# A Logical Model for Relational Abstract Domains

ROBERTO GIACOBAZZI and FRANCESCA SCOZZARI
Università di Pisa

In this article we introduce the notion of Heyting completion in abstract interpretation. We prove that Heyting completion provides a model for Cousot's reduced cardinal power of abstract domains and that it supplies a logical basis to specify relational domains for program analysis and abstract interpretation. We study the algebraic properties of Heyting completion in relation with other well-known domain transformers, like reduced product and disjunctive completion. This provides a uniform algebraic setting where complex abstract domains can be specified by simple logic formulas, or as solutions of recursive abstract domain equations, involving few basic operations for domain construction, all characterized by a clean logical interpretation. We apply our framework to characterize directionality and condensing in downward closed analysis of (constraint) logic programs.

Categories and Subject Descriptors: D.3.4 [**Programming Languages**]: Processors—*optimization*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*program analysis*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic

General Terms: Languages, Theory

Additional Key Words and Phrases: Abstract interpretation, closure operators, condensing analysis, constraint logic programming, directional types, domains, Heyting completion, intuitionistic logic, reduced cardinal power, static analysis

## 1. INTRODUCTION

Abstract interpretation has been introduced in 1977 by Patrick and Radhia Cousot as a general theory for specifying and then validating program analyses [Cousot and Cousot 1977]. In recent years, this theory has increasingly gained popularity as a theory for approximating the semantics of generic discrete dynamic systems, and it has been applied in many different areas of computer science (cf. Cousot [1996]). The idea of abstract interpretation is that the analysis of a system, e.g., a program, can be obtained as an approximation of its formal semantics. The (concrete) semantics of a programming language is traditionally specified by a concrete domain $C$ of objects and a semantic function $[\![\cdot]\!] : Program \to C$, associating with

A preliminary version of this article appeared in *Proceedings of the 9th International Symposium on Programming Languages, Implementations, Logics, and Programs* (*PLILP '97*), Southampton UK.

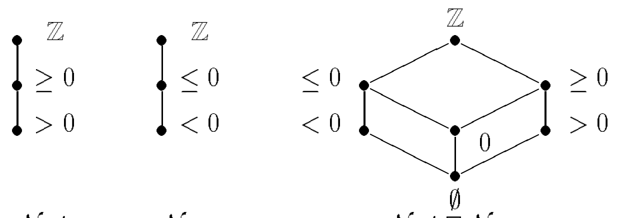
Authors' addresses: R. Giacobazzi, Università di Pisa, Dipartimento di Informatica, Corso Italia 40, 56125 Pisa, Italy; email: giaco@di.unipi.it; F. Scozzari, Università di Pisa, Dipartimento di Informatica, Corso Italia 40, 56125 Pisa, Italy; email: scozzari@di.unipi.it.

each (well-formed) program $P \in \mathit{Program}$ its semantics $[\![P]\!] \in C$. According to the classical Cousot and Cousot [1977] framework, an abstract interpretation is defined as a nonstandard (approximated) program semantics, which is obtained from the standard (or concrete) one by substituting the concrete domain $C$ of computation and its semantics $[\![\cdot]\!]$ with, respectively, an *abstract domain* $A$ and an *abstract semantics* $[\![\cdot]\!]^\sharp$.

One of the most fundamental facts of abstract interpretation is that most properties of the abstract semantics $\langle A, [\![\cdot]\!]^\sharp \rangle$, like precision, completeness, and compositionality, where $[\![\cdot]\!]^\sharp$ may involve complex operators, fixpoints etc., all depend on the notion of *abstraction*, which is precisely and uniquely specified by the chosen domain of properties $A$ (cf. Cousot and Cousot [1979]). Central in the construction of an abstract interpretation is therefore the notion of *domain*. A domain of properties is a set of mathematical objects which represent the properties about the dynamic system we are interested in. The notion of approximation is encoded in domains by a suitable partial order, and both $C$ and $A$ are assumed to be complete lattices with respect to this order. The orderings on the concrete and abstract domains describe the relative precision of domain values, somehow in a dual fashion with respect to the standard domains for denotational semantics—the top element representing no information. Therefore, $a \in A$ approximates $c \in C$ if $a$ represents some concrete object $\tilde{c} \in C$ such that $c \leq \tilde{c}$. Consider for instance the concrete lattice of sets of integers $\wp(\mathbb{Z})$, whose points are traditionally used to represent the set of possible values of integer variables at a given program point. The following simple complete lattice $\mathcal{N}at$ is an abstract domain which provides information to detect whether an integer variable is a strictly positive number. In this case, the elements: $\mathbb{Z}$, $\geq 0$, and $> 0$ represent the corresponding sets of integers:

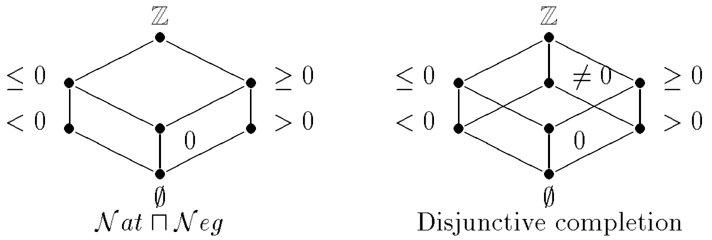
$$\begin{array}{l} \bullet\ \mathbb{Z} \\ \mid \\ \bullet\ \geq 0 \\ \mid \\ \bullet\ > 0 \end{array}$$

Because of the key role of domains in abstract interpretation, any formal method to compare or transform abstract interpretations is inherently based on corresponding methods to compare and transform abstract domains: *modifying domains corresponds to modify abstract interpretations.* The foundation of a theory of domains for abstract interpretation was fixed by Cousot and Cousot [1979], and later expanded by Giacobazzi and Ranzato [1997b]. A number of operations have been designed to systematically construct new abstract domains from given ones, by refinement [Cortesi et al. 1994; Cousot and Cousot 1979; 1994; Filé and Ranzato 1994; Giacobazzi and Ranzato 1998; 1997a; Jensen 1992; Nielson 1985], simplification [Giacobazzi and Ranzato 1997a], decomposition [Cortesi et al. 1997], and compression [Giacobazzi and Ranzato 1996], and some of them have been included as tools for design aid in modern systems for program analysis, like in GAIA [Cortesi et al. 1994], PLAI [Codish et al. 1995], and System Z [Yi and Harrison 1993], providing high-level facilities to tune program analysis in accuracy and costs. Most of these operations arise from the common logical operations on program properties. A typical example is *reduced product* [Cousot and Cousot 1979], which is, among these operations, the most common and widely used one (e.g., see Click and Cooper [1995], Codish et al. [1995], Muthukumar and Hermenegildo [1991], and Sundararajan and Conery [1992]). Reduced product corresponds to cartesian product of domains, where tuples having equivalent meaning, namely representing the same concrete property, are identified, viz., reduced. Consider for instance the domains $\mathcal{N}at$ and $\mathcal{N}eg$ depicted below. The reduced product of the two domains is a new richer domain, denoted by $\mathcal{N}at \sqcap \mathcal{N}eg$, which includes both the given domains and is closed under intersection. Intersection plays the role of conjunction of properties. For instance, the new object 0 denotes the conjunctive property for a program variable of being simultaneously both nonnegative ($\geq 0$) and nonpositive ($\leq 0$). From the domain point of view, the reduced product of two domains corresponds to the simplest (most abstract) domain which is more precise than the two given domains, i.e., which derives at least the same properties, as follows:




$\mathcal{N}at$ $\mathcal{N}eg$ $\mathcal{N}at \sqcap \mathcal{N}eg$

Another well-known operation for domain transformation, which has a clean logical interpretation, is *disjunctive completion*, also introduced in Cousot and Cousot [1979]. The disjunctive completion of a domain is the most abstract domain able to represent the concrete disjunction of its objects. This operation has been widely used in static program analysis (e.g., see Cousot and Cousot [1994; 1992], Filé and Ranzato [1998], and Jensen [1992]) to improve precision in case of multiple branches in approximated computations, and more recently in analysis of reactive systems and abstract model checking [Dams et al. 1997]. From a logical point of view, disjunctive completion upgrades a domain with disjunctive information: the disjunctive completion of a given domain $X$ is the least domain which includes $X$ and is closed under possibly infinite disjunctions. In the case of our example, where the disjunction on the concrete lattice $\wp(\mathbb{Z})$ is simply set union, the disjunctive completion of the domain $\mathcal{N}at \sqcap \mathcal{N}eg$ is obtained by adding the only missing disjunctive point $\neq 0$, which is obtained as union of $< 0$ and $> 0$ and which represents the disjunctive property for a program variable of being either positive or negative, as follows:

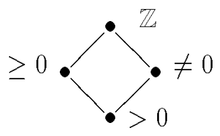



$\mathcal{N}at \sqcap \mathcal{N}eg$ Disjunctive completion

*The Problem.* The success of reduced product and disjunctive completion in systematic design of domains, for instance in program analysis, stems from their clean logical interpretation, respectively as conjunction and disjunction of domain's properties. Unfortunately, there are operations for domain construction and transformation for which a corresponding logical interpretation is not known. These include, among the others, the basic operations for transforming a domain to include some sort of relational information about its objects. The very first example of domain constructor for this purpose is Cousot's *reduced cardinal power*, introduced together with reduced product and disjunctive completion in their POPL'79 paper [Cousot and Cousot 1979]. In reduced cardinal power, monotone functions are used to represent relational information between properties of programs. A given domain is upgraded by first including all monotone functions between its objects (which corresponds to the lattice-theoretic notion of *cardinal power* [Birkhoff 1967]) and then by identifying (viz., reducing) those functions that represent the same relational property between domain's objects. The relational information supplied by reduced cardinal power is particularly important in program analysis, e.g., for transforming programs in such a way that they can be efficiently executed on parallel computers, and in comparative semantics, where similar relational information has been used to design compositional semantics for logic programs in Giacobazzi and Ranzato [1995]. In particular in program analysis, the possibility to tune the analysis in independent attribute and relational mode is probably one of the most attractive features of the abstract interpretation technology. The efficient but less precise *independent attribute* analysis method assigns properties to program objects, independently from the properties of other objects. In analysis, this is often insufficient for accurately detecting important program properties, like variable aliasing. This method is generalized by the more precise but costly *relational* one, which exploits at each program point the relations between the properties of program objects (cf. Jones and Muchnick [1981]), e.g., to express input/output relations. Although the relational information supplied by reduced cardinal power is clearly fundamental for most analyses, this domain constructor is rarely applied in the literature. To the best of our knowledge, the first use of reduced cardinal power after Cousot's original definition, has been considered about 20 years later in Giacobazzi and Ranzato [1995]. This is quite surprising because, as we prove in this article, most well-known domains for relational program analysis can be specified in this way. We believe that this failure is mostly due to the lack of a clean logical interpretation for this operation, which makes it difficult to understand the meaning of the objects in the reduced cardinal power of domains.

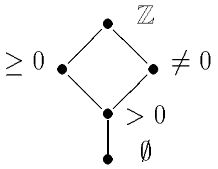
*The Idea: Implications to Model Reduced Cardinal Power.* Relational information in domains is traditionally used to express relations between properties of program's objects (e.g., variables) at different program points. The idea of this article originates by observing that, in order to exploit the relational information gathered at different program points, we need some objects in the domain which represent the implicational relation $\varphi \rightarrow \psi$: given two properties $\varphi$ and $\psi$, the property $\varphi \rightarrow \psi$ encodes the fact that whenever $\varphi$ holds, then $\psi$ holds too.

In the case of abstract interpretation, the properties of interest about the execution of a program are approximated by the objects of the abstract domain. The idea is therefore to enhance a given abstract domain of basic properties to include the space of all above implications built from every pair of its elements: for any pair of objects $a$ and $b$ in a domain $A$, a relational object, called *implication* and denoted by $a \to b$, is included in the enhancement of $A$, in such a way that the conjunction of $a$ and $a \to b$ is still approximated by $b$. In general, there are many possible choices for a concrete object $c$ such that the above *modus ponens* law holds, i.e., $a \wedge c \leq b$. In order to have a unique best representative object to express the above relational information, we are interested in those lattices where there exists a most abstract object, yet satisfying the modus ponens law. These are well-known lattices called *Heyting algebras*, which are precisely the models of *intuitionistic logic* [Birkhoff 1967; van Dalen 1994].[1] Intuitionistic logic is the weakest framework where relational information can be exploited by implicational objects. We can think of $\varphi \to \psi$ as an automatic method for transforming proofs of $\varphi$ into proofs of $\psi$: if we are able to prove $\varphi$ at given program point, then we are able to prove $\psi$ too. The general idea of interpreting implications as functions mapping proofs into proofs goes back to Heyting [Girard et al. 1989] and gives semantics to a logic weaker than classical logic, which is known as intuitionistic (constructive) logic. In this new logic, neither proofs by contradiction nor proofs by cases are allowed. The basic idea of intuitionistic logic is to model the notion of "provable sentence." For instance, the implicational relation $\varphi \to \varphi$ is interpreted as the identity function, which associates with each proof of $\varphi$ the same proof. It is easy to imagine that there always exists a proof constructor which simply returns the input proof, for every formula $\varphi$. Therefore $\varphi \to \varphi$ is always a true sentence, i.e., it is a tautology: $\varphi \to \varphi = \top$. What makes intuitionistic logic weaker than classical logic is the treatment of negation: $\neg\varphi$ means that $\varphi$ is not provable. It denotes the implication $\varphi \to \bot$, where $\bot$ represents a sentence with no possible proof. The intuitive meaning of $\varphi \to \bot$ is a method for transforming proofs of $\varphi$ in proofs of an unprovable sentence. As a consequence, the well-known tautology of classical logic $\varphi \vee \neg\varphi$ does not hold anymore, because in general we might not be able to exhibit a proof of either $\varphi$ or $\neg\varphi$, for some sentence $\varphi$. We lose therefore the tautology of classical logic $\varphi \to \psi = \neg\varphi \vee \psi$, since the formula on the right-hand side does not provide a constructive method to obtain a proof of $\psi$ from a proof of $\varphi$. The notion of negation is therefore weaker in intuitionistic logic than in classical logic. While $\varphi \wedge (\varphi \to \bot) = \bot$ always holds (it is a sort of consistency statement: we cannot prove both $\varphi$ and its contrary), it is not in general true that $\varphi \vee (\varphi \to \bot) = \top$, i.e., in some cases we are not able to prove neither $\varphi$ nor its contrary. When this latter equality holds, the whole system boils down to classical logic. Consider the domain $\mathcal{N}at$ described above. We are in the case of classical logic, because the concrete domain is the Boolean algebra $\wp(\mathbb{Z})$. In this case the sets of integers $\neq 0$ represents the implication $\geq 0 \to > 0$, which, according to the law $p \to q = \neg p \vee q$, is $\neg \geq 0 \cup > 0$.




[1]Some authors use the term *Brouwerian lattice* [Birkhoff 1967] or *frame* [Vickers 1989] to denote some algebras which are slightly different from Heyting algebras. But, in case of complete lattices, all the definitions become equivalent.

Clearly, Heyting algebras include, among others, Boolean algebras which are the models of classical logic [Birkhoff 1967]. As an example of non-Boolean Heyting algebras, consider the lattice depicted below. Note that $\geq 0 \rightarrow > 0$ cannot be described in terms of disjunction and complementation. In particular, the complement $\neg \geq 0$, namely an object $d$ such that $d \wedge \geq 0 = \emptyset$ and $d \vee \geq 0 = \mathbb{Z}$, does not exist . However, being an Heyting algebra, $\geq 0 \rightarrow > 0$ still exists, and it is $\neq 0$.




*Contribution of this Work.* In this article we introduce a new operation for abstract domain refinement, called *Heyting completion*, and we prove that it provides a precise and clean logical model for reduced cardinal power of domains. The Heyting completion of a domain $A$ corresponds to enhance $A$ with all the above-mentioned implicational objects. Heyting algebras provide here a weak enough generalization of classical implication, and therefore negation, which can be the right basis for giving a comprehensive logical account on a variety of relational information in abstract domains. We show that this fact is true, by proving that Heyting completion provides a logical characterization for a generalization of Cousot's reduced cardinal power of abstract domains. In particular we prove that the reduced cardinal power of a domain $A$ with respect to a domain $B$ is precisely the Heyting completion of $A$ with respect to $B$, i.e., the most abstract domain containing $a \rightarrow b$ with $a \in A$ and $b \in B$. Therefore, the domains obtained by reduced cardinal power of any domain with respect to itself gets an immediate and clean logical interpretation as the collection of all intuitionistic formulas built, without nested implication, in the fragment $\wedge, \rightarrow$ of propositional logic. One might think that this result is expected, because Heyting algebras are the models of untyped $\lambda$-calculus, and reduced cardinal power of domains is a space of functions. On the contrary, this is quite surprising, because the structure of domains in standard abstract interpretation theory is rather different from domains in standard denotational semantics. As the reader can check from Theorem 5.3, the structure of abstract domains is fully used in the proof. This result has both practical and theoretical consequences in abstract interpretation and static program analysis. Theoretically we establish a strong analogy between well-known domains for relational program analysis and Heyting algebras, which are models of intuitionistic logic and untyped $\lambda$-calculus. Practically, we make applicable to reduced cardinal power relevant results in intuitionistic logic, which may be particularly helpful to both characterize the least amount of information which is necessary to construct the reduced cardinal power of domains and provide the domain designer with a number of tools for specifying complex domains from simpler basic ones, directly in terms of the logical properties that their objects have to satisfy. In particular we study the algebraic properties of Heyting completion in relation with the remaining operators introduced in Cousot and Cousot [1979], i.e., reduced product and disjunctive completion. The result is that all the operations introduced in Cousot and Cousot [1979] for systematic domain design can be embedded into an algebra of domain operations, all characterized by a clean logical interpretation, where abstract domains can be specified by simple logic formulas. In this algebraic setting, we study the solutions of implicational domain equations, i.e., equations between abstract domains involving Heyting completion as basic domain constructor. A new family of abstract domains for analysis is introduced as solutions of these equations. These domains are closed under implication construction and represent somehow an upper bound on how much implicational information can be derived systematically from a given domain. We prove that they define precisely the family of abstract domains for which the abstraction function $\alpha$ satisfies one postulate of quantifiers in universal algebra: $\alpha(\alpha(a) \wedge b) = \alpha(a) \wedge \alpha(b)$, for all concrete objects $a$ and $b$. This result has interesting practical consequences in downward closed domains for analysis of (constraint) logic programs, where conjunction models abstract unification of constraints. Logic programming is a high-level language having a simple declarative semantics, and this is particularly helpful to show practical applications of our theory. We prove that the solutions of these domain equations characterize the domains which are *condensing* [Langen 1990], i.e., such that the analysis of a goal can be derived from a goal-independent analysis with the same precision as the one obtainable for the same goal in a goal-driven way. Moreover we apply Heyting completion as domain refinement to generalize directional type inference in logic programming. Type inference is general enough to include relevant examples of analyses in logic programs. We prove that the same construction by Heyting completion surfaces most of the well-known applications of type inference, from groundness [Armstrong et al. 1994] to more structured directional types [Aiken and Lakshman 1994; Bronsard et al. 1992]. This approach to directionality by systematic domain refinement has the advantage of being independent from specific applications. It is worth noting that our results, although applied in the context of logic programming, are obtained in the standard theory of abstract interpretation and are therefore independent from the choice of particular programming languages.

*Structure of the Article.* After some preliminary definitions in Section 2, we define the notion of Heyting completion in abstract interpretation in Section 3. In Section 4 we study the main algebraic and logic properties of Heyting completion with respect to other well-known operations for transforming domains. The main results are in Sections 5 and 7. In Section 5 we prove the equivalence between a generalization of reduced cardinal power and Heyting completion of abstract domains. In Section 7 we characterize the family of domains closed under Heyting completion, and we study the properties of solutions of recursive domain equations involving Heyting and disjunctive completion as basic domain constructors. The application of our operator to directional types, ground dependencies, and condensation in logic program analysis is in Sections 6 and 8. The article ends by addressing related works in Section 9 and further research directions in Section 10.

## 2. PRELIMINARIES

Throughout the article we assume familiarity with lattice theory (e.g., see Birkhoff [1967] and Davey and Priestley [1990]), closure operators (see Cousot [1978] and Morgado [1960]), and abstract interpretation (see Cousot and Cousot [1977; 1979]).

### 2.1 Notation and Basic Notions

Let $A$, $B$, and $C$ be sets. The powerset of $A$ is denoted by $\wp(A)$. If $X \subseteq A$, $\overline{X}$ is the set-theoretic complement of $X$ in $A$. If $A$ is a poset, we usually denote by $\leq_A$ the corresponding partial order. The dual of a poset $A$ is denoted by $A^{\mathrm{op}}$. If $A$ is a poset with bottom $\bot$ then an element $a \in A$ is an *atom* if $a$ covers $\bot$, i.e., $a \neq \bot$ and for all $x \in A$, $\bot < x \leq a$ implies $x = a$. $At(A)$ denotes the set of atoms in $A$. A complete lattice $A$ with partial ordering $\leq_A$, least upper bound (*lub*) $\vee_A$, greatest lower bound (*glb*) $\wedge_A$, least element $\bot_A$, and greatest element $\top_A$ is denoted $\langle A, \leq_A, \vee_A, \wedge_A, \top_A, \bot_A\rangle$. Note that, being $A$ complete, the lub and glb operations are defined on $\wp(A)$ and return an element of $A$. A complete lattice $A$ is *atomic* if every element $x \in A$ different from $\bot$ is the *lub* of the atoms which precede $x$, i.e., $x = \bigvee\{a \in At(A) \mid a \leq x\}$. *Dual-atoms* and *dual-atomicity* are defined by duality. If $A$ is a poset and $I \subseteq A$ then $\downarrow I = \{x \in A \mid \exists y \in I.\ x \leq_A y\}$. For $x \in A$, $\downarrow x$ is a shorthand for $\downarrow\{x\}$. $\wp^{\downarrow}(A)$ denotes the set of *ideals* of $A$, where $I \subseteq A$ is an ideal if $I = \downarrow I$. $\wp^{\downarrow}(A)$ is a complete lattice with respect to set-theoretic inclusion, where the join is set union, and the meet is set intersection. We write $f : A \longmapsto B$ to mean that $f$ is a total function from $A$ to $B$. $f : A \overset{m}{\longmapsto} B$ denotes that $f$ is a monotone function. If $C \subseteq A$ then $f(C) = \{f(x) \mid x \in C\}$. By $g \circ f$ we denote the composition $\lambda x.g(f(x))$. Let $\langle A, \leq_A, \vee_A, \wedge_A, \top_A, \bot_A\rangle$ and $\langle B, \leq_B, \vee_B, \wedge_B, \top_B, \bot_B\rangle$ be complete lattices. A function $f : A \longmapsto B$ is *(co-)additive*, denoted $f : A \overset{a(ca)}{\longmapsto} B$, if for any $C \subseteq A$, $f(\vee_A C) = \vee_B f(C)$ $(f(\wedge_A C) = \wedge_B f(C))$. $f$ is (co-)continuous when the above statement on (co-)additivity holds for chains $C$ only. $Ord$ denotes the proper class of ordinals, where $\omega \in Ord$ is the first limit ordinal.

### 2.2 Abstract Interpretation, Galois Connections and Closure Operators

The standard Cousot and Cousot [1977] theory of abstract interpretation is based on the notion of Galois connection. If $C$ and $A$ are posets and $\alpha : C \overset{m}{\longmapsto} A$, $\gamma : A \overset{m}{\longmapsto} C$ are such that $\forall c \in C.\ c \leq_C \gamma(\alpha(c))$ and $\forall a \in A.\ \alpha(\gamma(a)) \leq_A a$, then we call the quadruple $\langle C, \alpha, A, \gamma\rangle$ a *Galois connection* (G.c.) between $C$ and $A$. In the setting of abstract interpretation, $C$ and $A$ are called, respectively, *concrete* and *abstract domain*, and they are assumed to be complete lattices. If in addition $\forall a \in A.\ \alpha(\gamma(a)) = a$, then $\langle C, \alpha, A, \gamma\rangle$ is a *Galois insertion* (G.i.) of $A$ in $C$. Any G.c. $\langle C, \alpha, A, \gamma\rangle$ can be lifted to a G.i. by identifying (in an equivalence class) those objects in $A$ having the same image (meaning) in $C$. This process is known as *reduction* of the abstract domain, and the result is an abstract domain containing no redundant objects; namely, different abstract objects denoting the same concrete ones are collapsed into a single object. A reduction of a G.c. $\langle C, \alpha, A, \gamma\rangle$ is the G.i. $\langle C, \alpha, \alpha(C), \gamma\rangle$. In the following we will always consider abstract domains which are parts of a G.i. If $\langle C, \alpha, \alpha(C), \gamma\rangle$ is a G.i. then $\alpha(C)$ is isomorphic to $\gamma(\alpha(C))$, and $\langle C, \gamma \circ \alpha, \gamma(\alpha(C)), id\rangle$ is also a G.i. This allows us to consider only G.i.'s of type $\langle C, \rho, \rho(C), id\rangle$ with $\rho : C \longmapsto C$: The abstraction function and the abstract domain uniquely determinate each other. Hence, to specify a G.i. $\langle C, \rho, A, id\rangle$, it suffices to give either the abstraction function $\rho$ (and the abstract domain will be $\rho(C)$) or an abstract domain $A \subseteq C$ (and the abstraction function is the unique function $\rho : C \longmapsto C$ such that $\langle C, \rho, \rho(C), id\rangle$ is a G.i. and $\rho(C) = A$).

Cousot and Cousot [1979] observed that an often more practical way of reasoning about abstract domains in abstract interpretation is by closure operators. A function $\rho : L \longmapsto L$ is a closure operator on a complete lattice $L$ if it is monotone, idempotent (i.e., $\forall x \in L.\ \rho(\rho(x)) = \rho(x)$), and extensive (i.e., $\forall x \in L.\ x \leq_L \rho(x)$). The equivalence between G.i. and closure operators is well known (cf. Birkhoff [1967]). In view of this equivalence, abstract domains can be studied independently from the representation chosen for their abstract objects. In the following we briefly recall the relation between G.i. and closure operators. In the rest of the article $L$ is a complete lattice $\langle L, \leq_L, \vee_L, \wedge_L, \top_L, \bot_L\rangle$ playing the role of the concrete domain. The following result from Cousot and Cousot [1979] will be essential to move from one representation of domains into another.

THEOREM 2.1 [COUSOT AND COUSOT 1979]. *Let $A \subseteq L$ be a complete lattice and $\rho : L \longmapsto L$. Then the following properties are equivalent.*

—*$\langle L, \rho, A, id\rangle$ is a G.i.;*

—*$\rho$ is a closure operator such that $\rho(L) = A$.*

We identify with $uco(L)$, the set of all upper closure operators on $L$, the so-called *lattice of abstract interpretations* of $L$ (cf. Cousot and Cousot [1977, Section 7] and [1979, Section 8]), i.e., the complete lattice of all possible abstract domains (modulo isomorphic representation of their objects) of the concrete domain $L$. Thus, in the rest of the article, we often abuse notation and consider an abstract domain and the corresponding closure as the same object, and whenever we say that $A$ is an abstraction of $L$, we mean that $A$ is isomorphic to $\rho(L)$, for some closure $\rho \in uco(L)$. When closures are used to denote abstract domains, they will be denoted as sets, with Latin capital letters. Recall that $\rho(L)$ is a *Moore-family* of $L$, i.e., a complete meet-sublattice of $L$ ($\top_L \in \rho(L)$, and $\rho(L)$ is completely meet-closed, i.e., for any nonempty $Y \subseteq \rho(L)$, $\bigwedge_L Y \in \rho(L)$). Moreover, any Moore-family of $L$ is the image of some closure operator on $L$. Note that $\rho(L)$ is not, in general, a complete sublattice of $L$, since the join in $\rho(L)$ might be different from $\vee_L$. $\rho(L)$ is a complete sublattice of $L$ if and only if $\rho$ is additive. For any $X \subseteq L$, we denote by $\curlywedge(X) = \{\, \bigwedge_L I \mid I \subseteq L \,\}$ the *Moore-closure* of $X$, i.e., the least Moore-family of $L$ containing $X$.

The lattice of abstract interpretations inherits all the relevant algebraic properties of the lattice of all closure operations [Morgado 1960; Ward 1942]. Ward [1942, Theorem 4.2] proved that if $L$ is a complete lattice, then $\langle uco(L), \sqsubseteq, \sqcap, \sqcup, \lambda x.\top, \lambda x.x\rangle$ is a complete lattice, where for every $\rho, \eta \in uco(L)$, $\{\rho_i\}_{i\in I} \subseteq uco(L)$, and $x \in L$ we have

—$\rho \sqsubseteq \eta$ if and only if $\forall x \in L.\ \rho(x) \leq_L \eta(x)$, or, equivalently, $\rho \sqsubseteq \eta$ if and only if $\eta(L) \subseteq \rho(L)$;

—$(\sqcap_{i\in I}\rho_i)(x) = \bigwedge_{i\in I} \rho_i(x)$;

—$(\sqcup_{i\in I}\rho_i)(x) = x \Leftrightarrow \forall i \in I.\ \rho_i(x) = x$;

—$\lambda x.\top$ is the top element, whereas $\lambda x.x$ is the bottom element.

It is also known that, for any complete lattice $L$, $uco(L)$ is dual-atomic, where the dual-atoms are all and only those closures $\varphi_x = \{\top, x\}$, for any $x \in L \setminus \{\top\}$, i.e., if $y \leq x$ then $\varphi_x(y) = x$, otherwise $\varphi_x(y) = \top$. We call any $\varphi_x$ simply atomic closure. All domains in the lattice of abstract interpretations are partially ordered by the ordering on $uco(L)$, which corresponds precisely to the standard order used to compare domains with regard to their precision: $A_1$ is *more precise* than $A_2$ (or $A_2$ is an abstraction of $A_1$) if and only if $A_1 \sqsubseteq A_2$ in $uco(L)$, which holds if and only if $A_2 \subseteq A_1$. The top closure $\lambda x.\top$ is the top domain $\{\top\}$, which contains no information about $L$. The bottom closure $\lambda x.x$ is the bottom domain $L$ which is the most precise approximation of $L$. The *lub* and *glb* on $uco(L)$ have therefore the following meaning as operators on domains. Suppose $\{A_i\}_{i\in I} \subseteq uco(L)$: (i) $\sqcup_{i\in I} A_i$ is the most concrete among the domains which are abstractions of all the $A_i$'s, i.e., $\sqcup_{i\in I} A_i$ is the *least common abstraction* of all $A_i$'s; (ii) $\sqcap_{i\in I} A_i$ is (isomorphic to) the well-known *reduced product* (basically cartesian product plus reduction) of all the $A_i$'s, or, equivalently, it is the most abstract among the domains (abstracting $L$) which are more concrete than every $A_i$.

## 3. DOMAIN REFINEMENT BY COMPLEMENTS

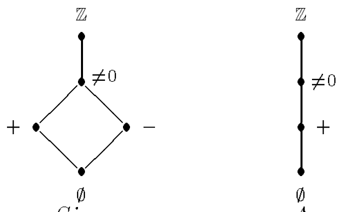
The notions of *domain refinement*, and its dual operations of *domain simplification* and *domain compression*, have been introduced in Filé et al. [1996] and Giacobazzi and Ranzato [1997b] as a generalization of most well-known operations for transforming domains. Intuitively a refinement is any operator performing an action of refinement with respect to the standard ordering of precision, e.g., by adding information to domains; while simplificators and compressors perform the dual action of "taking out" information from domains. The refinement $\Re(A)$ of a domain $A$ is a new domain which includes more information than $A$, i.e., $\Re(A) \sqsubseteq A$. Moreover a refinement respects the order of precision among domains—if we have more information then we add more information. Therefore a refinement is a reductive and monotone operator on $uco(L)$. Relevant examples of these operators include Cousot and Cousot's *reduced product* discussed above [Cousot and Cousot 1979], *disjunctive completion* [Cousot and Cousot 1979; 1994; 1992; Filé and Ranzato 1998; Jensen 1992], and *reduced cardinal power* [Cousot and Cousot 1979]; Nielson's *tensor product* [Nielson 1985]; Giacobazzi and Ranzato's *dual-Moore-set completion* [Giacobazzi and Ranzato 1995], and *complete extension* [Giacobazzi and Ranzato 1997a], and the *open product* and *pattern completion* of Cortesi et al. [1994].

In this section we gradually move from complementation in complete lattices to implication in abstract interpretation. In particular we introduce a new operation for domain refinement by complements. This provides a simple way to determine complementary information in abstract interpretation. As we will see later, this information is fundamental to enhance domains to represent relational information and to supply a logical interpretation of Cousot's reduced cardinal power of domains.

### 3.1 Complementary Information in Domains

Given two domain objects $a$ and $b$, we are interested in characterizing an object $c$ such that, when combined with $a$, the result is approximated by $b$, i.e., $a \wedge c \leq b$. The following example shows how this operation works on a simple domain for sign analysis.

*Example* 3.1.1. Consider the following lattice *Sign*, which is an abstract interpretation of $\wp(\mathbb{Z})$, and one of its strict abstractions $A$, both depicted below. The objects in $Sign$ and $A$ have the obvious meaning as fixpoints of closures on $\wp(\mathbb{Z})$. Here $- \in Sign$ represents the necessary complementary information which, when combined with $+ \in A$, gives $\emptyset$ as result.




$Sign$ $A$

A simple refinement introducing negative information in domains has been firstly studied in Filé et al. [1996]. This refinement is based on the notion of complement [Birkhoff 1967], viz. $a^*$ is a complement for $a$ in $L$ if $a \wedge_L a^* = \bot_L$ and $a \vee_L a^* = \top_L$. When the complement exists unique for an element $a$, it summarizes the common information shared by all the elements $c$ such that $c \wedge_L a = \bot_L$. However, the previous example shows that the requirement of complementation that $a \vee_L a^* = \top_L$ is too strong to enhance the simple domain of signs $A$ (note that $-$ is not the complement for $+$ in $Sign$). In order to characterize complementary information, we need at least to include those points for which $a \wedge_L a^* = \bot_L$ holds, namely those elements $a^*$ which do not share anything but the bottom with $a$. We weaken the definition of complementation in two ways. First, by relaxing that $a \vee_L a^* = \top_L$, the above observation boils down to the well-known notion of *pseudocomplement*, which, when it exists, is unique; indeed $a \to \bot_L = \bigvee_L \{\, c \mid a \wedge_L c = \bot_L \,\}$.[2] Second, we relativize this notion by considering the *pseudocomplement* of $a$ *relatively* to $b$ which, if it exists, is the unique element $a \to b \in L$ such that for any $x \in L$: $a \wedge_L x \leq_L b$ if and only if $x \leq_L a \to b$. Relative pseudocomplements, when they exist, are uniquely given by $a \to b = \bigvee_L \{\, c \mid a \wedge_L c \leq_L b \,\}$. In other words, the pseudocomplement of $a$ relative to $b$ exists (and it is given by the above formula) if and only if $a \wedge_L (\bigvee_L \{\, c \mid a \wedge_L c \leq_L b \,\}) \leq_L b$. It is worth noting that $a \to b = \bigvee_L \{\, c \mid a \wedge_L c \leq_L b \,\}$ is always defined in any complete lattice, for every pair of elements $a$ and $b$. This holds even if $L$ is not relative pseudocomplemented, i.e., when $a \to b$ may fail to be the relative pseudocomplement of $a$ and $b$. Because of this, in the following, when no hypotheses on $L$ are assumed, we denote by $a \to b$ simply the object $\bigvee_L \{\, c \mid a \wedge_L c \leq_L b \,\}$. The following properties are straightforward by definition.

Proposition 3.1.2. *Let $a, b \in L$.*

$$a \to \top_L = \top_L \tag{1}$$

$$\top_L \to a = a \tag{2}$$

$$\bot_L \to a = \top_L \tag{3}$$

$$a \wedge b \to a = \top_L \tag{4}$$

$$a \to a \vee b = \top_L \tag{5}$$

[2] Note, that while pseudocomplements, when they exist, are always unique, in general we may have infinitely many complements in complemented lattices [Birkhoff 1967].

We extend implication between objects to implication between sets in the obvious way.

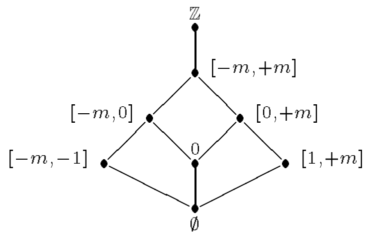
*Definition* 3.1.3. For any $A, B \subseteq L$, $A \longrightarrow B = \{\, a \rightarrow b \in L \mid a \in A, b \in B \,\}$.

The following property is immediate by definition.

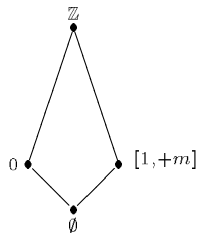
PROPOSITION 3.1.4. *Let $L$ be a complete lattice. $\lambda X, Y. X \longrightarrow Y : \wp(L) \longmapsto \wp(L)$ is argumentwise additive in $\langle \wp(L), \subseteq \rangle$.*

It is worth noting that in the above definition we do not require any further hypothesis on the structure of $L$, apart of being a complete lattice. As observed above, although $a \rightarrow b = \bigvee_L \{\, c \mid a \wedge_L c \leq_L b \,\}$ always exists unique in complete lattices, it may in general fail to be the pseudocomplement of $a$ relative to $b$. The following example shows this phenomenon in a simple lattice combining sign and overflow analysis.

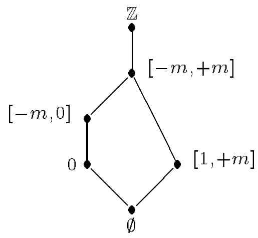
*Example* 3.1.5. Consider the following simple lattice $L$ for *sign and overflow analysis* of integer variables, which is an abstract interpretation of the standard lattice for *interval analysis* in Cousot and Cousot [1977]. In this example, $m \in \mathbb{N}$ is a constant, playing the role of *maxint*.




Note that $L$ is neither distributive, nor pseudocomplemented. If $N$ is the domain depicted below, which is a strict abstraction of $L$, although being defined, $0 \rightarrow \emptyset = 0 \rightarrow [1, +m] = [-m, +m]$ is neither the pseudocomplement of 0 nor the pseudocomplement of 0 relative to $[1, +m]$.




$N$




$N \longrightarrow N$

In the case of $[1, +m] \rightarrow 0 = [1, +m] \rightarrow \emptyset$ instead, we obtain precisely the relative pseudocomplement which is indeed $[-m, 0]$. Note that when an object $a \rightarrow b$ is not the relative pseudocomplement, it still provides an approximation for the information which is complementary to $a$. In this case, the complementary information for 0, which should be given by the union of the intervals $[-m, -1] \cup [1, +m]$, is approximated by their convex hull $0 \rightarrow \emptyset = [-m, +m]$.

| | | | |
|---|---|---|---|
| $(i)$. | $a \to a = \top$ | $(ii)$. | $a \wedge \bot = \bot$ |
| $(iii)$. | $(a \to b) \wedge b = b$ | $(iv)$. | $a \wedge (a \to b) = a \wedge b$ |
| $(v)$. | $a \to (b \wedge c) = (a \to b) \wedge (a \to c)$ | $(vi)$. | $(a \vee b) \to c = (a \to c) \wedge (b \to c)$ |

Fig. 1. Axioms for Heyting algebras.

### 3.2 Intuitionistic Implication

Considering the implication $a \to b$ as a domain object becomes particularly interesting in case it is properly a relative pseudocomplement, i.e., $a \wedge (a \to b) \leq b$. Lattices where this always happens, for every pair of objects, are called *Heyting algebras*. Heyting algebras play a central role in our construction, as they correspond to the models of propositional intuitionistic logic. In fact, meet, join, and relative pseudocomplement of Heyting algebras correspond precisely to conjunction, disjunction, and intuitionistic implication of intuitionistic logic (see Birkhoff [1967]). In this section we recall the notion of Heyting algebra and some of its properties that will be frequently used in the rest of the article.

A complete lattice $L$ is a *complete Heyting algebra* (cHa) if it is *relatively pseudocomplemented*, i.e., the pseudocomplement of $a$ relative to $b$ ($a \to b$) exists for every $a, b \in L$. A different (but equivalent) definition of (cHa) is the following: for each $X \subseteq L$ and $y \in L$, $y \wedge_L (\bigvee_L X) = \bigvee_L \{\, y \wedge_L x \mid x \in X \,\}$, which is the (complete) $\wedge$-distributivity law. From a logical point of view, a complete lattice $L$ is a cHa if we can define an (infix) binary operation $\to$ which satisfies, for every $a, b, c \in L$, the equalities in Figure 1. The following proposition recalls some key properties of cHa's [van Dalen 1994].

Proposition 3.2.1. *Let $L$ be a cHa, $\{b_i\}_{i \in I} \subseteq L$, and $a, b, c \in L$.*

$$(a \wedge b) \to c = a \to (b \to c) \tag{6}$$

$$a \to \bigwedge_{i \in I} b_i = \bigwedge_{i \in I} (a \to b_i) \tag{7}$$

Proof of (6). See van Dalen [1994]. $\square$

Proof of (7). Let $a \in L$ and $\{b_i\}_{i \in I} \subseteq L$. By definition, for any $i \in I$, $a \wedge (a \to b_i) \leq b_i$. Therefore we have $\bigwedge_{i \in I}(a \wedge (a \to b_i)) \leq \bigwedge_{i \in I} b_i$. However we have that $\bigwedge_{i \in I}(a \wedge (a \to b_i)) = a \wedge \bigwedge_{i \in I}(a \to b_i)$. Therefore $\bigwedge_{i \in I}(a \to b_i) \leq a \to \bigwedge_{i \in I} b_i$. The other inequality holds because $\forall k \in I.\ \bigwedge_{i \in I} b_i \leq b_k$ implies that $\forall k \in I.\ a \to \bigwedge_{i \in I} b_i \leq a \to b_k$. Therefore $a \to \bigwedge_{i \in I} b_i \leq \bigwedge_{k \in I}(a \to b_k)$. $\square$

### 3.3 Heyting Completion in Abstract Interpretation

In this section we introduce the notion of *Heyting completion* which formalizes, in a unique operation on domains, the ideas described above. We extend $\longrightarrow$ to become a binary domain transformation: given two abstract domains $X$ and $Y$, this is obtained precisely by the least Moore family (viz. the most abstract domain) containing $X \longrightarrow Y$.

*Definition* 3.3.1. Let $L$ be a complete lattice and $A, B \in uco(L)$. The *Heyting* *completion of* $A$ *with respect to* $B$ is $\curlywedge(A \longrightarrow B)$, and it is denoted by $A \stackrel{\curlywedge}{\longrightarrow} B$.

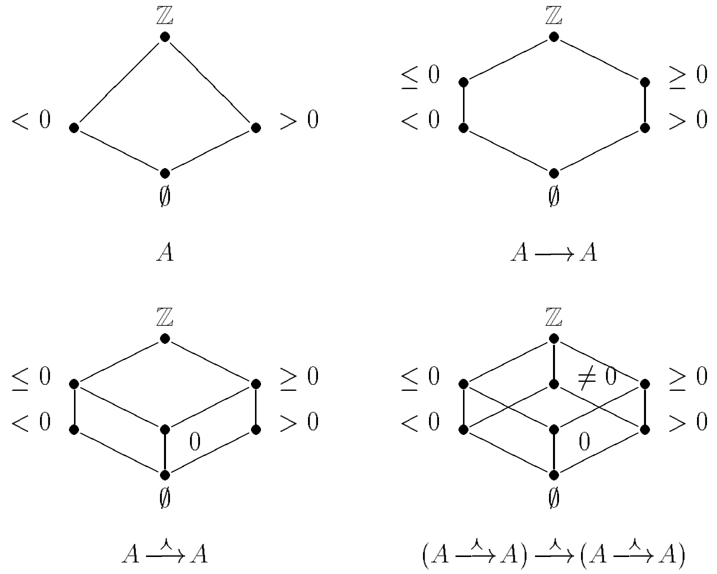
Note that the Heyting completion is defined for every complete lattice. Even if the concrete lattice is not a cHa, $\longrightarrow$ still captures some complementary information, as shown in Example 3.1.5.

THEOREM 3.3.2. *Let* $A \in uco(L)$. $\lambda X, Y.\ X \stackrel{\curlywedge}{\longrightarrow} Y$ *is argumentwise monotonic on* $uco(L)$ *and* $\lambda X.\ A \stackrel{\curlywedge}{\longrightarrow} X$ *is reductive on* $uco(L)$.

PROOF. Monotonicity follows from $\curlywedge$ and $\longrightarrow$ monotonicity in Proposition 3.1.4. Reductivity in its second argument holds by Eq. (2): for any $y \in Y$, $y = \top_L \to y$. Since $A$ is a Moore-family, then $\top_L \in A$ and therefore $Y \subseteq A \to Y \subseteq \curlywedge(A \to Y) = A \stackrel{\curlywedge}{\longrightarrow} Y$. $\square$

In the following, given an abstract domain $X$, $X \stackrel{\curlywedge}{\longrightarrow} X$ is called the Heyting completion of $X$. This is a unary operation which is, by Theorem 3.3.2, monotone and reductive on $uco(L)$. Therefore, the operation of Heyting completion $\lambda X.\ X \stackrel{\curlywedge}{\longrightarrow} X : uco(L) \longmapsto uco(L)$ is a domain refinement in the sense of Filé et al. [1996] and Giacobazzi and Ranzato [1997b]. Note that, while it is monotonic and reductive, in general it is not idempotent, as shown in next example.

*Example* 3.3.3. Consider the abstract domain $A$ for sign analysis depicted below, whose concrete domain is $\wp(\mathbb{Z})$ and whose objects have an obvious meaning as elements in $\wp(\mathbb{Z})$. Note that $A \longrightarrow A$ is strictly included in $A \stackrel{\curlywedge}{\longrightarrow} A$, which is strictly included in $(A \stackrel{\curlywedge}{\longrightarrow} A) \stackrel{\curlywedge}{\longrightarrow} (A \stackrel{\curlywedge}{\longrightarrow} A)$.




## 4. ALGEBRAIC PROPERTIES

In this section we show some basic algebraic properties of Heyting completion with respect to other operations for domain manipulation. To this aim, we embed the lattice of abstract domains $uco(L)$ into an algebra with primitive operators for domain refinement. In particular, we consider the meet operation ($\sqcap$) of closure operators, which is (isomorphic to) the well-known *reduced product* [Cousot and Cousot 1979] of abstract domains, the operation for *disjunctive completion* ($\curlyvee$) [Cousot and Cousot 1979; 1994; Giacobazzi and Ranzato 1996], and the above Heyting completion of abstract domains. As we will see, in most cases, the algebraic properties Heyting completion are direct consequences of analogous logical properties of intuitionistic implication.

### 4.1 Heyting Completion and Reduced Product

Reduced product was introduced in Cousot and Cousot [1979] as the basic operation for combining domains by including conjunction of domain objects in the refined domain. The following properties hold on generic complete lattices.

Proposition 4.1.1. *Let $L$ be a complete lattice, and $A, B, C \in uco(L)$.*

*a.* $A \xrightarrow{\curlywedge} \{\top_L\} = \{\top_L\}$
*b.* $\{\top_L\} \xrightarrow{\curlywedge} A = A$
*c.* $\{\top_L, x\} \xrightarrow{\curlywedge} A = A$ *if* $x \in L$ *and* $x \leq_L \bot_A$
*d.* $(A \xrightarrow{\curlywedge} C) \sqcap (B \xrightarrow{\curlywedge} C) = \curlywedge((A \cup B) \longrightarrow C)$[3]

Proof.

*a.* By Eq. (1), $\forall a \in A.\ a \to \top_L = \top_L$.
*b.* By Eq. (2), $\forall a \in A.\ \top_L \to a = a$.
*c.* By Eq. (4), $\forall a \in A.\ x \to a = (x \wedge a) \to a = \top_L$.
*d.* By definition of $\sqcap$, we have to prove that

$$\curlywedge((A \xrightarrow{\curlywedge} C) \cup (B \xrightarrow{\curlywedge} C)) = \curlywedge((A \cup B) \longrightarrow C).$$

By Lemma 3.1.4 $(A \cup B) \longrightarrow C = (A \longrightarrow C) \cup (B \longrightarrow C) \subseteq A \xrightarrow{\curlywedge} C \cup B \xrightarrow{\curlywedge} C$, and therefore $\curlywedge((A \cup B) \longrightarrow C) \subseteq \curlywedge((A \xrightarrow{\curlywedge} C) \cup (B \xrightarrow{\curlywedge} C))$. By monotonicity $A \longrightarrow C \subseteq (A \cup B) \longrightarrow C$, and therefore, $A \xrightarrow{\curlywedge} C \subseteq \curlywedge((A \cup B) \longrightarrow C)$. Similarly, $B \xrightarrow{\curlywedge} C \subseteq \curlywedge((A \cup B) \longrightarrow C)$ and $(A \xrightarrow{\curlywedge} C) \cup (B \xrightarrow{\curlywedge} C) \subseteq \curlywedge((A \cup B) \longrightarrow C)$. By monotonicity and idempotency of $\curlywedge$ we conclude the proof. ☐

Complete Heyting algebras play a central role in our construction, as they correspond precisely to the models of propositional intuitionistic logic. Let $L$ be a cHa. From a logical point of view, the Heyting completion of a domain is the collection of all intuitionistic formulas built with the $\wedge$ and $\to$ connectives, without nested implication. Hence, $A \xrightarrow{\curlywedge} A$ is the most abstract domain containing all the intuitionistic implications $a \to b$ for $a, b \in A$. The following results are consequences of the basic logical properties of implication in cHa.

Proposition 4.1.2. *Let $L$ be a cHa and $A, B_i, C \in uco(L)$ with $i \in I$.*

*a.* $A \xrightarrow{\curlywedge} \bigsqcap_{i \in I} B_i = \bigsqcap_{i \in I}(A \xrightarrow{\curlywedge} B_i)$
*b.* $(A \sqcap B) \xrightarrow{\curlywedge} C = A \xrightarrow{\curlywedge} (B \xrightarrow{\curlywedge} C)$
*c.* $A \xrightarrow{\curlywedge} \curlywedge(X) = \bigsqcap_{x \in X} A \xrightarrow{\curlywedge} \{\top_L, x\}$ *if* $X \subseteq L$.

[3]Note that $A \cup B$ may not be, in general, a Moore-family, and $A \sqcap B = \curlywedge(A \cup B)$.

PROOF.

*a.* $A \xrightarrow{\curlywedge} \bigsqcap_{i \in I} B_i \supseteq \bigsqcap_{i \in I}(A \xrightarrow{\curlywedge} B_i)$ is straightforward by monotonicity of $\xrightarrow{\curlywedge}$ and reduced product. Assume that $x \in A \longrightarrow \bigsqcap_{i \in I} B_i$. $x$ has the form $a \to \bigwedge_{j \in J} b_j$ where $b_j \in \cup_{i \in I} B_i$. By Eq. (7): $a \to (\bigwedge_{j \in J} b_j) = \bigwedge_{j \in J}(a \to b_j)$. We show that for each $j \in J$, $a \to b_j \in \bigsqcap_{i \in I}(A \xrightarrow{\curlywedge} B_i)$. Since $b_j \in \cup_{i \in I} B_i$, there exists $B_k$, $k \in I$, such that $b_j \in B_k$. It follows that $a \to b_j \in A \xrightarrow{\curlywedge} B_k$ and thus $a \to b_j \in \bigsqcap_{i \in I}(A \xrightarrow{\curlywedge} B_i)$. The result follows from the fact that $\bigsqcap_{i \in I}(A \xrightarrow{\curlywedge} B_i)$ is a Moore family, and therefore $\bigwedge_{j \in J}(a \to b_j) \in \bigsqcap_{i \in I}(A \xrightarrow{\curlywedge} B_i)$.

*b.* From Eq. (7), it follows that

$$a \to (\bigwedge_{j \in J}(b_j \to c_j)) = \bigwedge_{j \in J}(a \to (b_j \to c_j)) = \bigwedge_{j \in J}((a \wedge b_j) \to c_j)$$

which proves that $A \xrightarrow{\curlywedge} (B \xrightarrow{\curlywedge} C) \sqsubseteq (A \sqcap B) \xrightarrow{\curlywedge} C$.

For the other inclusion, by Eq. (6), it follows

$$\bigwedge_{i \in I, j \in J}(a_i \wedge b_j) \to c = (\bigwedge_{i \in I}(a_i) \wedge \bigwedge_{j \in J}(b_j)) \to c = \bigwedge_{i \in I}(a_i) \to (\bigwedge_{j \in J}(b_j) \to c).$$

*c.* One inclusion is straightforward by $\xrightarrow{\curlywedge}$ and $\sqcap$ monotonicity, and because, for any $x \in X$: $\curlywedge(X) \sqsubseteq \{\top_L, x\}$. We prove $A \xrightarrow{\curlywedge} \curlywedge(X) \sqsubseteq \sqcap_{x \in X} A \xrightarrow{\curlywedge} \{\top_L, x\}$. Let $x \in A \longrightarrow \curlywedge(X)$. Then $x$ has the form $a \to \bigwedge_{i \in I} b_i$ for some set $I$, $a \in A$ and $\{b_i\}_{i \in I} \subseteq X$. By Eq. (7), $a \to \bigwedge_{i \in I} b_i = \bigwedge_{i \in I}(a \to b_i)$, and by definition $a \to b_i \in A \xrightarrow{\curlywedge} \{\top_L, b_i\}$ for any $i \in I$. $\square$

As observed above, any abstract interpretation $A$ of a concrete domain $L$, i.e., $A \in uco(L)$, can be always decomposed into atomic domains of the form $\{\top_L, x\}$ for $x \in A$. In order to use the above result for decomposing the Heyting completion into the completion of simpler domains, we consider lattices generated by some suitable strict subsets of their points. Let $L$ be a complete lattice, and $A \in uco(L)$. We say that $A$ is *generated by the set* $B \subseteq A$ if $A = \curlywedge(B)$. Hence, by (c) in Proposition 4.1.2, Heyting completion can be constructed from *atomic domains*, i.e., closures of the form $\{\top_L, x\}$, by considering only the elements $x$ in a generating set $X$. This is particularly important for domains generated by meet-irreducible elements. An element $x$ of a complete lattice $L$ is *completely meet-irreducible* if $\forall Y \subseteq L.\ x = \bigwedge_L Y \Rightarrow x \in Y$. $MIrr(L)$ denotes the set of completely meet-irreducible elements of $L$ ($\top_L \notin MIrr(L)$). Most domains in semantics are generated by their meet-irreducible elements. They include *algebraic* [Birkhoff 1967; Davey and Priestley 1990] and *collecting domains* of the form $\wp(S)$ for some set $S$, which are typical domains in collecting semantics and analysis (e.g., see Cousot and Cousot [1994], Jones and Søndergaard [1987], and Mycroft and Nielson [1983]). In the case of algebraic domains, the set of meet-irreducible elements is the least set generating the domain (cf. Birkhoff [1967]). It is therefore immediate by (c) in Proposition 4.1.2 that, if $B \in uco(L)$ is algebraic and $L$ is cHa, then the following statement holds:

$$A \xrightarrow{\curlywedge} B = \bigsqcap_{x \in MIrr(B)} A \xrightarrow{\curlywedge} \{\top_L, x\} \tag{8}$$

This result can be improved by noting that the implications with meet-irreducible elements of $L$ on the right side are always reduced, i.e., if $b \in MIrr(L) \cap B$ then $a \to b \in B$. Therefore, we can get rid of them in our decomposition. The following result characterizes the new information added to $B$, by $A \xrightarrow{\curlywedge} B$, in terms of the meet-irreducible elements in $B$. Note that, in general, it does not hold that $MIrr(B) = MIrr(L) \cap B$, but $MIrr(B) \supseteq MIrr(L) \cap B$.

PROPOSITION 4.1.3. *If $L$ is cHa and $B \in uco(L)$ is algebraic, then*

$$A \xrightarrow{\curlywedge} B = B \sqcap \left( \bigsqcap_{x \in MIrr(B) \setminus MIrr(L)} A \xrightarrow{\curlywedge} \{\top_L, x\} \right).$$

PROOF. We show that for every $a \in A$ and $b \in MIrr(L) \cap B$ it holds $a \to b = b$ or $a \to b = \top$. The result then follows by Eq. (8). Let $a \in A$ and $b \in MIrr(L) \cap B$. If $a \leq b$ then $a \to b = \top$. Suppose $a \not\leq b$. Then we have

$$\begin{aligned} a \wedge a \to b \leq b \ &\Rightarrow\ (a \wedge a \to b) \vee b = b \\ &\Rightarrow\ (a \vee b) \wedge (a \to b \vee b) = b \\ &\Rightarrow\ (a \vee b) \wedge a \to b = b. \end{aligned}$$

Since $b \in MIrr(L)$, then either $b = a \vee b$ or $b = a \to b$ holds. By hypothesis, $a \not\leq b$. Therefore $b = a \to b$. □

## 4.2 Heyting Completion and Disjunctive Completion

In this section we study the relation between Heyting and disjunctive completion in abstract interpretation. Disjunctive completion was introduced in Cousot and Cousot [1979] to express *merge over all paths* data-flow analysis in fixpoint form. The idea is to lift a given abstract domain $A$ to the most abstract domain which includes $A$ and which is a complete (join-) sublattice of $L$, viz. no approximation is induced in the disjunctive completed domain by considering the join (i.e., disjunction) of abstract objects. The disjunctive completion of $A$ is defined as the most abstract domain which is an additive closure and includes $A$ (cf. Cousot and Cousot [1979] and Giacobazzi and Ranzato [1996]): $\curlyvee(A) = \sqcup\{\, X \in uco(L) \mid X \sqsubseteq A \text{ and } X \text{ is additive}\,\}$. A domain $A$ is disjunctive if $\curlyvee(A) = A$. The following result specifies the disjunctive completion of an abstract domain $A$ when the concrete domain is either completely distributive or a cHa with $A$ finite.

PROPOSITION 4.2.1. *If $L$ is completely distributive or $L$ is a cHa and $A$ is finite, then $\curlyvee(A) = \{\, \bigvee_L X \,|\, X \subseteq A\}$.*

PROOF. If $L$ is completely distributive, the result is immediate. Assume $L$ to be a cHa and $A$ to be finite. Therefore $\curlyvee(A)$ is finite too. Thus we only need to prove that for each $X, Y \subseteq A$, $(\bigvee_L X) \wedge (\bigvee_L Y) \in \curlyvee(A)$. Being $X$ and $Y$ finite, the thesis follows by $L$ distributivity. □

The next result follows immediately from the logical properties of intuitionistic implication, and it specifies the relation between Heyting and disjunctive completion.

PROPOSITION 4.2.2. *If $L$ is completely distributive or $L$ is a cHa and $A$ is finite, then $\curlyvee(A) \xrightarrow{\curlywedge} B = A \xrightarrow{\curlywedge} B$.*

PROOF. Assume $L$ to be completely distributive or $L$ to be a cHa and $A$ to be finite. Let $a \in \curlyvee(A) \longrightarrow B$. By Proposition 4.2.1, there exists $X \subseteq A$ such that $a = \bigvee_L X \to b$. If $L$ is a cHa and $A$ is finite, then $X$ is finite too, and the result follows by $L$ distributivity; otherwise, because if $L$ is complete distributive then $\bigvee_L X \to b = \bigwedge_{x \in X} x \to b$ from which follows the thesis. $\square$

The inverse operation to disjunctive completion, which is called *least disjunctive basis*, was introduced in Giacobazzi and Ranzato [1996] to characterize the least amount of information which is necessary to construct a given domain by disjunctive completion. Given an abstract domain $A$, the idea is to find the most abstract domain $X$ (when it exists) such that $\curlyvee(A) = \curlyvee(X)$. This domain, denoted $\Omega(A)$, is such that $A \sqsubseteq \Omega(A)$. Giacobazzi and Ranzato [1996] proved that $\Omega(A)$ exists under some hypotheses, e.g., when $L$ or $A$ are finite, or when $L^{\mathrm{op}}$ is a completely distributive algebraic lattice. In this case, if $JIrr(A)$ denotes the set of points in $A$ which are *completely join irreducible* in $L$, i.e., $JIrr(A) = \{ x \in A \mid \forall Y \subseteq A.\, x = \bigvee_L Y \Rightarrow x \in Y \}$, then $\Omega(A) = \curlywedge(JIrr(A))$. The following result, which is a consequence of Proposition 4.2.2, specifies the relation between Heyting completion and the least disjunctive bases operation.

COROLLARY 4.2.3. *If $L^{op}$ is a completely distributive algebraic lattice or $L$ is a cHa and $A$ is finite, then $A \xrightarrow{\curlywedge} B = \Omega(A) \xrightarrow{\curlywedge} B$.*

This result specifies that $A \xrightarrow{\curlywedge} B$ can be obtained by considering the more abstract, and hence less expensive, domain $\Omega(A)$. This provides a systematic method to simplify the construction of Heyting completions of abstract domains, because the implicational objects generated by combining disjunctive objects in $A$, i.e., those objects $x \to b$ where $x \in A$ is such that there exists $C \subseteq A$ and $x = \bigvee_L C$, can be fully reconstructed by conjunction of relative pseudocomplements of the form $c \to b$, with $c \in C$. Here, the least disjunctive basis plays the role of the least domain generating $A$ by disjunctive completion. By combining the above result with Proposition 4.1.3 we obtain the following result:

$$A \xrightarrow{\curlywedge} B = B \sqcap \left( \bigsqcap_{x \in MIrr(B) \setminus MIrr(L)} \Omega(A) \xrightarrow{\curlywedge} \{\top_L, x\} \right)$$

This result may be useful to implement Heyting completion of abstract domains, from the most abstract domains as arguments. Note that the implementation of the Heyting completion $A \xrightarrow{\curlywedge} B$ involving the least disjunctive basis of $A$ and an atomic decomposition of $B$ requires a lower number of reductions to identify pairs $a \to b$ having the same meaning. The following corollary is immediate by the above observations and can be applied in any cHa to represent the Heyting completion of any finite domain.

COROLLARY 4.2.4. *If $L$ is collecting or $L$ is a cHa and $A$ is finite, then*

$$A \xrightarrow{\curlywedge} A = \curlywedge(A \cup (JIrr(A) \longrightarrow (MIrr(A) \setminus MIrr(L)))).$$

It is known that it is always possible to represent any complete lattice in terms of the set of its join and meet-irreducible elements, and this yields the so-called *concept representation of lattices*. According to this reconstruction of lattices (cf. Davey and Priestley [1990]), a complete lattice $L$ is always isomorphic to the collection of all pairs $(A, B)$ such that $A \subseteq JIrr(L)$, $B \subseteq MIrr(L)$, and $B = \{ m \in MIrr(L) \mid \forall j \in A.\ j \leq_L m \}$ and $A = \{ j \in JIrr(L) \mid \forall m \in B.\ m \leq_L j \}$, ordered by set inclusion: $(A_0, B_0) \leq (A_1, B_1)$ if $A_0 \subseteq A_1$, or equivalently $B_1 \subseteq B_0$. Because $JIrr(L)$ and $MIrr(L)$ can be logarithmic in the size of $L$, this representation can be very efficient for finite lattices. Moreover, as shown in Freese et al. [1995], most known representations for lattices, e.g., by representing the $\leq$-relation, can be transformed into a concept representation in $O(n^{5/2})$, with $n = |L|$. Therefore, in view of the above results, the theory of concept lattices can be particularly helpful to provide efficient representation for finite domains and their Heyting completion. In order to represent the domain $A \xrightarrow{\curlywedge} A$ it is therefore sufficient to consider pairs in the concept representation of $A$, i.e., elements in $JIrr(A) \longrightarrow MIrr(A) \setminus MIrr(L)$. This representation is particularly efficient when $A = \wp(S)$, for some set $S$, and $A$ is a disjunctive domain. In this case an upper bound to the number of elements in $A \longrightarrow A$ is not $|A|^2$, but $2|A|$, because $|\Omega(A)| = log(|A|)$, $A = \bigcurlywedge(\{ S \setminus \{ x \} \mid x \in S \})$, and $|\{ S \setminus \{ x \} \mid x \in S \}| = log(|A|)$.

## 5. REDUCED CARDINAL POWER AND HEYTING COMPLETION

Reduced cardinal power has been introduced by Cousot and Cousot [1979] in the very last section of their POPL'79 paper. The idea is that, given a pair of abstract domains $A$ and $B$, a new, and more concrete domain can be systematically derived by considering the lattice of all monotone mappings from $A$ to $B$, which is known in lattice theory as the cardinal power with base $B$ and exponent $A$. The following result is the main result in Cousot and Cousot [1979, Section 10.2].

THEOREM 5.1 [COUSOT AND COUSOT 1979]. *Let $X$ be a set, and assume that $(\wp(X), \alpha_A, A, \gamma_A)$ and $(\wp(X), \alpha_B, B, \gamma_B)$ to be G.i.'s. Then $(\wp(X), \alpha, A \stackrel{m}{\longmapsto} B, \gamma)$ is a G.c. with $\alpha = \lambda P.\ \lambda x.\ \alpha_B(P \cap \gamma_A(x))$.*

The *reduced cardinal power with base $B$ and exponent $A$*, denoted $B^A$, is the domain $A \stackrel{m}{\longmapsto} B$, reduced with respect to concretization (cf. Cousot and Cousot [1979]). Monotone functions are considered equivalent, and therefore reduced, if they represent the same function between the objects of $A$ and $B$. It is worth noting that the definition given by Cousot and Cousot [1979] strongly depends on the Boolean structure of the complete lattice of assertions, which is always isomorphic to $\wp(X)$, for some set $X$, and on the properties of set intersection, which is conjunction of assertions in the early definition of Cousot and Cousot [1979]. Given any set $P \subseteq X$, the function $\lambda x.\ \alpha_B(P \cap \gamma_A(x)) \in B^A$ represents, on the base abstract domain $B$, the way $P$ changes when put in conjunction with objects in the exponent abstract domain $A$. This construction has been generalized in Giacobazzi and Ranzato [1995], where the weaker hypotheses of cHa on the concrete domain is considered for the existence of reduced cardinal power of domains. We follow this latter more general presentation, and we call *dependencies* the objects in the reduced cardinal power of domains. If $A = \rho_A(L)$ and $B = \rho_B(L)$ for $\rho_A, \rho_B \in uco(L)$, a dependency is specified by the function $\lambda x \in A.\ \rho_B(d \wedge_L x) : A \longmapsto B$, with $d \in L$. The set of all such dependencies is the reduced cardinal power $B^A = \{ \lambda x \in A.\ \rho_B(d \wedge_L x) \mid d \in L \}$. The next result extends Theorem 5.1

above to cHa's and proves that $B^A$ refines $B$.

Theorem 5.2 [Giacobazzi and Ranzato 1998]. *Let $L$ be a cHa, $A = \rho_A(L)$, and $B = \rho_B(L)$ for $\rho_A, \rho_B \in uco(L)$. Then $\langle L, \vec{\alpha}, B^A, \vec{\gamma}\rangle$ is a G.i. where $\vec{\alpha} = \lambda d \in L.\, (\lambda x \in A.\, \rho_B(d \wedge_L x))$. Moreover $B^A \sqsubseteq B$.*

The following representation result for dependencies is the main result of this section, and it specifies that reduced cardinal power[4] is equivalent to Heyting completion. This result proves a strong link between two apparently unrelated domain refinements, namely between domain refinement by reduced cardinal power and domain completion by relative pseudocomplements.

Theorem 5.3. *If $L$ is a cHa and $A, B \in uco(L)$, then $B^A = A \xrightarrow{\curlywedge} B$.*

In order to prove this result we need some additional lemmata to give a representation, in terms of relative pseudocomplements, for dependencies in $B^A$. In the following, given any $X \in uco(L)$, we denote by $\rho_X$ the corresponding closure operator.

Lemma 5.4. *Let $L$ be a cHa and $A, B \in uco(L)$. Then, for any $d \in L$ we have*

$$\lambda x \in A.\, \rho_B(d \wedge x) = \lambda x \in A.\, \rho_B(\bigwedge_{a \in A} (a \rightarrow \rho_B(d \wedge_L a)) \wedge_L x).$$

Proof. It is sufficient to prove that for any $d \in L, x \in A$ we have

$$\rho_B(d \wedge x) = \rho_B(\bigwedge_{a \in A} (a \rightarrow \rho_B(d \wedge_L a)) \wedge x).$$

For any $x \in A$ we have, by extensivity of $\rho_B$,

$$\begin{array}{lll}
\forall a \in A.\ d \wedge a \leq \rho_B(d \wedge a) & \Rightarrow & [\text{ by definition of } \rightarrow\ ] \\
\forall a \in A.\ d \leq a \rightarrow \rho_B(d \wedge a) & \Rightarrow & \\
\qquad d \leq \bigwedge_{a \in A}(a \rightarrow \rho_B(d \wedge_L a)) & \Rightarrow & [\text{ left monotony of } \wedge\ ] \\
\qquad d \wedge x \leq \bigwedge_{a \in A}(a \rightarrow \rho_B(d \wedge_L a)) \wedge x & \Rightarrow & [\text{ monotony of } \rho_B\ ] \\
\qquad \rho_B(d \wedge x) \leq \rho_B(\bigwedge_{a \in A}(a \rightarrow \rho_B(d \wedge_L a)) \wedge x). & &
\end{array}$$

For the other part we have, for any $x \in A$,

$$\begin{array}{ll}
\bigwedge_{a \in A}(a \rightarrow \rho_B(d \wedge_L a)) \leq x \rightarrow \rho_B(d \wedge x) & \Rightarrow \\
\bigwedge_{a \in A}(a \rightarrow \rho_B(d \wedge_L a)) \wedge x \leq (x \rightarrow \rho_B(d \wedge x)) \wedge x \leq \rho_B(d \wedge x) & \Rightarrow \\
\rho_B(\bigwedge_{a \in A}(a \rightarrow \rho_B(d \wedge_L a)) \wedge x) \leq \rho_B(d \wedge x). \quad \Box &
\end{array}$$

Lemma 5.5. *Let $L$ be a cHa and $A, B \in uco(L)$. For any $d \in L$ we have*

$$\vec{\gamma}(\lambda x \in A.\, \rho_B(d \wedge_L x)) = \bigwedge_{a \in A} (a \rightarrow \rho_B(d \wedge_L a)).$$

Proof. Let $b = \bigwedge_{a \in A}(a \rightarrow \rho_B(d \wedge_L a))$. By Lemma 5.4 $\lambda x \in A.\, \rho_B(d \wedge_L x) = \lambda x \in A.\, \rho_B(b \wedge_L x)$. Now we prove that $b$ is the most abstract object with this

[4] In the following we call reduced cardinal power the generalization of Cousot's reduced cardinal power made in Giacobazzi and Ranzato [1995].

property. Let $f \in L$ be such that $\lambda x \in A.\ \rho_B(d \wedge_L x) = \lambda x \in A.\ \rho_B(f \wedge_L x)$. Then

$$
\begin{array}{ll}
\forall x \in A.\ f \wedge x \leq \rho_B(f \wedge x) & \Rightarrow \ [\ \text{because } \rho_B(d \wedge x) = \rho_B(f \wedge x)\ ] \\
\forall x \in A.\ f \wedge x \leq \rho_B(d \wedge x) & \Rightarrow \\
\forall x \in A.\ f \leq x \rightarrow \rho_B(d \wedge x) & \Rightarrow \\
f \leq \bigwedge_{a \in A}(a \rightarrow \rho_B(d \wedge_L a)) = b. \quad \Box &
\end{array}
$$

We are now in the position to give the proof of Theorem 5.3.

Proof of Theorem 5.3. Let $A, B \in uco(L)$. By Lemma 5.5 we know that

$$B^A = \{\ \bigwedge_{a \in A}(a \rightarrow \rho_B(d \wedge_L a)) \mid d \in L\,\}.$$

Hence $B^A \subseteq A \xrightarrow{\curlywedge} B$. We prove that $A \xrightarrow{\curlywedge} B \subseteq B^A$. Let $a \rightarrow b \in A \longrightarrow B$. We show that $a \rightarrow b = \bigwedge_{x \in A}(x \rightarrow \rho_B(a \rightarrow b \wedge_L x))$, which implies, by Lemma 5.5, that $a \rightarrow b \in B^A$. By Lemma 5.5, for any $d \in L$, $d \leq \bigwedge_{x \in A}(x \rightarrow \rho_B(d \wedge_L x))$ (since $d \leq \vec{\gamma}(\vec{\alpha}(d))$). Thus $a \rightarrow b \leq \bigwedge_{x \in A}(x \rightarrow \rho_B(a \rightarrow b \wedge_L x))$. For the other part, for any $a \in A, b \in B$ we have

$$
\begin{array}{ll}
a \rightarrow b \wedge a \leq b & \Rightarrow \ [\ \text{because } \rho_B \text{ is monotone and } b \in B\ ] \\
\rho_B(a \rightarrow b \wedge a) \leq b & \Rightarrow \ [\ \text{because} \rightarrow \text{is right-monotone}\ ] \\
a \rightarrow \rho_B(a \rightarrow b \wedge a) \leq a \rightarrow b. &
\end{array}
$$

Hence, because $a \in A$, we have that for any $b \in B$

$$\bigwedge_{x \in A}(x \rightarrow \rho_B(a \rightarrow b \wedge_L x)) \leq a \rightarrow \rho_B(a \rightarrow b \wedge a) \leq a \rightarrow b. \quad \Box$$

## 6. APPLICATIONS I: DIRECTIONALITY IN LOGIC PROGRAM ANALYSIS

In this section we apply Heyting completion to systematic design of domains for relational analysis in logic programming. In particular we prove that Heyting completion surfaces most domains and applications aimed to detect directional information in type and groundness analysis.

We fix a first-order language $\mathcal{L}$, with variables ranging in $\mathcal{V}$. The set of terms and idempotent substitutions, viz. mappings from $\mathcal{V}$ to terms in $\mathcal{L}$, are denoted respectively $T_{\mathcal{L}}$ and $Sub$. Objects in $T_{\mathcal{L}}$ and $Sub$ are partially ordered by instantiation: $a \leq b$ if and only if $\exists \theta \in Sub.\ a = b\theta$. For any syntactic object $s$, $var(s)$ denotes the set of its variables.

### 6.1 Directionality in Type Inference

In this section we consider the problem of inferring types in logic programs by abstract interpretation. Given a domain for (basic) type analysis, we show that its Heyting completion captures precisely the notion of directionality among types and, more in general, how this construction surfaces directionality in any property closed under instantiation. In this case, Heyting completion supplies a domain for type inference, including the same relational information between types as traditionally obtained in standard directional type checking by verification methods (e.g., see Aiken and Lakshman [1994]).

The operational semantics of a logic program $P$, viz. a finite set of Horn clauses in $\mathcal{L}$, is defined by SLD-resolution (cf. Apt [1990]). If $G$ is a goal, viz. a conjunction of atoms in $\mathcal{L}$, then we write $G \xrightarrow{\vartheta}_P^* \Box$ if and only if $\vartheta \in Sub$ is a computed answer substitution for $G$ in $P$. As usual (cf. Aiken and Lakshman [1994]) a type is a set of terms closed under substitution . Hence, the concrete domain in which we interpret types is $\wp^{\downarrow}(T_{\mathcal{L}})$. An (abstract) domain of types $\mathcal{T}$ is therefore any abstraction of $\wp^{\downarrow}(T_{\mathcal{L}})$, i.e., any element in $uco(\wp^{\downarrow}(T_{\mathcal{L}}))$.

A type for a predicate $p$ of arity $n$ is an n-tuple of types. In the following, for the sake of simplicity, we assume that all predicates in the program have the same arity $n$, and abuse $T_{\mathcal{L}}$ to denote $n$-tuples of terms. A *directional type* for a predicate $p$ has the form $\langle I, O\rangle$ where $I$ and $O$ are types for $p$. The type $I$ is the "input" type, and type $O$ is the "output" type of $p$. A predicate $p$ has type $\langle I, O\rangle$ in a program $P$ (denoted $p \in \langle I, O\rangle$) if for any goal $p(\bar{t})$, $\bar{t} \in I \;\Rightarrow\; \theta(\bar{t}) \in O$ for any answer substitution $\theta$ (cf. Aiken and Lakshman [1994]). The next result relates the Heyting completion refinement to the notion of directional type. The idea is to prove that an implicational type $\tau_1 \to \tau_2 \in \mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$ models exactly the above notion of directional type $\langle \tau_1, \tau_2\rangle$.

THEOREM 6.1.1. *Let $P$ be a program, $\mathcal{T}$ be a domain of types, $\tau_1, \tau_2 \in \mathcal{T}$, and $p$ be a predicate. Then the following equivalence holds.*

$$(\forall\theta.\; p(\bar{X}) \xrightarrow{\theta}_P^* \Box \Rightarrow \bar{X}\theta \in \tau_1 \to \tau_2) \;\Leftrightarrow\; (\forall \bar{t} \in \tau_1.\forall\sigma.\; p(\bar{t}) \xrightarrow{\sigma}_P^* \Box \Rightarrow \bar{t}\sigma \in \tau_2)$$

PROOF. Assume that

$$\forall\theta.\; p(\bar{X}) \xrightarrow{\theta}_P^* \Box \Rightarrow \bar{X}\theta \in \tau_1 \to \tau_2. \tag{9}$$

We prove that

$$\forall \bar{t} \in \tau_1.\forall\sigma.\; p(\bar{t}) \xrightarrow{\sigma}_P^* \Box \Rightarrow \bar{t}\sigma \in \tau_2. \tag{10}$$

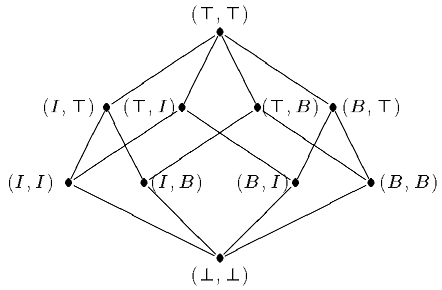
Let $\bar{t} \in \tau_1$ and $\sigma \in Sub$ such that $p(\bar{t}) \xrightarrow{\sigma}_P^* \Box$. Let $\bar{X}$ be new variables, not contained in $\bar{t}\sigma$. Let $\alpha$ such that $\bar{t}\sigma = \bar{X}\alpha$ (it exists since $\bar{X}$ are new variables). By definition of $\longrightarrow_P^*$, there exists a substitution $\beta$ such that $\alpha \leq \beta$ and $p(\bar{X}) \xrightarrow{\beta}_P^* \Box$. By (9), $\bar{X}\beta \in \tau_1 \to \tau_2$. Since $\bar{t} \in \tau_1$ and $\tau_1$ is closed under instantiation, it follows that $\bar{t}\sigma \in \tau_1$. Moreover $\bar{t}\sigma = \bar{X}\alpha \leq \bar{X}\beta$. Therefore, from $\bar{X}\beta \in \tau_1 \to \tau_2$, it follows that $\bar{t}\sigma \in \tau_2$.

For the other direction, assume (10), and let $\theta \in Sub$ such that $p(\bar{X}) \xrightarrow{\theta}_P^* \Box$. In order to prove that $\bar{X}\theta \in \tau_1 \to \tau_2$, we have to show that for every instance $\bar{t}$ of $\bar{X}\theta$ such that $\bar{t} \in \tau_1$, then $\bar{t} \in \tau_2$. From $p(\bar{X}) \xrightarrow{\theta}_P^* \Box$ it follows that $p(\bar{t}) \xrightarrow{\epsilon}_P^* \Box$. Therefore, by (10), $\bar{t} \in \tau_2$. $\Box$

COROLLARY 6.1.2. *In the hypothesis of Theorem 6.1.1, $p$ has type $\tau_1 \to \tau_2$ in $\mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$ if and only if $p \in \langle \tau_1, \tau_2\rangle$.*

Clearly, the domain $\mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$ is still closed under instantiation. Directional types in $\mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$, that satisfy the theorem above, can be obtained by abstract interpretation, with abstract domain $\mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$, by means of any bottom-up/top-down analyzer which is able to approximate correct or even computed answer substitutions (e.g., see Barbuti et al. [1993] and Bruynooghe [1991]).

*Example* 6.1.3. Consider the program $P = \{\, p(X, X).,\, r(2, Y) : -p(2, Y).\}$ and the following simple domain of basic types $\mathcal{T}$, where symbols I (integer) and B (Boolean) have the obvious meaning as fixpoints of a closure operator on $\wp^{\downarrow}(T_{\mathcal{L}})$.




A standard type inference which can be obtained from the above domain of types $\mathcal{T}$ returns the following type patterns: $\{ p(\top, \top), r(I, \top) \}$. Although correct, this result is not satisfactory. This is due to the fact that the domain $\mathcal{T}$ is not able to represent the property that both arguments of the predicate $p$ have the same type. Instead, this information can be expressed in the refined domain $\mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$. Since the concrete domain is $\wp^{\downarrow}(T_{\mathcal{L}})$, it is easy to prove that we have an explicit definition for the element $a \rightarrow b$, i.e., $a \rightarrow b = \{ t \in T_{\mathcal{L}} \mid \forall t' \leq t.\, t' \in a \Rightarrow t' \in b \}$. For example $(I, \top) \rightarrow (I, I) = \{ t \mid \forall t' \leq t.\, t' \in (I, \top) \Rightarrow t' \in (I, I) \}$, which intuitively denotes the set of pairs of terms such that, for any instance, whenever the first term of the instance is an integer, then the second term of the instance is an integer too. In the domain $\mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$, we can express the missing property in terms of suitable conjunctions of directional types. Let us denote by $p(\tau, \tau)$ the conjunction $\bigwedge_{(\tau_1, \tau_2) \in \mathcal{T}} p(\tau_1, \tau_2) \rightarrow p(\tau_1 \cap \tau_2, \tau_1 \cap \tau_2)$, i.e., $p(I, \top) \rightarrow p(I, I) \wedge p(\top, I) \rightarrow p(I, I) \wedge p(B, \top) \rightarrow p(B, B) \wedge p(\top, B) \rightarrow p(B, B) \wedge p(I, B) \rightarrow p(\bot, \bot) \wedge p(B, I) \rightarrow p(\bot, \bot)$. The implicational objects in $p(\tau, \tau)$ force both arguments of $p$ to have the same type, by assigning them the common (most precise) type, which is precisely type intersection. By abstract interpretation in the refined domain $\mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$, we therefore obtain the more precise type patterns: $\{ p(\tau, \tau), r(I, I) \}$.

### 6.2 The Essence of Directionality: From Types to Groundness

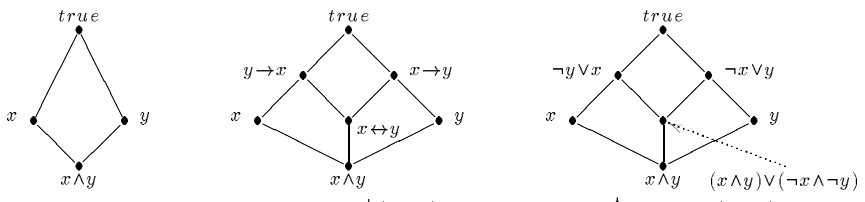
Giacobazzi and Ranzato [1995] proved that the abstract domain $Def$ [Armstrong et al. 1994], of propositional formulas whose models are closed under intersection, is the reduced cardinal power of a simpler domain for pure groundness introduced by Jones and Søndergaard's [1987]. $Def$ is a well-known domain for ground dependency analysis which is able to characterize both pure groundness, i.e., whether a variable is instantiated to ground terms during program execution, and the relations between the groundness of different program variables, providing in this sense another clear example of relational analysis.

Let $Var \subseteq \mathcal{V}$ be a finite set, representing the set of interesting variables. $Def$ is traditionally presented as the set of (classic) propositional formulas built on $Var$ with connectives $\wedge$ and $\rightarrow$, whose models are closed under set-intersection. The interpretation of the connectives is the classical one: $\phi \leq \psi$ if and only if $\psi$ is a logical consequence of $\phi$. We say that $I \subseteq Var$ is a model for $\phi$, denoted $I \models \phi$, if $\phi$ is true in the interpretation which assigns $true$ to all the variables in $I$ and $false$ to the other ones [Armstrong et al. 1994]. For the sake of simplicity we write $\theta\, \mathtt{grounds}\, \phi$ to denote $\{ x \in Var \mid var(x\theta) = \emptyset \} \models \phi$, for $\theta \in Sub$. Because logic programs compute substitutions, the concrete domain we consider is $\wp^{\downarrow}(Sub)$, which is completely distributive, and therefore a cHa. $Def$ is an abstraction of $\wp^{\downarrow}(Sub)$, and the concretization function is $\gamma_{Def}(\phi) = \{\, \theta \in Sub \mid \forall \sigma \leq \theta.\ \sigma\, \mathtt{grounds}\, \phi \,\}$ (cf. Cortesi et al. [1996]). $\mathcal{G}$ is the standard domain for pure groundness analysis introduced by Jones and Søndergaard [1987], which is a strict abstraction of $Def$, including formulas on $Var$ with the only connective $\wedge$. $\mathcal{G}$ is (isomorphic to) the complete lattice $\langle \wp(Var), \supseteq \rangle$, and it is well known that it enjoys a Galois insertion $\langle \wp^{\downarrow}(Sub), \gamma_{Var}, \mathcal{G}, \alpha_{Var} \rangle$, where for any $V \subseteq Var$, $\gamma_{Var}(V) = \{\, \theta \in Sub \mid \forall v \in V.\ \theta\, \mathtt{grounds}\, v \,\}$. As usual, we identify $Def$ and $\mathcal{G}$ with the corresponding closures on $\wp^{\downarrow}(Sub)$: $\gamma_{Def}(Def)$ and $\gamma_{Def}(\mathcal{G})$. By applying Theorem 5.3 and a result in Giacobazzi and Ranzato [1995], we immediately obtain a characterization of $Def$ in terms of Heyting completion.

Corollary 6.2.1. *$Def = \mathcal{G} \stackrel{\curlywedge}{\longrightarrow} \mathcal{G}$.*

Proof. In Giacobazzi and Ranzato [1995] it has been proven that $Def = \mathcal{G}^{\mathcal{G}}$. The thesis follows by Theorem 5.3. □

In view of the observations in Section 6.1, pure groundness $\mathcal{G}$ can be viewed as some basic type definition, and ground dependency in $Def$ can be interpreted as directional type inference. The key point here, as well as in directional types, is that the concrete domain is some lattice of ideals, i.e., sets of objects closed under instantiation, which are in general non-Boolean cHa. It is worth noting that a different concrete domain would completely change the interpretation of directionality. Consider the Boolean concrete domain $\wp(Sub)$. If $Var = \{x, y\}$ we have the following domain $\mathcal{G} \stackrel{\curlywedge}{\longrightarrow} \mathcal{G}$, which differs from the analogous completion $Def$ on the non-Boolean lattice $\wp^{\downarrow}(Sub)$.




$\mathcal{G}$ $\qquad$ $Def$ in $\wp^{\downarrow}(Sub)$ $\qquad$ $\mathcal{G} \stackrel{\curlywedge}{\longrightarrow} \mathcal{G}$ in $\wp(Sub)$

While $\mathcal{G} \stackrel{\curlywedge}{\longrightarrow} \mathcal{G}$ is no longer able to express ground dependencies, it is now able to express negative groundness information. This is because intuitionistic implication is replaced by the classical one. This is a consequence of the peculiar nature of the concrete (collecting) domain, which is a Boolean lattice. In this case, the relative pseudocomplements introduced by Heyting completion provide negation of properties. This is because, in Boolean lattices, $a \to b = \neg a \vee b$ and, for instance $\neg x \vee y = \overline{\gamma_{Def}(x)} \cup \gamma_{Def}(y)$, while this is not the case in complete Heyting algebras. For instance, the empty substitution belongs to $\neg x \vee y$, but it provides no groundness information. Clearly, because the set-theoretic complement of a set closed under instantiation is not, in general, closed under instantiation, then $\mathcal{G} \stackrel{\curlywedge}{\longrightarrow} \mathcal{G}$ above is not an abstraction of $\wp^{\downarrow}(Sub)$. In this sense, when the concrete domain is $\wp(Sub)$, the analysis obtained from $\mathcal{G} \stackrel{\curlywedge}{\longrightarrow} \mathcal{G}$ cannot be regarded as a type inference.

## 7. IMPLICATIONAL DOMAIN EQUATIONS

In the previous sections we have given a logical interpretation to a generalization of Cousot's reduced cardinal power of domains in terms of Heyting completion. We are now in the position where all the operators defined in Cousot and Cousot [1979], i.e., reduced product, disjunctive completion, and reduced cardinal power, have a clean logical interpretation. In this section we consider these operators, together with the lattice of abstract interpretations, as an algebra of domains $\mathcal{A} = \langle uco(L), \sqcap, \curlyvee, \xrightarrow{\curlywedge} \rangle$, involving conjunctive information by reduced product, disjunctive information by disjunctive completion, and relational information by Heyting completion.

It is natural at this point to ask "What is the structure of domains that can be systematically built with the basic operators defined in Cousot and Cousot [1979], i.e., out of the above algebra $\mathcal{A}$?" In particular, which domains can be specified as solutions of simple domain equations on $\mathcal{A}$ using Heyting completion? For instance, the examples in Section 6 have shown that well-known domains for logic program analysis can be constructed by combining the operators in $\mathcal{A}$, and the results in Section 4 may help in simplifying complex formulas involving these operators. As usual in domain theory, the most interesting case is the one of recursive domain equations. The presence of recursion leads us to get a constructive definition for domains which are closed under a suitable combination of properties involving disjunction, conjunction, and implication. Clearly, by definition of abstract domains, i.e., Moore families, any domain is closed by conjunction. Therefore the above equations make sense when they combine disjunctive and Heyting completion. Let $A \in uco(L)$ be a given abstract domain of basic properties. We want to design a domain which refines $A$ and is closed under some combination of $\bigsqcap$, $\curlyvee$, and $\xrightarrow{\curlywedge}$. Since Heyting completion is the only nonidempotent operator, equations which do not contain Heyting completion always have trivial solutions. For instance, it is immediate to observe that the most abstract solution of $X = A \sqcap X$ is the domain $A$ itself and that the most abstract solution of $X = A \sqcap \curlyvee(X)$ is $\curlyvee(A)$, because $\curlyvee$ is idempotent. The ultimate goal is therefore to construct domains which are closed with respect to Heyting completion, i.e., which are a fixpoint of the Heyting completion operator.

Given a domain $A$, we are interested in the most abstract abstract domain $X$ which is more concrete than $A$, and it is closed under $\xrightarrow{\curlywedge}$. This domain can be obtained as the greatest (most abstract) solution of the following recursive domain equation:

$$X = A \sqcap (X \xrightarrow{\curlywedge} X) \tag{11}$$

We are also interested in the family of domains closed with respect to both Heyting and disjunctive completion. This corresponds to find the greatest (most abstract) solution of the following domain equation:

$$X = A \sqcap (X \xrightarrow{\curlywedge} X) \sqcap \curlyvee(X) \tag{12}$$

Therefore, we study recursive domain equations $X = F(X)$, where the right-hand side $F(X)$ is for either $A \sqcap (X \xrightarrow{\curlywedge} X)$ or $A \sqcap (X \xrightarrow{\curlywedge} X) \sqcap \curlyvee(X)$. By Theorem 3.3.2, $\xrightarrow{\curlywedge}$ is monotone. Moreover, both reduced product and disjunctive completion are monotone operators on $uco(L)$. Therefore $X = F(X)$ has always a greatest (most abstract) and least (most concrete) solution. Note that the least fixpoint of $\lambda X.\ F(X)$ is $L$, which is of little interest being the identity closure $\lambda x.x$. Therefore, we are mostly interested in the greatest fixpoint of $\lambda X.\ F(X)$ in $uco(L)$, which exists and can be constructively obtained as the limit of the following transfinite sequence of domains $\{F_\alpha\}_{\alpha \in Ord}$, all approximating the greatest solution from above:

$$\begin{array}{lll} F_0 & = & \top \\ F_{\alpha+1} & = & F(F_\alpha) \\ F_\beta & = & \bigsqcap_{\gamma<\beta} F_\gamma \quad \text{if } \beta \text{ is a limit ordinal} \end{array}$$

Clearly, all refinements built in this way are idempotent for the Heyting completion refinement. The following two sections are devoted to study the solutions of the above equations.

### 7.1 Heyting Completion Closed Domains

In this section we study the solutions to Eq. (11). The most abstract solution is the greatest fixpoint of $\lambda X.\ A \sqcap (X \xrightarrow{\curlywedge} X)$ in $uco(L)$, which can be constructively obtained as the limit of the following transfinite sequence of domains $\{A_\alpha^{\rightarrow}\}_{\alpha \in Ord}$:

$$\begin{array}{lll} A_0^{\rightarrow} & = & \top \\ A_1^{\rightarrow} & = & A \\ A_{\alpha+1}^{\rightarrow} & = & A \sqcap A_\alpha^{\rightarrow} \xrightarrow{\curlywedge} A_\alpha^{\rightarrow} \\ A_\beta^{\rightarrow} & = & \bigsqcap_{\gamma<\beta} A_\gamma^{\rightarrow} \quad \text{if } \beta \text{ is a limit ordinal} \end{array}$$

Equation (11) can be presented in different but equivalent ways, as shown in the following theorem.

THEOREM 7.1.1. *The following equations have the same greatest solution:*

—$X = A \sqcap (X \xrightarrow{\curlywedge} X)$

—$Y = A \sqcap (Y \xrightarrow{\curlywedge} A)$

—$Z = Z \xrightarrow{\curlywedge} A$

PROOF. Let $\lambda X.\ A \sqcap (X \xrightarrow{\curlywedge} X)$, $\lambda Y.\ A \sqcap (Y \xrightarrow{\curlywedge} A)$ and $\lambda Z.\ Z \xrightarrow{\curlywedge} A$ be the refinement operators associated to the equations. Let $\{X_\alpha^{\rightarrow}\}_{\alpha \in Ord}$, $\{Y_\alpha^{\rightarrow}\}_{\alpha \in Ord}$, and $\{Z_\alpha^{\rightarrow}\}_{\alpha \in Ord}$ be the corresponding sequences of domains built from each operator.

First note that $X_0^{\rightarrow} = Y_0^{\rightarrow} = Z_0^{\rightarrow} = \top$ and for $\alpha \geq 1$ it holds that $X_\alpha^{\rightarrow} \sqsubseteq A$, $Y_\alpha^{\rightarrow} \sqsubseteq A$, and $Z_\alpha^{\rightarrow} \sqsubseteq A$. It immediately follows that for each $\alpha \in Ord$ we have $Y_\alpha^{\rightarrow} = Z_\alpha^{\rightarrow}$. Now we prove by transfinite induction that for each $\alpha \in Ord$ we have $X_\alpha^{\rightarrow} = Y_\alpha^{\rightarrow}$.

$\alpha = 0$. $X_0^{\rightarrow} = Y_0^{\rightarrow} = \top$.

$\alpha = 1$. $X_1^{\rightarrow} = Y_1^{\rightarrow} = A$.

$\alpha = 2$. $X_2^{\rightarrow} = Y_2^{\rightarrow} = A \xrightarrow{\curlywedge} A$.

$\alpha = \beta + 2,\ \beta \neq 0.$

$$
\begin{array}{lll}
X^{\rightarrow}_{\alpha} & = A \sqcap (X^{\rightarrow}_{\beta+1} \xrightarrow{\curlywedge} X^{\rightarrow}_{\beta+1}) & [\ \beta \geq 1\ ] \\
& = X^{\rightarrow}_{\beta+1} \xrightarrow{\curlywedge} X^{\rightarrow}_{\beta+1} & [\ \text{by inductive hypothesis}\ ] \\
& = Y^{\rightarrow}_{\beta+1} \xrightarrow{\curlywedge} Y^{\rightarrow}_{\beta+1} & \\
& = Y^{\rightarrow}_{\beta+1} \xrightarrow{\curlywedge} (A \sqcap (Y^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} A)) & [\ \beta \geq 1\ ] \\
& = Y^{\rightarrow}_{\beta+1} \xrightarrow{\curlywedge} (Y^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} A) & [\ \text{by (b) in Prop. 4.1.2}\ ] \\
& = (Y^{\rightarrow}_{\beta+1} \sqcap Y^{\rightarrow}_{\beta}) \xrightarrow{\curlywedge} A & [\ \text{by monotonicity}\ ] \\
& = Y^{\rightarrow}_{\beta+1} \xrightarrow{\curlywedge} A & \\
& = Y^{\rightarrow}_{\beta+2} = Y^{\rightarrow}_{\alpha} &
\end{array}
$$

$\alpha = \beta + 1$, *$\beta$ limit ordinal.*

$$
\begin{array}{lll}
X^{\rightarrow}_{\alpha} & = A \sqcap (X^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} X^{\rightarrow}_{\beta}) & [\ \beta \geq 1\ ] \\
& = X^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} X^{\rightarrow}_{\beta} & [\ \text{by inductive hypothesis}\ ] \\
& = Y^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} Y^{\rightarrow}_{\beta} & \\
& = Y^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} \bigsqcap_{\gamma<\beta} Y^{\rightarrow}_{\gamma} & [\ \text{by (a) in Prop. 4.1.2}\ ] \\
& = \bigsqcap_{\gamma<\beta} Y^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} Y^{\rightarrow}_{\gamma} & [\ \text{since } \gamma < \beta\ ] \\
& = \bigsqcap_{\gamma<\beta} (Y^{\rightarrow}_{\beta} \sqcap Y^{\rightarrow}_{\gamma}) \xrightarrow{\curlywedge} Y^{\rightarrow}_{\gamma} & [\ \text{by (b) in Prop. 4.1.2}\ ] \\
& = \bigsqcap_{\gamma<\beta} Y^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} (Y^{\rightarrow}_{\gamma} \xrightarrow{\curlywedge} Y^{\rightarrow}_{\gamma}) & [\ \text{by inductive hypothesis}\ ] \\
& = \bigsqcap_{\gamma<\beta} Y^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} (Y^{\rightarrow}_{\gamma} \xrightarrow{\curlywedge} A) & [\ \text{by (b) in Prop. 4.1.2}\ ] \\
& = \bigsqcap_{\gamma<\beta} (Y^{\rightarrow}_{\beta} \sqcap Y^{\rightarrow}_{\gamma}) \xrightarrow{\curlywedge} A & [\ \text{since } \gamma < \beta\ ] \\
& = \bigsqcap_{\gamma<\beta} Y^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} A & \\
& = Y^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} A & [\ \beta \geq 1\ ] \\
& = A \sqcap (Y^{\rightarrow}_{\beta} \xrightarrow{\curlywedge} A) & \\
& = Y^{\rightarrow}_{\alpha} &
\end{array}
$$

*$\alpha$ limit ordinal.* Trivial by inductive hypothesis. □

The following result characterizes the most abstract solution of Eq. (11). Recall that Heyting algebras are defined in terms of three basic operations: meet, join, and relative pseudocomplement. Because any abstract interpretation is always a meet-sublattice of the concrete domain (cf. Cousot and Cousot [1979]), when $L$ is a cHa, $X \subseteq L$ is a solution for Eq. (11) if and only if the abstract operation of relative pseudocomplementation is precisely the restriction of the concrete one to the elements of $X$. Hence, when $X$ is a solution of the above recursive domain equation, then $X$ is always a cHa, where only the join operator may differ from the concrete one. Moreover we also characterize the solution of Eq. (11) in terms of its achieved precision. It turns out that the most abstract solution of Eq. (11) is the most abstract domain which includes $A$, and it is *complete* for a suitable set of functions. Recall that an abstract domain $A$, associated with a closure operator $\rho_A \in uco(L)$, is *complete* for a function $f : L \longmapsto L$ if $\rho_A \circ f \circ \rho_A = \rho_A \circ f$ (see Cousot and Cousot [1979], Giacobazzi and Ranzato [1997a], and Mycroft [1993]). A domain is complete for a family of functions $\{f_i\}_{i\in I}$ if it is complete for all of them.

THEOREM 7.1.2. *Let $L$ be a cHa, $X, A \in uco(L)$, and $X \sqsubseteq A$. Then the following are equivalent:*

*a. $X$ is a solution of Eq.* (11)*;*

*b. $X$ is a subalgebra of $L$ with respect to the relative pseudocomplement;*

*c. $X$ is complete for $\{\lambda x.\ c \wedge x\}_{c \in X}$;*

*d. $\rho_X(\rho_X(a) \wedge b) = \rho_X(a) \wedge \rho_X(b)$ for any $a, b \in L$.*

PROOF. $a \Rightarrow b$. Assume that $X$ is a solution of Eq. (11). We have to prove that $X$ is relatively pseudocomplemented and that the relative pseudocomplements computed in $X$ are exactly the relative pseudocomplements in $L$; that is, for any $a, b \in X$ there exists $a \rightsquigarrow b \in X$ such that $\forall c \in X.\ a \wedge c \leq b \Leftrightarrow c \leq a \rightsquigarrow b$ and $a \rightsquigarrow b = a \rightarrow b$. This is equivalent to proving that $\forall a, b \in X.\ a \rightarrow b \in X$ and $\forall c \in X.\ a \wedge c \leq b \Leftrightarrow c \leq a \rightarrow b$. Since $X$ is a solution of Eq. (11), $X = X \xrightarrow{\wedge} X$; therefore $a \rightarrow b \in X$. By definition of $\rightarrow$, it holds that $\forall c \in L.\ a \wedge c \leq b \Leftrightarrow c \leq a \rightarrow b$. The thesis follows because $X \subseteq L$.

$b \Rightarrow c$. Assume that for any $a, b \in X$: $a \rightarrow b \in X$. We prove that $\rho_X$ is complete for the set of functions $\{\lambda x.\ c \wedge x\}_{c \in X}$, namely that for any $x \in L$ and $c \in X$, $\rho_X(c \wedge \rho_X(x)) = \rho_X(c \wedge x)$. Let $x \in L$ and $c \in X$ be fixed. It is obvious that $\rho_X(c \wedge x) \leq \rho_X(c \wedge \rho_X(x))$. By hypothesis, $c \rightarrow \rho_X(c \wedge x) \in X$. Moreover, by extensivity, $c \wedge x \leq \rho_X(c \wedge x)$. Therefore, $L$ being a cHa, $x \leq c \rightarrow \rho_X(c \wedge x)$. By monotonicity and idempotence we have that $\rho_X(x) \leq \rho_X(c \rightarrow \rho_X(c \wedge x)) = c \rightarrow \rho_X(c \wedge x)$. Therefore we conclude with the following inequalities:

$$\begin{aligned} \rho_X(c \wedge \rho_X(x)) &= \rho_X(\rho_X(c) \wedge \rho_X(x)) \\ &= \rho_X(c) \wedge \rho_X(x) \\ &\leq \rho_X(c) \wedge (c \rightarrow \rho_X(c \wedge x)) \\ &\leq \rho_X(c \wedge x). \end{aligned}$$

$c \Rightarrow a$. Assume that $X$ be complete for the set of functions $\{\lambda x.\ c \wedge x\}_{c \in X}$. We prove that for any $a, b \in X$, $a \rightarrow b \in X$, i.e., $\rho_X(a \rightarrow b) = a \rightarrow b$. By extensivity, $\rho_X(a \rightarrow b) \geq a \rightarrow b$. Moreover, $L$ being a cHa, we have that $a \wedge a \rightarrow b \leq b$. By hypothesis, $X$ is complete for the function $\lambda x.\ a \wedge x : L \longmapsto L$. Therefore we have the following inequalities:

$$\begin{aligned} a \wedge \rho_X(a \rightarrow b) &\leq \rho_X(a \wedge \rho_X(a \rightarrow b)) \\ &= \rho_X(a \wedge a \rightarrow b) \\ &\leq \rho_X(b) \\ &= b. \end{aligned}$$

Therefore $a \wedge \rho_X(a \rightarrow b) \leq b$ which implies that $\rho_X(a \rightarrow b) \leq a \rightarrow b$.

$c \Leftrightarrow d$. Straightforwardly by definition of completeness. □

By Theorem 7.1.2, the greatest solution of Eq. (11) is therefore the greatest (i.e., most abstract) domain which contains $A$ and is complete for the set of functions $\{\lambda x.\ c \wedge x\}_{c \in X}$. Completeness represents the key notion to characterize the solutions of Eq. (11). Completeness here means that no loss of precision is accumulated by abstracting the conjunction of abstract objects with respect to what can be achieved by abstracting only one of its arguments. The notion of *least complete extension* of a domain has been recently introduced by Giacobazzi and Ranzato [1997a] and Giacobazzi et al. [1998a], where the authors proved that there always exists the (unique) most abstract domain which contains a given domain $A$ and is complete for any given continuous function $f$. In this sense, Point (c) in Theorem 7.1.2 can be viewed as an instance of these more general results.

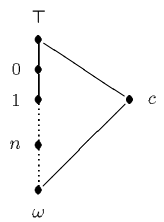
*Example* 7.1.3. $Pos$ is definitely the most widely known and used domain for groundness analysis of logic programs (cf. Armstrong et al. [1994]). $Pos$ is traditionally presented as the set of (classic) propositional formulas built on $Var$ and the constant $true$, by using the connectives $\wedge, \vee, \rightarrow$, and ordered by logical implication. $Pos$ is an abstraction of $\wp^{\downarrow}(Sub)$ and $\gamma_{Pos}(\phi) = \{\, \theta \in Sub \mid \forall \sigma \leq \theta.\ \sigma\, \mathtt{grounds}\, \phi \,\}$ (cf. Cortesi et al. [1996]). With respect to $Def$, disjunction of variables are allowed in $Pos$. In this sense, $Pos$ is strictly more expressive than $Def$, i.e., $Def \subset Pos$.

Consider the simple domain $\mathcal{G}$ for groundness analysis defined in Section 6.2. It is proved in Scozzari [1997] that $Pos = (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \xrightarrow{\curlywedge} \mathcal{G}$ and $Pos \xrightarrow{\curlywedge} Pos = Pos$. Therefore the domain $Pos$ is the greatest solution to the equation: $X = \mathcal{G} \sqcap (X \xrightarrow{\curlywedge} X)$. By exploiting our characterization of Theorem 7.1.2, it turns out that $Pos$ is the most abstract domain which is complete for the family of functions $\{\lambda x.\ c \wedge x\}_{c \in Pos}$. Recall that the meet operation in $Pos$ is precisely abstract unification. Therefore $Pos$ represents the most abstract domain where abstract unification is complete, provided that at least one of the arguments belongs to $Pos$.

By Theorem 7.1.2, it is clear that the greatest solution of Eq. (11) is a cHa. The meet and implication operations of the greatest solution are exactly the restriction of the corresponding ones on the concrete domain. Since the greatest solution may fail to be disjunctive, the join in the abstract domain can be different from the concrete one. For the case of groundness analysis, this is coherent with a result by Filé and Ranzato [1994], proving that $Pos$ is not disjunctive domain, i.e., $Pos \neq \curlyvee(Pos)$.

It is worth noting that the operator associated with the Eq. (11), $\lambda X.\ A \sqcap (X \xrightarrow{\curlywedge} X)$ is not, in general, co-continuous in $uco(L)$, as shown by the following example. Hence $\omega$ may not in general be the least closure ordinal for the sequence $\{A^{\rightarrow}_{\alpha}\}_{\alpha \in Ord}$, i.e., it may well happen that $A^{\rightarrow}_{\omega} \subset A^{\rightarrow}_{\omega+1}$.

*Example* 7.1.4. Consider the following complete lattice $L = \mathbb{N} \cup \{\omega, \top, c\}$ depicted below, and the family of its abstractions $A_i \in uco(L)$, such that for any $i \in \mathbb{N}$, $A_i = \{\, n \in \mathbb{N} \mid n \leq i \,\} \cup \{\top\}$.




$\{A_i\}_{i \in \mathbb{N}}$ is a chain, and for any $i \in \mathbb{N}$, $A_i = A_i \xrightarrow{\curlywedge} A_i$, but $\sqcap_{i \in \mathbb{N}} A_i = \mathbb{N} \cup \{\omega\}$ is not implication closed, since $c = 0 \rightarrow \omega \in (\sqcap_{i \in \mathbb{N}} A_i) \xrightarrow{\curlywedge} (\sqcap_{i \in \mathbb{N}} A_i)$. Note that $L$ above is not a cHa. However, unfortunately, it is not co-continuous even if the concrete domain is a Boolean algebra. It is sufficient to consider $L = \langle \wp(\omega), \omega, \emptyset, \cap, \cup, \subseteq \rangle$, and the chain of domains $A_i = \curlywedge(\cup_{1 \leq j \leq i} \{\, \emptyset, \omega, \omega \setminus \{\, i \,\}, \{\, i \,\} \,\}) \in uco(\wp(\omega))$, for $i \geq 1$. Note that $\sqcap_{i \geq 1} A_i = \curlywedge(\cup_{i \geq 1} A_i)$ and $\{\, 0 \,\}, \emptyset \in \sqcap_{i \geq 1} A_i$, but being $\cap$-irreducible, $\{\, 0 \,\} \rightarrow \emptyset = \omega \setminus \{\, 0 \,\} \notin \sqcap_{i \geq 1}(A_i \xrightarrow{\curlywedge} A_i)$.

Clearly, co-continuity is ensured when $L$ is a finite domain, because $uco(L)$ is finite too, or when $uco(L)$ satisfies the descending chain condition, i.e., it does not contains infinite descending chains. Because $uco(L)$ is a domain of functions, both of these conditions are far too restrictive for a concrete domain in most applications of abstract interpretation, e.g., in static program analysis. However, co-continuity is maintained when the descending chain condition holds on the concrete domain $L$, as proved by the next result. Note that the operator we want to prove co-continuous is defined on $uco(L)$ and that our condition, in general, does not imply that $uco(L)$ enjoys the descending chain property.

LEMMA 7.1.5. *Let $L$ be a complete lattice which enjoys the descending chain property and $\{ B_i \}_{i\in I} \subseteq uco(L)$ be a chain. Then $\sqcap_{i\in I} B_i = \bigcup_{i\in I} B_i$.*

PROOF. We know that $\sqcap_{i\in I} B_i = \curlywedge(\bigcup_{i\in I} B_i)$. Therefore we have only to prove that $\bigcup_{i\in I} B_i$ is closed under (possibly infinite) $\wedge$. Let $\{ b_j \}_{j\in J} \subseteq \bigcup_{i\in I} B_i$. By the descending chain condition, there exists a finite $J' \subseteq J$ such that $\bigwedge_{j\in J} b_j = \bigwedge_{j\in J'} b_j$ and an index $k \in I$ such that $\forall j \in J'.\ b_j \in B_k$ (because $J'$ is finite). Since $B_k$ is in $uco(L)$, it follows that $\bigwedge_{j\in J'} b_j \in B_k$ and thus $\bigwedge_{j\in J'} b_j \in \bigcup_{i\in I} B_i$. $\square$

THEOREM 7.1.6. *If $L$ enjoys the descending chain property, $\lambda X.\ A \sqcap (X \xrightarrow{\curlywedge} X)$ is co-continuous in $uco(L)$.*

PROOF. Let $\{ B_i \}_{i\in I}$ be a chain in $uco(L)$ and $F = \lambda X.\ A \sqcap (X \xrightarrow{\curlywedge} X)$. We have to prove that $F(\sqcap_{i\in I} B_i) = \sqcap_{i\in I} F(B_i)$. By Lemma 7.1.5 this is equivalent to proving that $F(\bigcup_{i\in I} B_i) = \bigcup_{i\in I} F(B_i)$. The direction $F(\bigcup_{i\in I} B_i) \supseteq \bigcup_{i\in I} F(B_i)$ trivially holds. For the other direction, let $f \in F(\bigcup_{i\in I} B_i)$. By definition of $F$, there exist $a \in A, b_j, c_j \in \bigcup_{i\in I} B_i$ for $j \in J$, such that $f = a \wedge \bigwedge_{j\in J}(b_j \rightarrow c_j)$. From the descending chain property, there exists a finite $J' \subseteq J$ such that $\bigwedge_{j\in J}(b_j \rightarrow c_j) = \bigwedge_{j\in J'}(b_j \rightarrow c_j)$. Let $k \in I$ be such that $\forall j \in J'.\ b_j, c_j \in B_k$ (such an index exists since $J'$ is finite). Hence $\bigwedge_{j\in J'}(b_j \rightarrow c_j) \in B_k \xrightarrow{\curlywedge} B_k$. Since $f = a \wedge \bigwedge_{j\in J'}(b_j \rightarrow c_j)$ it follows that $f \in F(B_k)$ and thus $f \in \bigcup_{i\in I} F(B_i)$. $\square$

It is therefore immediate that, under the condition in Theorem 7.1.6, $\omega$ is the least closure ordinal, i.e., $A^{\rightarrow}_{\omega}$ is the greatest solution of Eq. (11).

## 7.2 Heyting and Disjunctive Closed Domains

In this section we study the family of domains which are solutions to the equation $X = A \sqcap (X \xrightarrow{\curlywedge} X) \sqcap \curlyvee(X)$. This equation characterizes all domains which are closed with respect to both Heyting and disjunctive completion. The following result is an immediate extension of Theorem 7.1.2 to the case of disjunctive solutions.

THEOREM 7.2.1. *Let $L$ be a cHa, $X, A \in uco(L)$, and $X \sqsubseteq A$. The following statements are equivalent:*

—*$X$ is a solution of Eq.* (12)*;*

—*$X$ is a solution of Eq.* (11) *and is disjunctive;*

—*$X$ is a complete Heyting subalgebra of $L$.*

COROLLARY 7.2.2. *If $B$ is a solution of Eq.* (12) *then $B$ is a solution of Eq.* (11).

As before for Eq. (11), also Eq. (12) can be presented in different, but equivalent, ways. The following result is straightforward from monotonicity of all involved operators.

Theorem 7.2.3. *The following equations have the same greatest fixpoint:*

—$X = A \sqcap (X \xrightarrow{\curlywedge} X) \sqcap \curlyvee(X)$
—$Y = A \sqcap \curlyvee(Y \xrightarrow{\curlywedge} Y)$
—$Z = \curlyvee(A \sqcap (Z \xrightarrow{\curlywedge} Z))$

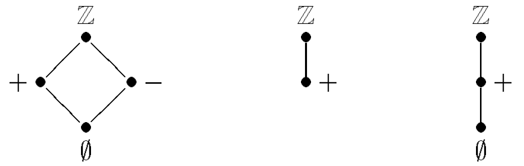
*Example* 7.2.4. We keep on Example 7.1.3 and consider the equation

$$X = \mathcal{G} \sqcap (X \xrightarrow{\curlywedge} X) \sqcap \curlyvee(X).$$

Since it has been proven in Scozzari [1997] that $\curlyvee(Pos) = \curlyvee(Pos) \xrightarrow{\curlywedge} \curlyvee(Pos)$, it follows that $\curlyvee(Pos)$ is the greatest solution of the above equation.

It is worth noting that $Z = Z \xrightarrow{\curlywedge} A$ in Theorem 7.1.1 cannot be directly extended by applying disjunctive completion, i.e., by considering the equation $X = \curlyvee(X \xrightarrow{\curlywedge} A)$, to obtain the same solutions as Eq. (12) does. The next example shows this phenomenon.

*Example* 7.2.5. Consider the simple lattice $L$ depicted below and one of its abstractions $A$. The greatest solution of the equation $X = \curlyvee(X \xrightarrow{\curlywedge} A)$ is the domain $B$, while the greatest solution of the equation $X = A \sqcap (X \xrightarrow{\curlywedge} X) \sqcap \curlyvee(X)$ is the domain $L$ itself. Note that while $B$ is still disjunctive, it is not closed by Heyting completion.




$L$ $A$ $B$

The next result gives a characterization of both generic and most abstract solutions to Eq. (11), under the additional hypothesis that $L$ is a collecting domain and $\bot_L \in A$. Under these hypotheses, Eq's. (11) and (12) both have the same solutions.

Theorem 7.2.6. *Let $L = \wp(S)$ for some set $S$ and $A \in uco(L)$, with $\bot_L \in A$.*

—$A = A \xrightarrow{\curlywedge} A$ *if and only if $A$ is disjunctive and complemented, i.e., $A = \curlyvee(A)$ and $A \xrightarrow{\curlywedge} \{\top_L, \bot_L\} \subseteq A$;*
—$\curlyvee(A \sqcap (A \xrightarrow{\curlywedge} \{\top_L, \bot_L\}))$ *is the most abstract solution for* (11) *and* (12).

Proof. Let $A = A \xrightarrow{\curlywedge} A$. By monotonicity, $A = A \xrightarrow{\curlywedge} A \sqsubseteq A \xrightarrow{\curlywedge} \{\top, \bot_L\}$. Hence $A \xrightarrow{\curlywedge} \{\top_L, \bot_L\} \subseteq A$. We now prove that $A$ is disjunctive. Let $\{ a_i \}_{i \in I} \subseteq A$ for some set $I$. Since $A$ is complemented, then $\forall i \in I.\ a_i \to \bot_L \in A$. Thus being

$A \in uco(L)$, $\bigwedge_{i\in I}(a_i \to \bot_L) \in A$. But also $\bigwedge_{i\in I}(a_i \to \bot_L) \to \bot_L \in A$ because $A$ is complemented. Moreover

$$\begin{aligned}\bigwedge_{i\in I}(a_i \to \bot_L) \to \bot_L &= \neg(\bigwedge_{i\in I}(a_i \to \bot_L)) \vee \bot_L\\ &= \bigvee_{i\in I}(\neg(\neg a_i \vee \bot_L)) \vee \bot_L = \bigvee_{i\in I}((a_i \wedge \neg\bot_L) \vee \bot_L) = \bigvee_{i\in I}(a_i \wedge \top_L) = \bigvee_{i\in I} a_i.\end{aligned}$$

Thus $A$ is closed under (possibly) infinite concrete disjunctions and $\bot_L \in A$. By Proposition 4.2.1 it is a disjunctive domain or equivalently an additive closure.

Now assume $A$ to be complemented and disjunctive. Clearly $A \subseteq A \longrightarrow A$ by Theorem 3.3.2, and therefore $A = \curlywedge(A) \subseteq A \stackrel{\curlywedge}{\longrightarrow} A$. Let $a \to b \in A \longrightarrow A$. Since $L$ is a Boolean algebra, it holds $a \to b = \neg a \vee b = (a \to \bot_L) \vee b$. Hence $a \to b \in A$ because $A$ is complemented and disjunctive. Thus $A \longrightarrow A \subseteq A$, and therefore $A = A \stackrel{\curlywedge}{\longrightarrow} A$.

Let $S$ be a solution to Eq. (11), i.e., $S \sqsubseteq A$ and $S = S \stackrel{\curlywedge}{\longrightarrow} S$. We first prove that

$$S \sqsubseteq \curlyvee(A \sqcap (A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\})).$$

Note that $\bot_L \in S$, since $\bot_L \in A \subseteq S$. Hence, by the previous point, $S = \curlyvee(S)$. Moreover, $S = S \stackrel{\curlywedge}{\longrightarrow} S \sqsubseteq S \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\}$. It follows that $S = S \sqcap S \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\}$ and $S = \curlyvee(S) = \curlyvee(S \sqcap S \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\})$. Since $S \sqsubseteq A$, by monotonicity, it holds that $S = \curlyvee(S \sqcap S \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\}) \sqsubseteq \curlyvee(A \sqcap (A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\}))$.

Now we prove that $\curlyvee(A \sqcap A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\})$ is a solution to Eq. (11). Clearly $\curlyvee(A \sqcap A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\}) \sqsubseteq A$, and it is trivially disjunctive by construction. Hence we have only to show that it is complemented. Let $a \in \curlyvee(A \sqcap A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\})$. We prove that $a \to \bot_L \in \curlyvee(A \sqcap A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\})$. By definition of reduced product, $a = \bigvee_{i\in I} a_i \wedge b_i$ with $a_i \in A$ and $b_i \in A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\}$, for some set $I$. Because $L$ is a Boolean algebra, we have

$$\begin{aligned} a \to \bot_L &= (\textstyle\bigvee_{i\in I} a_i \wedge b_i) \to \bot_L\\ &= \neg(\textstyle\bigvee_{i\in I} a_i \wedge b_i) \vee \bot_L\\ &= (\textstyle\bigwedge_{i\in I} \neg(a_i \wedge b_i)) \vee \bot_L\\ &= \textstyle\bigwedge_{i\in I}(\neg(a_i \wedge b_i) \vee \bot_L)\\ &= \textstyle\bigwedge_{i\in I}(\neg a_i \vee \neg b_i).\end{aligned}$$

With $\curlyvee(A \sqcap A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\})$ being a Moore family, we have only to prove that, for each $i \in I$, $\neg a_i \vee \neg b_i \in \curlyvee(A \sqcap A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\})$. Since $a_i \in A$ and $b_i \in A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\}$, it follows that $\neg a_i \in A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\}$ and $\neg b_i \in A$, because a Boolean algebra is uniquely complemented. Hence $\neg a_i, \neg b_i \in A \sqcap A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\}$ and $\neg a_i \vee \neg b_i \in \curlyvee(A \sqcap A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\})$.

Moreover, $\curlyvee(A \sqcap A \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\})$ being disjunctive, by Theorem 7.2.1, it is the most abstract solution of Eq. (12). $\square$

We conclude this section by observing that, from a purely algebraic point of view, the solutions of Eq. (11), and therefore those of Eq. (12), have relevant properties that provide a strong link between abstraction functions and *algebraic quantifiers*. The above results, together with Point (d) in Theorem 7.1.2, show that the solutions of Eq. (11) are all and only those closures $\rho \in uco(L)$ that satisfy the postulate $\rho(\rho(x) \wedge y) = \rho(x) \wedge \rho(y)$ of *quantifiers* in universal algebra [Halmos 1962]. This apparently useless observation has instead relevant applications in program analysis. In the case of logic program analysis, for instance, this fact implies that goal-directed and goal-independent analyses are equivalent in precision. The next section provides a domain-theoretic characterization of this phenomenon, which is known in logic program analysis as condensation.

## 8. APPLICATIONS II: CONDENSATION IN LOGIC PROGRAM ANALYSIS

In this section we prove that, for downward closed properties, the domains which are solutions of the recursive domain equation $X = A \sqcap X \xrightarrow{\wedge} X$ all satisfy the relevant property of being *condensing* (see Langen [1990] and Marriott and Søndergaard [1993]), and under additional nonrestrictive hypotheses, condensing domains are all and only the solutions of $X = A \sqcap X \xrightarrow{\wedge} X$. This property gives computational relevance in static program analysis to the results in Section 7 and shows a surprising link between Cousot's reduced cardinal power of domains and condensation.

The definition of condensation for a domain $X \in uco(L)$ can be given with respect to a generic abstract function $f^\sharp : X \longmapsto X$. The following definition is due to Marriott and Søndergaard [1993] and formalizes, for downward closed domains, an idea due to Langen [1990].

*Definition* 8.1. A domain $X \in uco(L)$ is *condensing* for $f^\sharp : X \longmapsto X$ if for any $x, y \in X$: $x \wedge f^\sharp(y) = f^\sharp(x \wedge y)$.

The above property of being condensing is relevant in particular in analysis of downward closed properties, e.g., type inference and groundness of (constraint) logic programs. This is because in this case $\wedge$ plays the role of (abstract) unification of constraints [Langen 1990; Marriott and Søndergaard 1993].

It is well known that, in general, goal-independent analyses of logic programs, like those obtainable with generic bottom-up/top-down analyzers [Barbuti et al. 1993; Bruynooghe 1991; Codish et al. 1994], are less precise than goal-directed ones (cf. Marriott and Søndergaard [1993]). This is because goal-independent analysis can be thought to be an analysis for all possible initial states, while in goal-directed analysis, a given query with a fixed initial state is executed in the abstract domain. In this latter case the result may be refined by knowing in advance the initial calls. More formally, if $true$ represents the top (weakest) constraint, representing any initial store, then, by Definition 8.1, for any constraint $c$ we have $c \wedge f^\sharp(true) = f^\sharp(c)$. The idea here is that computing the semantics of a query with initial constraint $c$, i.e., $f^\sharp(c)$, is the same as unifying $c$ with the result of the semantics of the same query in the most general environment $true$, which is $f^\sharp(true)$. This encodes the typical way we derive the analysis of a query in an initial state from a goal-independent analysis: we filter out, by unification, those computations which do not satisfy the given initial state (cf. Barbuti et al. [1993]). Unfortunately, for most domains, this information cannot be derived with the same precision as that obtainable in goal-directed analysis, in particular $c \wedge f^\sharp(true) \geq f^\sharp(c)$. However, as observed in Langen [1990] and Jacobs and Langen [1992], under the hypothesis that the abstract domain is condensing, no loss of precision is accumulated in deriving information about specific calls from goal-independent analyses.

Consider a simple constraint logic programming (*clp*) language (cf. Jaffar and

Lassez [1987]). We consider here a *ccp*-like presentation of *clp*: processes interact with each other by monotonically refining a common store, which is represented by a set of constraints [Saraswat et al. 1991]. For simplicity we do not consider here any synchronization mechanism, like blocking-ask; therefore processes are only allowed to *tell* constraints to a global store. Because of this restriction, our language boils down to a standard (nondeterministic) *clp* language. The syntax of programs is as follows, where $P$ are programs and $A$ are *clp*-agents:

$$\begin{array}{lcl} P & ::= & \varepsilon \mid p(\bar{x}) \leftarrow A \mid P.P \\ A & ::= & c \mid A \wedge A \mid \sum_{i=1}^{n} A_i \mid p(\bar{x}) \end{array}$$

We abuse by considering programs as (finite) sets of predicate or process declarations, and we assume that each process can be declared in at most one clause of the form $p(\bar{x}) \leftarrow A$. Also, we assume here that *Store* is a poset of constraints (i.e., a constraint system), ordered by logical implication, including equational constraints. This is a weak enough condition to include most constraint systems developed in the literature (e.g., see Saraswat et al. [1991]). The semantic domain is $\wp^{\downarrow}(Store)$, which is a cHa. We consider therefore downward closed properties only. The (forward) semantics $[\![\langle p(\bar{x}), \{c\}\rangle]\!]_P$ of a process $p$, executed with initial store $c \in Store$ in a program $P$, is defined as usual as the fixpoint of the following recursive function $\mathcal{S} : \wp^{\downarrow}(Store) \longmapsto \wp^{\downarrow}(Store)$, where for any $\sigma \in \wp^{\downarrow}(Store)$ we have

$$\begin{array}{lcl} \mathcal{S}_{P,c}(\sigma) & = & \downarrow c \wedge \sigma \\ \mathcal{S}_{P,A_1 \wedge A_2}(\sigma) & = & \mathcal{S}_{P,A_1}(\sigma) \wedge \mathcal{S}_{P,A_2}(\sigma) \\ \mathcal{S}_{P,\sum_{i=1}^{n} A_i}(\sigma) & = & \bigvee_{i=1}^{n} \mathcal{S}_{P,A_i}(\sigma) \\ \mathcal{S}_{P,p(\bar{x})}(\sigma) & = & \mathcal{S}_{P,A}(\sigma \wedge \bar{x} = \bar{y}) \ \text{ if } p(\bar{y}) \leftarrow A \ll P. \end{array}$$

Note that parameter passing is performed, as usual in logic programming, by telling an equation to the global store, which makes formal and actual parameters matching. In this definition $\ll$ selects a clause from $P$ whose variables are renamed apart from the rest of the computation.

The following result, which is a consequence of Theorem 7.1.2, states that it is always possible to systematically design a condensing domain for a given basic downward closed property of interest $A \in uco(\wp^{\downarrow}(Store))$, by solving the recursive domain equation $X = A \sqcap X \xrightarrow{\curlywedge} X$. In order to prove this statement, we consider the following abstract semantics of processes. The abstract semantics is associated with an abstract domain $X \in uco(\wp^{\downarrow}(Store))$, such that for any $\sigma \in X$ we have

$$\begin{array}{lcl} \mathcal{S}^X_{P,c}(\sigma) & = & \rho_X(\mathcal{S}_{P,c}(\sigma)) \\ \mathcal{S}_{P,A_1 \wedge A_2}(\sigma) & = & \mathcal{S}^X_{P,A_1}(\sigma) \wedge \mathcal{S}^X_{P,A_2}(\sigma) \\ \mathcal{S}_{P,\sum_{i=1}^{n} A_i}(\sigma) & = & \rho_X(\bigvee_{i=1}^{n} \mathcal{S}^X_{P,A_i}(\sigma)) \\ \mathcal{S}^X_{P,p(\bar{x})}(\sigma) & = & \mathcal{S}^X_{P,A}(\sigma \wedge \rho_X(\downarrow \bar{x} = \bar{y})) \ \text{ if } p(\bar{y}) \leftarrow A \ll P. \end{array}$$

In this case, for any process $p(\bar{x})$, abstract initial store $\sigma \in X$, and program $P$, we define $[\![\langle p(\bar{x}), \sigma\rangle]\!]^X_P = fix(\mathcal{S}^X_{P,p(\bar{x})}(\sigma))$. Note that, in each equation above, $\mathcal{S}^X$ is recursively defined as the best correct approximation of $\mathcal{S}$ in $X$.

THEOREM 8.2. *Let $P$ be a program, $A, X \in uco(\wp^{\downarrow}(Store))$, and $p(\bar{x})$ be a process. If $X = A \sqcap X \xrightarrow{\curlywedge} X$ then $X$ is condensing for $\lambda\sigma \in X.\ [\![\langle p(\bar{x}), \sigma\rangle]\!]^X_P$.*

PROOF. Let $X$ be a solution of the recursive domain equation $X = A \sqcap X \stackrel{\curlywedge}{\longrightarrow} X$. Let $\psi, \varphi \in X \subseteq \wp^{\downarrow}(Store)$. Because $X$ is a cHa by Theorem 7.1.2, then it is sufficient to prove, by induction on the recursive calls to the function $\mathcal{S}^X$ in the definition of $[\![\langle p(\bar{x}), \psi \wedge \varphi \rangle]\!]_P^X$, that $\mathcal{S}^X$ is condensing; i.e., $\mathcal{S}^X(\psi \wedge \varphi) = \psi \wedge \mathcal{S}^X(\varphi)$.

—Consider $\mathcal{S}_{P,c}^X$. By Point (d) in Theorem 7.1.2 we have

$$\begin{aligned} \mathcal{S}_{P,c}^X(\psi \wedge \varphi) &= \rho_X(\mathcal{S}_{P,c}(\psi \wedge \varphi)) \\ &= \rho_X(\downarrow c \wedge \psi \wedge \varphi) \\ &= \rho_X(\downarrow c) \wedge \psi \wedge \varphi \\ &= \psi \wedge \rho_X(\downarrow c \wedge \varphi) \\ &= \psi \wedge \mathcal{S}_{P,c}^X(\varphi). \end{aligned}$$

—Consider $\mathcal{S}_{P,A_1 \wedge A_2}^X$. By Point (d) in Theorem 7.1.2 and inductive hypothesis we have

$$\begin{aligned} \mathcal{S}_{P,A_1 \wedge A_2}^X(\psi \wedge \varphi) &= \mathcal{S}_{P,A_1}^X(\psi \wedge \varphi) \wedge \mathcal{S}_{P,A_2}^X(\psi \wedge \varphi) \\ &= \psi \wedge \mathcal{S}_{P,A_1}^X(\varphi) \wedge \mathcal{S}_{P,A_2}^X(\varphi) \\ &= \psi \wedge \mathcal{S}_{P,A_1 \wedge A_2}^X(\varphi). \end{aligned}$$

—Consider $\mathcal{S}_{P,\sum_{i=1}^n A_i}^X$. Because $X$ is a cHa by Theorem 7.1.2, then $X$ is a completely meet-distributive lattice. By inductive hypothesis and Point (d) in Theorem 7.1.2 we have

$$\begin{aligned} \mathcal{S}_{P,\sum_{i=1}^n A_i}^X(\psi \wedge \varphi) &= \rho_X(\textstyle\bigvee_{i=1}^n \mathcal{S}_{P,A_i}^X(\psi \wedge \varphi)) \\ &= \rho_X(\textstyle\bigvee_{i=1}^n \psi \wedge \mathcal{S}_{P,A_i}^X(\varphi)) \\ &= \rho_X(\psi \wedge \textstyle\bigvee_{i=1}^n \mathcal{S}_{P,A_i}^X(\varphi)) \\ &= \psi \wedge \rho_X(\textstyle\bigvee_{i=1}^n \mathcal{S}_{P,A_i}^X(\varphi)) \\ &= \psi \wedge \mathcal{S}_{P,\sum_{i=1}^n A_i}^X(\varphi). \end{aligned}$$

—Consider $\mathcal{S}_{P,p(\bar{x})}$. The thesis follows immediately by definition and inductive hypothesis. □

In order to prove the converse of Theorem 8.2, i.e., that condensing domains are solutions of Eq. (11), we need some additional hypotheses. Next result, together with Theorem 8.2, provides a constructive characterization of condensing domains which are at least as precise as a given domain $A$, as all and only those domains which are solutions of a recursive domain equation of the form $X = A \sqcap X \stackrel{\curlywedge}{\longrightarrow} X$.

THEOREM 8.3. *Let $P$ be a program, $A, X \in uco(\wp^{\downarrow}(Store))$, and $p(\bar{x})$ be a process. Assume that for any $x, y \in X$ there exist $\{c_1, \ldots c_n\} \subseteq Store$, $n < \omega$, such that the following conditions hold:*

—$\bigvee_{i=1}^n \downarrow c_i \leq x \to y$
—$\rho_X(\bigvee_{i=1}^n \downarrow c_i) = \rho_X(x \to y)$

*If $X$ is condensing for $\lambda\sigma \in X.\ [\![\langle p(\bar{x}), \sigma \rangle]\!]_P^X$ and $X \sqsubseteq A$ then $X = A \sqcap X \stackrel{\curlywedge}{\longrightarrow} X$.*

PROOF. Let $X$ be condensing. By Theorem 7.1.2, we have to prove that for any $a, b \in X$ we have $\rho(a \to b) = a \to b$. Suppose, by contradiction, that there exist $a, b \in X$ such that $\rho(a \to b) > a \to b$. By hypothesis there exists $\{c_1, \ldots c_n\} \subseteq$

*Store*, $n < \omega$ such that the above conditions hold. Consider the program $P = \{\, p :- \sum_{i=1}^{n} c_i \,\}$. Then, by hypothesis, we have

$$\begin{aligned} a \wedge [\![\langle p, \top\rangle]\!]_P^X &= a \wedge \rho_X(\textstyle\bigvee_{i=1}^{n} \rho_X(\downarrow c_i)) \\ &= a \wedge \rho_X(\textstyle\bigvee_{i=1}^{n} \downarrow c_i) \\ &= a \wedge \rho_X(a \to b) \\ &> a \wedge b \\ &= \rho_X(a \wedge a \to b) \\ &\geq \rho_X(a \wedge \textstyle\bigvee_{i=1}^{n} \downarrow c_i) \\ &= \rho_X(\textstyle\bigvee_{i=1}^{n} a \wedge \downarrow c_i) \\ &= \rho_X(\textstyle\bigvee_{i=1}^{n} \rho_X(a \wedge \downarrow c_i)) \\ &= \rho_X(\textstyle\bigvee_{i=1}^{n} \mathcal{S}_{P,c_i}^X(a)) \\ &= [\![\langle p, a\rangle]\!]_P^X. \end{aligned}$$

Therefore $X$ would not be condensing, which is a contradiction. $\square$

The hypothesis in Theorem 8.3 is not restrictive for most well-known downward closed domains in logic program analysis. Conditions (1) and (2) say that any implicational object $x \to y$ always approximates a finite disjunction of basic stores. This allows us, in Theorem 8.3, to construct, for any implicational object $x \to y$, a (finite) program having that object in the abstract semantics. The idea is that, in order to prove the converse of Theorem 8.2, any implicational object has to be the semantics of some well-defined program. The following characterization result of condensing is therefore immediate by Theorems 8.2 and 8.3.

COROLLARY 8.4. *Under the hypothesis of Theorem 8.3, $X$ is condensing for $\lambda\sigma \in X.\, [\![\langle p(\bar{x}), \sigma\rangle]\!]_P^X$ and $X \sqsubseteq A$ if and only if $X = A \sqcap X \xrightarrow{\curlywedge} X$.*

As observed in Example 7.1.3 (see Scozzari [1997]), *Pos* is the greatest solution of the recursive domain equation $X = \mathcal{G} \sqcap X \xrightarrow{\curlywedge} X$. This confirms, in view of Theorem 8.2, a result by Marriott and Søndergaard [1993], which was derived from some results in Langen [1990], saying that *Pos* is condensing. Moreover it is immediate to prove that *Pos* satisfies the hypothesis of Theorem 8.3. Therefore, we can conclude that *Pos* is the most abstract domain, including $\mathcal{G}$, which is condensing for our simple *clp*-language. This statement strengthens the result in Marriott and Søndergaard [1993] and proves that being disjunctive is not a necessary condition for being condensing. In fact we observe that, although *Pos* is not disjunctive, i.e., $Pos \neq \curlyvee(Pos)$ (cf. Filé and Ranzato [1994]), it is still a condensing domain. The key point in the treatment of disjunction in Theorems 8.2 and 8.3 is that the concrete domain and the solutions of the recursive domain equation $X = A \sqcap X \xrightarrow{\curlywedge} X$ are always cHa, i.e., these domains are completely meet-distributive. Therefore, even if they are not disjunctive, still no loss of precision is accumulated by distributing the glb (abstract unification) over nondeterministic computations, which are modeled by abstract disjunction.

As observed above, condensing domains are fundamental in order to apply goal-independent analysis to logic and constraint-based languages, yet providing the same precision as in a goal-directed one [Langen 1990; Marriott and Søndergaard 1993]. Because of this key property, condensation is known to be an important property of domains for logic program analysis. Clearly, by our results, the same property of condensation holds for any downward closed domain which is a solution of Eq. (11), including those for directional type inference in Section 6. The following example shows this phenomenon for a simple domain for directional type inference.

*Example* 8.5. Consider the domain of basic types $\mathcal{T}$ defined in Example 6.1.3 and the simple program $P = \{\, p(X,Y) :- X = Y.\, \}$. We fix the variables $X$ and $Y$ as variables of interest for the predicate $p(X,Y)$ and consider the abstract constraint $p(I,\top)$ whose concretization is the set of all constraints where $X$ is an integer.

$$\begin{aligned}
[\![\langle p(X,Y), p(I,\top)\rangle]\!]^{\mathcal{T}}_P &= \mathcal{S}^{\mathcal{T}}_{P,p(X,Y)}(p(I,\top)) \\
&= \mathcal{S}^{\mathcal{T}}_{P,\{\, X=Y\,\}}(p(I,\top)) \\
&= \rho_{\mathcal{T}}(\mathcal{S}_{P,\{\, X=Y\,\}}(p(I,\top))) \\
&= \rho_{\mathcal{T}}(\downarrow \{\, X=Y\,\} \wedge p(I,\top)) \\
&= p(I,I)
\end{aligned}$$

$$\begin{aligned}
[\![\langle p(X,Y), p(\top,\top)\rangle]\!]^{\mathcal{T}}_P &= \mathcal{S}^{\mathcal{T}}_{P,p(X,Y)}(p(\top,\top)) \\
&= \mathcal{S}^{\mathcal{T}}_{P,\{\, X=Y\,\}}(p(\top,\top)) \\
&= \rho_{\mathcal{T}}(\mathcal{S}_{P,\{\, X=Y\,\}}(p(\top,\top))) \\
&= \rho_{\mathcal{T}}(\downarrow \{\, X=Y\,\} \wedge p(\top,\top)) \\
&= \rho_{\mathcal{T}}(\downarrow \{\, X=Y\,\}) \\
&= p(\top,\top)
\end{aligned}$$

In this case $p(I,\top) \wedge [\![\langle p(X,Y), p(\top,\top)\rangle]\!]^{\mathcal{T}}_P = p(I,\top) \wedge p(\top,\top) = p(I,\top)$ is different from $[\![\langle p(X,Y), p(I,\top)\rangle]\!]^{\mathcal{T}}_P = p(I,I)$. This proves that the abstract domain $\mathcal{T}$ is not condensing. By Theorem 8.2, a condensing domain for directional type inference can be systematically designed by solving the recursive domain equation $X = \mathcal{T} \sqcap X \xrightarrow{\curlywedge} X$.

## 9. RELATED WORKS

In the very last section of Cousot and Cousot [1979], the authors introduced the notion of reduced cardinal power, as a domain of monotone functions between two given abstract domains. In Cousot and Cousot [1979, Theorem 10.2.0.1] the authors observed that the concretization of objects in the reduced cardinal power can be expressed by classical implication. This was mainly a consequence of the structure of the underlying concrete domain, which was assumed to be a complete Boolean algebra. This construction was reconsidered and generalized in Giacobazzi and Ranzato [1998], where the authors observed that the weaker hypothesis of cHa is sufficient to ensure the existence of reduced cardinal power of abstract domains. Although in Cousot and Cousot [1979] the authors already observed that functions in reduced cardinal power represent classical implications, none of these papers gave a full logical characterization for the operation of reduced cardinal power in the most general case of cHa. This is a limitation of the above approaches, because in most cases, e.g., those considered in Sections 6 and 8, the concrete domain is not a Boolean algebra. Moreover, as observed in Section 6.2, Boolean algebras may provide the wrong basis to characterize directionality in terms of reduced cardinal power. In addition, none of these papers relate reduced cardinal power with other known operations for domain manipulation, and none of them considered the case of domain equations involving reduced cardinal power and other refinements as basic operators. This is particularly important in abstract interpretation because the design of a domain is usually the combination of several refinements, and in this process, a clean logical interpretation for all these operations and their interaction makes proofs and technical development more easy to understand. The idea of domain refinement by (classical) complements was introduced in Filé et al. [1996], and later used in Giacobazzi and Ranzato [1997b], to show that there exist refinements that are not invertible, i.e., for which it is not possible to define an inverse operation which returns the (unique) most abstract domain which has the same refinement as the refinement of a given input domain. None of these works, however, generalized the idea to intuitionistic implications. Nielson's *tensor product* [Nielson 1985] is also known as a basic operator for relational analyses construction. The tensor product $\otimes$ is a binary operation on domains such that $A \otimes B = (A \xrightarrow{a} B^{\mathrm{op}})^{\mathrm{op}}$. The relation between reduced cardinal power and Nielson's tensor product has been studied in Giacobazzi and Ranzato [1998], where, under suitable hypothesis, it is proven that $A \xrightarrow{a} B = B^A$. Therefore, by Theorem 5.3, under these conditions, it is possible to give a logical interpretation of tensor product in terms of intuitionistic implication. Recently, the relation between completeness in abstract interpretation and logical implication has been studied in [Giacobazzi et al. 1998b]. This result relies upon a generalization of our construction from intuitionistic to linear implication.

On the side of domains for program analysis, it would be impossible to give here a full and comprehensive account on all domains for relational program analysis, which include some functionalities which can be provided by reduced cardinal power. These include, among the others, the domains proposed in Aiken and Lakshman [1994], Armstrong et al. [1994], Bronsard et al. [1992], Dart [1991], Giacobazzi and Ranzato [1995], Codish and Demoen [1994], and Yi and Harrison [1993]. However, to the best of our knowledge, the first explicit use of reduced cardinal power to design domains in abstract interpretation, after its original definition in Cousot and Cousot [1979], is in Giacobazzi and Ranzato [1995], where the authors derive the domain $Def$ for ground dependency analysis and a compositional semantics for logic programs as the reduced cardinal power of corresponding abstractions for groundness and noncompositional semantics, respectively. An interesting example on a practical way for implementing domains which can be derived by reduced cardinal power has been recently proposed in logic program analysis, by Codish and Demoen [1994]. In this article the authors used the directional information obtainable for groundness analysis in $Pos$ (also called $Prop$ [Armstrong et al. 1994]) to efficiently implement a variety of type inference algorithms by abstract compilation. In Section 6 we have shown that both $Pos$ and directional types can be obtained by Heyting completion. Because Heyting completion captures precisely the common structure which surfaces directionality in all these applications, it seems natural to think that the approach in Codish and Demoen [1994] can be used to derive prototype implementations of reduced cardinal power of any domain which is downward closed. In Scozzari [1997] Heyting completion has been applied to groundness analysis in logic programming. These results extend those sketched in Section 6.2 and characterize a variety of abstract domains for groundness analysis in terms of Heyting completion. The use of (recursive) domain equations for specifying domains in static analysis was already considered in Benton [1992a]. In his PhD thesis, Benton used recursive domain equations to extend a simply typed lazy functional language to include algebraic data-types. The author proposes a method where abstract domains for analysis of algebraic data-types are constructed as solutions of a recursive domain equation associated with a given type. This construction is similar to what we propose here for characterizing domains by Heyting completion (e.g., see the characterization of *Pos* in Section 6 and that of condensing in Section 8). However, these two methods differ on the way equations are defined. In our case, domain equations are not derived from any concrete data-type definition, but they are instead specified in terms of a combination of suitable domain refinements. Domain refinements may in fact be not related with any concrete data-type used in the considered programming language. In principle, a domain refinement is just an operation to enhance an input domain by including new information, which can be either derived by some concrete data-type, or be just part of the concrete domain of computation (see Giacobazzi and Ranzato [1997b]). This makes our approach applicable to derive domains for any (possibly untyped) programming language. As a matter of fact, in the case of Heyting completion considered here, the greatest solution to Eq. (11) does not strictly reflect any concrete data-type, but instead it specifies the least amount of information that a domain has to contain in order to both include a basic domain of properties and allow that these properties can be propagated on domain's objects by conjunction (see Theorem 7.1.2). For instance, in logic programming, the solution of the above domain equation amounts to finding the most abstract domain which is condensing and includes a given domain of basic properties.

## 10. CONCLUSIONS AND FUTURE WORK

The main contribution of this work, beside the logical characterization of Cousot's reduced cardinal power in terms of Heyting completion, is to supply a logical interpretation for the algebra of operators introduced in Cousot and Cousot [1979] for manipulating abstract domains. Domains can no longer be specified only in terms of their elements, but they can also be specified in terms of the property they have to meet, e.g., domains can be specified in equational form, as domains in denotational semantics. Heyting completion is, in this construction, the basic step to build relational domains and analyses. This made it possible to give a clean logical characterization of important properties of domains for program analysis, like condensing and directionality. Although this use of Heyting completion in domain refinement provides a better understanding for a number of properties of domains, the equivalence between reduced cardinal power and Heyting completion opens interesting problems that still deserve further investigation. Our result in fact makes explicit a strong link between domains traditionally used in program analysis and *nonstandard type inference*. The idea is that the domain $A$, in the domain equation $X = A \sqcap X \xrightarrow{\curlywedge} X$, may represent some sort of *basic type* information which can be unrelated to any concrete data-type. The solution of the above domain equation therefore specifies a domain of nonstandard types, where product and function types still have the meaning, respectively, as conjunction and implication, similarly to the standard semantics of typed $\lambda$-calculus. It seems therefore natural to associate with each abstract domain, obtained as solution of the above domain equation, e.g., *Pos* for ground dependency in logic programming (see Example 7.1.3), a corresponding domain of nonstandard types, and with each abstract interpretation based on that domain, a corresponding nonstandard type inference algorithm, which can be used to derive static program analysis information. The idea of using type information in static program analysis was first considered in Jensens's [1995] and Benton's [1992b] strictness logic and later used by Hankin and Le Metayer [1994]. In these works, the authors proved that some relevant properties of data-structures during program execution can be inferred by a simple type system, in a kind of *proof-theoretic* approach to program analysis. However, this approach is different from what we are looking for. As observed above in Section 9, the type inference based program analysis in Benton [1992a] is always derived by abstracting the definition of concrete types for the considered programming language. Because of this, nowadays, the use of these so-called type inference based methods for static analysis is mostly restricted to functional languages. We believe instead that any abstract interpretation, for any programming language, whose domains are solutions of the above domain equation, may be joint with a type inference system, which can be in general completely unrelated to any concrete data-type, but still provides correct information about program's data-flow. In this sense, the relation between types and solutions of Eq. (11) can be further investigated in order to let nonstandard type inference be applicable in the analysis of other programming languages, including logic and constraint-based languages.

## REFERENCES


AIKEN, A. AND LAKSHMAN, T. 1994. Directional type checking of logic programs. In *Proceedings of the 1st International Static Analysis Symposium (SAS '94)*, B. L. Charlier, Ed. Lecture Notes in Computer Science, vol. 864. Springer-Verlag, Berlin, 43–60.

APT, K. 1990. Introduction to logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B, *Formal Models and Semantics.* Elsevier, Amsterdam, 495–574.

ARMSTRONG, T., MARRIOTT, K., SCHACHTE, P., AND SØNDERGAARD, H. 1994. Boolean functions for dependency analysis: Algebraic properties and efficient representation. In *Proceedings of the 1st International Static Analysis Symposium (SAS '94)*, B. Le Charlier, Ed. Lecture Notes in Computer Science, vol. 864. Springer-Verlag, Berlin, 266–280.

BARBUTI, R., GIACOBAZZI, R., AND LEVI, G. 1993. A general framework for semantics-based bottom-up abstract interpretation of logic programs. *ACM Trans. Program. Lang. Syst. 15,* 1, 133–181.

BENTON, P. 1992a. Strictness analysis of lazy functional programs. Ph.D. thesis, University of Cambridge, Cambridge, U.K.

BENTON, P. 1992b. Strictness logic and polymorphic invariance. In *Proceedings of the 2nd International Symposium on Logical Foundation of Computer Science*, A. Nerode and M. Taitslin, Eds. Lecture Notes in Computer Science, vol. 620. Springer-Verlag, Berlin, 33–44.

BIRKHOFF, G. 1967. *Lattice Theory.* AMS Colloquium Publication, 3rd edition. AMS, Providence, R.I.

BRONSARD, F., LAKSHMAN, T., AND REDDY, U. 1992. A framework of directionality for proving termination of logic programs. In *Proceedings of the 1992 Joint International Conference and Symposium on Logic Programming.* MIT Press, Cambridge, Mass., 321–335.

BRUYNOOGHE, M. 1991. A practical framework for the abstract interpretation of logic programs. *J. Logic Programm. 10*, 91–124.

CLICK, C. AND COOPER, K. D. 1995. Combining analyses, combining optimizations. *ACM Trans. Program. Lang. Syst. 17,* 2, 181–196.

CODISH, M., DAMS, D., AND YARDENI, E. 1994. Bottom-up abstract interpretation of logic programs. *Theor. Comput. Sci. 124,* 1, 93–126.

CODISH, M. AND DEMOEN, B. 1994. Deriving type dependencies for logic programs using multiple incarnations of *prop*. In *Proceedings of the 1st International Static Analysis Symposium*

(*SAS '94*), B. L. Charlier, Ed. Lecture Notes in Computer Science, vol. 864. Springer-Verlag, Berlin, 281–297.

Codish, M., Mulkers, A., Bruynooghe, M., García de la Banda, M., and Hermenegildo, M. 1995. Improving abstract interpretations by combining domains. *ACM Trans. Program. Lang. Syst. 17,* 1, 28–44.

Cortesi, A., Filé, G., Giacobazzi, R., Palamidessi, C., and Ranzato, F. 1997. Complementation in abstract interpretation. *ACM Trans. Program. Lang. Syst. 19,* 1, 7–47.

Cortesi, A., Filé, G., and Winsborough, W. 1996. Optimal groundness analysis using propositional logic. *J. Logic Program. 27,* 2, 137–167.

Cortesi, A., Le Charlier, B., and Van Hentenryck, P. 1994. Combinations of abstract domains for logic programming. In *Conference Record of the 21st ACM Symposium on Principles of Programming Languages* (*POPL '94*). ACM Press, New York, 227–239.

Cousot, P. 1978. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes. Ph.D. thesis, Université Scientifique et Médicale de Grenoble, Grenoble, France.

Cousot, P. 1996. Abstract interpretation. *ACM Comput. Surv. 28,* 2, 324–328.

Cousot, P. and Cousot, R. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages* (*POPL '77*). ACM Press, New York, 238–252.

Cousot, P. and Cousot, R. 1979. Systematic design of program analysis frameworks. In *Conference Record of the 6th ACM Symposium on Principles of Programming Languages* (*POPL '79*). ACM Press, New York, 269–282.

Cousot, P. and Cousot, R. 1992. Abstract interpretation and application to logic programs. *J. Logic Program. 13,* 2-3, 103–179.

Cousot, P. and Cousot, R. 1994. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages). In *Proceedings of the IEEE International Conference on Computer Languages* (*ICCL '94*). IEEE Computer Society Press, Los Alamitos, Calif., 95–112.

Dams, D., Gerth, R., and Grumberg, O. 1997. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst. 19,* 2, 253–291.

Dart, P. 1991. On derived dependencies and connected databases. *J. Logic Program. 11,* 2, 163–188.

Davey, B. A. and Priestley, H. A. 1990. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, U.K.

Filé, G., Giacobazzi, R., and Ranzato, F. 1996. A unifying view of abstract domain design. *ACM Comput. Surv. 28,* 2, 333–336.

Filé, G. and Ranzato, F. 1994. Improving abstract interpretations by systematic lifting to the powerset. In *Proceedings of the 1994 International Logic Programming Symposium* (*ILPS '94*), M. Bruynooghe, Ed. The MIT Press, Cambridge, Mass., 655–669.

Filé, G. and Ranzato, F. 1998. The powerset operator on abstract interpretations. To appear in *Theor. Comput. Sci.*. Preliminary version in Filé and Ranzato [1994].

Freese, R., Jezek, J., and Nation, J. 1995. *Free lattices.* Mathematical surveys and monographs, vol. 42. AMS, Providence, R.I.

Giacobazzi, R. and Ranzato, F. 1995. Functional dependencies and Moore-set completions of abstract interpretations and semantics. In *Proceedings of the 1995 International Symposium on Logic Programming* (*ILPS '95*), J. Lloyd, Ed. The MIT Press, Cambridge, Mass., 321–335.

Giacobazzi, R. and Ranzato, F. 1996. Compositional optimization of disjunctive abstract interpretations. In *Proc. of the 1996 European Symposium on Programming*, H. Nielson, Ed. Lecture Notes in Computer Science, vol. 1058. Springer-Verlag, Berlin, 141–155.

Giacobazzi, R. and Ranzato, F. 1997a. Completeness in abstract interpretation: A domain perspective. In *Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology* (*AMAST '97*), M. Johnson, Ed. Lecture Notes in Computer Science, vol. 1349. Springer-Verlag, Berlin, 231–245.

GIACOBAZZI, R. AND RANZATO, F. 1997b. Refining and compressing abstract domains. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP '97)*, P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, Eds. Lecture Notes in Computer Science, vol. 1256. Springer-Verlag, Berlin, 771–781.

GIACOBAZZI, R. AND RANZATO, F. 1998. The reduced relative power operation of abstract domains. To appear in *Theor. Comput. Sci.*. Preliminary version in Giacobazzi and Ranzato [1995].

GIACOBAZZI, R., RANZATO, F., AND SCOZZARI, F. 1998b. Building complete abstract interpretations in a linear logic-based setting. In *Proceedings of Fifth International Static Analysis Symposium (SAS'98)*, G. Levi, Ed. Lecture Notes in Computer Science. Springer-Verlag, Berlin. To appear.

GIACOBAZZI, R., RANZATO, F., AND SCOZZARI, F. 1998a. Complete abstract interpretations made constructive. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS '98)*, J. Gruska and J. Zlatuska, Eds. Lecture Notes in Computer Science, vol. 1450. Springer-Verlag, Berlin, 366–377.

GIRARD, J.-Y., LAFONT, Y., AND TAYLOR, P. 1989. *Proofs and Types*. Cambridge University Press, Cambridge, U.K.

HALMOS, P. 1962. *Algebraic Logic*. Chelsea Publ. Co., New York.

HANKIN, C. AND METAYER, D. L. 1994. Deriving algorithms from type inference systems: Applications to strictness analysis. In *Conference Record of the 21st ACM Symposium on Principles of Programming Languages (POPL '94)*. ACM Press, New York, 202–212.

JACOBS, D. AND LANGEN, A. 1992. Static analysis of logic programs for independent AND-parallelism. *J. Logic Program. 13,* 2-3, 154–165.

JAFFAR, J. AND LASSEZ, J.-L. 1987. Constraint Logic Programming. In *Conference Record of the ACM Symposium on Principles of Programming Languages (POPL '87)*. ACM Press, New York, 111–119.

JENSEN, T. 1995. Conjunctive type systems and abstract interpretation of higher-order functional languages. *J. Logic Comput. 5,* 4, 397–421.

JENSEN, T. P. 1992. Disjunctive strictness analysis. In *Proceedings of the 7th IEEE Symposium on Logic in Computer Science (LICS '92)*. IEEE Computer Society Press, Los Alamitos, Calif., 174–185.

JONES, N. D. AND MUCHNICK, S. S. 1981. Complexity of flow analysis, inductive assertion synthesis and a language due to dijkstra. In *Program Flow Analysis: Theory and Applications*, S. S. Muchnick and N. D. Jones, Eds. Prentice-Hall, Upper Saddle River, N.J., 380–393.

JONES, N. D. AND SØNDERGAARD, H. 1987. A semantics-based framework for the abstract interpretation of Prolog. In *Abstract Interpretation of Declarative Languages*, S. Abramsky and C. Hankin, Eds. Ellis Horwood Ltd, Chichester, UK, 123–142.

LANGEN, A. 1990. Static analysis for independent And-parallelism in logic programs. Ph.D. thesis, Univ. of Southern California, Los Angeles, Calif.

MARRIOTT, K. AND SØNDERGAARD, H. 1993. Precise and efficient groundness analysis for logic programs. *ACM Lett. Program. Lang. Syst. 2,* 1-4, 181–196.

MORGADO, J. 1960. Some results on the closure operators of partially ordered sets. *Portugaliæ-Mathematica 19,* 2, 101–139.

MUTHUKUMAR, K. AND HERMENEGILDO, M. 1991. Combined determination of sharing and freeness of program variables through abstract interpretation. In *Proceedings of the 8th International Conference on Logic Programming (ICLP '91)*, K. Furukawa, Ed. The MIT Press, Cambridge, Mass., 49–63.

MYCROFT, A. 1993. Completeness and predicate-based abstract interpretation. In *Proceedings of the ACM Symposium on Partial Evaluation and Program Manipulation (PEPM '93)*. ACM Press, New York, 179–185.

MYCROFT, A. AND NIELSON, F. 1983. Strong abstract interpretation using power domains. In *Proceedings of the 10th International Colloquium on Automata, Languages and Programming (ICALP '83)*, J. Díaz, Ed. Lecture Notes in Computer Science, vol. 154. Springer-Verlag, Berlin, 536–547.

NIELSON, F. 1985. Tensor products generalize the relational data flow analysis method. In *Proceedings of the 4th Hungarian Computer Science Conference*, M. Arató, I. Kátai, and L. Varga, Eds. 211–225.

SARASWAT, V., RINARD, V. A., AND PANANGADEN, P. 1991. Semantic foundations of concurrent constraint programming. In *Conference Record of the 18th ACM Symposium on Principles of Programming Languages* (*POPL '91*). ACM Press, New York, 333–353.

SCOZZARI, F. 1997. Logical optimality of groundness analysis. In *Proceedings of the 4th International Static Analysis Symposium* (*SAS '97*), P. Van Hentenryck, Ed. Lecture Notes in Computer Science, vol. 1302. Springer-Verlag, Berlin, 83–97.

SUNDARARAJAN, R. AND CONERY, J. 1992. An abstract interpretation scheme for groundness, freeness, and sharing analysis of logic programs. In *Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science* (*FST&TCS '92*), R. Shyamasundar, Ed. Lecture Notes in Computer Science, vol. 652. Springer-Verlag, Berlin, 203–216.

VAN DALEN, D. 1994. *Logic and Structure.* 3rd ed. Springer-Verlag, Berlin.

VICKERS, S. 1989. *Topology via Logic.* Cambridge University Press, Cambridge, U.K.

WARD, M. 1942. The closure operators of a lattice. *Ann. Math. 43,* 2, 191–196.

YI, K. AND HARRISON, W. L. 1993. Automatic generation and management of interprocedural program analyses. In *Conference Record of the 20th ACM Symposium on Principles of Programming Languages* (*POPL '93*). ACM Press, New York, 246–259.

Received January 1998; accepted May 1998