in the following we will consider only normal constraints.

In order for a domain to be used for program analysis, a conjunction and a restriction operation must be provided, together with *diagonal elements*, used for variable renaming.

Definition 1. *Let $\{W_1, W_2\} \subseteq \mathcal{W}$ with $W_1 \cap W_2 = \emptyset$ [1] and $x \in V$. We define*

$$(\exists_{W_1} c_1) \star^{H_V} (\exists_{W_2} c_2) = \begin{cases} \exists_{W_1\cup W_2} \mathsf{mgu}(c_1 \cup c_2) & \textit{if } \mathsf{mgu}(c_1 \cup c_2) \textit{ exists} \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

$$\exists_x^{H_V}(\exists_W c) = \exists_{W\cup N} c[N/x] \qquad \textit{where } N \in \mathcal{W} \textit{ is fresh.}$$

Finally, given two disjoint sequences of variables $\langle x_1, \ldots, x_n\rangle$ and $\langle y_1, \ldots, y_n\rangle$ in V, we define $\delta^{H_V}_{\langle x_1,\ldots,x_n\rangle,\langle y_1,\ldots,y_n\rangle} = \{x_i = y_i \mid i = 1, \ldots, n\}$.

2.3 Abstract interpretation

Abstract interpretation [8] is a theory developed to reason about the abstraction relation between two different domains (the *concrete* and the *abstract* domain).

Definition 2. *Let $(C, \leq)$ and $(A, \preceq)$ be two complete lattices (the concrete and the abstract domain). A* Galois connection *$\langle\alpha, \gamma\rangle : (C, \leq) \rightleftharpoons (A, \preceq)$ is a pair of maps $\alpha : C \to A$ and $\gamma : A \to C$ such that α and γ are monotonic, for each $x \in C$ we have $x \leq (\gamma \circ \alpha)(x)$, and for each $y \in A$ we have $(\alpha \circ \gamma)(y) \preceq y$. Moreover, a* Galois insertion *(of $(C, \leq)$ into $(A, \preceq)$) $\langle\alpha, \gamma\rangle : (C, \leq) \rightleftharpoons (A, \preceq)$ is a Galois connection where $\alpha \circ \gamma = Id_A$.*

The map α (γ) is called the *abstraction* (*concretisation*). The composition of Galois insertions is a Galois insertion.

In a Galois insertion, γ is one-to-one if and only if α is onto if and only if $\alpha \circ \gamma = Id_A$, α is additive and γ is co-additive, the abstraction map uniquely

[1] Note that this is not restrictive because the names of existential variables are irrelevant: given an existential Herbrand constraint $\exists_W c$, the constraint $\exists_{W'} c[W'/W]$ is equivalent to it. Hence we can always assume existential Herbrand constraints to be renamed apart with respect to existential variables.

determines the concretisation map and vice versa. Namely, $\gamma = \lambda y.\ \sqcup_C \{x \in C \mid \alpha(x) \preceq y\}$ and $\alpha = \lambda x.\ \sqcap_A \{y \in A \mid x \leq \gamma(y)\}$. Conversely, if C and A are complete lattices and $\alpha : C \to A$ is additive or $\gamma : A \to C$ is co-additive, then $\langle \alpha, \gamma \rangle$ is a Galois connection from C to A.

When, in a Galois connection $\langle \alpha, \gamma \rangle$, γ is not injective, several distinct elements of the abstract domain $(A, \preceq)$ have the same meaning (by γ). This is usually considered useless.

A map $\rho : L \to L$ is a *closure operator* if it is monotonic, extensive and idempotent. The equivalence between Galois insertions and closure operators is well-known [2]. In view of this equivalence, abstract domains can be studied independently from the representation chosen for their abstract objects. Assume L to be a complete lattice playing the role of the concrete domain.

Theorem 1 ([7]). *Let $A \subseteq L$ be a complete lattice and $\rho : L \to L$. We have that $\langle \rho, id \rangle : (L, \leq_L) \leftrightharpoons (A, \leq_L)$ is a Galois insertion if and only if ρ is an upper closure operator such that $\rho(L) = A$.*

Any closure operator $\rho : L \to L$ is identified by the set of its fixpoints, which is its image $\rho(L)$. Recall that $\rho(L)$ is a *Moore family* of L, i.e., a complete meet-sublattice of L (i.e., $\top_L \in \rho(L)$ and for any non-empty $Y \subseteq \rho(L)$ we have $\sqcap_L Y \in \rho(L)$). Moreover, any Moore family of L is the image of some upper closure operator on L. For any $X \subseteq L$, we denote by $\curlywedge(X) = \{\sqcap_L I \mid I \subseteq X\}$ the *Moore closure* of X, i.e., the least Moore family of L containing X.

Let $f : C^n \to C$ be a concrete operator and assume that $\tilde{f} : A^n \to A$. Then $\tilde{f}$ is correct with respect to f if and only if for all $y_1, \dots, y_n \in A$ we have $\alpha(f(\gamma(y_1), \dots, \gamma(y_n))) \preceq \tilde{f}(y_1, \dots, y_n)$. For each operator f, there exists an optimal (most precise) correct abstract operator $\tilde{f}$ defined as $\tilde{f}(y_1, \dots, y_n) = \alpha(f(\gamma(y_1), \dots, \gamma(y_n)))$.

Given an abstract domain A, a domain refinement operator R yields an abstract domain $R(A)$ which is more precise than A, i.e., which contains more points than A. Classical domain refinement operators are *reduced product* and *disjunctive completion* [10]. The reduced product $A \sqcap B$ of two domains A and B, both contained in C, is isomorphic to the Cartesian product of A and B, modulo the equivalence relation $\langle a_1, b_1 \rangle \equiv \langle a_2, b_2 \rangle$ if and only if $a_1 \sqcap b_1 = a_2 \sqcap b_2$. This means that pairs having the same *meaning* are identified.

Linear refinement, proposed in [11], allows to include in a domain the information relative to propagation of the abstract property of interest before and after the application of a concrete operator $\boxtimes$ (in our case, $\boxtimes$ is the point-wise extension of unification).

Let L and C be complete lattices such that $L \subseteq C$, and let $\boxtimes$ be a commutative partial map $\boxtimes : L \times L \rightharpoonup L$. For any $a, b \in L$, we define $a \multimap^{\boxtimes} b = \bigsqcup_C \{c \in C \mid \text{if } a \boxtimes c \text{ is defined then } a \boxtimes c \leq_C b\}$. The linear refinement domain $L \multimap^{\boxtimes} L$ is $L \multimap^{\boxtimes} L = L \sqcap \curlywedge\{a \multimap^{\boxtimes} b \mid a, b \in L\}$, which can be simplified into

$$L \multimap^{\boxtimes} L = \curlywedge\{a \multimap^{\boxtimes} b \mid a, b \in L\} \tag{1}$$

if the elements of L can be obtained as greatest lower bounds of arrows.

3 The domains *Down* and *Sol*

In this section we define a domain of downward closed sets of substitutions. Given a substitution θ, its downward closure represents the set of substitutions which are *consistent* with θ, i.e., which can be derived from θ with computation. For instance, if in a program point we have $y = \mathtt{f}(x)$, then it is possible that, as computation proceeds, we have $y = \mathtt{f}(\mathtt{g}(w))$ and $x = \mathtt{g}(w)$. It is not possible, however, that $y = \mathtt{g}(w)$. With this interpretation in mind, we can say that a downward closed set of substitutions contains exactly all substitutions which are consistent with the prosecution of computation. For instance, if S_1 is the (downward closed) set of substitutions which are consistent with a procedure call p_1 and S_2 is the (downward closed) set of substitutions which are consistent with a procedure call p_2, then $S_1 \cap S_2$ is the (downward closed) set of substitutions which are consistent with the calls p_1, p_2 and p_2, p_1.

Definition 3 (The *Down* **domain).** *We define the family of sets* $Down = \{Down_V\}_{V \in \wp_f(\mathcal{V})}$ *where* $Down_V = \wp{\downarrow}(\Theta_V)$. *Let* $\{S, S_1, S_2\} \subseteq Down_V$ *and* $x \in V$. *We define*

$$S_1 \star^{Down_V} S_2 = S_1 \cap S_2$$

$$\exists_x^{Down_V} S = \left\{ \theta' \in \Theta_V \,\middle|\, \begin{array}{l} \textit{there exist } \sigma \in S[n/x],\ \theta \in \Theta_{V \cup n, V} \\ \textit{such that } \theta|_V = \theta' \textit{ and } \theta \leq_{V \cup n} \sigma \end{array} \right\},$$

where $n \in \mathcal{V} \setminus V$ *is fresh,* $S[n/x] = \{\sigma[n/x] \mid \sigma \in S\}$ *and* $\sigma[n/x] \in \Theta_{V \cup n, V \cup n}$ *is defined as* $\sigma[n/x](x) = x$, $\sigma[n/x](n) = \sigma(x)[n/x]$ *and* $\sigma[n/x](y) = \sigma(y)[n/x]$ *if* $y \neq x$ *and* $y \neq n$.

Given two disjoint sequences $\langle x_1, \ldots, x_n \rangle$ *and* $\langle y_1, \ldots, y_n \rangle$ *in* V, *we define*

$$\delta^{Down_V}_{\langle x_1 \ldots, x_n \rangle, \langle y_1, \ldots, y_n \rangle} = \{\theta \in \Theta_V \mid \theta(x_i) = \theta(y_i) \textit{ for all } i = 1, \ldots, n\} .$$

While the definition of conjunction is the classical one for the case of downward closed sets of substitution (see, for instance, [15]), and is justified by the above considerations, it turns out that an *explicit* definition of restriction on downward closed sets of substitutions was never given. The conjunction operator is closed on $Down_V$, as it can be checked easily. The same holds for restriction.

Proposition 1. *Let* $S, S_1, S_2 \subseteq \Theta_V$.

i) If $S = {\downarrow} S$ *then* $\exists_x^{Down_V} S = {\downarrow}(\exists_x^{Down_V} S)$ *(restriction is closed on the set of downward closed sets of substitutions).*
ii) If $S_1 \subseteq S_2$, *then* $\exists_x^{Down_V} S_1 \subseteq \exists_x^{Down_V} S_2$ *(restriction is monotonic).*
iii) $S \subseteq \exists_x^{Down_V}(S)$ *(restriction is extensive).*

An existential Herbrand constraint $h \in H_V$ can be mapped into a downward closed set of substitutions through the map sol_V which yields the set of its solutions. However, this map is not onto.

Proposition 2. *If* Σ *contains at least a constant and a functor symbol, then* $\{\mathsf{sol}_V(h) \mid h \in H_V\} \subset \wp{\downarrow}(\Theta_V)$.

In spite of this result, the following proposition shows that $\star^{Down_V}$ and $\exists^{Down_V}$ are closed on the set $\mathsf{sol}_V(H_V)$.

Proposition 3. *Let $\{h, h_1, h_2\} \subseteq H_V$ and $\tilde{x}, \tilde{y} \subseteq V$ disjoint and of the same length.*

i) $\mathsf{sol}_V(h_1) \star^{Down_V} \mathsf{sol}_V(h_2) = \mathsf{sol}_V(h_1 \star^{H_V} h_2)$,
ii) $\exists_x^{Down_V} \mathsf{sol}_V(h) = \mathsf{sol}_V\left(\exists_x^{H_V} h\right)$ *for every* $x \in V$,
iii) $\delta_{\tilde{x},\tilde{y}}^{Down_V} = \mathsf{sol}_V(\delta_{\tilde{x},\tilde{y}}^{H_V})$.

Therefore, we can introduce the following definition.

Definition 4 (The *Sol* domain). *We define $Sol = \{Sol_V\}_{V \in \wp_f(\mathcal{V})}$, where $Sol_V = \{\mathsf{sol}_V(h) \mid h \in H_V\}$. The $\star^{Sol_V}$ and $\exists^{Sol_V}$ operators on Sol_V are the restriction of the corresponding operators of $Down_V$ to Sol_V. The diagonal elements of Sol_V are the diagonal elements of $Down_V$.*

Since sol_V is one-to-one and onto from H_V into Sol_V, and for every $\{h_1, h_2\} \subseteq H_V$ we have $h_1 \leq h_2$ if and only if $\mathsf{sol}_V(h_1) \subseteq \mathsf{sol}_V(h_2)$, we conclude that sol_V, endowed with the $\subseteq$ partial ordering, is isomorphic to H_V, and we know that the corresponding operations coincide (Proposition 3). The usefulness of *Sol* is that of presenting an existential Herbrand constraint as the set of its solutions. This will be very important in the next section.

4 The collecting semantics

Now we know that existential Herbrand constraints are essentially the same as their sets of solutions. This isomorphism can be lifted to an isomorphism between $\wp(H)$ and $\wp(Sol)$, the domains used for the collecting semantics. The domain $\wp(Sol)$ is not exactly what we are looking for. Indeed, we want to be able to represent every downward closed set of substitutions rather every set of sets of solutions. This means that we want to use $Down_V$ to represent our abstract properties. We prove here that there is a Galois insertion from $\wp(Sol_V)$ into $Down_V$ for every $V \in \wp_f(\mathcal{V})$.

We use $\cup$ as abstraction map from $\wp(Sol_V)$ into $Down_V$. The map $\cup$ is onto from $\wp(Sol_V)$ into $Down_V$. This is because every $d \in Down_V$ can be written as the infinite union $d = \cup\{\downarrow\{\theta\} \mid \theta \in d\}$ and every single $\downarrow\{\theta\}$ belongs to Sol_V, as shown by the following proposition.

Proposition 4. *Given $\theta \in \Theta_V$ we have* $\mathsf{sol}_V(\mathsf{Eq}(\theta)) = \downarrow\{\theta\}$.

It can be shown that $\cup$ is additive from $\wp(Sol_V)$ into $Down_V$.

Proposition 5. *Given $\{p_i\}_{i \in I} \subseteq \wp(Sol_V)$ with $I \subseteq \mathbb{N}$, we have* $\cup\left(\vee_{i \in I}^{\wp(Sol_V)} p_i\right) = \vee_{i \in I}^{Down_V} (\cup p_i)$.

The above proposition and the fact that $\cup$ is onto entail that $\cup$ induces a Galois insertion from $\wp(Sol_V)$ into $Down_V$. Moreover, it can be shown that the operators and the diagonal elements on $Down_V$ are the best abstraction through $\cup$ of the corresponding operators and diagonal elements on $\wp(Sol_V)$.

Proposition 6. *Let* $\{p_1, p_2, p\} \subseteq \wp(Sol_V)$ *and* $\tilde{x}$ *and* $\tilde{y}$ *be two disjoint sequences in* V *of the same length.*

i) $\cup \left(p_1 \star^{\wp(Sol_V)} p_2\right) = (\cup p_1) \star^{Down_V} (\cup p_2)$.
ii) $\cup \left(\exists_x^{\wp(Sol_V)} p\right) = \exists_x^{Down_V} (\cup p)$.
iii) $\cup \left(\delta_{\tilde{x},\tilde{y}}^{\wp(Sol_V)}\right) = \delta_{\tilde{x},\tilde{y}}^{Down_V}$.

5 Type systems and type domains

We have shown that the collecting semantics over existential Herbrand constraints can be abstracted into a semantics over the domain *Down*. This domain allows to represent *every* downward closed property of logic programs. In general, the full power of *Down* is not needed. For instance, we might be interested in *some* downward closed property, like a set of types and their dependencies. This means that we want to abstract *Down* into a domain for a specific *type system*, where a type system specifies which downward closed properties we are interested in. From now on the abstraction becomes type-dependent.

Definition 5. *A* type system *is* $\langle \Delta, \Sigma, I \rangle$ *where* Δ *and* Σ*, called* type signature *and* term signature*, respectively, are sets of function symbols with associated arities and* $I(\mathtt{f}^n)$ *is a total map from* $(\wp{\downarrow}(\mathsf{terms}(\Sigma, \mathcal{V})))^n$ *to* $\wp{\downarrow}(\mathsf{terms}(\Sigma, \mathcal{V}))$ *for every* $\mathtt{f}^n \in \Delta$*. We define* $[\![\mathtt{f}^0]\!]I = I(\mathtt{f}^0)$ *and* $[\![\mathtt{f}^n(t_1, \ldots, t_n)]\!]I = I(\mathtt{f}^n)([\![t_1]\!]I, \ldots, [\![t_n]\!]I)$.

Example 1 (Type system for groundness). Let $\mathsf{G} = \langle \{\mathtt{g}^0\}, \Sigma, I \rangle$ and let $I(\mathtt{g}) = \mathsf{terms}(\Sigma, \emptyset)$. The functor symbol $\mathtt{g}$ is interpreted as the set of terms which do not contain variables.

Example 2 (Type system for non-freeness). Let $\mathsf{NF} = \langle \{\mathtt{nf}^0\}, \Sigma, I \rangle$ where $I(\mathtt{nf}) = \mathsf{terms}(\Sigma, \mathcal{V}) \setminus \mathcal{V}$. The functor symbol $\mathtt{nf}$ is interpreted as the set of terms which are not in $\mathcal{V}$, i.e., the set of non-free terms.

Example 3 (Type system for integers, lists and trees). Let $\mathsf{ILT} = \langle \{\mathtt{int}^0, \mathtt{list}^1, \mathtt{tree}^1\}, \Sigma, I \rangle$, where Σ contains $\{0^0, \mathtt{s}^1, []^0, [|]^2, \mathtt{void}^0, \mathtt{tree}^3\}$ and

$$I(\mathtt{int}) = \mu i.\{0\} \cup \{\mathtt{s}(n) \mid n \in i\} \qquad I(\mathtt{list}) = \lambda\tau.\mu l.\{[]\} \cup \{[h|t] \mid h \in \tau, t \in l\}$$
$$I(\mathtt{tree}) = \lambda\tau.\mu t.\{\mathtt{void}\} \cup \{\mathtt{tree}(n, l, r) \mid n \in \tau, l \in t, r \in t\} \ .$$

Since `list` and `tree` are unary, we can write `list(int)` and `tree(list(int))`.

Given a type system, we can define a type domain. Type domains are formed by transfinite propositional formulas, defined below.

Definition 6. *Given a type signature* Δ *and a set of variables* V*, the set* $\Phi_{\Delta,V}$ *of* transfinite formulas *over* Δ *and* V *is defined as the least set such that* $x \in t$ *is a transfinite formula, where* $x \in V$ *and* $t \in \mathsf{terms}(\Delta, \emptyset)$*, if* S *is a set of transfinite formulas then* $\wedge S$ *and* $\vee S$ *are transfinite formulas, if* ϕ_1 *and* ϕ_2 *are transfinite* *formulas then* $\phi_1 \Rightarrow \phi_2$ *is a transfinite formula and if* ϕ *is a transfinite formula then* $\neg\phi$ *is a transfinite formula. We write* $\phi_1 \wedge \phi_2$ *for* $\wedge\{\phi_1, \phi_2\}$, *true for* $\wedge\emptyset$ *and* $false$ *for* $\vee\emptyset$.

Definition 7 (Type domain). *Let* $\mathsf{T} = \langle \Delta, \Sigma, I \rangle$ *be a type system. A* type domain *for* T *is a family* $T = \{T_V\}_{V \in \wp_f(\mathcal{V})}$ *where* $T_V \subseteq \Phi_{\Delta,V}$ *contains the diagonal elements* $\delta^{T_V}_{\langle x_1,\ldots,x_n\rangle,\langle y_1,\ldots,y_n\rangle} = \wedge_{d \in \mathsf{terms}(\Delta,\emptyset), i=1,\ldots,n}\{(x_i \in d \Rightarrow y_i \in d) \wedge (y_i \in d \Rightarrow x_i \in d)\}$ *for every* $\langle x_1, \ldots, x_n\rangle$ *and* $\langle y_1, \ldots, y_n\rangle$ *in* V *disjoint, and is closed w.r.t.*

$$\phi_1 \star^{T_V} \phi_2 = \phi_1 \wedge \phi_2$$

$$\exists^{T_V}_x \phi = \vee \left\{ \phi[P/x] \;\middle|\; \begin{array}{l} \textit{there exists } t \in \mathsf{terms}(\Sigma, V) \textit{ such} \\ \textit{that } P = \{d \in \mathsf{terms}(\Delta,\emptyset) \mid t \in [\![d]\!]I\} \end{array} \right\},$$

where, for any $P \in \wp(\mathsf{terms}(\Delta, \emptyset))$,

$$(x \in t)[P/x] = \begin{cases} true & \textit{if } \cap_{p \in P}[\![p]\!]I \subseteq [\![t]\!]I \\ false & \textit{otherwise} \end{cases} \qquad (y \in t)[P/x] = (y \in t)$$

and the other cases are similar.

Note that if $\mathsf{terms}(\Delta, \emptyset)$ is finite then the set $\Phi_{\Delta,V}$ is finite too, and the resulting type domains are finite. Moreover, the conjunction and the restriction operations become effective, while they are not computable in the general case. Finally, note that, for the type domains of Examples 1 and 2, restriction becomes the classical Schröder elimination [1], since $\mathsf{terms}(\Delta, \emptyset)$ contains only a constant symbol whose evaluation is not empty.

Definition 8. *Given a type system* $\mathsf{T} = \langle \Delta, \Sigma, I \rangle$, *we define the map* $[\![]\!]_\mathsf{T} : \Phi_{\Delta,V} \to (\Theta_V \to \{0, 1\})$ *as*

$$[\![x \in t]\!]_\mathsf{T}\sigma = \begin{cases} 1 & \textit{if } \sigma(x) \in [\![t]\!]I \\ 0 & \textit{otherwise} \end{cases} \qquad [\![\wedge S]\!]_\mathsf{T}\sigma = \begin{cases} 1 & \textit{if } [\![\phi]\!]_\mathsf{T}\sigma = 1 \textit{ for any } \phi \in S \\ 0 & \textit{otherwise.} \end{cases}$$

The other cases are defined similarly. When it is clear from the context, we write $[\![]\!]$ *for* $[\![]\!]_\mathsf{T}$. *Given* $V \in \wp_f(\mathcal{V})$ *and* $\phi_1, \phi_2 \in \Phi_{\Delta,V}$, *we define* $\phi_1 \leq_{\mathsf{T},V} \phi_2$ *if and only if for every* $\theta \in \Theta_V$ *we have that* $[\![\phi_1]\!]_\mathsf{T}\theta = 1$ *entails* $[\![\phi_2]\!]_\mathsf{T} = 1$. *When it is clear from the context, we write* $\leq$ *for* $\leq_{\mathsf{T},V}$. *We define* $\phi_1 \equiv_{\mathsf{T},V} \phi_2$ *if and only if* $\phi_1 \leq_{\mathsf{T},V} \phi_2$ *and* $\phi_2 \leq_{\mathsf{T},V} \phi_1$. *This equivalence is called* (T, V)-equivalence. *Again, we drop the subscripts* T *and* V *when it is clear from the context.*

From now on, if $T = \{T_V\}_{V \in \wp_f(\mathcal{V})}$ is a type domain and $V \in \wp_f(\mathcal{V})$, every transfinite formula in T_V will stand for its (T, V)-equivalence class. Note that, for every $V \in \wp_f(\mathcal{V})$, T_V is a complete lattice w.r.t. $\leq_{\mathsf{T},V}$, since by Definition 7 it is completely $\wedge$-closed and topped (since $true = \wedge\emptyset$). We must check that the operations $\star^{T_V}$ and $\exists^{T_V}$ of Definition 7 are independent from the representants chosen for the (T, V)-equivalence classes. This is obvious for conjunction. For restriction, it is true if the following property holds.

P1: Let $T = \{T_V\}_{V \in \wp_f(\mathcal{V})}$ be a type domain for $\langle \Delta, \Sigma, I \rangle$. For every $V \in \wp_f(\mathcal{V})$ and $\{\phi_1, \phi_2\} \subseteq T_V$, if $\phi_1 \equiv \phi_2$ then $\phi_1[P/x] \equiv \phi_2[P/x]$ for any $x \in V$ and $P = \{d \in \mathsf{terms}(\Delta, \emptyset) \mid t \in [\![d]\!]I\}$ with $t \in \mathsf{terms}(\Sigma, V)$.

For any $V \in \wp_f(\mathcal{V})$, a Moore family of $Down_V$ (i.e., an abstract interpretation of $Down_V$) can be defined once a type domain is given.

Definition 9. *Given a type domain $T = \{T_V\}_{V \in \wp_f(\mathcal{V})}$ for the type system $\langle \Delta, \Sigma, I \rangle$ and $\phi \in T_V$, we define $\gamma_{T_V}(\phi) = \{\theta \in \Theta_V \mid$ for all $\sigma \in \Theta_V$ s.t. $\sigma \leq \theta$ we have $[\![\phi]\!]_{\mathsf{T}}\sigma = 1\}$. Note that $\gamma_{T_V}(\phi) \in Down_V$ for any $\phi \in T_V$. Given $\phi_1, \phi_2 \in T_V$, we define $\phi_1 \equiv_{\gamma_{T_V}} \phi_2$ if and only if $\gamma_{T_V}(\phi_1) = \gamma_{T_V}(\phi_2)$. This equivalence relation will be called* γ_{T_V}-equivalence*. When it is clear from the context, we drop the subscript T_V from γ_{T_V}.*

It is easy to realise that $\equiv$ entails $\equiv_\gamma$, though the converse does not hold.

Example 4. Consider the type system G of Example 1 and the type domain $\{\Phi_{\Delta,V}\}_{V \in \wp_f(\mathcal{V})}$ for G. Consider V such that $\{x, y\} \subseteq V$. Let $\phi_1 = false$ and $\phi_2 = x \in \mathsf{g} \Rightarrow false$. Note that there is no θ such that $[\![\phi_1]\!]_{\mathsf{G}}\theta = 1$. But any substitution θ such that $\theta(x)$ is not ground satisfies $[\![\phi_2]\!]_{\mathsf{G}}\theta = 1$. Hence ϕ_1 and ϕ_2 are *not* (G, V)-equivalent. However, $\gamma(\phi_1) = \emptyset$ and $\gamma(\phi_2) = \emptyset$. This is obvious for ϕ_1. For ϕ_2, assume $\theta \in \gamma(\phi_2)$. Any instance of θ must belong to $\gamma(\phi_2)$. But this is a contradiction as soon as we consider an instance θ' of θ which makes x ground, because in such a case it would be $[\![false]\!]_{\mathsf{G}}\theta' = 1$.

Proposition 7. *Given a type domain $T = \{T_V\}_{V \in \wp_f(\mathcal{V})}$ for T, γ_{T_V} is co-additive and $\gamma_{T_V}(T_V)$ is a Moore family of $Down_V$ for any $V \in \wp_f(\mathcal{V})$.*

Since γ is co-additive and $Down_V$ and T_V are complete lattices, we conclude that γ is the concretisation map of a Galois connection from $Down_V$ into T_V. Let α (i.e., α_{T_V}) be the corresponding abstraction map. Since, in general, γ is not one-to-one (Example 4), we are not guaranteed to have a Galois insertion instead of just a Galois connection. This is what the property below requires.

P2: Let $T = \{T_V\}_{V \in \wp_f(\mathcal{V})}$ be a domain for T. For any $V \in \wp_f(\mathcal{V})$ and $\phi_1, \phi_2 \in T_V$, if $\phi_1 \equiv_\gamma \phi_2$ then $\phi_1 \equiv \phi_2$.

Definition 10. *Given a type system $\mathsf{T} = \langle \{\mathsf{c}^0\}, \Sigma, I \rangle$ and $V = \{x_1, \ldots, x_n\} \in \wp_f(\mathcal{V})$, $\phi \in \Phi_{\Delta,V}$ is called* positive *if and only if for any ground term $t \in \mathsf{terms}(\Sigma, \emptyset)$ such that $t \in [\![\mathsf{c}]\!]I$, we have $[\![\phi]\!]\{x_1 \mapsto t, \ldots, x_n \mapsto t\} = 1$.*

Definition 11. *A type domain $T = \{T_V\}_{V \in \wp_f(\mathcal{V})}$ for the type system $\langle \Delta, \Sigma, I \rangle$ is* positive *if Δ is formed by just one constant, i.e., $\Delta = \{\mathsf{c}^0\}$ for some c, and for every $V \in \wp_f(\mathcal{V})$ and every $\phi \in T_V$, ϕ is positive (Definition 10).*

A type domain $T = \{T_V\}_{V \in \wp_f(\mathcal{V})}$ for the type system $\mathsf{T} = \langle \Delta, \Sigma, I \rangle$ is structural *if and only if T is* structural*, i.e., if and only if for every $V \in \wp_f(\mathcal{V})$ and $t_1, \ldots, t_n \in \mathsf{terms}(\Sigma, V)$ there exists $\sigma \in \Theta_V^V$ such that, for every $i = 1, \ldots, n$, the term $t_i\sigma$ is ground and for every $d \in \mathsf{terms}(\Delta, \emptyset)$ we have $t_i \in [\![d]\!]I$ if and only if $t_i\sigma \in [\![d]\!]I$.*

In a structural type domain, every finite set of terms can be instantiated into a finite set of ground terms with the same type properties. The following proposition shows the importance of positive or structural type domains.

Proposition 8. *Every positive or structural type domain satisfies both properties P1 and P2.*

Proposition 9. *Let $T = \{T_V\}_{V \in \wp_f(\mathcal{V})}$ be a type domain for the type system $\langle \Delta, \Sigma, I \rangle$, satisfying property P1.*

i) The operator $\star^{T_V}$ is always correct w.r.t. $\star^{Down_V}$, and it is its best possible approximation if property P2 holds for T.
ii) The operator $\exists^{T_V}$ is always correct w.r.t. $\exists^{Down_V}$, and it is its best possible approximation if T is positive or structural.
iii) The element $\delta^{T_V}_{\tilde{x},\tilde{y}}$ is a correct approximation of $\delta^{Down_V}_{\tilde{x},\tilde{y}}$, and it is its best possible approximation if T is positive or structural.

In conclusion, a positive or structural type domain enjoys interesting properties. Namely, restriction is well-defined (property P1), a Galois insertion can be established between $Down_V$ and $Type_V$ (property P2), the operators over $Type_V$ are the best possible approximations of the corresponding operators over $Down_V$ and the diagonal elements of $Type_V$ are exactly the abstraction of the diagonal elements of $Down_V$ (Proposition 9).

It is not easy to apply the definition of structural type domain (Definition 11). Therefore, we provide a sufficient condition which entails that a type domain is structural, and shows how large the class of structural type domains is.

Proposition 10. *Let $T = \{T_V\}_{V \in \wp_f(\mathcal{V})}$ be a type domain for the type system $\langle \Delta, \Sigma, I \rangle$. Assume there exists a ground term $t \in \mathsf{terms}(\Sigma, \emptyset)$ such that for every $d \in \mathsf{terms}(\Delta, \emptyset)$ and for all $t' \in [\![d]\!]I$, every term obtained from t' by substituting some occurrences of t with variables of $\mathcal{V}$ is still in $[\![d]\!]I$. Then T is structural.*

6 Applications

Example 5. Consider the type system of Example 1 and the type domain $Pos_{\{\mathsf{g}\}} = \{Pos_{\{\mathsf{g}\},V}\}_{V \in \wp_f(\mathcal{V})}$ where $Pos_{\{\mathsf{g}\},V}$ is formed by the set of positive transfinite formulas in $\Phi_{\{\mathsf{g}\},V}$. Note that this set is finite. In [1] it is shown that $Pos_{\{\mathsf{g}\},V}$ is closed under conjunction and restriction and contains the diagonal elements. This type domain is well-known [1, 5, 6].

A formula is called definite if the set of its propositional models is closed under instantiation. Consider the type domain $Def_{\{\mathsf{g}\}} = \{Def_{\{\mathsf{g}\},V}\}_{V \in \wp_f(\mathcal{V})}$ where $Def_{\{\mathsf{g}\},V}$ is the set of (positive) definite formulas in $\Phi_{\{\mathsf{g}\},V}$. Even this type domain is well-known [1]. In [1] it is shown that the set of definite formulas is closed under conjunction and restriction and contains the diagonal elements.

Since $Pos_{\{\mathsf{g}\}}$ and $Def_{\{\mathsf{g}\}}$ are positive, we conclude that they are related to *Down* through a Galois insertion and that the operations and diagonal elements of Definition 7 are the best possible approximations of the corresponding operations and diagonal elements of *Down* (Proposition 9).

Example 6. The same construction done above for groundness can be applied to non-freeness (Example 2). The resulting type domains will be called $Pos_{\{\mathtt{nf}\}}$ and $Def_{\{\mathtt{nf}\}}$. Note that they are positive type domains.

Since Proposition 9 considers every positive type domain, we have generalised the result contained in [6] for the case of groundness. Therefore, we can use many *incarnations* of positive type analyses, combining them with a reduced product operation and using the operations and diagonal elements of Definition 7 as done in [3], where, however, the authors did not provide any justification of the correctness of this approach.

The following example shows that we can apply our theory to more complicated cases than those considered above, dealing with many types, dependencies among different types and polymorphism.

Example 7. Given the type system of Example 3, consider the type domain $ILT = \{ILT_V\}_{V \in \wp_f(\mathcal{V})}$ where $ILT_V = \Phi_{\Delta,V}$. Since Proposition 10 holds with $t = \mathtt{s}([])$, we conclude that ILT is structural, is related to *Down* through a Galois insertion and the operators and diagonal elements of Definition 7 are optimal w.r.t. the corresponding operators and diagonal elements of *Down*.

For groundness and non-freeness we considered only positive formulas (Examples 5 and 6), while we considered the whole set of formulas for ILT. Indeed, in the case of groundness or non-freeness, a variable can always eventually belong to the type, and a formula like $\neg(x \in \mathtt{g})$ has an empty concretisation. Instead, the formula $\neg(x \in \mathtt{int})$ has a clear meaning. Namely, it says that x is not and can never become an integer. For instance, it might be bound to a list.

Note that ILT_V is not a finite set, although V is finite, since we can write arbitrarily complex types, and the definition of the restriction operator (Definition 7) is not effective. In [12] it is shown how to overcome these problems using a domain of type dependencies with type variables which is an abstract interpretation of our type domains of transfinite formulas.

7 Type systems and linear refinement

We have shown that every type domain induces an abstract interpretation of *Down* (Proposition 7). In this section, we want to show how a hierarchy of abstract interpretations of *Down* can be defined starting from a basic one modelling just the type properties of interest. Every domain in this hierarchy will be the linear refinement of the previous one w.r.t. concrete conjunction. This means that our hierarchy will be a chain of domains which induce an approximation of concrete conjunction which is more and more precise as long as we refine. This generalises an analogous result about groundness analysis shown in [15].

Definition 12. *Let* $\mathsf{T} = \langle \Delta, \Sigma, I \rangle$ *be a type system and* $V \in \wp_f(\mathcal{V})$*. We define* $\mathbf{v}_t = \{\theta \in \Theta_V \mid \theta(v) \in [\![t]\!]I\}$ *with* $v \in V$ *and* $t \in \mathsf{terms}(\Delta, \emptyset)$*,* $Basic_{\mathsf{T},V} = \curlywedge\{\mathbf{v}_t \mid v \in V$ *and* $t \in \mathsf{terms}(\Delta, \emptyset)\}$*,* $Basic^{i+1}_{\mathsf{T},V} = Basic^{i}_{\mathsf{T},V} \xrightarrow{\star^{Down_V}} Basic^{i}_{\mathsf{T},V}$ *for* $i \geq 0$.

We show now what is $\rightharpoonup^{Down_V}$ for downward closed properties and that $Basic^{i+1}_{\mathsf{T},V}$ contains $Basic^{i}_{\mathsf{T},V}$, which is why we used Equation (1) in Definition 12.

Proposition 11. *Let $V \in \wp_f(\mathcal{V})$.*

i) If $\{d_1, d_2\} \subseteq Down_V$, $d_1 \rightharpoonup^{\star^{Down_V}} d_2 = \{\theta \in \Theta_V \mid$ for all $\sigma \leq \theta$ if $\sigma \in d_1$ then $\sigma \in d_2\}$.
ii) Given a type system T, $Basic^{i}_{\mathsf{T},V} \subseteq Basic^{i+1}_{\mathsf{T},V}$ for any $i \geq 0$.

For a large class of type systems, a *best* domain exists.

Proposition 12. *Let $\mathsf{T} = \langle \Delta, \Sigma, I \rangle$ be a structural type system. For any $V \in \wp_f(\mathcal{V})$, $Basic_{\mathsf{T},V}$ is condensing, $Basic^{2}_{\mathsf{T},V} = Basic^{i}_{\mathsf{T},V}$ for $i \geq 2$ and $Basic^{2}_{\mathsf{T},V} = (Basic_{\mathsf{T},V} \rightharpoonup^{Down_V} Basic_{\mathsf{T},V}) \rightharpoonup^{Down_V} Basic_{\mathsf{T},V}$.*

Note that we do not know if the analogous of Proposition 12 holds for the type system of a positive type domain (Definition 11). However, in [15] it is shown that Proposition 12 holds for the type system for groundness of Example 1, and that $Basic_{\mathsf{G},V}$ is isomorphic to $Pos_{\{\mathtt{g}\},V}$ (Example 5). The proofs of [15] hold for the type system for non-freeness too (Example 2). However, they cannot be generalised to every type system of a positive type domain.

The domains $Basic_{\mathsf{T},V}$, $Basic^{1}_{\mathsf{T},V}$ and $Basic^{2}_{\mathsf{T},V}$ can be represented easily.

Definition 13. *Let $\langle \Delta, \Sigma, I \rangle$ be a type system. Let $V \in \wp_f(\mathcal{V})$. We define $And_{\Delta,V} = \{\wedge S \mid S \subseteq \{v \in t \mid v \in V \text{ and } t \in \mathsf{terms}(\Delta, \emptyset)\}\}$, $Or_{\Delta,V} = \{\vee S \mid S \subseteq \{v \in t \mid v \in V \text{ and } t \in \mathsf{terms}(\Delta, \emptyset)\} \text{ and } S \neq \emptyset\}$, $Def_{\Delta,V} = \{A_1 \Rightarrow A_2 \mid \{A_1, A_2\} \subseteq And_{\Delta,V}\}$ and $Pos_{\Delta,V} = \{A \Rightarrow O \mid A \in And_{\Delta,V} \text{ and } O \in Or_{\Delta,V}\}$.*

Proposition 13. *Given a positive or structural type domain T for $\langle \Delta, \Sigma, I \rangle$, $Def_{\Delta,V}$ is isomorphic to $Basic^{1}_{\Delta,V}$ and $Pos_{\Delta,V}$ is isomorphic to $Basic^{2}_{\Delta,V}$.*

Therefore, for structural type domains $Pos_{\Delta,V}$ allows to approximate the concrete conjunction in the best possible way, among the abstract domains which do not consider the name of the functors (Proposition 12).

8 Conclusions

We have defined a large class of type domains that enjoy the same desirable properties of the well-known domain for groundness analysis [1, 5, 6]. This leads to the use of transfinite formulas and operators on transfinite formulas for the type analysis of logic programs. In [12] the authors tackle the problem of the finiteness of the analysis. Finiteness is achieved through the use of *type variables*, which allow one to represent infinite conjunctions by using a finite object. The resulting domains are made of logic programs, and the abstract operations are operations over logic programs.

It would be interesting to know if the condition of being positive or structural, which entails all the desirable properties of a type domain, is necessary or, at least, can be further weakened.

References

1. T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two Classes of Boolean Functions for Dependency Analysis. *Science of Computer Programming*, 31(1):3–45, 1998.
2. G. Birkhoff. Lattice Theory. In *AMS Colloquium Publication, third ed.*, 1967.
3. M. Codish and B. Demoen. Deriving Polymorphic Type Dependencies for Logic Programs Using Multiple Incarnations of Prop. In *Proc. of the first International Symposium on Static Analysis*, volume 864 of *Lecture Notes in Computer Science*, pages 281–296. Springer-Verlag, 1994.
4. M. Codish and V. Lagoon. Type Dependencies for Logic Programs Using ACI–Unification. In *Proceedings of the 1996 Israeli Symposium on Theory of Computing and Systems*, pages 136–145. IEEE Press, June 1996. Extended version to appear in Theoretical Computer Science.
5. A. Cortesi, G. Filè, and W. Winsborough. *Prop* Revisited: Propositional Formula as Abstract Domain for Groundness Analysis. In *Proc. Sixth IEEE Symp. on Logic In Computer Science*, pages 322–327. IEEE Computer Society Press, 1991.
6. A. Cortesi, G. Filè, and W. Winsborough. Optimal Groundness Analysis Using Propositional Logic. *Journal of Logic Programming*, 27(2):137–167, 1996.
7. P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proc. 6th ACM Symp. on Principles of Programming Languages*, pages 269–282, 1979.
8. P. Cousot and R. Cousot. Abstract Interpretation and Applications to Logic Programs. *Journal of Logic Programming*, 13(2 & 3):103–179, 1992.
9. N.J. Cutland. *Computability: An Introduction to Recursive Function Theory.* Cambridge University Press, 1980.
10. G. Filé, R. Giacobazzi, and F. Ranzato. A Unifying View on Abstract Domain Design. *ACM Computing Surveys*, 28(2):333–336, 1996.
11. R. Giacobazzi, F. Ranzato, and F. Scozzari. Building Complete Abstract Interpretations in a Linear Logic-based Setting. In G. Levi, editor, *Static Analysis, Proceedings of the 5th International Static Analysis Symposium SAS 98*, volume 1503 of *Lecture Notes in Computer Science*, pages 215–229. Springer-Verlag, 1998.
12. Giorgio Levi and Fausto Spoto. An Experiment in Domain Refinement: Type Domains and Type Representations for Logic Programs. In Catuscia Palamidessi, Hugh Glaser, and Karl Meinke, editors, *Principles of Declarative Programming*, volume 1490 of *Lecture Notes in Computer Science*, pages 152–169, Pisa, Italy, September 1998. ©Springer-Verlag.
13. K. Marriott and H. Søndergaard. Precise and Efficient Groundness Analysis for Logic Programs. *ACM Letters on Programming Languages and Systems*, 2(1–4):181–196, 1993.
14. A. Martelli and U. Montanari. An Efficient Unification Algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.
15. F. Scozzari. Logical Optimality of Groundness Analysis. In P. Van Hentenryck, editor, *Proceedings of the 4th International Static Analysis Symposium SAS'97*, volume 1302 of *Lecture Notes in Computer Science*, pages 83–97. Springer-Verlag, 1997.
16. Jan-Georg Smaus, Patricia Hill, and Andy King. Mode Analysis for Typed Logic Programs. In *Proc. of the LOPSTR'99 Workshop*, pages 163–170, Venice, Italy, September 1999.