Università degli Studi di Siena
Dipartimento di Matematica
Dottorato di Ricerca in Logica Matematica e Informatica Teorica

Ph.D. Thesis

# Domain theory in abstract interpretation: equations, completeness and logic

Francesca Scozzari

26 February 1999

Dipartimento di Matematica, Via del Capitano 15, I-53100 Siena
E-mail: scozzari@di.unipi.it

# Acknowledgements

This Ph.D. thesis is the work of many people, who made it possible to conceive and write it, and therefore deserve to be acknowledged. I would like to start by thanking my advisors Roberto Giacobazzi and Giorgio Levi. Roberto guided and helped me develop most of my work during the last three years. He taught me to unfold my ideas and to be a bit more enthusiastic in my daily work. Above all, it has been very stimulating to come to know and even to dispute his ideas. His guidance is surely one of the main reasons for the existence in this form of this thesis.

Giorgio has been successively one of my teachers at the university, advisor in my Master thesis, and finally in my Ph.D. thesis. Most of the motivations which led me to write this thesis and, in general, to work on research, come from him and from his vision of the research world. I am sincerely indebted to him for his generous advice, his help in so many forms, and, mainly, for his constant smiling, in spite of everything.

I have also been lucky to work closely with Francesco Ranzato. I thank him for lots of stimulating and extremely helpful discussions.

I would like to thank all my colleagues at the Department of Mathematics in Siena, the coordinator of my Ph.D. program, Franco Montagna, and, particularly, Michele Pinna, who is an especially precious friend. I also thank all my colleagues and friends at the Department of Computer Science in Pisa, where I have written most of the thesis. Amongst them, special thanks to my colleagues, but above all special friends, Marco Comini, Fabio Gadducci, Roberta Gori, and Alessandra Raffaetà, all for a variety of (and very different) reasons, which would be too difficult and too long to list here. I would like to thank the *Laboratoire d'Informatique (LIX)*, at *Ècole Polytechnique*, Paris, for hosting me several months, during and after the preparation of the thesis. In particular, I would like to express my gratitude to Radhia Cousot for introducing me in her research group, and to Ian Mackie and Jorge Sousa Pinto for all the coffee breaks.

I am also indebted to the two referees of this thesis, Patrick Cousot and Harald Søndergaard, for a careful reading of the thesis and for lots of suggestions.

Most of all my greatest thanks go to Raffaella, Marco, Sabrina and my parents for being, at the same time, so far from and so close to me.

This thesis was conceived and written with the constant and tender encouragement of Lorenzo, for whom I reserve my very special thanks.

To Lorenzo

# Contents

# Chapter 1

# From theory to practice

## 1.1 Abstract interpretation

Abstract interpretation has been introduced by Patrick and Radhia Cousot in [1977] as a general theory for specifying and then validating program analyses. In recent years, abstract interpretation 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 $[\![\cdot]\!] : Program \longmapsto C$, associating with each program $P \in Program$ its semantics $[\![P]\!] \in C$. According to the classical Cousot and Cousot's framework, an abstract interpretation is defined as a nonstandard (approximated) program semantics, which is obtained from the standard (or concrete) one by replacing the concrete domain $C$ of computation and its semantics $[\![\cdot]\!]$ with, respectively, an *abstract domain* $A$ and an *abstract semantics* $[\![\cdot]\!]^\sharp : Program \longmapsto A$, providing the abstract semantics to correctly approximate the concrete semantics, that is to say, for each program $P$, $[\![P]\!]^\sharp$ represents some concrete object $c \in C$ such that $[\![P]\!] \leq c$. One of the most fundamental facts of abstract interpretation is that most properties of the abstract interpretation $\langle A, [\![\cdot]\!]^\sharp\rangle$, like precision, completeness, and compositionality, all depend on the notion of *abstraction* only, which is precisely and uniquely specified by the chosen domain of properties $A$ (cf. Cousot and Cousot [1979b]). Central in the construction of an abstract interpretation is therefore the notion of *abstract domain*.

### 1.1.1 Abstract domains

An abstract domain 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 simple lattice $\mathcal{N}at$ in Figure 1.1 is an abstract domain which provides information to detect whether an integer variable is a (strictly) positive number. In this domain, the elements $\mathbb{Z}$, $\geq 0$, and $> 0$ represent the corresponding sets of integers. Intuitively, a set $A$ of abstract objects is an abstract domain (relatively to some concrete domain $C$) if, for each concrete object $c \in C$, there exists in $A$ a (unique) best approximation of $c$. In other words, it is possible to define an abstraction function $\alpha : C \longmapsto A$ which associates with any concrete value $c$ its best approximation $\alpha(c)$ in such a way that, for any other abstract value $b \in A$ which still approximates $c$, it holds that $a \leq b$. An important feature of abstract interpretation is that any abstract domain $A$ is isomorphic to a subset of the concrete domain $C$, up to the representation of its objects. Thus we can consider only those abstract domains which are subsets of the concrete domain, without losing any generality. This allows us to easily compare abstract domains with respect to their relative precision. We say that an abstract domain $A$ is more concrete than $B$ (or, equivalently, $B$ is more abstract than $A$) if $B \subseteq A$. More formally, the set of all abstract domains is isomorphic to the set of all upper closure operators on the concrete domain $C$. Therefore, in the following, given any concrete domain $C$, we denote by $uco(C)$ the set of all abstract domains on $C$ and given any abstract domain $A \in uco(C)$ we denote by $\alpha_A$ the associated abstraction function.

Because of the key rôle 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: *modify domains corresponds to modify abstract interpretations.* The foundation of a theory of domains for abstract interpretation was fixed by Cousot and Cousot [1979b], 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 1979b; Cousot and Cousot 1994; Filé

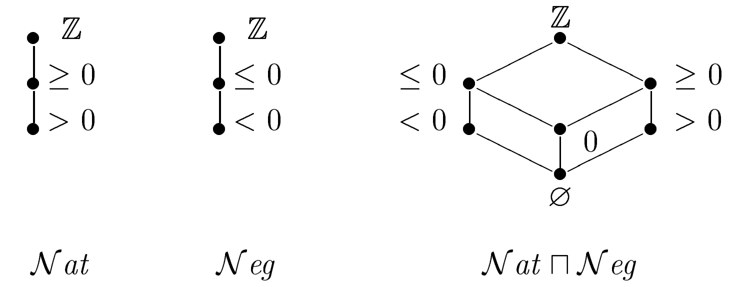



Figure 1.1: Reduced product

and Ranzato 1994; Giacobazzi and Ranzato 1998b; Giacobazzi and Ranzato 1997a; Jensen 1992; Nielson 1985], simplification [Giacobazzi and Ranzato 1997a], decomposition [Cortesi et al. 1997], and compression [Giacobazzi and Ranzato 1998a], and some of them have been included as tools for design aid in modern systems for program analysis, like in GAIA [Le Charlier and Van Hentenryck 1994], PLAI [Muthukumar and Hermenegildo 1992], 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 1979b], 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 $\mathcal{N}eg$ in Figure 1.1. The reduced product of the two domains, denoted by $\mathcal{N}at \sqcap \mathcal{N}eg$, is a new, richer domain which includes both the given domains and is closed under intersection. Intersection plays the rôle 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 non-positive ($\leq 0$). 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.

Another well-known operation for domain transformation, which has a clean logical interpretation, is *disjunctive completion*, also introduced in Cousot and Cousot [1979b]. 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; 1992a], Filé and

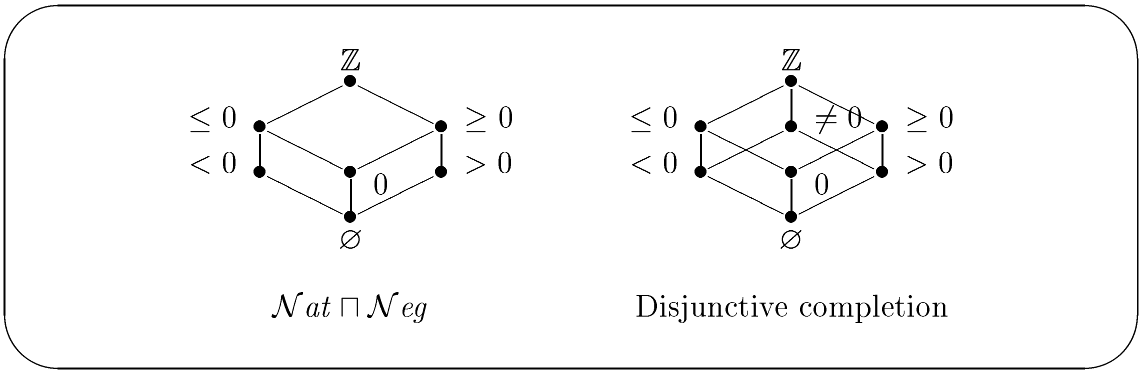



Figure 1.2: Disjunctive completion

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]. The disjunctive completion of a given domain $A$ is the most abstract domain which includes $A$ 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 property for a program variable of being either positive or negative, as shown in Figure 1.2.

### 1.1.2 Abstract functions: correctness and completeness

Given a concrete domain $C$ and a concrete function $f : C \longmapsto C$, an abstract interpretation is defined by an abstract domain $A$ and an abstract function $f^\sharp : A \longmapsto A$, providing the abstract function to correctly approximate the concrete one, that is to say, for each $c \in C$, $f^\sharp(\alpha_A(c))$ is an approximation of $f(c)$, where $\alpha_A : C \longmapsto A$ is the abstraction function associating with any concrete object $c \in C$ its best approximation $\alpha_A(c)$ in $A$. Associated with any abstract domain $A$, there always exists the best correct approximation of the concrete function $f$, which is precisely $\alpha \circ f$. As a consequence, it turns out that many interesting properties (e.g., concerning the precision of abstract computations) uniquely depend on the choice of the abstract domain. This justifies the increasing amount of work to describe techniques to combine and manipulate abstract domains, disregarding the associated abstract functions. Nevertheless, choosing the best correct approximation as abstract function does not ensure us that the abstract computations are as precise as the concrete ones. In other words, it may happen that for an abstract interpretation $\langle A, f^\sharp \rangle$, the result of an abstract computation is not the best approximation in $A$ of the result of the corresponding concrete one, even if $f^\sharp$ is the best correct approximation of $f$. In this case we say that the abstract interpretation $\langle A, f^\sharp\rangle$ is not complete for the concrete function $f$.

As an example, consider the abstract domain $\mathcal{N}at$ in Figure 1.1, and suppose we are interested in computing the successor of a set (which is the set of all successors), i.e., $Succ(N) = \{\, n+1 \mid n \in N\,\}$. In this case we want to observe when the sign of the successor is (strictly) positive. The best correct approximation of the function $Succ$ is defined as follows:

$$\begin{array}{lcl} Succ^{\mathcal{N}at}(>0) & = & >0 \\ Succ^{\mathcal{N}at}(\geq 0) & = & >0 \\ Succ^{\mathcal{N}at}(\mathbb{Z}) & = & \mathbb{Z} \end{array}$$

Consider now the concrete object $\{\,-1\,\}$ and suppose we want to determine the sign of its successor. A concrete computation would answer $\geq 0$. If we consider its best approximation in $\mathcal{N}at$, which is $\mathbb{Z}$, we obtain the result $\mathbb{Z}$, which is strictly less precise than $\geq 0$. This shows that the abstract interpretation $\langle \mathcal{N}at, Succ^{\mathcal{N}at}\rangle$ is not complete, even though $Succ^{\mathcal{N}at}$ is the best correct approximation in $\mathcal{N}at$ of the concrete function $Succ$.

Formally, an abstract interpretation $\langle A, f^\sharp\rangle$ is complete for the concrete function $f$ if, for any $c \in C$, the result of an abstract computation $f^\sharp(\alpha(c))$ is the best approximation in $A$ of the result of the concrete computation $f(c)$, namely $\alpha(f(c)) = f^\sharp(\alpha(c)))$. Therefore, *completeness* for an abstract interpretation is a much richer property than plain mandatory correctness. Roughly speaking, a complete abstract interpretation turns out to be as precise as possible, relatively to its underlying abstract domains where approximated computations are encoded. This simple intuition explains why, although being a rather uncommon property in practice, notably in static program analysis, completeness is a highly desirable feature for an abstract interpretation, especially in abstract model checking (indeed, some authors term it "optimality", e.g., [Dams et al. 1997]). Although in static program analysis decidability issues often force to sacrifice completeness for achieving termination and/or efficiency, examples of complete abstract interpretations are common in other fields of application. For instance, several complete abstractions of algebraic polynomial systems have been studied by Cousot and Cousot in [Cousot and Cousot 1997] and many complete abstract interpretations can be found in comparative program semantics [Cousot 1997a; Cousot and Cousot 1992b; Giacobazzi 1996]. Moreover, completeness being a notion relative to the concrete semantics of reference, complete abstract interpretations which are more concrete than a certain, possibly approximated and decidable, property of interest, yield an absolute upper bound for the precision that one can achieve in computing that property. Thus, complete abstract interpretations may play a useful rôle in static program analysis as well. These considerations probably stimulated the recent trend of research on completeness in abstract interpretation [Cousot 1995; Giacobazzi and Ranzato 1997a; Giacobazzi et al. 1998a; Mycroft 1993; Reddy and Kamin 1993; Sekar et al. 1997; Steffen 1989].

A key feature of completeness in abstract interpretation is that this property uniquely depends upon the abstraction function. In fact, it is easily seen that if $\langle A, f^\sharp \rangle$ is complete then the abstract interpretation obtained by using the best correct approximation $\langle A, \alpha \circ f \rangle$, is complete as well, and $f^\sharp$ indeed coincides with $\alpha \circ f$ (cf. [Giacobazzi and Ranzato 1997a]). Thus, given an abstract domain $A$, one can define a complete abstract interpretation $\langle A, f^\sharp \rangle$ if and only if $\langle A, \alpha \circ f \rangle$ is complete and $f^\sharp = \alpha \circ f$. This simple observation makes completeness an abstract domain property, namely a characteristic of the abstract domain. As an example, consider the abstract domain $\mathcal{N}at$ in Figure 1.1 and the concrete function which computes the square of a set (which is the set of all squares) $Square(N) = \{ n^2 \mid n \in N \}$. It is easily seen that $\langle \mathcal{N}at, Square^{\mathcal{N}at} \rangle$ is complete for the function $Square$, where $Square^{\mathcal{N}at}$ is the best correct approximation of $Square$ in $\mathcal{N}at$, which, in turn, is defined as follows:

$$
\begin{array}{lcl}
Square^{\mathcal{N}at}(>0) & = & >0 \\
Square^{\mathcal{N}at}(\geq 0) & = & \geq 0 \\
Square^{\mathcal{N}at}(\mathbb{Z}) & = & \geq 0
\end{array}
$$

It is not hard to check that, by considering any other abstract function, which correctly approximates the concrete one, the new abstract interpretation is not complete anymore. Therefore, in the following, we say that an abstract domain $A$ is complete for a concrete function $f$ if the abstract interpretation$\langle A, \alpha \circ f \rangle$ is complete for $f$, i.e., $\alpha \circ f = \alpha \circ f \circ \alpha$.

## 1.2 Properties by completeness problems

Given an abstract domain $A$ which is not complete for a concrete function $f$, a question naturally arises: can we enrich the domain $A$ in order to obtain a complete domain for $f$? The standard methods for combining domains disregarding the associated functions (like reduced product and disjunctive completion) do not help, in general, to find an answer. What we need is some function-driven technique for transforming abstract domains which ensures us that the resulting domains are complete with respect to a given concrete function. Moreover, when transforming abstract domains, we usually require the new complete domain to be the most abstract one with this property. This ensures that the resulting domain has been obtained by adding the least amount of objects to the original domain. Given a domain $A$, the most abstract domain which contains $A$ and is complete for $f$, when it exists, is called *least complete extension* of $A$ for $f$. Dually, we wonder if we can simplify the domain $A$, in order to get completeness. In other words, we are interested in finding a complete domain which is contained in $A$ and which is the more concrete one with this property. In this case we speak of *greatest complete kernel* of $A$ for $f$. Consider, for instance, the domain $\mathcal{N}eg$ in Figure 1.1 and the concrete function $Square$ defined above. The domain $\mathcal{N}eg$ is not complete for $Square$, e.g., $Square(\{0\}) = \{0\}$, whose best approximation in $\mathcal{N}eg$ is $\leq 0$, while $Square^{\mathcal{N}eg}(\leq 0) = \mathbb{Z}$. The least extension of $\mathcal{N}eg$ for the function $Square$ is obtained by adding to $\mathcal{N}eg$ the objects $\{0\}$ and $\varnothing$. Intuitively, they represent the (greatest) concrete objects on which the function $Square$ is not complete. Similarly, the greatest complete kernel of $\mathcal{N}eg$ is simply the abstract domain $\{\mathbb{Z}\}$, i.e., we need to remove from $\mathcal{N}eg$ both the objects $\leq 0$ and $< 0$ in order to get completeness for the function $Square$.

The idea of this thesis arises from the observation that we can describe the precision degree of an abstract domain by a suitable generalization of the standard notion of completeness (as presented in [Cousot and Cousot 1979b]). Therefore, completeness becomes the language by which we survey the relative precision of abstract domains in computing a given function. In fact, standard completeness can be thought of as an intensional description of all domains whose abstract computations are as precise as the concrete ones. We carry out this basic idea and generalize the completeness notion to generic (possibly typed) functions. In addition, we split the problem into two parts, and consider two separate abstract domains. In the observation domain, which is an abstraction on the range of the concrete function, we only observe the result of an abstract computation. Therefore, the observation domain exactly represents the property we are interested in. A computation domain is used to actually perform the computation, and it is an abstraction of the domain of the concrete function. The idea of considering two different abstract domains allows us to relativize the notion of completeness, and consider abstract domains whose computations are as precise as possible, when the results are observed in some other abstract domain. For instance, given two concrete domains $C$ and $D$ and a function $f : C \longmapsto D$, we can fix a domain of observation $A \in uco(D)$, which intuitively represents the property we are interested in, and look for the most abstract domain $B \in uco(C)$ such that any abstract computation on the domain $B$ is as precise as the corresponding concrete computation, when the results are observed in the domain $A$. Formally, if $\alpha_A$ and $\alpha_B$ are the abstraction functions associated, respectively, with $A$ and $B$, the domain $B$ has to be a solution of the following completeness problem:

$$\alpha_A \circ f = \alpha_A \circ f \circ \alpha_B.$$

Moreover, by requiring $B$ to be the most abstract domain with this property, we guarantee that $B$ is obtained by adding the least amount of objects, yet providing the desired level of precision. Dually, we can fix a domain of computation $B \in uco(C)$ and look for the most concrete observation domain $A \in uco(C)$ which satisfies the above completeness problem. In this case we intuitively look for the most concrete abstract domain in which we can observe only complete computations. In other words, the solution to this problem tells us how much we can observe in an abstract computation in order to be sure that no approximation errors are introduced. Therefore, completeness problems represent a compact and elegant language to describe the precision degree of the abstract domain we are interested in.

## 1.3 From completeness problems to abstract domain equations

Once we have expressed the desired precision level in terms of a completeness problem, we need a method to solve such a problem. The idea is to associate with each completeness problem a system of (possibly recursive) abstract domain equations in such a way that the solutions to the abstract domain equations satisfy the completeness problem. Intuitively, an abstract domain equation is some equation of kind $X = F(X)$, where $X$ is the abstract domain we look for and $F$ is a monotone function transforming abstract domains. Abstract domain equations can then be solved iteratively by looking for the least fixpoint, since $F$ is a monotone function. The way to move from completeness problems to domain equations is therefore the main point of this construction.

For each completeness problem, we provide a constructive method to build either the observation or the computation domain, given the other one. More precisely, given a continuous semantic function $f : C \longmapsto D$, we prove that the relation between any pair of observation and computation domains which satisfy the completeness problem $\alpha_A \circ f = \alpha_A \circ f \circ \alpha_B$ is precisely an adjunction, and study in detail the property of such a relation. We show that the completeness problem $\alpha_A \circ f = \alpha_A \circ f \circ \alpha_B$ holds if and only if $A$ is more abstract than a certain domain $L_f(B)$ depending on $B$. Intuitively, $L_f(B)$ represents the most concrete domain where all the abstract computations computed in $B$ are precise. Dually, this happens if and only if $B$ is more concrete than a certain domain $R_f(A)$ depending on $A$. $R_f(A)$ summarizes the least amount of information which is needed in the computation domain in order to get completeness.

Thus, the mappings $R_f : uco(D) \longmapsto uco(C)$ and $L_f : uco(C) \longmapsto uco(D)$ form an adjunction. By exploiting these results, we are able to characterize:

- the least complete extension of $B$ with respect to the observation domain $A$, i.e., the most abstract complete domain which includes $B$ and satisfies the above completeness equation;
- the greatest complete kernel of $A$ with respect to the computation domain $B$, i.e., the most concrete domain which is included in $A$ and satisfies the above completeness problem.

As a remarkable point of our result, we provide explicit constructive characterizations for both operators $R_f$ and $L_f$, and therefore for least complete extensions and greatest complete kernels. Finally, we address the problem of representing the objects in complete abstract domains. To this end, we supply a natural way to represent the abstract objects, by resorting to suitable fragments of (propositional) classic, intuitionistic and linear logic.

## 1.4 Plan of the thesis

After some preliminary definitions, we present in Chapter 3 the notion of completeness and its basic properties. We show how to associate with each completeness problem an abstract domain equation and we provide a constructive method to set up least complete extensions and greatest complete kernels.

In Chapter 4, our results are applied to Wadler and Hughes' projection-based analysis for functional programs [1987]. The key problem in projection-based backward analysis consists in finding a certain least projection, called *least backward abstraction* [Davis 1992; Davis 1993]. It turns out that dualizing the basic projection equation, one gets exactly a completeness problem, and, notably, least backward abstractions correspond to least complete extensions. We show how to exploit our results on least complete extensions to considerably improve on Davis' result, and, more importantly, we supply a constructive method to set up such abstractions.

In Chapter 5, we investigate the relationship between completeness and the concept of *quotient* of an abstract interpretation, recently introduced by Cortesi et al. [1998] for comparing the precision of abstract interpretations in computing a given property. We show that, in general, quotients do not always exist. However, we observe that quotients, when they exist, turn out to be certain least complete extensions, which naturally formalize the intuition behind the notion of quotient. Thus, a simple and natural generalization of the notion of quotient is proposed, which retains the advantage of being always well-defined, under the hypothesis of continuity of the semantic functions.

In Chapter 6, we extend our results for dealing with binary concrete functions. We introduce some typical completeness problems arising when considering functions with multiple arguments. Moreover, we show how propositional linear logic can be used to characterize objects in complete abstract domains in a natural and elegant way, which allows us to inherit the basic features of linear logic and to simplify most of our results on abstract domain equations.

The application of our theory to "optimal" semantics for logic programs is in Chapter 7. We present a simple logic program semantics for observing computed answer substitutions and, by exploiting our results, we propose a method to construct optimal semantics, i.e., semantics which are neither too concrete nor too abstract for characterizing a given property of interest. Such semantics are obtained by solving suitable completeness problems in a concrete domain of substitutions.

Then, in Chapter 8, we apply our framework to study condensing domains in analyses of (constraint) logic programs, i.e., domains where goal-independent analyses have the same precision as goal-driven ones, and we show that condensing domains can be characterized as solutions of completeness problems.

We define the notion of Heyting completion in Chapter 9. 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. We show that Heyting completion can be characterized by a completeness problem and study the main algebraic properties of Heyting completion, when combined with other well-known operations for transforming domains, like reduced product and disjunctive completion. Finally, we characterize the family of domains closed under Heyting completion.

In Chapter 10 we apply our results to groundness analysis of logic programs. We study the relations among various abstract domains proposed in the literature for groundness analysis of logic programs. We show that the relations between these domains are exactly captured by our notion of Heyting completion. This allows us to easily compare the relative precision of the different abstract domains and, notably, to fully characterize them in terms of the completeness problems they solve. Finally, we propose a new domain for groundness analysis, which is automatically obtained by applying the Heyting completion operator, and which has the advantage of being independent from the specific program.

In Chapter 11, we consider the problem of inferring types in logic programs by abstract interpretation. Given a domain for (basic) type analysis, we show that Heyting completion precisely captures the notion of directionality among types and, more generally, that this construction surfaces directionality in any property closed under instantiation.

# Chapter 2

# Notation and basic notions

This chapter is intended to fix the notation of the basic mathematical notions used throughout the thesis. We assume familiarity with lattice theory (e.g., see [Birkhoff 1967; Gierz et al. 1980; Davey and Priestley 1990]), closure operators (see [Morgado 1960; Birkhoff 1967; Cousot 1978]), and abstract interpretation (see [Cousot and Cousot 1977, 1979b]).

## 2.1 Algebraic notions

### 2.1.1 Sets and lattices

In dealing with sets we use these standard notations: membership $\in$, the empty set $\varnothing$, inclusion $\subseteq$, proper inclusion $\subset$, and set-theoretic difference $\smallsetminus$. If $A$ and $B$ are sets, $\overline{A}$ is set-theoretic complement, $A \subseteq_f B$ denotes that $A$ is a finite subset of $B$, $\wp(A)$ is the power set of $A$ and the set of all finite subset of $A$ is denoted by $\wp_f(A)$.

If $A$ is a partially ordered set (**poset**), we usually denote by $\leq_A$ the corresponding partial order. With any poset $\langle A, \leq_A \rangle$ we have its *dual* $\langle A, \geq_A \rangle$ denoted by $A^{\text{op}}$. The set of *maximal elements* of $A$ is $max(A) = \{\, a \in A \mid \forall b \in A.\, a \leq_A b \Rightarrow a = b \,\}$. If $A$ is a poset with least element (bottom) $\bot_A$, an element $a \in A$ is an *atom* if $a$ covers $\bot_A$, i.e., $a \neq \bot_A$ and for all $x \in A$, $\bot <_A x \leq_A a$ implies $x = a$. $At(A)$ denotes the set of atoms in $A$.

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 $\{\, I \subseteq A \mid I = \downarrow I \,\}$. $I$ is *Scott-closed* if $I = \downarrow I$ and for any chain $C \subseteq I$ there exists the least upper bound $\vee C$ and $\vee C \in I$. We denote by $\wp^{\mathcal{H}}(A)$ the *Hoare powerdomain*, i.e., the collection of all subsets of $A$ which are Scott-closed.

A poset $A$ is called a **cpo** (complete, partially ordered set) if $A$ admits a least element $\bot_A$ and each chain has a least upper bound.

A poset $A$ is called a **lattice** if least upper bound $\vee_A$ (l.u.b., join) and greatest lower bound $\wedge_A$ (g.l.b., meet) exist for all finite (nonempty) subsets of $A$. A lattice $A$ is *distributive* if $(a \vee_A b) \wedge_A c = (a \wedge_A c) \vee_A (b \wedge_A c)$, for all $a, b, c \in A$. Moreover, $a$ is a *complement* of $b$ if $a \vee_A b = \top_A$ and $a \wedge_A b = \bot_A$. A lattice is complemented if every element has a complement. A **complete** lattice is any lattice for which l.u.b. and g.l.b. exist for any subset of $A$. A complete lattice $A$ with partial ordering $\leq_A$, l.u.b. $\vee_A$, g.l.b. $\wedge_A$, greatest element $\top_A$ and least element $\bot_A$, is denoted by $\langle A, \leq_A, \vee_A, \wedge_A, \top_A, \bot_A \rangle$. As usual, we follow the standard conventions that $\wedge\varnothing = \top_A$ and $\vee\varnothing = \bot_A$. Both $\wp(A)$ and $\wp^{\downarrow}(A)$ are complete lattices with respect to set-theoretic inclusion, where the join is set union and the meet is set intersection. A complete lattice $A$ is *atomic* if every element $x \in A$ different from $\bot_A$ is the l.u.b. of the atoms less than or equal to $x$, i.e., $x = \bigvee_A \{ a \in At(A) \mid a \leq_A x \}$. *Dual-atoms* and *dual-atomicity* are defined by duality.

### 2.1.2 Functions

We write $f : A \longmapsto B$ to mean that $f$ is a total function from $A$ to $B$. In the following we sometimes use Church's lambda notation for functions, so that a function $f$ will be denoted by $\lambda x.f(x)$. By $g \circ f$ we denote the composition $\lambda x.g(f(x))$. If $B$ is a poset and $f, g : A \longmapsto B$, we write $f \sqsubseteq g$ if for all $x \in A$, $f(x) \leq_B g(x)$. If $C \subseteq A$ then $f(C) = \{ f(x) \mid x \in C \}$ and $f^{-1} : B \longmapsto \wp(A)$ is the function given by $f^{-1}(b) = \{ a \in A \mid f(a) = b \}$.

Let $A$ and $B$ be complete lattices. A function $f : A \longmapsto B$ is (completely) **additive** if for any $C \subseteq A$, $f(\vee_A C) = \vee_B f(C)$. $f$ is **continuous** when the above statement on additivity holds for chains $C$ only. $A \overset{m}{\longmapsto} B$, $A \overset{c}{\longmapsto} B$, and $A \overset{a}{\longmapsto} B$ denote, respectively, the set of all monotone, continuous, and additive functions from $A$ to $B$. $A \overset{co-a}{\longmapsto} B$ and $A \overset{co-c}{\longmapsto} B$ denote the set of all *co-additive* and *co-continuous* functions, which are defined by duality.

*Ord* denotes the proper class of *ordinals*, where $\omega \in Ord$ is the first infinite ordinal. For a complete lattice $C$, given $f : C \longmapsto C$, for any $\alpha \in Ord$, the *$\alpha$-th (upper) power* $f^\alpha : C \longmapsto C$ of $f$ is inductively defined as follows, for $x \in C$:

$$\begin{array}{lcl} f^0(x) & = & \bot_C \\ f^{\alpha+1}(x) & = & f(f^\alpha(x)) \\ f^\beta(x) & = & \bigvee_{\gamma<\beta} f^\gamma(x) \ \text{ if } \beta \text{ is a limit ordinal} \end{array}$$

A *fixpoint* for a function $f : C \longmapsto C$ is any element $x \in C$ such that $f(x) = x$. By Knaster-Tarski Fixpoint Theorem ([Tarski 1955]), the set of fixpoints of any monotone function $f : C \longmapsto C$ on a complete lattice $C$ is a (nonempty) complete lattice. Therefore, $f$ has both the least fixpoint $\mathit{lfp}(f)$ and the greatest fixpoint

$gfp(f)$. When $f$ is continuous, it holds that $lfp(f) = f^{\omega}(\bot_C)$. A constructive characterization of the lattice of all fixpoints for monotone functions was found by Cousot and Cousot [1979a].

## 2.2 Abstract interpretation and Galois connections

The standard Cousot and Cousot [1977; 1979b] theory of abstract interpretation is based on the notion of Galois connection. If $C$ and $A$ are posets and $\alpha : C \stackrel{m}{\longmapsto} A$, $\gamma : A \stackrel{m}{\longmapsto} C$ are such that:

- $\forall c \in C \ c \leq_C \gamma(\alpha(c))$
- $\forall a \in A \ \alpha(\gamma(a)) \leq_A a$

then we call the quadruple $\langle C, \alpha, A, \gamma \rangle$ a *Galois connection* (or, equivalently, an *adjunction*) between $C$ and $A$. In the setting of abstract interpretation, $C$ and $A$ are called, respectively, the *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 Galois connection $\langle C, \alpha, A, \gamma \rangle$ can be lifted to a G.i. by identifying (in an equivalence class) those objects in $A$ having the same image (meaning) in $C$. Different abstract objects denoting the same concrete ones are collapsed into a single object, and the result is an abstract domain containing no redundant objects. This process is known as *reduction* of the abstract domain. It ensures that any abstract object is the image of some concrete object, or equivalently, that the abstraction function $\alpha$ is surjective. A reduction of a Galois connection $\langle C, \alpha, A, \gamma \rangle$ 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 determine 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 $\rho(C) = A$).

### 2.2.1 From Galois insertions to closure operators

Cousot and Cousot [1979b] observed that an often more practical way of reasoning about abstract domains in abstract interpretation is by closure operators. This allows us to be independent from specific representations of abstract domain's objects.

A function $\rho : L \longmapsto L$ is an *(upper) closure operator* on a complete lattice $L$ if it is:

- monotone;
- idempotent: $\forall x \in L.\ \rho(\rho(x)) = \rho(x)$;
- and extensive: $\forall x \in L.\ x \leq_L \rho(x)$.

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 $L$ is a complete lattice $\langle L, \leq_L, \vee_L, \wedge_L, \top_L, \bot_L \rangle$ playing the rôle of the concrete domain. The next result from Cousot and Cousot [1979b] will be essential to move from one representation of domains into another.

**Theorem 2.2.1 ([Cousot and Cousot 1979b])** *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 [1979b, Section 8]), i.e., the complete lattice of all possible abstract domains (modulo isomorphic representation of their objects) of the concrete domain $L$. We often abuse the 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.

Any closure operator $\rho : L \longmapsto L$ is identified by the set of its fixpoints, which is its image $\rho(L)$. Recall that $\rho(L)$ is a *Moore family* of $L$, i.e., a complete meet-sublattice of $L$ (i.e., $\top_L \in \rho(L)$ and for any 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 $\vee_\rho$ in $\rho(L)$, which is precisely given by $\rho \circ \vee_L$, 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 $\curlywedge(X) = \{ \bigwedge_L I \mid I \subseteq X \}$ the *Moore closure* of $X$, i.e., the least Moore family of $L$ containing $X$. Moreover, we say that $X$ generates $A$, or equivalently, that $X$ is a set of *generators* for $A$, when $\curlywedge(X) = A$. Given a Moore family $A$, we denote by $\rho_A$ the corresponding closure operator.

### 2.2.2 The lattice of abstract interpretations

The lattice of abstract interpretations inherits all the relevant algebraic properties of the lattice of all closure operations [Morgado 1960; Ward 1942; Cousot 1978]. Ward [1942, Theorem 4.2] proved that if $L$ is a complete lattice, then $\langle uco(L), \sqsubseteq, \sqcap, \sqcup, \lambda x.\top_L, \lambda x.x\rangle$ 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 that:

- $\rho \sqsubseteq \eta$ if and only if $\forall x \in L.\ \rho(x) \leq_L \eta(x)$ if and only if $\eta(L) \subseteq \rho(L)$;
- $(\sqcap_{i\in I}\rho_i)(x) = \bigwedge_{i\in I} \rho_i(x)$;
- $(\sqcup_{i\in I}\rho_i)(x) = x \iff \forall i \in I.\ \rho_i(x) = x$;
- $\lambda x.\top_L$ is the top element, whereas $\lambda x.x$ is the bottom element.

Moreover, for any $\rho \in uco(C)$ and $Y \subseteq C$, the following helpful properties hold:

$$\rho(\bigvee_{y\in Y} \rho(y)) = \rho(\bigvee Y) \tag{2.2.1}$$

$$\rho(\bigwedge_{y\in Y} \rho(y)) = \bigwedge_{y\in Y} \rho(y). \tag{2.2.2}$$

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 $\varphi_x = \{\, \top_L, x \,\}$, for any $x \in L \setminus \{\, \top_L \,\}$, i.e., if $y \leq x$ then $\varphi_x(y) = x$, otherwise $\varphi_x(y) = \top_L$. We call any such $\varphi_x$ an *atomic domain*. 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 $\lambda x.\top_L$ is the top domain $\{\, \top_L \,\}$, 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 l.u.b. and g.l.b. on $uco(L)$ have therefore the following meaning as operators on domains. Suppose $\{\, A_i \,\}_{i\in I} \subseteq uco(L)$:

- $\sqcup_{i\in I}A_i$ is the *least common abstraction* of all $A_i$'s, i.e., $\sqcup_{i\in I}A_i$ is the most concrete among the domains which are abstractions of all the $A_i$'s;
- $\sqcap_{i\in I}A_i$ 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 $A_i$.

*Projections* (also called lower closure operators) are defined by duality of upper closer operators (and therefore are monotone, idempotent and *reductive* functions). Given a complete lattice $L$, the lattice of all projections on $L$ is denoted by $\langle prj(L), \sqsubseteq \rangle$, and it is dually isomorphic to $\langle uco(L), \sqsubseteq \rangle$.

## 2.3 Logic in brief

### 2.3.1 Boolean algebras and classic logic

A *Boolean algebra* is a distributive, complemented lattice. A complete Boolean algebra (**cBa**) is a Boolean algebra whose underlying set is a complete lattice. Examples of cBa are $\wp(P)$, for any set $P$.

Complete Boolean algebras turn out to be the models of (propositional) *classic logic*. The basic connectives of propositional classic logic are conjunction, disjunction and (classic) negation. Classic negation has the fundamental property of being involutive, that is to say, that the double negation of a formula is the formula itself, i.e., $\neg\neg\phi = \phi$, for any formula $\phi$. Moreover, given any two formulas $\phi$ and $\psi$, a notion of *classic implication* $\phi \Rightarrow \psi$ can be defined as the disjunction of $\psi$ and the negation of $\phi$, i.e., $\phi \Rightarrow \psi \stackrel{\text{def}}{=} \neg\phi \vee \psi$.

The conjunction, disjunction, and (classic) negation logical connectives have their algebraic counterparts as meet, join and complement operations in cBa. The algebraic notion corresponding to (classic) implication is straightforwardly obtained by exploiting the previous definition.

### 2.3.2 Heyting algebras and intuitionistic logic

The intuitive idea of *Heyting algebras* is to consider only lattices where, for any $a, b \in L$, there exists some object $c \in L$, such that $a \wedge c \leq b$ and, in addition, $c$ is the greatest element with this property. These are well-known lattices called Heyting algebras, which are precisely the models of *intuitionistic logic* [Birkhoff 1967; van Dalen 1994]. In this case, the unique object $c$ is called *intuitionistic implication* and denoted by $a \rightarrow b$, and the property $a \wedge a \rightarrow b \leq b$ is called *modus ponens* law. Therefore, intuitionistic logic is the weakest framework where implicational objects do exist. Intuitively, given two formulas $\varphi$ and $\psi$, we can think of $\varphi \rightarrow \psi$ 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

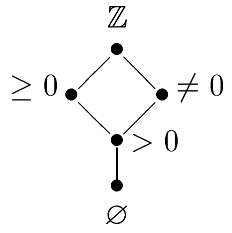



Figure 2.1: An example of (non-Boolean) Heyting algebra

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 able to prove neither $\varphi$ nor its contrary. When this latter equality holds, the whole system boils down to classical logic.

Clearly, Heyting algebras include, among others, Boolean algebras. As an example of non-Boolean Heyting algebras, consider the lattice *Sign* depicted in Figure 2.1, which is an abstraction of $\wp(\mathbb{Z})$, whose points have the obvious meaning. Note that $\geq 0 \to > 0$ cannot be described in terms of disjunction and complementation. In particular, the complement $\neg \geq 0$, namely an object $d$ such that $d \wedge \geq 0 = \varnothing$ and $d \vee \geq 0 = \mathbb{Z}$, does not exist. However, being a Heyting algebra, $\geq 0 \to > 0$ still exists, and it is $\neq 0$.

More formally, given a lattice $L$ and $a, b \in L$, the *pseudocomplement* of $a$ *rela-*

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

Figure 2.2: Axioms for Heyting algebras

*tively* to $b$, if it exists, is the unique element $a \to b$ in $L$ such that, for any $x \in L$, it holds that:

$$a \wedge_L x \leq_L b \iff x \leq_L a \to b.$$

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 $a \wedge_L (\bigvee_L \{\, c \mid a \wedge_L c \leq_L b \,\}) \leq_L b$. A lattice $L$ is called a *Heyting algebra*[1] when the pseudocomplement of $a$ relatively to $b$, $a \to b$, exists for every $a, b \in L$. A complete Heyting algebra (**cHa**) is a Heyting algebra whose underlying set is a complete lattice. Examples of cHa are $\wp^{\downarrow}(P)$, for a poset $P$, and any cBa. Moreover, cHa enjoys the following useful property (see [Birkhoff 1967]).

**Proposition 2.3.1** *A complete lattice $L$ is a cHa if and only if the following complete $\wedge$-distributivity law holds, for each $X \subseteq L$ and $y \in L$:*

$$y \wedge_L (\bigvee_L X) = \bigvee_L \{\, y \wedge_L x \mid x \in X \,\}.$$

Meet, join and relative pseudocomplement of Heyting algebras precisely correspond to conjunction, disjunction and intuitionistic implication of intuitionistic logic. In the following, in order to keep distinct classic and intuitionistic implication, we denote by $\Rightarrow$ the classic one and by $\to$ the intuitionistic one. From a logical point of view, a cHa $L$ is a set equipped with three operations $\wedge, \vee, \to$ and constants $\bot, \top$ which satisfies, for every $a, b, c \in L$, the equalities in Figure 2.2.

The following proposition recalls some key properties of cHa's (see [van Dalen 1994; Birkhoff 1967]).

[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.

**Proposition 2.3.2** *Let $L$ be a cHa, $\{ b_i \}_{i \in I} \subseteq L$, and $a, b, c \in L$.*

$$(a \wedge b) \to c = a \to (b \to c) \tag{2.3.1}$$

$$a \to \bigwedge_{i \in I} b_i = \bigwedge_{i \in I} (a \to b_i) \tag{2.3.2}$$

$$(\bigvee_{i \in I} b_i) \to a = \bigwedge_{i \in I} (b_i \to a) \tag{2.3.3}$$

### 2.3.3 Quantales and linear logic.

*Quantales* originated as algebraic foundations of the so-called *quantum logic.* They have been successively considered for the lattice-theoretic semantics of Girard's *linear logic* (see [Rosenthal 1990] for an exhaustive treatment of quantales). Informally, quantales can be thought of as a generalization of cHa's, where the modus ponens law $a \wedge a \to b \leq b$ holds for a generic binary operation $\otimes$. Given a binary operation $\otimes$ and two objects $a$ and $b$, we look for the most abstract object $c$ which satisfies $a \otimes c \leq b$ and, dually, for the most abstract object $d$ which satisfies $d \otimes a \leq b$.

More formally, a quantale is an algebra $\langle C, \otimes \rangle$ such that:

- $C$ is a complete lattice;
- $\otimes : C \times C \longmapsto C$ is a binary, associative operation, i.e., $(a \otimes b) \otimes c = a \otimes (b \otimes c)$, for any $a, b, c \in C$;
- $a \otimes (\bigvee_{i \in I} b_i) = \bigvee_{i \in I} (a \otimes b_i)$ and $(\bigvee_{i \in I} b_i) \otimes a = \bigvee_{i \in I} (b_i \otimes a)$, for any $\{ b_i \}_{i \in I} \subseteq C$.

In other words, a quantale is a complete lattice endowed with an associative "product" $\otimes$ which distributes over arbitrary l.u.b.'s on both sides. Quantales are a generalization of cHa, and any cHa is a quantale, with $\otimes = \wedge$. Given a set $A$ and an associative operation $\cdot : A \times A \longmapsto A$, an example of quantale is $\langle \wp(A), \otimes \rangle$ where $X \otimes Y = \bigcup \{ x \cdot y \mid x \in X, y \in Y \}$.

**Example 2.3.3** As an example of quantale, consider the complete lattice $\wp(list(\mathbb{N}))$ where $list(\mathbb{N})$ is the set of all finite lists of natural numbers. Let *append* be the operation of list concatenation which, given two lists $a, b \in list(\mathbb{N})$, returns the list obtained by appending $b$ to $a$. Consider the "collecting" version of list concatenation, $\otimes : \wp(list(\mathbb{N})) \times \wp(list(\mathbb{N})) \longmapsto \wp(list(\mathbb{N}))$, which is defined as follows, for any $L, M \in \wp(list(\mathbb{N}))$:

$$L \otimes M \stackrel{\text{def}}{=} \{ append(l, m) \mid l \in L,\ m \in M \}$$

By definition, the operation $\otimes$ is argumentwise additive and associative. Therefore, $\langle \wp(list(\mathbb{N})), \otimes \rangle$ is a quantale. ∎

The fundamental property of quantales is that, for any $a \in C$, both functions $\lambda x.a \otimes x$ and $\lambda x.x \otimes a$ have *right adjoints* denoted, respectively, by $\lambda x.a \multimap x$ and $\lambda x.x \multimapinv a$. This is equivalent to saying that, for all $a, b, c \in C$, the following two properties hold.

$$a \otimes b \leq c \iff b \leq a \multimap c$$

$$a \otimes b \leq c \iff a \leq c \multimapinv b$$

In other words, for all $a \in C$, $\lambda x.a \otimes x$ and $\lambda x.x \otimes a$ are complete join morphism. Hence they have unique right adjoints $\lambda x.a \multimap x$ and $\lambda x.x \multimapinv a$ which form Galois connections. Two functions $\multimap, \multimapinv: C \times C \longmapsto C$ can therefore be defined as follows:

$$a \multimap c \stackrel{\text{def}}{=} \bigvee \{ b \in C \mid a \otimes b \leq c \}$$

$$c \multimapinv b \stackrel{\text{def}}{=} \bigvee \{ a \in C \mid a \otimes b \leq c \}$$

From a logical point of view, it is well known that quantales turn out to be models of noncommutative linear logic [Yetter 1990; Rosenthal 1990], where the operations $\multimap$ and $\multimapinv$ are called *right* and *left linear implications*. The next proposition summarizes the main properties of linear implications (see [Rosenthal 1990]). It is worth noting that the first two properties can be thought of as a generalization of the modus ponens law in cHa.

**Proposition 2.3.4** *Let $\langle C, \otimes \rangle$ be a quantale, $\{ x_i \}_{i \in I} \subseteq C$ and $a, b, c \in C$.*

| | | | |
|---|---|---|---|
| $(i)$. | $a \otimes (a \multimap c) \leq c$ | $(ii)$. | $(c \multimapinv b) \otimes b \leq c$ |
| $(iii)$. | $a \multimap (\bigwedge_{i \in I} x_i) = \bigwedge_{i \in I} (a \multimap x_i)$ | $(iv)$. | $(\bigwedge_{i \in I} x_i) \multimapinv b = \bigwedge_{i \in I} (x_i \multimapinv b)$ |
| $(v)$. | $(\bigvee_{i \in I} x_i) \multimap c = \bigwedge_{i \in I} (x_i \multimap c)$ | $(vi)$. | $c \multimapinv (\bigvee_{i \in I} x_i) = \bigwedge_{i \in I} (c \multimapinv x_i)$ |
| $(vii)$. | $a \multimap (b \multimap c) = (b \otimes a) \multimap c$ | $(viii)$. | $(c \multimapinv b) \multimapinv a = c \multimapinv (a \otimes b)$ |
| $(ix)$. | $a \multimap (c \multimapinv b) = (a \multimap c) \multimapinv b$ | | |

A quantale $\langle C, \otimes \rangle$ is called *commutative* when $\otimes$ is commutative, and this is equivalent to requiring that, for all $a, b \in C$, $a \multimap b = b \multimapinv a$. Also, a commutative quantale $\langle C, \otimes \rangle$ is called *unital* if there exists an object $1 \in C$ such that $1 \otimes a = a = a \otimes 1$, for all $a \in C$. Examples of unital, commutative quantales are any cHa, with $\otimes = \wedge$ and unity $\top$. The next proposition shows some additional properties of unital, commutative quantales (see [Rosenthal 1990]).

**Proposition 2.3.5** *Let $\langle C, \otimes \rangle$ be a unital commutative quantale with unity* $1$*. The following additional properties hold, for any* $a, b, c \in C$*.*

| | | | |
|---|---|---|---|
| $(x)$. | $a \multimap (b \multimap c) = b \multimap (a \multimap c)$ | $(xi)$. | $1 \multimap a = a$ |
| $(xii)$. | $c \leq_C (c \multimap a) \multimap a$ | $(xiii)$. | $((c \multimap a) \multimap a) \multimap a = c \multimap a$ |

# Chapter 3

# Completeness and equations

Abstract

The idea of setting up "precise" abstract domains relies mainly in two points: how to describe the desired precision level of precision and how to construct the new domains. In this chapter we show that the precision of abstract domains can be described by means of completeness problems. As a main result, we provide a constructive method for associating with each completeness problem an abstract domain equation, which enables us to construct the complete domains.

The results presented in this chapter are a joint work with R. Giacobazzi and F. Ranzato and appeared in [Giacobazzi et al. 1998a].

A typical framework for analyses of programming languages (e.g., in [Cousot and Cousot 1979b]) is given by specifying, for any program $P$, a concrete semantic function $[\![P]\!] : \mathit{Input} \longmapsto \mathit{Output}$, where the concrete semantics domains $\mathit{Input}$ and $\mathit{Output}$ represent the set of all possible inputs and outputs for the program $P$. Note that, this definition is general enough to include most semantics of programming languages, which differ on how the semantic function $[\![P]\!]$ is actually computed. Moreover, we consider here the most general case of analyses where two different abstractions, respectively on the input and output domains, can be specified. Therefore, an abstract interpretation is given by two abstractions $A$ and $B$, respectively, on the input and output domains, and by an abstract semantic function $[\![P]\!]^{A,B} : A \longmapsto B$, providing the abstract semantics to correctly approximate the concrete one, i.e., for any possible input $i \in \mathit{Input}$, it holds

$$\alpha_B([\![P]\!](i)) \leq_B [\![P]\!]^{A,B}(\alpha_A(i)).$$

In this scenario, an abstract interpretation is complete when equality holds, i.e.,

$$\alpha_B([\![P]\!](i)) = [\![P]\!]^{A,B}(\alpha_A(i)).$$

It is then clear that a key problem consists in devising systematic and constructive methodologies for transforming abstract domains in such a way that completeness is achieved and the resulting complete abstract domains are as close as possible to the initial (noncomplete) ones.

## 3.1 Completeness by closures

Let $f : C \longmapsto D$ be any concrete function occurring in some complex semantic specification, where $C$ and $D$ are complete lattices playing the rôle of concrete semantic domains. Let an abstract interpretation $\langle A, B, f^\sharp\rangle$ of $\langle C, D, f\rangle$ be specified by the G.i.'s $(\alpha_{C,A}, C, A, \gamma_{A,C})$ and $(\alpha_{D,B}, D, B, \gamma_{B,D})$, and by an abstract function $f^\sharp : A \longmapsto B$. The concept of *correctness* is standard and well-known (see [Cousot and Cousot 1979b]). $\langle A, B, f^\sharp\rangle$ is a *correct* abstract interpretation – or $f^\sharp$ is a correct approximation of $f$ relatively to $A$ and $B$ – when $\alpha_{D,B} \circ f \sqsubseteq f^\sharp \circ \alpha_{C,A}$. This is equivalent to saying that $\alpha_{D,B} \circ f \circ \gamma_{A,C} \sqsubseteq f^\sharp$. Thus, $f^{A,B} \stackrel{\text{def}}{=} \alpha_{D,B} \circ f \circ \gamma_{A,C} : A \longmapsto B$ is called the best correct approximation of $f$ relatively to the abstract domains $A$ and $B$. On the other hand, $\langle A, B, f^\sharp\rangle$ is called *complete* when equality holds, i.e.,

$$\alpha_{D,B} \circ f = f^\sharp \circ \alpha_{C,A}$$

In this case, $f^\sharp = f^\sharp \circ \alpha_{C,A} \circ \gamma_{A,C} = \alpha_{D,B} \circ f \circ \gamma_{A,C} = f^{A,B}$, i.e., whenever $f^\sharp$ is a complete approximation of $f$, then $f^\sharp$ is even the best correct approximation of $f$. This means that, given two abstract domains $A$ and $B$, there exists $f^\sharp$ such that $\langle A, B, f^\sharp\rangle$ is complete if and only if $\langle A, B, f^{A,B}\rangle$ is complete. Once a concrete function $f$ is fixed, it turns out that $f^{A,B}$ only depends on $A$ and $B$, and therefore we get that *completeness is an abstract domain property.* Thus, given $A$ and $B$, we refer to completeness of $A$ and $B$ in order to refer to completeness of the whole abstract interpretation $\langle A, B, f^{A,B}\rangle$.

The notion of completeness can be also formalized by resorting to the Cousot and Cousot *closure operator* approach to abstract interpretation [Cousot 1978; Cousot and Cousot 1979b]. This allows us to be independent from specific representations of abstract domain's objects. By using closure operators, if $\rho = \gamma_{A,C} \circ \alpha_{C,A} \in uco(C)$ and $\eta = \gamma_{B,D} \circ \alpha_{D,B} \in uco(D)$ are the closures associated, respectively, with $A$ and $B$, one can show that $A$ and $B$ are complete if and only if $\eta \circ f = \eta \circ f \circ \rho$. This justifies the following general definition of completeness.

**Definition 3.1.1** Let $C$ and $D$ be complete lattices, $f : C \stackrel{m}{\longmapsto} D$, $\rho \in uco(C)$, and $\eta \in uco(D)$. Then, the pair $\langle \rho, \eta\rangle$ is **complete** for $f$ if $\eta \circ f = \eta \circ f \circ \rho$. Also, if $F \subseteq C \stackrel{m}{\longmapsto} D$ then $\langle \rho, \eta\rangle$ is complete for $F$ whenever $\forall f \in F.\ \eta \circ f = \eta \circ f \circ \rho$. ■

In the following, we shall refer to $\rho$ (the abstraction on the domain of the function $f$) as the computation domain and to $\eta$ as the observation domain. Let us first see some basic examples of complete abstract interpretations which directly follow from the definition.

- $\langle \lambda x.x, \eta\rangle$ is complete for any $\eta$ and any function $f$;
- $\langle \rho, \lambda x.\top_D\rangle$ is complete for any $\rho$ and any function $f$;
- any $\langle \rho, \eta\rangle$ is complete for the constant function $\lambda x.d$, for any $d \in D$.

The first point states that, when the computation domain coincides with the whole concrete domain, we always obtain a complete abstract interpretation, whatever property we observe. Intuitively, this happens since we consider the set of all possible inputs for the function $f$, and therefore, any computation is precise, independently from the abstract domain where the results are observed. Also, when the observation domain is $\lambda x.\top_D$ (that is to say that we do not observe any property) again we get completeness. Note that the set of possible outputs is the singleton $\{\top_D\}$, and thus all abstract computations are precise simply because we are not able to distinguish between any two different outputs.

It is worth remarking that our definition encompasses the case where $f : C \longmapsto C$ and one is interested in two different abstractions of input and output, i.e., $\rho, \eta \in uco(C)$ with $\rho \neq \eta$. Whenever $f : C \longmapsto C$ and $\rho = \eta$, the above definition of completeness boils down to the equation $\rho \circ f = \rho \circ f \circ \rho$ considered in [Cousot and Cousot 1977; Giacobazzi and Ranzato 1997a]. The next proposition relates our definition of completeness with the standard one, given in terms of G.i.'s.

**Proposition 3.1.2** *Let $C$ and $D$ be complete lattices, and $f : C \stackrel{m}{\longmapsto} D$. Let $(\alpha_{C,A}, C, A, \gamma_{A,C})$ and $(\alpha_{D,B}, D, B, \gamma_{B,D})$ be G.i.'s, and $f^\sharp : A \stackrel{m}{\longmapsto} B$ an abstract function. $\langle A, B, f^\sharp\rangle$ is a complete abstract interpretation for $f$ (i.e., $\alpha_{D,B} \circ f = f^\sharp \circ \alpha_{C,A}$) if and only if $f^\sharp = \alpha_{D,B} \circ f \circ \gamma_{A,C}$ and $\eta \circ f = \eta \circ f \circ \rho$, where $\rho = \gamma_{A,C} \circ \alpha_{C,A} \in uco(C)$ and $\eta = \gamma_{B,D} \circ \alpha_{D,B} \in uco(D)$.*

**Proof.** Assume $\alpha_{D,B} \circ f = f^\sharp \circ \alpha_{C,A}$. From $f^\sharp = f^\sharp \circ \alpha_{C,A} \circ \gamma_{A,C} = \alpha_{D,B} \circ f \circ \gamma_{A,C}$ we get that $f^\sharp$ is the best correct approximation $f^{A,B}$. Therefore $\alpha_{D,B} \circ f = f^\sharp \circ \alpha_{C,A} = \alpha_{D,B} \circ f \circ \gamma_{A,C} \circ \alpha_{C,A}$ and this is equivalent to $\gamma_{B,D} \circ \alpha_{D,B} \circ f = \gamma_{B,D} \circ \alpha_{D,B} \circ f \circ \gamma_{A,C} \circ \alpha_{C,A}$. The converse is straightforward. ∎

The previous result shows that, even in this general context where typed functions and different abstract domains are allowed, completeness is still an abstract domain property, which is independent from the abstract function. This gives rise to the question whether abstract domains can be minimally modified, by adding or removing objects from the abstract domains, so that completeness is achieved. Let us give a simple example concerning Mycroft's strictness analysis for functional programs [Burn et al. 1986; Mycroft 1981].

**Example 3.1.3** Consider the following function $F$ of type $\mathsf{Nat} \times \mathsf{Nat} \longmapsto \mathsf{Bool}$:

$$F(\langle x, y\rangle) \stackrel{\text{def}}{=} if\ (x = 3\ and\ y = 3)\ then\ true\ else\ \bot$$

Following Burn et al. [Burn et al. 1986], from $F$ one gets in the most natural way its denotational "collecting" semantics

$$f : \wp^{\mathcal{H}}(\mathbb{N}_\bot \times \mathbb{N}_\bot) \longmapsto \wp^{\mathcal{H}}(Bool_\bot)$$

where $\wp^{\mathcal{H}}$ is the Hoare powerdomain operator and $\bot$ denotes undefinedness (i.e., both nontermination and error). Let $S = \{\, \mathsf{0} < \mathsf{1} \,\}$ be the basic strictness domain, abstracting both $\wp^{\mathcal{H}}(\mathbb{N}_\bot)$ and $\wp^{\mathcal{H}}(Bool_\bot)$, and such that $S \times S$ abstracts $\wp^{\mathcal{H}}(\mathbb{N}_\bot \times \mathbb{N}_\bot)$. Concretization and abstraction maps are the usual ones, e.g. $\gamma(\langle \mathsf{0}, \mathsf{0}\rangle) = \{\, \langle \bot, \bot\rangle \,\}$ and $\gamma(\langle \mathsf{0}, \mathsf{1}\rangle) = \{\, \langle \bot, x\rangle \mid x \in \mathbb{N}_\bot \,\}$. Then, the best correct approximation $f^S : S \times S \longmapsto S$ of $f$ is as follows:

$$\begin{array}{rcl} f^S(\langle \mathsf{0}, \mathsf{0}\rangle) & = & \mathsf{0} \\ f^S(\langle \mathsf{0}, \mathsf{1}\rangle) & = & \mathsf{0} \\ f^S(\langle \mathsf{1}, \mathsf{0}\rangle) & = & \mathsf{0} \\ f^S(\langle \mathsf{1}, \mathsf{1}\rangle) & = & \mathsf{1}. \end{array}$$

Clearly, $f^S$ is not complete. For instance, $\alpha(f(\{\, \langle \bot, \bot\rangle, \langle 4, 5\rangle \,\})) = \alpha(\{\, \bot \,\}) = \mathsf{0}$, while $f^S(\alpha(\{\, \langle \bot, \bot\rangle, \langle 4, 5\rangle \,\})) = f^S(\langle \mathsf{1}, \mathsf{1}\rangle) = \mathsf{1}$. These phenomena of incompleteness in strictness analysis are analyzed in depth in [Reddy and Kamin 1993; Sekar et al. 1997], which, however, do not investigate the issue of achieving completeness by minimally modifying the abstract domains. Moreover, because the range and domain of $f$ are different, the method of [Giacobazzi and Ranzato 1997a] is not applicable here. Instead, the methodology proposed here allows us to constructively derive the least extension $\mathcal{E}(S \times S)$ of $S \times S$ which induces a complete abstract interpretation. It should be clear that by adding a point to $S \times S$ which is able to represent the information that the first and second components are surely not simultaneously equal to $3 \in \mathbb{N}_\bot$, one gets a domain inducing a complete abstract interpretation. Indeed, our methodology allows us to constructively derive that $\mathcal{E}(S \times S) = (S \times S) \cup \{\, \langle \neq, \neq\rangle \,\}$, where $\gamma(\langle \neq, \neq\rangle) = (\mathbb{N}_\bot \times \mathbb{N}_\bot) \smallsetminus \{\, \langle 3, 3\rangle \,\}$. In this way, one gets a best correct approximation $f^{S^*} : \mathcal{E}(S \times S) \longmapsto S$ such that $f^{S^*}(\langle \neq, \neq\rangle) = \mathsf{0}$, and therefore completeness has been achieved. ∎

In the following, we will use the following helpful notation to denote that a pair $\langle\rho,\eta\rangle$ of abstractions is complete for a given set of functions $F \subseteq C \xrightarrow{m} D$.

$$\Gamma(C,D,F) \stackrel{\text{def}}{=} \{\, \langle\rho,\eta\rangle \in uco(C) \times uco(D) \mid \forall f \in F.\ \eta \circ f = \eta \circ f \circ \rho \,\}.$$

Whenever $F = \{\, f \,\}$, we simply write $\Gamma(C,D,f)$. The following result lists some interesting properties of completeness.

**Proposition 3.1.4** *Let $f : C \xrightarrow{m} D$, $g : D \xrightarrow{m} E$, $\rho \in uco(C)$, $\eta \in uco(D)$, $\mu \in uco(E)$ and $d \in D$.*

1. *$\langle\rho,\eta\rangle \in \Gamma(C,D,f)$ if and only if $f \circ \rho \sqsubseteq \eta \circ f$;*
2. *$\langle\rho,\eta\rangle \in \Gamma(C,D,f)$ if and only if $\langle\rho,\eta\rangle \in \Gamma(C,D,\eta \circ f)$;*
3. *If $\langle\rho,\eta\rangle \in \Gamma(C,D,f)$ and $\langle\eta,\mu\rangle \in \Gamma(D,E,g)$ then $\langle\rho,\mu\rangle \in \Gamma(C,E,g \circ f)$;*
4. *If $\langle\rho,\eta\rangle \in \Gamma(C,D,f)$, $\delta \sqsubseteq \rho$ and $\beta \sqsupseteq \eta$, then $\langle\delta,\beta\rangle \in \Gamma(C,D,f)$.*

**Proof.**

1. Straightforward by definition.
2. Follows from $\eta$ idempotency.
3. We have to verify that $\mu \circ g \circ f \circ \rho = \mu \circ g \circ f$, and this is shown by the following equalities: $\mu \circ g \circ f = \mu \circ g \circ \eta \circ f = \mu \circ g \circ \eta \circ f \circ \rho = \mu \circ g \circ f \circ \rho$.
4. By Point 1, it is enough to show that $f \circ \delta \sqsubseteq \eta \circ f$, as given by the following inequalities: $f \circ \delta \sqsubseteq f \circ \rho \sqsubseteq \eta \circ f \circ \rho = \eta \circ f \sqsubseteq \beta \circ f$. ■

Point 2 states that, a pair $\langle\rho,\eta\rangle$ is complete for a function $f$ if and only if it is complete for the best correct approximation of $f$, i.e., $\eta \circ f$. Therefore, completeness for a function $f$ actually depends only on the behaviour of $f$ in the observation domain. Moreover, it is easy to observe that, given any other function $g$ in between, i.e., $f \sqsubseteq g \sqsubseteq \eta \circ f$, completeness for $f$ is equivalent to completeness for $g$, whenever the abstraction domain is $\eta$. This observation suggests that, when testing for completeness, we can replace the concrete function with any function between $f$ and $\eta \circ f$. The last point is maybe the most interesting one and it shows an important property of completeness which will be frequently used later in the thesis. It states that, whenever a pair of abstractions $\langle\rho,\eta\rangle$ is complete for a function $f$, any other pair of abstractions obtained by replacing $\rho$ with a more concrete domain and $\eta$ with a more abstract domain, is still complete for $f$. Intuitively, this result confirms our intuition that a given pair $\langle\rho,\eta\rangle$ is complete when the computation domain $\rho$ is concrete "enough" for computing $f$, when the results are observed in $\eta$. Therefore, once we have a complete abstract interpretation, we do not loose completeness by considering more concrete computation domains or, dually, more abstract observation domains. According to this intuition, completeness can be achieved in two different ways: we can either simplify (reduce) the observation domain, or refine (extend) the computation domain. In general, we are interested in achieving completeness by minimally modifying the given abstractions. Therefore, if we fix an observation domain $\eta$, we shall look for the most abstract domain $\varphi$ which is more concrete than $\rho$ and such that $\langle\varphi,\eta\rangle$ is complete, if it exists. Note that, when this happens, any other domain more concrete than $\varphi$ still ensures completeness. Moreover, it does not make sense to search for the most concrete domain $\varphi$ which is contained in $\rho$ such that $\langle\varphi,\eta\rangle$ is complete, because either it coincides with $\rho$ or it does not exist. Dually, when a computation domain $\rho$ is fixed, we shall look for the most concrete domain $\varphi$ which is contained in $\eta$ and such that $\langle\rho,\varphi\rangle$ is complete. Therefore, as a consequence of Point 4 above, one can draw the following two important remarks.

- Let $\rho' = \sqcap\{\,\varphi \in uco(C) \mid \rho \sqsubseteq \varphi,\ \langle\varphi,\eta\rangle \in \Gamma(C,D,F)\,\}$. If $\langle\rho',\eta\rangle \in \Gamma(C,D,F)$ then $\rho' = \rho$;
- Let $\eta' = \sqcup\{\,\varphi \in uco(D) \mid \varphi \sqsubseteq \eta,\ \langle\rho,\varphi\rangle \in \Gamma(C,D,F)\,\}$. If $\langle\rho,\eta'\rangle \in \Gamma(C,D,F)$ then $\eta' = \eta$.

The above observations motivate why we introduce just the following operators. Given $F \subseteq C \xmapsto{m} D$ and $\eta \in uco(D)$, the following operator of type $uco(C) \longmapsto uco(C)$ is meant to refine computation domains.

$$\mathcal{E}_F^{\eta}(\rho) \stackrel{\text{def}}{=} \sqcup\,\{\,\varphi \in uco(C) \mid \varphi \sqsubseteq \rho,\ \langle\varphi,\eta\rangle \in \Gamma(C,D,F)\,\}$$

Also, given $\rho \in uco(C)$, an analogous operator $\mathcal{K}_F^{\rho}$ of type $uco(D) \longmapsto uco(D)$ is introduced.

$$\mathcal{K}_F^{\rho}(\eta) \stackrel{\text{def}}{=} \sqcap\,\{\,\varphi \in uco(D) \mid \eta \sqsubseteq \varphi,\ \langle\rho,\varphi\rangle \in \Gamma(C,D,F)\,\}$$

Thus, e.g., $\mathcal{E}_F^{\eta}(\rho)$ is the least common abstraction of all the domains $\varphi$ more concrete than $\rho$ and such that $\langle\varphi,\eta\rangle$ is complete for $F$. By exploiting these operators, we are now able to formalize the notions of least complete extensions and greatest complete kernels.

**Definition 3.1.5** Let $F \subseteq C \xmapsto{m} D$, $\rho \in uco(C)$ and $\eta \in uco(D)$. If $\langle\rho,\mathcal{K}_F^{\rho}(\eta)\rangle \in \Gamma(C,D,F)$ then $\mathcal{K}_F^{\rho}(\eta)$ is called the *greatest complete kernel* of $\eta$ relative to $\rho$. Dually, if $\langle\mathcal{E}_F^{\eta}(\rho),\eta\rangle \in \Gamma(C,D,F)$ then $\mathcal{E}_F^{\eta}(\rho)$ is called the *least complete extension* of $\rho$ relative to $\eta$. ∎

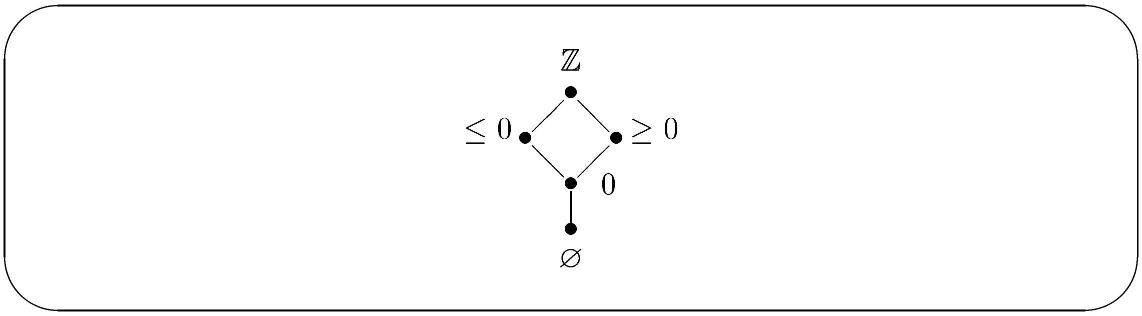



Figure 3.1: The lattice $Sign$

As far as complete kernels are concerned, it is an easy task to show that they always exist for monotone functions, although no explicit characterization can be given.

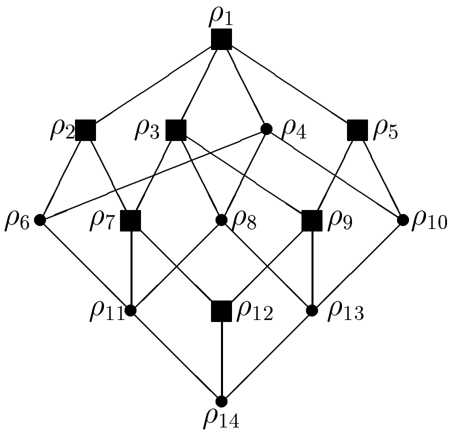
**Proposition 3.1.6** *Let $F \subseteq C \overset{m}{\longmapsto} D$, $\rho \in uco(C)$ and $\eta \in uco(D)$. There exists the greatest complete kernel of $\eta$ relative to $\rho$.*

**Proof.** Let us show that if $\{\mu_i\}_{i\in I} \subseteq uco(D)$ is such that, for any $i \in I$, $\langle \rho, \mu_i \rangle \in \Gamma(C, D, F)$, then $\langle \rho, \sqcap_{i\in I}\mu_i \rangle \in \Gamma(C, D, F)$. Let $f \in F$ and $x \in C$. We have to check that $\wedge_{i\in I}\mu_i(f(\rho(x))) = \wedge_{i\in I}\mu_i(f(x))$, but this is evidently true, since, for any $i \in I$, $\mu_i(f(\rho(x))) = \mu_i(f(x))$. Thus, $\langle \rho, \mathcal{K}^{\rho}_{F}(\eta) \rangle \in \Gamma(C, D, F)$. ∎

**Example 3.1.7** Consider as concrete domain the lattice $Sign$ depicted in Figure 3.1 and the concrete function $Square : Sign \longmapsto Sign$, defined as follows, whose meaning is the obvious one:

$$
\begin{array}{lcl}
Square(\varnothing) & = & \varnothing \\
Square(0) & = & 0 \\
Square(\leq 0) & = & \geq 0 \\
Square(\geq 0) & = & \geq 0 \\
Square(\mathbb{Z}) & = & \geq 0
\end{array}
$$

We consider as computation domain $\rho = \{\mathbb{Z}, 0\} \in uco(Sign)$ and as observation domain the whole concrete domain $Sign$, that is $\eta = \{\mathbb{Z}, \geq 0, \leq 0, 0, \varnothing\}$. It is easy to see that the pair $\langle \rho, \eta \rangle$ is not complete for $Square$. In fact, $\eta(Square(\varnothing)) = Square(\varnothing) = \varnothing$, while $\eta(Square(\rho(\varnothing))) = Square(0) = 0$. Note that $Square$ is trivially monotone, and therefore, by Proposition 3.1.6, there exists the greatest complete kernel of $\eta$ relative to $\rho$. To compute it, we fix the computation domain $\rho$ and we consider all the domains $\varphi$ more abstract than $\eta$ (and thus the whole lattice




$$
\begin{aligned}
\rho_1 &= \{\mathbb{Z}\} \\
\rho_2 &= \{\mathbb{Z}, \geq 0\} \\
\rho_3 &= \{\mathbb{Z}, 0\} \\
\rho_4 &= \{\mathbb{Z}, \varnothing\} \\
\rho_5 &= \{\mathbb{Z}, \leq 0\} \\
\rho_6 &= \{\mathbb{Z}, \geq 0, \varnothing\} \\
\rho_7 &= \{\mathbb{Z}, \geq 0, 0\} \\
\rho_8 &= \{\mathbb{Z}, 0, \varnothing\} \\
\rho_9 &= \{\mathbb{Z}, \leq 0, 0\} \\
\rho_{10} &= \{\mathbb{Z}, \leq 0, \varnothing\} \\
\rho_{11} &= \{\mathbb{Z}, \geq 0, 0, \varnothing\} \\
\rho_{12} &= \{\mathbb{Z}, \leq 0, \geq 0, 0\} \\
\rho_{13} &= \{\mathbb{Z}, \leq 0, 0, \varnothing\} \\
\rho_{14} &= \mathit{Sign}
\end{aligned}
$$

Figure 3.2: The lattice $uco(Sign)$

$uco(Sign)$) such that $\langle \rho, \varphi \rangle$ is complete for $Square$. The greatest complete kernel will be the most concrete abstraction with this property.

Figure 3.2 describes the lattice $uco(Sign)$, where the abstractions $\varphi$ such that $\langle \rho, \varphi \rangle$ is complete are marked by a black square (■). Hence, the greatest complete kernel is $\rho_{12} = \{\, \mathbb{Z}, \leq 0, \geq 0, 0 \,\}$. ■

## 3.1.1 Constructive characterization of completeness

The following key result characterizes complete abstract interpretations in a "constructive" way. In fact, it shows that a completeness equation $\eta \circ f = \eta \circ f \circ \rho$ holds if and only if $\rho$ contains a certain set of points depending on $\eta$, and, in a dual fashion, if and only if $\eta$ is contained in a certain set of points which depends on $\rho$. We will largely exploit the following compact notation: for any $f : C \longmapsto D$ and $y \in D$, $f^{-1}(\downarrow y) = \{\, x \in C \mid f(x) \leq y \,\}$. The proof makes use of a variant of the axiom of choice, known as Hausdorff's Maximal Principle (see e.g., [Birkhoff 1967, pag. 192]).

**Assumption 3.1.8 (Hausdorff's Maximal Principle)** *Every chain in a poset $P$ can be extended to a maximal chain in $P$.*

**Theorem 3.1.9** *Let $f : C \overset{c}{\longmapsto} D$, $\rho \in uco(C)$ and $\eta \in uco(D)$. The following are equivalent.*

1. $\langle \rho, \eta \rangle \in \Gamma(C, D, f)$;
2. $\cup_{y \in \eta} max(f^{-1}(\downarrow y)) \subseteq \rho$;
3. $\eta \subseteq \{ y \in D \mid max(f^{-1}(\downarrow y)) \subseteq \rho \}$.

*Moreover,* $\{ y \in D \mid max(f^{-1}(\downarrow y)) \subseteq \rho \} \in uco(D)$.

**Proof.** First note, that $\forall d \in D$, $f^{-1}(\downarrow d)$ is Scott-closed, since $f$ is continuous.

**(1)⇒(2)** Let $y \in \eta$ and $m \in max(\{ x \in C \mid f(x) \leq y \})$. Let us show that $\rho(m) = m$. Since $f(m) \leq y$, we have $\eta(f(\rho(m))) = \eta(f(m)) \leq \eta(y) = y$, and therefore, by extensivity of $\eta$, $f(\rho(m)) \leq y$. Thus, $\rho(m) \in \{ x \in C \mid f(x) \leq y \}$. By extensivity of $\rho$, it holds that $m \leq \rho(m)$, and therefore, by maximality of $m$, we have that $m = \rho(m)$.

**(2)⇒(1)** By Prop. 3.1.4 (1), we only need to prove that $f(\rho(z)) \leq \eta(f(z))$, for all $z \in C$. Let us consider the set $f^{-1}(\downarrow \eta(f(z))) = \{ x \in C \mid f(x) \leq \eta(f(z)) \}$. By extensivity of $\eta$, we have that $z \in f^{-1}(\downarrow \eta(f(z)))$. By Hausdorff's Maximal Principle, there exists a maximal chain $Z$ in $f^{-1}(\downarrow \eta(f(z)))$ which contains $z$. Since $f^{-1}(\downarrow \eta(f(z)))$ is Scott-closed, we have that $\vee Z \in f^{-1}(\downarrow \eta(f(z)))$, i.e., $f(\vee Z) \leq \eta(f(z))$. Moreover, since $Z$ is a maximal chain, we have that $\vee Z \in max(f^{-1}(\downarrow \eta(f(z))))$. Hence, by hypothesis, $\rho(\vee Z) = \vee Z$. Each of the following inequalities is justified on the left, and implies its successive.

| | | | |
|---|---|---|---|
| since $z \in Z$: | $\rho(z)$ | $\leq$ | $\rho(\vee Z)$ |
| since, $\rho(\vee Z) = \vee Z$: | $\rho(z)$ | $\leq$ | $\vee Z$ |
| by monotonicity of $f$: | $f(\rho(z))$ | $\leq$ | $f(\vee Z)$ |
| by continuity of $f$: | $f(\rho(z))$ | $\leq$ | $\vee f(Z)$ |
| since $\vee f(Z) \leq \eta(f(z))$: | $f(\rho(z))$ | $\leq$ | $\eta(f(z))$. |

The last inequality closes this implication.

**(2)⇔(3)** By a straightforward set-theoretic argument.

We prove that $\curlywedge(\{ y \in D \mid max(f^{-1}(\downarrow y)) \subseteq \rho \}) = \{ y \in D \mid max(f^{-1}(\downarrow y)) \subseteq \rho \}$. Since $max(f^{-1}(\downarrow \top_D)) = \{ \top_C \}$, we have that $\top_D \in \{ y \in D \mid max(f^{-1}(\downarrow y)) \subseteq \rho \}$. Let $Z \subseteq \{ y \in D \mid max(f^{-1}(\downarrow y)) \subseteq \rho \}$, with $Z \neq \varnothing$, and consider now $w = \wedge Z$. We show that $max(f^{-1}(\downarrow w)) \subseteq \rho$. Let $x \in max(f^{-1}(\downarrow w))$. Since $max(f^{-1}(\downarrow w)) = max(\{ c \in C \mid f(c) \leq \wedge Z \}) = max(\cap_{z \in Z} f^{-1}(\downarrow z))$, we have that $x \in \cap_{z \in Z} f^{-1}(\downarrow z)$. For all $z \in Z$, by Hausdorff's Maximal Principle, there exists a maximal chain $M_z$ in $f^{-1}(\downarrow z)$ which contains $x$. Let $m_z = \vee M_z$. Thus $x \leq m_z$, and therefore $x \leq \wedge_{z \in Z} m_z$. Moreover, $m_z \in f^{-1}(\downarrow z)$ (i.e., $f(m_z) \leq z$) since this is Scott-closed, $m_z \in max(f^{-1}(\downarrow z))$ since $M_z$ is a maximal chain and therefore $m_z \in \rho$.

On the other hand, for any $u \in Z$, $f(\wedge_{z\in Z} m_z) \leq f(m_u) \leq u$, from which we get $f(\wedge_{z\in Z} m_z) \leq \wedge Z = w$. Hence, $\wedge_{z\in Z} m_z \in f^{-1}(\downarrow w)$, and therefore, by maximality of $x$, $\wedge_{z\in Z} m_z \leq x$. Thus, $x = \wedge_{z\in Z} m_z$, and since each $m_z \in \rho$ and $\rho$ is a Moore family, we conclude that $x \in \rho$. ■

It is now an easy task to extend the previous result to sets of continuous functions, as the following theorem shows.

**Theorem 3.1.10** *Let $F \subseteq C \stackrel{c}{\longmapsto} D$, $\rho \in uco(C)$ and $\eta \in uco(D)$. The following are equivalent:*

1. $\langle \rho, \eta \rangle \in \Gamma(C, D, F)$;
2. $\eta \subseteq \{\, y \in D \mid \cup_{f\in F} max(f^{-1}(\downarrow y)) \subseteq \rho \,\}$;
3. $\cup_{f\in F, y\in\eta} max(f^{-1}(\downarrow y)) \subseteq \rho$.

*Moreover,* $\{\, y \in D \mid \cup_{f\in F} max(f^{-1}(\downarrow y)) \subseteq \rho \,\} \in uco(D)$.

**Proof.** By definition, $\langle \rho, \eta \rangle \in \Gamma(C, D, F) \iff \langle \rho, \eta \rangle \in \cap_{f\in F} \Gamma(C, D, f)$ holds. Thus, the thesis follows immediately by Theorem 3.1.9. ■

The next step is to constructively characterize the operators $\mathcal{E}_F^\eta$ and $\mathcal{K}_F^\rho$ for least complete extension and greatest complete kernel. First, as the above theorem suggests, given any set of continuous functions $F \subseteq C \stackrel{c}{\longmapsto} D$, we define two mappings $L_F : uco(C) \longmapsto uco(D)$ and $R_F : uco(D) \longmapsto uco(C)$ as follows:

$$L_F(\rho) \stackrel{\text{def}}{=} \{\, y \in D \mid \cup_{f\in F} max(f^{-1}(\downarrow y)) \subseteq \rho \,\}$$

$$R_F(\eta) \stackrel{\text{def}}{=} \curlywedge(\cup_{f\in F, y\in\eta} max(f^{-1}(\downarrow y))).$$

It is then useful to observe that, for any arbitrary set of points $S$ and closure $\rho$, the following equivalence holds: $S \subseteq \rho \iff \rho \sqsubseteq \curlywedge(S)$. Therefore, Theorem 3.1.10 can be restated as follows:

$$\langle \rho, \eta \rangle \in \Gamma(C, D, F) \;\Leftrightarrow\; L_F(\rho) \sqsubseteq \eta \;\Leftrightarrow\; \rho \sqsubseteq R_F(\eta). \tag{3.1.1}$$

In particular, $\langle uco(C), L_F, uco(D), R_F \rangle$ is an adjunction. Consequently, for any $\rho \in uco(C)$ and $\eta \in uco(D)$, one gets the following characterizations for the operators $\mathcal{K}_F^\rho$ and $\mathcal{E}_F^\eta$.

$$\mathcal{K}_F^\rho(\beta) = \sqcap\{\, \varphi \in uco(D) \mid \beta, L_F(\rho) \sqsubseteq \varphi \,\} = \beta \sqcup L_F(\rho)$$

$$\mathcal{E}_F^\eta(\delta) = \sqcup\{\, \varphi \in uco(C) \mid \varphi \sqsubseteq \delta, R_F(\eta) \,\} = \delta \sqcap R_F(\eta).$$

By Theorem 3.1.10, $\langle \rho, L_F(\rho)\rangle, \langle R_F(\eta), \eta\rangle \in \Gamma(C, D, F)$. Hence, by Proposition 3.1.4 (4), it holds that $\langle \rho, \beta \sqcup L_F(\rho)\rangle, \langle \delta \sqcap R_F(\eta), \eta\rangle \in \Gamma(C, D, F)$. Therefore, according to Definition 3.1.5, we can draw the following consequences.

- The greatest complete kernel of $\eta$ relative to $\rho$ is the least common abstraction of $\eta$ and $L_F(\rho)$.
- The least complete extension of $\rho$ relative to $\eta$ is the reduced product of $\rho$ and $R_F(\eta)$.

For any $\rho \in uco(C)$, $\eta \in uco(D)$, it is helpful to define two dual mappings $\mathcal{R}_F^\rho : uco(D) \longmapsto uco(C)$ and $\mathcal{L}_F^\eta : uco(C) \longmapsto uco(D)$ as follows:

$$\mathcal{R}_F^\rho(\mu) \stackrel{\text{def}}{=} \rho \sqcap R_F(\mu)$$

$$\mathcal{L}_F^\eta(\varphi) \stackrel{\text{def}}{=} \eta \sqcup L_F(\varphi).$$

The next results summarizes our results and explicitly states what one must add to $\rho$ in order to get its least complete extension relative to $\eta$ and, dually, what one must subtract from $\eta$ in order to get its greatest complete kernel relative to $\rho$.

**Corollary 3.1.11** *Let $F \subseteq C \stackrel{c}{\longmapsto} D$, $\rho \in uco(C)$ and $\eta \in uco(D)$.*

- $\mathcal{R}_F^\rho(\eta) = \curlywedge(\rho \cup (\cup_{f\in F, y\in\eta} max(f^{-1}(\downarrow y))))$ *is the least complete extension of $\rho$ relative to $\eta$;*
- $\mathcal{L}_F^\eta(\rho) = \eta \cap \{ y \in D \mid \cup_{f\in F} max(f^{-1}(\downarrow y)) \subseteq \rho \}$ *is the greatest complete kernel of $\eta$ relative to $\rho$.*

**Example 3.1.12** Consider Example 3.1.3 concerning Mycroft's strictness analysis and let $\rho \in uco(\wp^{\mathcal{H}}(\mathbb{N}_\perp \times \mathbb{N}_\perp))$ be the closure associated with the computation domain $S \times S$ and $\eta \in uco(\wp^{\mathcal{H}}(Bool_\perp))$ be the closure associated with the observation domain $S$, which are, therefore, defined as follows:

$$\rho = \{ \{ \langle \perp, \perp \rangle \}, \{ \perp \} \times \mathbb{N}_\perp, \mathbb{N}_\perp \times \{ \perp \}, \mathbb{N}_\perp \times \mathbb{N}_\perp \}$$

$$\eta = \{ \{ \perp \}, Bool_\perp \}.$$

The semantic function $f$ is obviously continuous and hence, by Corollary 3.1.11, the least complete extension of $\rho$ for $f$ relative to $\eta$ does exist, and it is given by the reduced product $\mathcal{R}_f^\rho(\eta) = \rho \sqcap R_f(\eta)$. If we compute $max(f^{-1}(\downarrow y))$, for any $y \in \eta$, we have that

- $max(f^{-1}(\downarrow Bool_\perp)) = \{ \mathbb{N}_\perp \times \mathbb{N}_\perp \}$;

- $$\begin{aligned} max(f^{-1}(\downarrow \{\bot\})) &= max(\{ Z \in \wp^{\mathcal{H}}(\mathbb{N}_\bot \times \mathbb{N}_\bot) \mid f(Z) \subseteq \{\bot\}\}) \\ &= max(\{ Z \in \wp^{\mathcal{H}}(\mathbb{N}_\bot \times \mathbb{N}_\bot) \mid \langle 3,3\rangle \notin Z \} \\ &= \{ (\mathbb{N}_\bot \times \mathbb{N}_\bot) \smallsetminus \{\langle 3,3\rangle\}\}. \end{aligned}$$

Hence, $\mathcal{R}^{\rho}_{F}(\eta) = \curlywedge(\rho \cup \{ (\mathbb{N}_\bot \times \mathbb{N}_\bot) \smallsetminus \{\langle 3,3\rangle\}\}) = \rho \cup \{ (\mathbb{N}_\bot \times \mathbb{N}_\bot) \smallsetminus \{\langle 3,3\rangle\}\}$. As announced in Example 3.1.3, and as one naturally expects, the least complete extension of $S \times S$ can be obtained by adding a point $\langle \neq, \neq\rangle$ with concrete meaning $(\mathbb{N}_\bot \times \mathbb{N}_\bot) \smallsetminus \{\langle 3,3\rangle\}$, i.e., denoting that the first and the second components are surely not simultaneously equal to 3. It should be clear that this refined computation abstract domain induces now a complete abstract interpretation. ■

It is worth noting that, for any complete abstract interpretation, continuity is preserved from the concrete to the abstract setting, as proved by the following theorem.

**Theorem 3.1.13** *Let $f : C \xrightarrow{c} D$, $\rho \in uco(C)$, and $\eta \in uco(D)$ such that $\langle \rho, \eta\rangle \in \Gamma(C, D, f)$. Then $\eta \circ f : \rho(C) \longmapsto \eta(D)$ is continuous on $\rho(C)$.*

**Proof.** Let $Y \subseteq \rho(C)$ be a chain. First note that $Y$ is also a chain in $C$, by definition of the ordering in $\rho(C)$.

$$\begin{aligned} \eta \circ f(\vee_\rho Y) &= \eta \circ f \circ \rho(\vee_C Y) && [\text{by definition of } \vee_\rho] \\ &= \eta \circ f(\vee_C Y) && [\text{by completeness of } f] \\ &= \eta(\vee_D f(Y)) && [\text{by continuity of } f \ (Y \text{ is a chain in } C)] \\ &= \eta(\vee_D \eta(f(Y))) && [\text{by Eq. (2.2.1)}] \\ &= \vee_\eta \eta \circ f(Y)) && [\text{by definition of } \vee_\eta]. \end{aligned}$$

Hence, $\eta \circ f$ is continuous on $\rho(C)$. ■

Therefore, for any complete abstraction, continuity is kept from the concrete function $f$ to the best correct approximation $\eta \circ f$.

### 3.1.2 Fixpoint and absolute completeness

In this section we consider the common case of programming languages where the concrete semantics of a program $P$ is specified as least fixpoint of some semantic function. This is the case, for instance, of standard semantics of logic and functional languages. Therefore, a typical framework for analyses (e.g., in [Cousot and Cousot 1979b]) is given by a concrete domain $C$ and a concrete semantic function $[\![\cdot]\!] : Program \longmapsto C$, where the semantics $[\![P]\!]$ of a program $P$ is defined as the least fixpoint of some concrete function $f_P : C \longmapsto C$, i.e., $[\![P]\!] = lfp(f_P)$. An abstract interpretation is then specified by an abstract domain $A$ and an abstract function $f_P^A : A \longmapsto A$. The abstract semantics $[\![P]\!]^A$ of a program $P$ is then defined as $[\![P]\!]^A = \mathit{lfp}(f_P^A)$, providing the abstract semantics to correctly approximate the concrete one, i.e., $\alpha([\![P]\!]) \leq_A [\![P]\!]^A$, that is to say that $\alpha(\mathit{lfp}(f_P)) \leq_A \mathit{lfp}(f_P^A)$.

In this scenario, an abstract interpretation is complete when equality holds, i.e.,

$$\alpha(\mathit{lfp}(f_P)) = \mathit{lfp}(f_P^A).$$

In order to distinguish this notion of completeness from that one in Definition 3.1.1, we shall refer to it as **fixpoint completeness***fixpoint completeness*. Actually, fixpoint completeness was the first notion to be presented in [Cousot and Cousot 1979b]. A question naturally arises: can we minimally modify (extend or reduce) the domain $A$ in order to get a fixpoint complete abstraction? In particular, does there exists a more abstract closure which is more concrete than $A$ and induces fixpoint completeness (the so-called least fixpoint complete extension).

Unfortunately, as observed in [Giacobazzi and Ranzato 1997a], the least fixpoint complete extension of a given abstract domain exists only under very restrictive hypotheses on the concrete function $f_P$. In particular, they show that fixpoint complete abstractions may fail to exist even for continuous semantics functions. Already [Cousot and Cousot 1979b] observed that testing for fixpoint completeness is a hard task, and therefore suggested the following stronger condition in order to construct fixpoint complete abstract interpretation.

$$\alpha \circ f_P = f_P^A \circ \alpha \qquad \text{for all programs } P$$

From Proposition 3.1.2, this is equivalent to saying that $f_P^A$ is the best correct approximation of $f$ (i.e., $f_P^A = \alpha \circ f_P$) and $\alpha \circ f_P = \alpha \circ f_P \circ \alpha$.

By resorting to our closure operator approach, this problem can be viewed as an instance of Definition 3.1.1 where $C = D$ and $\rho = \eta$. Therefore, the problem boils down to studying the completeness problem $\rho \circ f = \rho \circ f \circ \rho$. It is important to remark that if $\langle \mathcal{E}_F^\rho(\rho), \rho \rangle \in \Gamma(C, C, F)$, then $\mathcal{E}_F^\rho(\rho)$ is the most abstract among the domains $\varphi \sqsubseteq \rho$ such that $\rho \circ f = \rho \circ f \circ \varphi$, since the observation domain $\rho$ is thought of as fixed. Hence, by defining

$$E_F(\rho) \stackrel{\text{def}}{=} \sqcup \{\, \varphi \in \mathit{uco}(C) \mid \varphi \sqsubseteq \rho,\ \langle \varphi, \varphi \rangle \in \Gamma(C, C, F) \,\}$$

this latter least complete extension exists whenever $\langle E_F(\rho), E_F(\rho) \rangle \in \Gamma(C, C, F)$. Therefore, in this case, $\rho$ considered as observation domain is not fixed. Thus, we remark that this latter concept of least complete extension is different from that introduced in Definition 3.1.5 and, even though it is an instance of Definition 3.1.1, we can not use the same machinery to solve it. In order to distinguish them, we refer to this latter form of completeness as **absolute completeness**. Hence, when

$\langle E_F(\rho), E_F(\rho)\rangle \in \Gamma(C, C, F)$, we will call $E_F(\rho)$ the *absolute least complete extension* of $\rho$. Moreover, analogous dual considerations hold for greatest complete kernels: we will call them *absolute greatest complete kernels*. When clear from the context, we shall refer to absolute complete extensions and absolute complete kernels simply as complete extensions and complete kernels.

Assume that $C = D$, and let $F \subseteq C \xrightarrow{c} C$ and $\rho \in uco(C)$. By Eq. (3.1.1), for any $\varphi \in uco(C)$, we have that

- $\varphi \sqsubseteq \mathcal{R}^\rho_F(\varphi) \iff \varphi \sqsubseteq \rho$ and $\langle\varphi, \varphi\rangle \in \Gamma(C, C, F)$;
- $\mathcal{L}^\rho_F(\varphi) \sqsubseteq \varphi \iff \rho \sqsubseteq \varphi$ and $\langle\varphi, \varphi\rangle \in \Gamma(C, C, F)$.

Therefore, the operator $E_F$ introduced above can be rephrased as follows:

$$E_F(\rho) = \sqcup\{\, \varphi \in uco(C) \mid \varphi \sqsubseteq \mathcal{R}^\rho_F(\varphi) \,\}.$$

Since $\mathcal{R}^\rho_F : uco(C) \longmapsto uco(C)$ is clearly monotone for any $\rho$, and hence admits the greatest fixpoint, we get $E_F(\rho) = \mathit{gfp}(\mathcal{R}^\rho_F)$. Moreover, by Corollary 3.1.11, $\langle \mathit{gfp}(\mathcal{R}^\rho_F), \mathit{gfp}(\mathcal{R}^\rho_F)\rangle \in \Gamma(C, C, F)$. This means that the absolute least complete extension of $\rho$ exists, and it is $\mathit{gfp}(\mathcal{R}^\rho_F)$. Dual considerations hold for complete kernels. Thus, we get the following constructive characterization for absolute completeness.

**Corollary 3.1.14** *Let $F \subseteq C \xrightarrow{c} C$ and $\rho \in uco(C)$. Then, $\mathit{gfp}(\mathcal{R}^\rho_F)$ and $\mathit{lfp}(\mathcal{L}^\rho_F)$ are, respectively, the absolute least complete extension and absolute greatest complete kernel of $\rho$.*

## 3.2 Completeness: a historical vision

Completeness in abstract interpretation has been first considered by Cousot and Cousot [1979b], who studied the basic properties of complete abstractions in the approximation of least fixpoint semantics. After that seminal paper, a number of works considered complete abstract interpretations in specific applications. It would be impossible to give a full account of all these works; therefore, we consider only some representative works in different areas where abstract interpretation is traditionally used. Note that in all the following works, completeness is always meant as absolute completeness, unless otherwise stated.

In comparative semantics of programming languages, Cousot and Cousot [1992b] and Cousot [1997a] proved that some classical semantics of programming languages, like (demoniac, erratic and angelic) relational, denotational, predicate transformer and axiomatic, are all complete abstractions of a generalized common SOS operational semantics. In the field of logic program semantics, Giacobazzi [1996] proved that complete abstractions relating semantics provide a corresponding hierarchy of models for positive logic programs, while Comini and Levi [1994] based their taxonomy of observables (perfect, denotational, and semi-denotational) on the notion of completeness, applied to some basic additive operators for building SLD-trees. Recently, Amato and Levi [1997] studied the lattice-theoretic structure of these observables, which is basically the lattice of complete abstractions for the above additive operators for building SLD-trees.

In program analysis and type inference, Cousot [1997b] formalized Hindley monotype semantics as a (fixpoint) complete abstraction of the more concrete Church-Curry monotype semantics. Sekar et al. [1997] focussed on completeness of Mycroft's [1980; 1981] strictness analysis for functional programs. Their approach is rather different from ours: In fact, they identified the greatest class $Cl$ of functional programs (roughly, all the programs whose computational behavior is unaffected under any variation of constants) such that the strictness analysis of a program $P$ is complete if and only if $P \in Cl$. Thus, in our terminology, they have found the greatest set of programs $Cl$ such that $\mathit{Strictness} \in \Delta(C, \{T_P\}_{P\in Cl})$, where $C$ is the concrete domain of the standard collecting denotational semantics and $T_P$ is the standard least fixpoint semantic operator for a functional program $P$. Reddy and Kamin [1993] generalize the above approach of Sekar et al., which first appeared in POPL '91, by studying completeness with respect to a so-called similarity concrete semantics, which is defined in a denotational-like environment. They obtained two completeness results, respectively, for first-order and typed higher-order functional languages, that in a certain precise sense subsume the completeness result of Sekar et al. Thus, also their approach substantially differs from our perspective. Steffen [1989] is one of the first authors isolating completeness as a key property for analysis. He introduces a concept of observation equivalence for abstract interpretations, which is similar to our notion of absolute completeness. However, these approaches have a quite different viewpoint. Steffen's approach is "observation directed", in the sense that the design of a fully abstract (complete) abstract interpretation is directed by a certain observation level. This differs from our approach, which is, as the standard Cousot and Cousot theory of abstract interpretation, "semantics directed". A categorical generalization of Steffen's work has successively been investigated by Steffen et al. [1992]. More recently, Colby [1997] isolated the phenomenon of accumulated imprecision in abstract interpretation. This is essentially the same as the lack of completeness for the considered domain: An error is accumulated in the least fixpoint evaluation of an abstract semantics, analogously to the accumulated rounding error in numerical algorithms. To overcome these problems, Colby [1997] proposed to consider a new enhanced so-called transfer relations language to express the net behavior of finite control paths in the operational semantics of programming languages, and he showed that this allows us to solve the problem in some relevant examples. Colby's approach is therefore orthogonal to ours: The whole semantic metalanguage is changed to gain precision. We believe instead that the accumulated imprecision is inherently an abstract domain property, and, as we proved, precision can be enforced in any abstract domain of any abstract interpretation by either refining or simplifying them. Therefore, Colby's method can also be understood as a refinement of the abstract domain, where symbolic assertions are used between control points.

Completeness is particularly frequent in abstract model checking and in analysis of reactive systems. In abstract model checking, Clarke et al. [1994] observed that when dealing with complete abstract models (that they term exact), a given temporal CTL* formula can be equivalently checked either at the concrete or at the abstract level. Clarke et al. argue that this is an optimal situation, which however is unlikely to be useful in practice for reducing the complexity of verification. Completeness also plays a central rôle in [Dams et al. 1997] in abstract interpretation for checking properties of reactive systems, where the authors consider it a very desirable situation and hence call it optimality. In Section 5 of that work, the authors give some specific sufficient conditions so that the computed abstract model is complete. Furthermore, Dams et al. observe in Section 6 that the construction of a complete, i.e. optimal, abstract model may require a refinement of the precision of the underlying abstract domain, although they claim that abstract interpretation does not offer a systematic approach to accomplish this task.

Completeness is also frequent in complexity studies on program analysis systems. Debray [1995] considered a notion of exactness which is strictly stronger than completeness. He defined the complexity of a dataflow analysis algorithm $A$ as the complexity of typical programs in the class of exactness of $A$. If $A$ computes the least fixpoint of an abstract operator $T_P^\sharp$, then the above class of exactness is precisely the set of all programs $P \in \mathit{Program}$ such that $\mathit{lfp}(T_P) = \gamma(\mathit{lfp}(T_P^\sharp))$. Although this notion of exactness is evidently stronger than completeness, the author proves that it still represents a reasonable absolute (i.e. not relative to other algorithms) measure of precision for practical purposes. Recently, Deutsch [1997] improved on the precision of Park and Goldberg [1992] escape analysis of strongly typed functional languages, by specifying a complete abstraction which is polynomial in the first-order case. This represents a strong improvement with respect to the original exponential proposal. Similarly, Bagnara et al. [1997] obtained an abstract unification algorithm for pair-sharing analysis of logic programs, which is quadratic in complexity. This is obtained by characterizing a complete abstraction of the set-sharing domain, which leads traditionally to exponential algorithms for abstract unification.

Few works in abstract interpretation theory devote particular attention to completeness by its own. The most related works are certainly those of Mycroft [1993] and Giacobazzi and Ranzato [1997a]. Mycroft considers a notion of completeness which is essentially equal to our absolute completeness, except that he develops his theory using a predicate-based approach to abstract interpretation (a kind of logical view of classical abstract interpretation). Moreover, he argues that the well-known state minimization algorithm for finite deterministic automata can be used to produce a canonical complete abstract interpretation by removing useless domain elements. Although the technical approach followed by Mycroft is quite different from ours, the idea of systematically defining a canonical simplest complete abstract interpretation is basically the same idea of our absolute complete kernel operator. This relationship certainly deserves further investigation in order to see if these two systematic methodologies actually have the same behavior. On the other hand, Mycroft does not consider the dual problem analogous to our least complete extension operators.

More recently, Giacobazzi and Ranzato [1997a] pointed out that completeness is an abstract domain property, which only depends on the underlying abstract domain, and proved the existence of absolute complete extensions and absolute complete kernels, dealing with functions of type $C^n \longmapsto C$. In general, they only prove the existence and give a constructive methodology (similar to our Corollary 3.1.14) only when the semantic operations are additive. However, additivity is too restrictive a hypothesis to be widely applicable in practice. More importantly, functions of generic type $C \longmapsto D$, occurring frequently in denotational semantics for functional programming, cannot be handled.

Hence, the results in this chapter fully solve the problems and the limitations of the previous approaches. In fact, we deal with the most general hypothesis of continuous (possibly typed) semantic functions and, more importantly, we characterize in a constructive way the solutions to all completeness problems. We believe that our results can be straightforwardly and successfully applied to most fields where completeness is required, in particular to abstract model checking, where completeness turns out to be a key property (see, e.g., [Bensalem et al. 1998]).

# Chapter 4

# Projection-based program analysis

Abstract

In this chapter, we apply our completeness results in projection-based analysis of functional programs. The key problem in projection-based backward analysis consists in finding a certain least projection, called least backward abstraction. We show that the projection equation corresponds to a basic completeness problem, and, notably, least backward abstractions correspond to least complete extensions. Therefore, from our results, we derive a constructive method to set up least backward abstractions.

The results presented in this chapter are a joint work with R. Giacobazzi and F. Ranzato.

Wadler and Hughes [1987] projection-based program analysis is a well-known technique for statically finding out information about functional programs, used in compilers for lazy functional languages. This technique is particularly useful to detect certain forms of data structure strictness, such as head strictness for lists. Let us recall the basic points of this approach. Let $C$ and $D$ be cpo's, and $f : C \longmapsto D$ the (standard denotational semantics of the) function to analyze. Given two projections $\beta \in prj(C)$ and $\delta \in prj(D)$, one says that $f$ is $\beta$-strict in a $\delta$-context when $\delta \circ f = \delta \circ f \circ \beta$. Once a context $\delta$ is fixed, the key problem of projection-based backward analysis consists in finding a projection $\beta$ such that the following basic equation holds.

$$\delta \circ f = \delta \circ f \circ \beta \tag{4.0.1}$$

Of course, the intuition is that smaller projections $\beta$ (w.r.t. the pointwise order of $prj(C)$) carry more information. Thus, ideally, one aims at finding the so-called *least backward abstraction* of $f$ relative to $\delta$, when it exists. More generally, given $f$, the goal is to find the least backward abstraction operator $\mathbb{B}_f : prj(D) \longmapsto prj(C)$ which, for any context $\delta$, yields the corresponding least backward abstraction.

The theory developed by Davis [1992; 1993] for determining least backward abstraction is based on the notion of *stable* function (stability in the sense of Berry [1978]). A continuous function $f : C \longmapsto D$ is stable if for all $x \in C, y \in D$ such that $y \leq_D f(x)$, there exists a least value $z \leq_C x$ such that $y \leq_D f(z)$. Davis [1992] proved that least backward abstraction does exist for each stable function. We aim to show that Davis' results can be improved by relaxing stability to co-continuity.

Davis claimed that, in general, least backward abstractions do not exist for continuous functions [Davis 1992; Davis 1993]. He presented an example about parallel-or and showed that it does not admit a least backward abstraction. Note that, parallel-or is a typical example of continuous, nonstable function. However, his example was not correct. First, let us see that the parallel-or function does admit a least backward abstraction.

**Example 4.0.1** We follow [Davis 1992] and consider the well-known parallel-or function $por : Bool_\perp \times Bool_\perp \longmapsto Bool_\perp$ such that $por(x, \perp) = por(\perp, x) = x$, and, for $x \neq \perp \neq y$, $por(x, y) = x \text{ or } y$. Then, Davis considers the greatest projections $\beta_1, \beta_2 \in prj(Bool_\perp \times Bool_\perp)$ such that $\beta_1(\langle true, true\rangle) = \langle true, \perp\rangle$ and $\beta_2(\langle true, true\rangle) = \langle \perp, true\rangle$. Next, he observes that $por = por \circ \beta_i$, while $por \neq por \circ (\beta_1 \sqcap \beta_2)$. Instead, note that the set of fixpoints of $\beta_1$ is $Bool_\perp \times Bool_\perp \setminus \{ \langle true, true\rangle, \langle \perp, true\rangle \}$. In fact, $\langle \perp, true\rangle$ cannot be a fixpoint, otherwise we would loose monotonicity of $\beta_1$. A similar argument shows that $\beta_2 = Bool_\perp \times Bool_\perp \setminus \{ \langle true, true\rangle, \langle true, \perp\rangle \}$. Thus, it is not true that $por = por \circ \beta_i$. In fact, $por(\langle \perp, true\rangle) = true$, while $por(\beta_1(\langle \perp, true\rangle)) = por(\langle \perp, \perp\rangle) = \perp$. Further, it is not too hard to check that the only projection $\beta \in prj(Bool_\perp \times Bool_\perp)$ such that $por \circ \beta = por$ is the identity projection. Actually, this is the least backward abstraction of $por$ relative to the identity context $\lambda x.x \in prj(Bool_\perp)$. ∎

Since the parallel-or function is the archetypical example of nonstable function, it motivated the following result by Davis: "if $f$ is stable then, for any context $\delta$, there exists the least backward abstraction of $f$ relative to $\delta$". Actually, by using our approach, a stronger result can be proved.

In order to exploit our results for projections, we need the following technical remarks and notation. First note that, the lattice of abstract interpretations can be set up by considering the set of closures on a cpo $C$, instead of a complete lattice. In fact, Ranzato [1998] showed that if $C$ is a cpo then $\langle uco(C), \sqsubseteq\rangle$ is a complete lattice, or, dually, that if $C$ is a co-cpo then $\langle prj(C), \sqsubseteq\rangle$ is a complete lattice. As we recalled in Section 2.2.2, for any cpo $C$, $\langle uco(C), \sqsubseteq\rangle$ and $\langle prj(C), \sqsubseteq\rangle$ are dually-isomorphic. Thus, the results of Section 3.1.1 can be dually interpreted for projections. Moreover, let us notice that in the proof of the three equivalences in Theorem 3.1.9, the only used hypothesis on the structure of $C$ and $D$ is the existence of l.u.b.'s of chains in $C$, namely that $C$ is a cpo. Thus, $prj(C)$ being a complete lattice, for any $S \subseteq C$, let us denote by $\mathcal{N}(S)$ the least subset of $C$ containing $S$, which is the set of fixpoints of a projection on $C$, i.e., $\mathcal{N}(S)$ is the least projection whose set of fixpoints contains $S$. When $C$ is a complete lattice, it turns out that $\mathcal{N}(S)$ is the join-closure in $C$ of $S$. Also, $f : C \overset{co-c}{\longmapsto} D$ denotes that the function $f$ is co-continuous, and, for any $y \in D$, $f^{-1}(\uparrow y) = \{ x \in C \mid y \leq_D f(x) \}$. Given this notation, we can state the following dual version of Corollary 3.1.11.

**Theorem 4.0.2** *Let $C$ and $D$ be co-cpo's, $f : C \overset{co-c}{\longmapsto} D$, $\beta \in prj(C)$, and $\delta \in prj(D)$. Then $\mathcal{N}(\cup_{y \in \delta} min(f^{-1}(\uparrow y)))$ is the least backward abstraction of $f$ relative to $\delta$.*

The importance of the above theorem is twofold. First, as far as existence of least backward abstractions is concerned, it considerably improves on the corresponding theorem given by Davis [1992; 1993]. Second, and more remarkably, it explicitly shows how to compute such least backward abstractions, i.e., it gives a practical and simple program analysis methodology. To conclude, let us consider Davis' example of parallel-or recalled above.

**Example 4.0.3** We continue Example 4.0.1 and consider the parallel-or function $por : Bool_\perp \times Bool_\perp \longmapsto Bool_\perp$ recalled above. We have already noted that the least backward abstraction $\mathbb{B}_{por}(\lambda x.x)$ of $por$ relative to the identity context $\lambda x.x \in uco(Bool_\perp)$ does exist and it is the identity projection. Since $por$ is trivially co-continuous, the existence of $\mathbb{B}_{por}(\lambda x.x)$ is ensured by Theorem 4.0.2. Moreover, Theorem 4.0.2 also says that $\mathbb{B}_{por}(\lambda x.x) = \mathcal{N}(\cup_{y \in Bool_\perp} min(por^{-1}(\uparrow y)))$. Let us verify this identity.

- $min(por^{-1}(\uparrow \perp)) = \{ \langle \perp, \perp \rangle \}$
- $min(por^{-1}(\uparrow true)) = \{ \langle \perp, true \rangle, \langle true, \perp \rangle \}$
- $min(por^{-1}(\uparrow false)) = \{ \langle \perp, false \rangle, \langle false, \perp \rangle \}$.

Thus, the least projection containing

$$\{ \langle \perp, \perp \rangle, \langle \perp, true \rangle, \langle true, \perp \rangle, \langle \perp, false \rangle, \langle false, \perp \rangle \}$$

is just the identity closure $Bool_\perp \times Bool_\perp$. ■

# Chapter 5

# Generalized quotients of abstract interpretations

Abstract

We apply our completeness results in abstract domain comparison by quotients. We investigate the relationship between completeness and the concept of quotient of an abstract interpretation, recently introduced for comparing the precision of abstract interpretations in computing a given property. We prove that quotients, when they exist, turn out to be certain least complete extensions, which naturally formalize the intuition behind the notion of quotient. Thus, a simple and natural generalization of the notion of quotient is proposed, which retains the advantage of always being well-defined, under the hypothesis of continuity of the semantic functions.

The results presented in this chapter are a joint work with R. Giacobazzi and F. Ranzato and appeared in [Giacobazzi et al. 1998a].

The concept of *quotient* of an abstract interpretation has been recently introduced by Cortesi et al. [1998] in order to formalize the least amount of information of a complex abstract domain $A$ which is useful for computing some property that $A$ is able to represent. They show how to exploit this notion for comparing the precision of two abstract interpretations in computing a given common property. In logic programming analyses, quotient has been used to compare the well-known *Sharing* (Jacobs and Langen [1992]) and *Pos* (Marriott and Søndergaard [1993]) Prolog abstract interpretations, by proving that *Pos* is strictly more precise than *Sharing* for computing groundness information. Further, Bagnara et al. [1997] show that, in order to compute pair-sharing information, the use of the quotient of *Sharing* with respect to the pair-sharing domain (Søndergaard [1986]) leads to remarkable gains of efficiency, when compared with the full domain *Sharing*.

Let us recall from [Cortesi et al. 1998] the definition of quotient. Let $A$ be any complete lattice, $f : A \overset{m}{\longmapsto} A$ be a monotone semantic function on $A$, and $\rho \in$

$uco(A)$ be an abstraction of $A$. Here, $\langle A, f\rangle$ models any abstract interpretation of some reference semantic definition, while $\rho$ plays the rôle of the property (i.e., the abstraction of $A$) one is interested in. The equivalence relation $r_\rho \subseteq A\times A$ is defined as follows:[1]

$$\langle a_1, a_2\rangle \in r_\rho \;\text{ iff }\; \forall i \in \omega.\ \rho(f^i(a_1)) = \rho(f^i(a_2)).$$

Roughly speaking, $\langle a_1, a_2\rangle \in r_\rho$ when $A$ views $a_1$ and $a_2$ as "equivalent" w.r.t. the computation of the property $\rho$. Thus, according to this intuition, the quotient $\mathcal{Q}_\rho(A)$ of $A$ w.r.t. $\rho$ is defined as the subset of $A$ of the l.u.b.'s of all equivalence classes of $r_\rho$. In other words, if $[a]$ denotes a generic equivalence class for $r_\rho$, then $\mathcal{Q}_\rho(A) \stackrel{\text{def}}{=} \{\vee[a] \mid a \in A\}$, and the ordering is inherited from $A$. Cortesi et al. [1998, Theorem 3.6] show that if the equivalence $r_\rho$ is additive, i.e., $\forall i \in I.\ \langle a_i, b_i\rangle \in r_\rho \Rightarrow \langle \vee_{i\in I} a_i, \vee_{i\in I} b_i\rangle \in r_\rho$, then $\mathcal{Q}_\rho(A)$ is well-defined, i.e., it is an abstraction of $A$, and $\rho$ is more abstract than $\mathcal{Q}_\rho(A)$. However, they do not provide any constructive method to set up quotients, when they exist.

These results can be sharpened as follows. First, it is useful to recall (see [Cousot and Cousot 1979b, Section 6.3]) that, in general, given an equivalence relation $R$ on a complete lattice $L$, $R$ is additive if and only if $\lambda x.\ \vee_L [x]_R \in uco(L)$. Thus, the hypothesis that the equivalence relation $r_\rho$ is additive is indeed equivalent to the fact that $\lambda a.\ \vee[a] \in uco(A)$, i.e., that the quotient $\mathcal{Q}_\rho(A)$ is well-defined. In this case, that $\mathcal{Q}_\rho(A) \sqsubseteq \rho$ is an immediate consequence. In fact, if $a = \rho(a)$ and $b \in [a]$ then $a = \rho(a) = \rho(f^0(a)) = \rho(f^0(b)) = \rho(b)$, and hence $b \le a$, i.e., $\vee[a] = a$.

Moreover, we observe that the additivity of $f$ is an obvious sufficient condition guaranteeing that the quotient exists. Actually, the quotients presented in [Bagnara et al. 1997; Cortesi et al. 1998] exist just because the involved semantic functions are additive.

**Lemma 5.0.4** *If $f : A \xrightarrow{a} A$ and $\rho \in uco(A)$, then $\mathcal{Q}_\rho(A) \in uco(A)$.*

**Proof.** Let us show that $r_\rho$ is additive. First, note that for any $i \in \omega$, $f^i$ is additive. Suppose that for any $k \in K$, $\langle a_k, b_k\rangle \in r_\rho$. Then, for any $i \in \omega$, $\rho(f^i(\vee_{k\in K} a_k)) = \rho(\vee_{k\in K} f^i(a_k)) = \rho(\vee_{k\in K}\rho(f^i(a_k)))$. Thus, since $\langle a_k, b_k\rangle \in r_\rho$, for any $k \in K$, we have that $\rho(f^i(\vee_{k\in K} a_k)) = \rho(\vee_{k\in K}\rho(f^i(b_k)))$, and therefore, by reasoning as above, $\rho(f^i(\vee_{k\in K} a_k)) = \rho(f^i(\vee_{k\in K} b_k))$, which closes the proof. ■

It turns out that the quotient abstract domain satisfies the following remarkable property of "minimality": when a quotient $\mathcal{Q}_\rho(A)$ exists, if $\phi_\rho = \lambda a.\ \vee[a] \in uco(A)$ is the closure associated with $\mathcal{Q}_\rho(A)$, then $\phi_\rho$ is the most abstract solution in $uco(A)$ of the system of equations $\{\rho \circ f^i = \rho \circ f^i \circ \psi \mid i \in \omega\}$.

[1] This definition considers the case of the first limit ordinal $\omega$ for practical purposes – a generalization to any (possibly transfinite) ordinal would be straightforward.

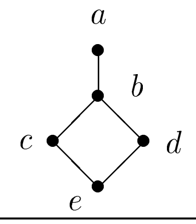



Figure 5.1: The domain $A$

**Lemma 5.0.5** *Let $\rho \in uco(A)$ such that $\mathcal{Q}_\rho(A) \in uco(A)$. Then, $\forall i \in \omega.\ \rho \circ f^i = \rho \circ f^i \circ \mathcal{Q}_\rho(A)$, and for any $\psi \in uco(A)$, $\forall i \in \omega.\ \rho \circ f^i = \rho \circ f^i \circ \psi$ implies $\psi \sqsubseteq \mathcal{Q}_\rho(A)$.*

**Proof.** Assume that $\phi_\rho = \lambda a. \vee [a] \in uco(A)$. Therefore, for any $i \in \omega$, we have that $\rho(f^i(a)) = \rho(f^i(\vee[a]))$, i.e., for any $i \in \omega$, $\rho \circ f^i = \rho \circ f^i \circ \mathcal{Q}_\rho(A)$. Then, let $\psi \in uco(A)$ such that $\forall i \in \omega.\ \rho \circ f^i = \rho \circ f^i \circ \psi$ and $a \in A$. Hence, $\psi(a) \in \{ x \in A \mid \forall i \in \omega.\ \rho(f^i(x)) = \rho(f^i(a)) \} = [a]$, and therefore $\psi(a) \le \vee[a] = \phi_\rho(a)$. ∎

Since the system of equations $\{ \rho \circ f^i = \rho \circ f^i \circ \psi \mid i \in \omega \}$ is clearly equivalent to the system $\{ \psi \sqsubseteq \rho \} \cup \{ \rho \circ f^i = \rho \circ f^i \circ \psi \mid i > 0 \}$, the above lemma says that, when a quotient $\mathcal{Q}_\rho(A)$ exists (i.e., $\mathcal{Q}_\rho(A) \in uco(A)$), it is characterized as follows:

$$\mathcal{Q}_\rho(A) = \sqcup \{ \psi \in uco(A) \mid \psi \sqsubseteq \rho,\ \forall i > 0.\ \rho \circ f^i = \rho \circ f^i \circ \psi \}.$$

This means that, when it exists, $\mathcal{Q}_\rho(A)$ is the least complete extension of $\rho$ for the set of functions $\{ f^i \}_{i>0}$ relative to $\rho$ itself. However, it may well happen that, for some $A$, $f$ and $\rho$, such least complete extension exists, while the quotient $\mathcal{Q}_\rho(A)$ does not exist, as the following simple example shows.

**Example 5.0.6** Let $A$ be the lattice depicted in the Figure 5.1, $f : A \longmapsto A$ be defined as $f = \{ a \mapsto a,\ b \mapsto a,\ c \mapsto e,\ d \mapsto e,\ e \mapsto e \}$, and $\rho \in uco(A)$ such that $\rho(A) = \{ a, b, e \}$. Trivially, $f$ is monotone (and therefore continuous) but not additive. Moreover, $f$ is idempotent, and therefore, for any $i \ge 1$, $f^i = f$. It turns out that, for any $i \ge 1$, $\rho \circ f^i = f$. As a consequence, $r_\rho$ is not an additive equivalence relation. In fact, for any $i \ge 0$, $\rho(f^i(c)) = \rho(f^i(d))$: If $i = 0$ then, $\rho(c) = \rho(d) = b$; if $i \ge 1$ then, $\rho(f^i(c)) = f(c) = e = f(d) = \rho(f^i(d))$. But, $\rho(f(c \vee d)) = \rho(f(b)) = f(b) = a$. Hence, this means that the quotient $\mathcal{Q}_\rho(A)$ does not exist. Instead, as each $f^i$ is continuous, by Corollary 3.1.11, the least complete extension of $\rho$ for $\{ f^i \}_{i>0}$ relative to $\rho$ does exist and it is given by the following reduced product: $\rho \sqcap (\cup_{i>0, y \in \rho} max((f^i)^{-1}(\downarrow y)))$. It is then a routine task to check that this is the domain $A$ itself, i.e., the identity closure $\lambda x.x$. ∎

Then, Lemma 5.0.5 and Example 5.0.6 hint to generalize the notion of quotient as the least complete extension of $\rho$ for $\{ f^i \}_{i>0}$ relative to $\rho$, whenever it exists.

**Definition 5.0.7** Given a complete lattice $A$, $f : A \overset{m}{\longmapsto} A$, and $\rho \in uco(A)$, the *generalized quotient* of $A$ w.r.t. $\rho$ is well-defined when there exists the least complete extension $\Phi_\rho(A)$ of $\rho$ for $\{ f^i \}_{i>0}$ relative to $\rho$; in such a case, the generalized quotient is defined to be $\Phi_\rho(A)$. ∎

It is worth noting that the above definition naturally extends the intuitive meaning of the concept of quotient. In fact, the abstract domain $\Phi_\rho(A)$ is the most abstract domain which is more concrete than the property $\rho$ and which is as good as $A$ for propagating the information through the semantic function $f$. In other words, $\Phi_\rho(A)$ encodes exactly the least amount of information of $A$ that is useful for computing the property $\rho$. Thus, this exactly formalizes the clear intuition behind the concept of quotient. As an immediate consequence of Corollary 3.1.11, we are then able to give the following theorem ensuring that, when the semantic function $f$ is continuous, generalized quotients always exist.

**Theorem 5.0.8** *If $f : A \overset{c}{\longmapsto} A$ then, for any $\rho \in uco(A)$, the generalized quotient $\Phi_\rho(A)$ exists.*

# Chapter 6

# Completeness and linear logic

Abstract

In this chapter, we study completeness problems arising when functions with multiple arguments are considered. We show that, in this context, propositional linear logic can be usefully exploited in order to characterize objects in complete abstract domains in a natural and elegant way, which allows us to inherit the basic features of linear logic and to simplify most of our results on abstract domain equations.

The results presented in this chapter are a joint work with R. Giacobazzi and F. Ranzato and appeared in [Giacobazzi et al. 1998b].

Completeness problems for functions with multiple arguments commonly arise when we face complex semantic definitions, obtained by composing simpler operations. Recall that, by Proposition 3.1.4 (3), completeness for a composed function $g \circ f$ holds whenever completeness for both functions $g$ and $f$ holds. This property suggests that, we can achieve completeness for a complex semantic function by inducing completeness for each simpler operation used in the semantic definition. A typical example of this situation is, for instance, in constraint-based programming languages, where the underlying structure is a constraint system endowed with a basic operation of conjunction. Conjunction of constraints is typically a binary operation which may play different rôles, depending on the specific language, e.g., consistency check, parameter passing, and synchronization mechanism ([Jaffar and Maher 1994; Saraswat et al. 1991]).

## 6.1 Basic completeness problems

In this chapter, we consider concrete interpretations $\langle C_1, C_2, D, \otimes \rangle$ consisting of three complete lattices $C_1$, $C_2$, and $D$, and a binary[1] operation $\otimes : C_1 \times C_2 \longmapsto D$.

[1] Clearly, a generalization to $n$-ary operations would be straightforward.

Therefore, an abstract interpretation is specified by a triple $\langle \rho, \mu, \eta \rangle$, with $\rho \in C_1$, $\mu \in C_2$, and $\eta \in D$. We show how to associate with each completeness problem an abstract domain equation, in order to build the complete domains. Our results are then instantiated to the case of quantales as concrete interpretations. Quantales are well-known algebraic structures which turn out to be the models of linear logic. Remarkably, in this logical setting, we inherit the basic features of linear logic, which allows us to considerably simplify most of our results on abstract domain equations. Moreover, we are able to represent the abstract objects in complete abstract interpretations by elegant linear logic-based formulas. Linear logic-based solutions find relevant applications in static program analysis and comparative semantics, for instance in logic programming, where unification — the prime computational step of any logic program semantics — turns out to be a binary operation in the quantale of substitutions.

The next example shows a simple completeness problem in abstract interpretation of list data structures. For the sake of simplicity, we consider lists of natural numbers, even if our discussion holds, more generally, for lists of objects of arbitrary type.

**Example 6.1.1** Let us consider the structure $\langle \wp(list(\mathbb{N})), \wp(\mathbb{N}), \wp(list(\mathbb{N})), [\cdot|\cdot] \rangle$, where $list(\mathbb{N})$ is the set of all finite lists of natural numbers and $[\cdot|\cdot] : \wp(\mathbb{N}) \times \wp(list(\mathbb{N})) \longmapsto \wp(list(\mathbb{N}))$ is a "collecting" version of concatenation defined as follows:

$$[\, N \mid L \,] \stackrel{\text{def}}{=} \{\, [\, n \mid l \,] \mid n \in N,\ l \in L \,\}$$

where $[\, \varnothing \mid L \,] = [\, N \mid \varnothing \,] = \varnothing$.

We say that a list is *irredundant* if it does not contain two occurrences of the same object. An abstract domain $\eta \in uco(\wp(list(\mathbb{N})))$ for detecting irredundant lists can be defined by

$$\eta \stackrel{\text{def}}{=} \{\, list(\mathbb{N}), Irr \,\}$$

where $Irr \subseteq list(\mathbb{N})$ is the set of irredundant lists over $\mathbb{N}$. Suppose we are interested in observing irredundant lists. Therefore, we fix $\eta$ as observation domain and we look for the most abstract solution $\langle \rho, \mu \rangle \in uco(\wp(\mathbb{N})) \times uco(\wp(list(\mathbb{N})))$ such that $\langle \rho, \mu, \eta \rangle$ is complete for $[\cdot|\cdot]$, i.e., for all $N \in \wp(\mathbb{N})$ and $L \in (\wp(list(\mathbb{N}))$ it holds

$$\eta([\, N \mid L \,]) = \eta([\, \rho(N) \mid \mu(L) \,])$$

The solution to this problem tells us which are the most abstract domains where we can use the operation of concatenation without approximation errors, when observing irredundant lists only. ∎

More formally, a basic completeness problem for binary operations arises when we fix an observation domain $\eta \in uco(D)$, and we look for two computation domains $\rho \in uco(C_1)$ and $\mu \in uco(C_2)$ which make the abstract interpretation $\langle \rho, \mu, \eta \rangle$ complete for $\otimes$, i.e., which satisfy the following equality[2].

$$\eta \circ \otimes = \eta \circ \otimes \circ \langle \rho, \mu \rangle \tag{6.1.1}$$

This is equivalent to saying that, for all $x \in C_1$ and $y \in C_2$, it holds that $\eta(x \otimes y) = \eta(\rho(x) \otimes \mu(y))$, or, by using G.i.'s, that $\alpha_{C_1,\rho}(x) \otimes^b \alpha_{C_2,\mu}(y) = \alpha_{D,\eta}(x \otimes y)$. Moreover, we are interested in achieving completeness by minimally modifying the original domains, and therefore the question naturally arises whether there exists the least complete extension of a given pair of computation domains $\langle A_1, A_2 \rangle$, i.e., the most abstract pair of domains $\langle \rho, \mu \rangle \in uco(C_1) \times uco(C_2)$, which is more concrete than the initial pair $\langle A_1, A_2 \rangle$ and simultaneously induces a complete abstract interpretation $\langle \rho, \mu, \eta \rangle$ with respect to $\otimes$.

**Example 6.1.2** We continue Example 6.1.1 and fix the domain $\eta$ for irredundancy as both computation and observation domain. We look for an abstraction on natural numbers which allows us to get a complete abstract interpretation. Therefore, we look for the most abstract domains $\beta \in uco(\wp(\mathbb{N}))$ such that, for any $N \in \wp(\mathbb{N})$ and $L \in (\wp(list(\mathbb{N}))$ it holds:

$$\eta([\, N \mid \eta(L)\,]) = \eta([\, \beta(N) \mid \eta(L)\,])$$

As another example, consider the following standard abstract domain $\rho \in uco(\wp(\mathbb{N}))$ for parity analysis.

$$\rho \stackrel{\text{def}}{=} \{\, \mathbb{N}, even, odd, \varnothing \,\}$$

We would like to combine the previous irredundancy analysis with a parity analysis and consider the following completeness problems: we look for the most abstract domains $\mu \in uco(\wp(list(\mathbb{N})))$ which induces a complete abstract interpretation when we observe irredundancy, yet computing on the parity abstraction. This means that, when computing, we can only see if the numbers are even or odd. This amounts to say that:

$$\eta([\, \rho(N) \mid L\,]) = \eta([\, \rho(N) \mid \mu(L)\,])$$

∎

More formally, in a *left completeness problem*, the abstraction $\mu \in uco(C_2)$ is thought of as fixed, and the solution to this problem exists when the following

[2]We prefer to use the operation $\otimes$ infix, instead of the common prefix notation, on order to better formulate its logical properties, by resembling the linear logic notion of tensor.

equality admits a solution, with $\rho \in uco(C_1)$ and $\eta \in uco(D)$.

$$\eta \circ \otimes \circ \langle id, \mu \rangle = \eta \circ \otimes \circ \langle \rho, \mu \rangle \tag{6.1.2}$$

Of course, *right completeness problems* are analogously formulated.

$$\eta \circ \otimes \circ \langle \rho, id \rangle = \eta \circ \otimes \circ \langle \rho, \mu \rangle \tag{6.1.3}$$

It turns out that a left completeness problem $\eta \circ \otimes \circ \langle id, \mu \rangle = \eta \circ \otimes \circ \langle \rho, \mu \rangle$ formulated in terms of G.i.'s amounts to require that for any $x \in C_1$ and $y \in \mu$, it holds that $\alpha_{C_1,\rho}(x) \otimes^b y = \alpha_{D,\eta}(x \otimes \gamma_{\mu,C_2}(y))$. Hence, a left completeness problem states that completeness for the abstractions $\rho \in uco(C_1)$, $\mu \in uco(C_2)$ and $\eta \in uco(D)$, holds when the operation $\otimes$ acts over $C_1 \times \mu$.

### 6.1.1 Absolute completeness

Absolute completeness problems clearly make sense only for concrete interpretations of type $\langle C, \otimes \rangle$. In the following example, we consider the simple quantale of lists presented in Example 2.3.3, with the operation "append".

**Example 6.1.3** Consider, the quantale $\langle \wp(list(\mathbb{N})), \otimes \rangle$ presented in Example 2.3.3, where $L \otimes M = \{ append(l, m) \mid l \in L,\ m \in M \}$, for any $L, M \in \wp(list(\mathbb{N}))$. We still consider the closure $\eta = \{ list(\mathbb{N}), Irr \}$ for detecting irredundant lists. Note that, $\eta$ is not absolute complete for $\otimes$. In fact, $\eta(\{ [1] \} \otimes \{ [2] \}) = \eta(\{ [1, 2] \}) = Irr$ while $\eta(\eta(\{ [1] \}) \otimes \eta(\{ [2] \})) = \eta(Irr \otimes Irr) = \wp(list(\mathbb{N}))$. We aim to find the most abstract domain $\rho$ which includes $\eta$ and is absolute complete for $\otimes$. ■

Formally, an *absolute completeness problem* arises when all the domains involved in the completeness problem are forced to be equal, as specified by the following equality.

$$\rho \circ \otimes = \rho \circ \otimes \circ \langle \rho, \rho \rangle \tag{6.1.4}$$

It is worth noting that this definition extends the absolute completeness problems for unary functions, presented in Section 3.1.2, to binary operations. In particular, observe that this problem always admits two trivial solutions, both the closures $\lambda x.x$ and $\lambda x.\top_C$. Therefore, when specifying absolute completeness problems, given an abstract domain $A \in uco(C)$, we are interested in finding a most abstract solution more concrete than $A$ (the so-called least absolute extension) or, dually, the most concrete solution more abstract than $A$ (the so-called greatest absolute kernel).

## 6.2 Solutions to completeness problems

Let $\langle C_1, C_2, D, \otimes \rangle$ be a concrete interpretation, where $C_1, C_2$ and $D$ are complete lattices and $\otimes : C_1 \times C_2 \longmapsto D$ is a binary, continuous operation. Let us first consider the left completeness problem given by Eq. (6.1.3). The idea is to reduce this problem to a problem of completeness of unary functions. Note that, in this formulation, the abstraction $\mu$ is fixed. Therefore, the concrete operator $\otimes : C_1 \times C_2 \longmapsto D$ can be thought of as the set of unary functions $F_\mu = \{\, \lambda x \in C_1 . x \otimes b \mid b \in \mu \,\}$. Hence, solving the left completeness problem is equivalent to solving the standard completeness problem $\langle \rho, \eta \rangle \in \Gamma(C_1, D, F_\mu)$. Moreover, since the $\otimes$ operation is argumentwise continuous, each function in $F_\mu$ is continuous as well. By Eq. (3.1.1), this is equivalent to saying that $\rho \sqsubseteq R_{F_\mu}(\eta)$. By definition, $R_{F_\mu}(\eta) = \curlywedge(\cup_{f \in F_\mu, y \in \eta} max(f^{-1}(\downarrow y)))$ and therefore, by exploiting the definition of $F_\mu$, this is equivalent to saying that $R_{F_\mu}(\eta) = \curlywedge(\cup_{a \in \mu, y \in \eta} max(\{\, x \in C_1 \mid x \otimes a \leq y \,\}))$. Of course, dual consideration holds when considering right completeness problems. Therefore, in order to characterize the solutions to such problems, we define two basic domain transformers $\stackrel{\curlywedge}{\rightarrow} : uco(C_1) \times uco(D) \longmapsto uco(C_2)$ and $\stackrel{\curlywedge}{\leftarrow} : uco(D) \times uco(C_2) \longmapsto uco(C_1)$ as follows, for any $\rho \in uco(C_1), \mu \in uco(C_2), \eta \in uco(D)$:

$$\rho \stackrel{\curlywedge}{\rightarrow} \eta \stackrel{\text{def}}{=} \curlywedge(\cup_{a \in \rho, y \in \eta} max\{\, z \in C_2 \mid b \otimes z \leq y \,\})$$

$$\eta \stackrel{\curlywedge}{\leftarrow} \mu \stackrel{\text{def}}{=} \curlywedge(\cup_{b \in \mu, y \in \eta} max\{\, x \in C_1 \mid x \otimes b \leq y \,\}).$$

**Proposition 6.2.1** *Let $\otimes : C_1 \times C_2 \stackrel{c}{\longmapsto} D$. $\stackrel{\curlywedge}{\rightarrow} : uco(C_1) \times uco(D) \longmapsto uco(C_2)$ is co-additive in its second argument, and $\stackrel{\curlywedge}{\leftarrow} : uco(D) \times uco(C_2) \longmapsto uco(C_1)$ is co-additive in its first argument.*

**Proof.** Let $\rho \in uco(C_1), \mu \in uco(C_2), \eta \in uco(D)$. First note that, by definition, $\rho \stackrel{\curlywedge}{\rightarrow} \eta = R_{F_\rho}(\eta)$ for $F_\rho = \{\, \lambda y \in C_2 . a \otimes y \mid a \in \rho \,\}$. Moreover, since the $\otimes$ operation is argumentwise continuous, each function in $F_\rho$ is continuous as well. Therefore, by Eq. (3.1.1), $\mu \sqsubseteq R_{F_\rho}(\eta)$ if and only if $L_{F_\rho}(\mu) \sqsubseteq \eta$ and thus $\langle D, R_{F_\rho}, C_2, L_{F_\rho} \rangle$ is an adjunction. Therefore $R_{F_\rho}$ is co-additive and, from $\rho \stackrel{\curlywedge}{\rightarrow} \eta = R_{F_\rho}(\eta)$, we have that $\stackrel{\curlywedge}{\rightarrow}$ is co-additive in its second argument. Dually for $\stackrel{\curlywedge}{\leftarrow}$. ■

Moreover, the following helpful property holds.

**Proposition 6.2.2** *Let $\otimes : C_1 \times C_2 \stackrel{c}{\longmapsto} D$, $\{\, \rho_i \,\}_{i \in I} \subseteq uco(C_1)$, $\{\, \mu_i \,\}_{i \in I} \subseteq uco(C_2)$ and $\eta \in uco(D)$.*

- *If* $\cup_{i \in I} \rho_i = \bigsqcap_{i \in I} \rho_i$ *then* $(\bigsqcap_{i \in I} \rho_i) \stackrel{\curlywedge}{\rightarrow} \eta = \bigsqcap_{i \in I} (\rho_i \stackrel{\curlywedge}{\rightarrow} \eta)$

- *If* $\cup_{i\in I}\mu_i = \bigsqcap_{i\in I}\mu_i$ *then* $\eta \overset{\curlywedge}{\leftarrow} (\bigsqcap_{i\in I}\mu_i) = \bigsqcap_{i\in I}(\eta \overset{\curlywedge}{\leftarrow} \rho_i)$.

**Proof.** Straightforward by definition of $\overset{\curlywedge}{\rightarrow}$ and $\overset{\curlywedge}{\leftarrow}$ ■

Therefore, by exploiting the previous definitions, we can state the following results.

**Theorem 6.2.3** *Let* $\otimes : C_1 \times C_2 \overset{c}{\longmapsto} D$, $\rho \in uco(C_1)$, $\mu \in uco(C_2)$ *and* $\eta \in uco(D)$. *The following are equivalent.*

1. $\eta \circ \otimes \circ \langle \rho, id\rangle = \eta \circ \otimes \circ \langle \rho, \mu\rangle$;
2. $\mu \sqsubseteq \rho \overset{\curlywedge}{\rightarrow} \eta$;
3. $\{\, y \in D \mid \rho \overset{\curlywedge}{\rightarrow} \{\, y\,\} \subseteq \mu \,\} \sqsubseteq \eta$.

**Proof.** By definition, $\eta \circ \otimes \circ \langle \rho, id\rangle = \eta \circ \otimes \circ \langle \rho, \mu\rangle$ if and only if $\langle \mu, \eta\rangle \in \Gamma(C, D, F_\rho)$ with $F_\rho = \{\, \lambda y \in C_2.a \otimes y \mid a \in \rho \,\}$. Since the $\otimes$ operation is argumentwise continuous, each function in $F_\rho$ is continuous as well. Therefore, by Eq. (3.1.1), $\langle \mu, \eta\rangle \in \Gamma(C, D, F_\rho)$ if and only if $L_{F_\rho}(\mu) \sqsubseteq \eta$ if and only if $\mu \sqsubseteq R_{F_\rho}(\eta)$. By definition, we have that $R_{F_\rho}(\eta) = \rho \overset{\curlywedge}{\rightarrow} \eta$ and $L_{F_\rho}(\mu) = \{\, y \in D \mid \rho \overset{\curlywedge}{\rightarrow} \{\, y\,\} \subseteq \mu \,\}$, which concludes the proof. ■

Thus, as an immediate consequence, right completeness problems admit the following solutions.

**Corollary 6.2.4** *Assume the same hypotheses of Theorem 6.2.3, and let* $A_2 \in uco(C_2)$ *and* $A \in uco(D)$.

- $A_2 \sqcap (\rho \overset{\curlywedge}{\rightarrow} \eta)$ *is the least (right) extension of* $A_2$ *with respect to* $\eta$ *and* $\rho$, *i.e., the most abstract solution of Eq.* (6.1.3), *which is more concrete than* $A_2$.
- $A \sqcup \{\, y \in D \mid \rho \overset{\curlywedge}{\rightarrow} \{\, y\,\} \subseteq \mu \,\}$ *is the greatest (right) kernel of* $A$ *with respect to* $\rho$ *and* $\mu$, *i.e., the most concrete solution of Eq.* (6.1.3), *which is more abstract than* $A$.

Of course, dual results can be given for left completeness problems.

**Theorem 6.2.5** *Let* $\otimes : C_1 \times C_2 \overset{c}{\longmapsto} D$, $\rho \in uco(C_1)$, $\mu \in uco(C_2)$ *and* $\eta \in uco(D)$. *The following are equivalent.*

1. $\eta \circ \otimes \circ \langle id, \mu\rangle = \eta \circ \otimes \circ \langle \rho, \mu\rangle$;
2. $\rho \sqsubseteq \eta \overset{\curlywedge}{\leftarrow} \mu$;
3. $\{\, y \in D \mid \{\, y\,\} \overset{\curlywedge}{\leftarrow} \mu \subseteq \rho \,\} \sqsubseteq \eta$.

**Proof.** By duality of Theorem 6.2.3 ■

**Corollary 6.2.6** *Assume the same hypotheses of Theorem 6.2.5, and let* $A_1 \in uco(C_1)$ *and* $A \in uco(D)$.

- $A_1 \sqcap (\eta \overset{\curlywedge}{\leftarrow} \mu)$ *is the least (left) extension of* $A_1$ *with respect to* $\eta$ *and* $\mu$*, i.e., the most abstract solution of Eq.* (6.1.2)*, which is more concrete than* $A_1$.
- $A \sqcup \{ y \in D \mid \{ y \} \overset{\curlywedge}{\leftarrow} \mu \subseteq \rho \}$ *is the greatest (left) kernel of* $A$ *with respect to* $\rho$ *and* $\mu$*, i.e., the most concrete solution of Eq.* (6.1.2)*, which is more abstract than* $A$.

**Example 6.2.7** The previous results enable us to constructively solve the problems presented in Example 6.1.2. Recall that, we look for the most abstract solutions $\beta$ and $\mu$ of the following equations, for any $N \in \wp(\mathbb{N})$ and $L \in (\wp(list(\mathbb{N}))$.

$$\eta([\, N \mid \eta(L)\,]) = \eta([\, \beta(N) \mid \eta(L)\,])$$

$$\eta([\, \rho(N) \mid L\,]) = \eta([\, \rho(N) \mid \mu(L)\,])$$

It is is clear that the operation $[\cdot|\cdot]$ is argumentwise additive, and therefore continuous. First, let us note that, for any $N \in \wp(\mathbb{N})$ and $L \in (\wp(list(\mathbb{N}))$, by additivity it holds:

$$max\{\, N \in \wp(\mathbb{N}) \mid [\, N \mid M\,] \subseteq L\,\} = \bigcup \{\, n \in \mathbb{N} \mid [\, \{\, n\,\} \mid M\,] \subseteq L\,\}$$

$$max\{\, M \in \wp(list(\mathbb{N})) \mid [\, N \mid M\,] \subseteq L\,\} = \bigcup \{\, m \in list(\mathbb{N}) \mid [\, N \mid \{\, m\,\}\,] \subseteq L\,\}$$

By Corollaries 6.2.6 and 6.2.4, we get the following solutions.

$$\begin{aligned}
\beta &= \{\, \mathbb{N}\,\} \sqcap (\eta \overset{\curlywedge}{\leftarrow} \eta) \\
&= \eta \overset{\curlywedge}{\leftarrow} \eta \\
&= \{\, \mathbb{N}, \varnothing\,\}
\end{aligned}$$

$$\begin{aligned}
\mu &= \{\, list(\mathbb{N})\,\} \sqcap (\rho \overset{\curlywedge}{\rightarrow} \eta) \\
&= \rho \overset{\curlywedge}{\rightarrow} \eta \\
&= \{\, \{\, []\,\}, Irr_{even}, Irr_{odd}, list(\mathbb{N})\,\}
\end{aligned}$$

where

$$Irr_{even} \stackrel{\text{def}}{=} \{\, l \in list(\mathbb{N}) \mid l \in Irr \text{ and } l \text{ does not contain even numbers}\,\}$$

$$Irr_{odd} \stackrel{\text{def}}{=} \{\, l \in list(\mathbb{N}) \mid l \in Irr \text{ and } l \text{ does not contain odd numbers}\,\}.$$

Thus, for the first problem, in order to get completeness, it is enough to check whether a given set of numbers is empty or not, while, for the second problem, we need to consider sets of irredundant lists which do not contain either even or odd numbers. ■

The above results for left and right completeness turn out to be the basic bricks for solving all the completeness problems. In fact, Eq. (6.1.1) can be thought of as a system of two equations for suitable instances of left and right completeness problems, as shown in the next proposition.

**Proposition 6.2.8** *Let* $\otimes : C_1 \times C_2 \overset{c}{\longmapsto} D$, $\rho \in uco(C_1)$, $\mu \in uco(C_2)$ *and* $\eta \in uco(D)$.

$\eta \circ \otimes = \eta \circ \otimes \circ \langle \rho, \mu \rangle$ *if and only if* $\eta \circ \otimes \circ \langle id, \mu \rangle = \eta \circ \otimes = \eta \circ \otimes \circ \langle \rho, id \rangle$.

**Proof.** Assume $\eta \circ \otimes = \eta \circ \otimes \circ \langle \rho, \mu \rangle$. By monotonicity of $\otimes$, we have that $\eta \circ \otimes \circ \langle id, \mu \rangle \leq \eta \circ \otimes \circ \langle \rho, \mu \rangle$ which, by hypothesis, is equal to $\eta \circ \otimes$. Therefore, $\eta \circ \otimes \circ \langle id, \mu \rangle = \eta \circ \otimes$. Dually, $\eta \circ \otimes \circ \langle \rho, id \rangle = \eta \circ \otimes$

Now assume $\eta \circ \otimes \circ \langle id, \mu \rangle = \eta \circ \otimes = \eta \circ \otimes \circ \langle \rho, id \rangle$. By $\eta \circ \otimes \circ \langle id, \mu \rangle = \eta \circ \otimes$ it follows that $\eta \circ \otimes \circ \langle \rho, \mu \rangle = \eta \circ \otimes \circ \langle \rho, id \rangle$ and therefore $\eta \circ \otimes \circ \langle \rho, \mu \rangle = \eta \circ \otimes$. ■

By using the previous proposition, we can now solve the completeness problem given by Eq. (6.1.1). The idea is to split the problem into two left and right completeness problems and to solve them separately.

**Theorem 6.2.9** *Let* $\otimes : C_1 \times C_2 \overset{c}{\longmapsto} D$, $\rho \in uco(C_1)$, $\mu \in uco(C_2)$ *and* $\eta \in uco(D)$. *The following are equivalent.*

- $\eta \circ \otimes = \eta \circ \otimes \circ \langle \rho, \mu \rangle$;
- $\rho \sqsubseteq \eta \overset{\curlywedge}{\leftarrow} C_2$ *and* $\mu \sqsubseteq C_1 \overset{\curlywedge}{\rightarrow} \eta$;
- $\{\, y \in D \mid \{\, y \,\} \overset{\curlywedge}{\leftarrow} \mu \subseteq \rho \,\} \sqsubseteq \eta$ *and* $\{\, y \in D \mid \rho \overset{\curlywedge}{\rightarrow} \{\, y \,\} \subseteq \mu \,\} \sqsubseteq \eta$.

**Proof.** By Prop.6.2.8, the problem boils down to solving two instances of left and right completeness problems where, respectively, $\rho = id$ and $\mu = id$. Therefore, the thesis follows by Theorems 6.2.5 and 6.2.3. ■

As a straight consequence, we get the following result, which characterizes the least extension and the greatest kernel.

**Corollary 6.2.10** *Assume the same hypotheses of Theorem 6.2.9 and let* $A_1 \in uco(C_1)$, $A_2 \in uco(C_2)$ *and* $A \in uco(D)$.

- $\langle A_1 \sqcap (\eta \overset{\curlywedge}{\leftarrow} C_2), A_2 \sqcap (C_2 \overset{\curlywedge}{\rightarrow} \eta)\rangle$ *is the least extension of* $\langle A_1, A_2\rangle$ *with respect to* $\eta$, *i.e., the most abstract solution of Eq.* (6.1.1), *which is more concrete than* $\langle A_1, A_2\rangle$.
- $A \sqcup \{\, y \in D \mid \{\, y\,\} \overset{\curlywedge}{\leftarrow} C_2 \subseteq C_2 \,\} \sqcup \{\, y \in D \mid C_1 \overset{\curlywedge}{\rightarrow} \{\, y\,\} \subseteq C_2 \,\}$ *is the greatest kernel of* $A$ *with respect to* $\rho$ *and* $\mu$, *i.e., the most concrete solution of Eq.* (6.1.1), *which is more abstract than* $A$.

**Example 6.2.11** We can now solve the completeness problem presented in Example 6.1.1. The solution is the most abstract pair of domains $\langle \rho, \mu\rangle$ for which abstract concatenation in $\rho$ and $\mu$ turns out to be complete when observing irredundancy, i.e., for all $N \in \wp(\mathbb{N})$ and $L \in (\wp(list(\mathbb{N}))$, it holds

$$\eta([\, N \mid L\,]) = \eta([\, \rho(N) \mid \mu(L)\,])$$

Given $N \subseteq \mathbb{N}$, let us denote by $list(N)$ the set of all (finite) lists of natural numbers contained in $N$. By Corollary 6.2.10, the solution $\langle \rho, \mu\rangle$ is as follows.

$$\begin{array}{rcl} \rho & = & \{\, \mathbb{N}\,\} \sqcap (\eta \overset{\curlywedge}{\leftarrow} \wp(list(\mathbb{N}))) \\ & = & \eta \overset{\curlywedge}{\leftarrow} \wp(list(\mathbb{N})) \\ & = & \wp(\mathbb{N}) \end{array}$$

The last step holds since, for all $n \in \mathbb{N}$, $max\{\, N \in \wp(\mathbb{N}) \mid [\, N \mid \{\, [n]\,\}\,] \subseteq Irr\,\} = \mathbb{N} \smallsetminus \{\, n\,\}$, and therefore we obtain the whole set $\wp(\mathbb{N})$.

$$\begin{array}{rcl} \mu & = & \{\, list(\mathbb{N})\,\} \sqcap (\wp(\mathbb{N}) \overset{\curlywedge}{\rightarrow} \eta) \\ & = & \wp(\mathbb{N}) \overset{\curlywedge}{\rightarrow} \eta \\ & = & \curlywedge(\{\, \bigcup\{\, m \in list(\mathbb{N}) \mid [\, N \mid \{\, m\,\}\,] \subseteq Irr\,\}\,\}) \\ & = & \{\, list(N) \cap Irr \mid N \subseteq \mathbb{N}\,\} \end{array}$$

Thus, in order to get completeness for concatenation when observing irredundancy, $\rho$ must coincide with the concrete domain $\wp(\mathbb{N})$, while it suffices that $\mu$ contains all the sets of irredundant lists which are closed by permutation of their objects. ■

### 6.2.1 Solutions to absolute completeness problems

It is now an easy task to provide a characterization of the solutions to the absolute completeness problems, by exploiting Theorem 6.2.9 with $\rho = \mu = \eta$.

**Theorem 6.2.12** *Let* $\otimes : C \times C \overset{c}{\longmapsto} C$ *and* $\rho \in uco(C)$. *The following are equivalent.*

- $\rho \circ \otimes = \rho \circ \otimes \circ \langle \rho, \rho\rangle$;

- $\rho \sqsubseteq \rho \overset{\curlywedge}{\leftarrow} C$ *and* $\rho \sqsubseteq C \overset{\curlywedge}{\rightarrow} \rho$*;*

- $\{\, y \in C \mid \{\, y \,\} \overset{\curlywedge}{\leftarrow} \rho \subseteq \rho \,\} \sqsubseteq \rho$ *and* $\{\, y \in C \mid \rho \overset{\curlywedge}{\rightarrow} \{\, y \,\} \subseteq \rho \,\} \sqsubseteq \rho$*.*

**Proof.** By applying Theorem 6.2.9 with $C_1 = C_2 = D$ and $\rho = \mu = \eta$, we obtain the result. ■

The previous result states that, the solutions to the absolute completeness problem are exactly the solutions of the following (recursive) abstract domain equation.

$$X = (X \overset{\curlywedge}{\leftarrow} C) \sqcap (C \overset{\curlywedge}{\rightarrow} X) \tag{6.2.1}$$

As a consequence, the least (absolute) complete extension of a given domain $A \in uco(C)$ is given by the greatest fixpoint of $\lambda X.A \sqcap (X \overset{\curlywedge}{\leftarrow} C) \sqcap (C \overset{\curlywedge}{\rightarrow} X)$. Thus, finding the least absolute complete extension of a given domain $A$ is equivalent to finding the greatest fixpoint of the following (recursive) abstract domain equation.

$$X = A \sqcap (X \overset{\curlywedge}{\leftarrow} C) \sqcap (C \overset{\curlywedge}{\rightarrow} X) \tag{6.2.2}$$

Dual considerations hold for greatest absolute kernels. Moreover, by Proposition 6.2.1, we know that both $\lambda X.X \overset{\curlywedge}{\leftarrow} C$ and $\lambda X.C \overset{\curlywedge}{\rightarrow} X$ are co-additive, and therefore the operator $\lambda X.A \sqcap (X \overset{\curlywedge}{\leftarrow} C) \sqcap (C \overset{\curlywedge}{\rightarrow} X)$ is co-additive as well. Thus, the least absolute extension, which is the greatest fixpoint of the above operator, is given by the (dual) $\omega$-th power of $f$.

**Proposition 6.2.13** *Assume the same hypothesis of Theorem 6.2.12 and let $A \in uco(C)$.*

- $\bigsqcap_{n<\omega}(\lambda X.A \sqcap (X \overset{\curlywedge}{\leftarrow} C) \sqcap (C \overset{\curlywedge}{\rightarrow} X))^n(\top_{uco(C)})$ *is the least absolute extension of $A$, i.e., the most abstract solution of Eq.* (6.1.4)*, which is more concrete than $A$.*

- $lfp(\lambda X.A \sqcup \{\, y \in C \mid \{\, y \,\} \overset{\curlywedge}{\leftarrow} X \subseteq X \,\} \sqcup \{\, y \in C \mid X \overset{\curlywedge}{\rightarrow} \{\, y \,\} \subseteq X \,\})$ *is the greatest absolute kernel of $A$, i.e., the most concrete solution of Eq.* (6.1.4)*, which is more abstract than $A$.*

**Proof.** By Proposition 6.2.1, both the operators $\lambda X.X \overset{\curlywedge}{\leftarrow} C$ and $\lambda X.C \overset{\curlywedge}{\rightarrow} X$ are co-additive. Therefore, $\lambda X.A \sqcap (X \overset{\curlywedge}{\leftarrow} C) \sqcap (C \overset{\curlywedge}{\rightarrow} X)$ is co-additive as well and the result follows by Theorem 6.2.12. ■

## 6.3 Disjunctive completeness

Absolute completeness represents an upper bound to the precision of an abstract domain, for a given operation. In fact, when an abstract domain $\rho$ is absolute complete, then the abstraction $\rho$ can be removed in any computation involving $\otimes$. This is a very powerful property, which ensures us that the results of abstract computations are not affected by approximation errors. Therefore, it is easy to observe that, except for trivial properties (i.e., decidable properties), solutions to absolute completeness problems always lead to infinite domains, and then to nondecidable analyses. This explains why, although example of absolute complete extensions can be easily found in comparative semantics, (e.g., [Cousot 1997a; Cousot and Cousot 1992b; Giacobazzi 1996; Comini and Meo 1999]), it is rarely applied in static program analysis.

The aim of this section is to present a novel kind of completeness problem, obtained by considering that, in many cases of abstract computations with binary operations, one of the two arguments is an abstract object. Following the previous approach for absolute completeness, we want to formalize the intuition that an abstraction $\rho$ can be removed in any computation involving $\otimes$, providing that the computation involves at least one abstract object each time $\otimes$ is applied. For instance, in logic programming this is the case when $\otimes$ is unification and the semantics is made by unifying concrete and abstract objects, which respectively come from program's clauses and the objects in the current interpretation (see [Barbuti et al. 1993; Codish et al. 1994]).

Given an abstract domain $\rho \in uco(C)$, a *disjunctive completeness problem* is formulated as follows:

$$\text{if either } x \in \rho \text{ or } y \in \rho \text{ then } \rho(\rho(x) \otimes \rho(y)) = \rho(x \otimes y).$$

This is equivalent to requiring that $\rho$ is a solution of the following equalities.

$$\rho \circ \otimes \circ \langle \rho, \rho \rangle = \rho \circ \otimes \circ \langle \rho, id \rangle = \rho \circ \otimes \circ \langle id, \rho \rangle \qquad (6.3.1)$$

Therefore, given an abstract domain $A \in uco(C)$, we look for the *least disjunctive complete extension* of $A$, i.e., the most abstract domain $\rho$ which is more concrete than $A$ and satisfy Eq. (6.3.1), and dually, for the *greatest disjunctive complete kernel* of $A$. In Chapter 8, we shall see how these notions can be usefully exploited to characterize condensing domains in logic programming analyses.

Disjunctive completeness problems can be solved by using the same method of absolute completeness, i.e., by resorting to a recursive abstract domain equation. The next theorem shows a (recursive) characterization of the solutions.

**Theorem 6.3.1** *Let* $\otimes : C \times C \overset{c}{\longmapsto} C$ *and* $\rho \in uco(C)$. *The following are equivalent.*

- $\rho \circ \otimes \circ \langle \rho, \rho \rangle = \rho \circ \otimes \circ \langle \rho, id \rangle = \rho \circ \otimes \circ \langle id, \rho \rangle$;
- $\rho \sqsubseteq \rho \overset{\curlywedge}{\leftarrow} \rho$ *and* $\rho \sqsubseteq \rho \overset{\curlywedge}{\rightarrow} \rho$;
- $\{\, y \in D \mid \{\, y \,\} \overset{\curlywedge}{\leftarrow} \rho \subseteq \rho \,\} \sqsubseteq \rho$ *and* $\{\, y \in D \mid \rho \overset{\curlywedge}{\rightarrow} \{\, y \,\} \subseteq \rho \,\} \sqsubseteq \rho$.

**Proof.** Solving Eq. (6.3.1) is equivalent to solving both the equalities $\rho \circ \otimes \circ \langle \rho, id \rangle = \rho \circ \otimes \circ \langle \rho, \rho \rangle$ and $\rho \circ \otimes \circ \langle id, \rho \rangle = \rho \circ \otimes \circ \langle \rho, \rho \rangle$. The first equation is an instance of a right completeness problem and the solutions are given by Theorem 6.2.3, applied with $C_1 = C_2 = D$ and $\rho = \mu = \eta$. Dually for the latter equality. ■

Therefore, the solutions to disjunctive completeness problem are exactly the solutions of the following (recursive) abstract domain equation.

$$X = (X \overset{\curlywedge}{\leftarrow} X) \sqcap (X \overset{\curlywedge}{\rightarrow} X) \tag{6.3.2}$$

In least disjunctive complete extensions, given $A \in uco(C)$, we are interested in the most abstract domain $\rho$ which is more concrete than $A$ and satisfies Eq. (6.3.1). As for absolute completeness, such a domain $\rho$ can be obtained as the greatest fixpoint of the following recursive domain equation.

$$X = A \sqcap X \overset{\curlywedge}{\leftarrow} X \sqcap X \overset{\curlywedge}{\rightarrow} X \tag{6.3.3}$$

The next corollary characterizes least disjunctive complete extensions and greatest disjunctive complete kernels, by exploiting Theorem 6.3.1. It is worth noting that, differently from least absolute extensions, the operator $\lambda X.A \sqcap (X \overset{\curlywedge}{\leftarrow} X) \sqcap (X \overset{\curlywedge}{\rightarrow} X)$ is not, in general, co-continuous. A counterexample will be presented in Example 9.4.3.

**Corollary 6.3.2** *Assume the same hypothesis of Theorem 6.3.1 and let* $A \in uco(C)$.

- $gfp(\lambda X.A \sqcap (X \overset{\curlywedge}{\leftarrow} X) \sqcap (X \overset{\curlywedge}{\rightarrow} X)$ *is the least disjunctive complete extension of* $A$, *i.e., the most abstract solution of Eq.* (6.3.1), *which is more concrete than* $A$.
- $lfp(\lambda X.A \sqcup \{\, y \in D \mid \{\, y \,\} \overset{\curlywedge}{\leftarrow} X \subseteq X \,\} \sqcup \{\, y \in D \mid X \overset{\curlywedge}{\rightarrow} \{\, y \,\} \subseteq X \,\}$ *is the greatest disjunctive complete kernel of* $A$, *i.e., the most concrete solution of Eq.* (6.3.1), *which is more abstract than* $A$.

## 6.4 Solutions in quantales

In this section, we instantiate the previous results on absolute completeness to the case of quantales as concrete interpretations. When $C_1 = C_2 = D$, and $\otimes$ is additive and associative, our structures boil down to quantales. The main feature of quantales is that they support a notion of left and right linear implication between domain's objects: given $a, b \in C_1$, there exists a unique most abstract object $a \multimap b \in C_1$ which, when combined by $\otimes$ with $a$, gives a result less than or equal to $b$. In other words, the following right modus ponens law $a \otimes (a \multimap b) \leq b$ holds. Analogous left implicational objects exist for a corresponding left modus ponens law.

When solving completeness problems in a setting where concrete interpretations are quantales, we inherit all the basic features of linear logic. As a consequence, least complete extensions can be explicitly characterized in terms of linear implications between domain objects, and more importantly, the operator for computing least absolute extensions (see Prop. 6.2.13) converges in finitely many steps, actually in three steps.

Let $\langle C, \otimes \rangle$ be a quantale playing the rôle of concrete interpretation. By Proposition 2.3.4 (i) and (ii), for any $a, b \in C$ we have that

$$max\{\, x \in C \mid a \otimes x \leq b \,\} = \bigvee \{\, x \in C \mid a \otimes x \leq b \,\} = a \multimap b$$

$$max\{\, x \in C \mid x \otimes a \leq b \,\} = \bigvee \{\, x \in C \mid x \otimes a \leq b \,\} = b \multimapinv a.$$

Therefore, the operators $\stackrel{\curlywedge}{\multimap}$ and $\stackrel{\curlywedge}{\multimapinv}$ can be rewritten by simply lifting left and right linear implications to abstract domains as follows, for any $\rho, \mu \in uco(C)$:

$$\rho \stackrel{\curlywedge}{\multimap} \mu = \curlywedge(\{\, a \multimap b \in C \mid a \in \rho, b \in \mu \,\})$$

$$\mu \stackrel{\curlywedge}{\multimapinv} \rho = \curlywedge(\{\, b \multimapinv a \in C \mid a \in \rho, b \in \mu \,\}).$$

Thus, the solutions to absolute completeness problems can be all characterized in terms of linear implications among domain's objects. Before proving our main result on least absolute extensions in quantales, let us first show the following helpful property. It states that, two successive applications of the operators $\stackrel{\curlywedge}{\multimap}$ and $\stackrel{\curlywedge}{\multimapinv}$ can be exchanged, without affecting the resulting domain.

**Proposition 6.4.1** *Let $\langle C, \otimes \rangle$ be a quantale and $A, B, C \in uco(L)$. $A \stackrel{\curlywedge}{\multimap} (C \stackrel{\curlywedge}{\multimapinv} B) = ((A \stackrel{\curlywedge}{\multimap} C) \stackrel{\curlywedge}{\multimapinv} B)$*

**Proof.** It suffices to note that, for $a_i \in A$, $b_i \in B$ and $c_i \in C$ it holds that $a \multimap \bigwedge_{i \in I}(c_i \multimapinv b_i) = \bigwedge_{i \in I}(a \multimap (c_i \multimapinv b_i)) = \bigwedge_{i \in I}((a \multimap c_i) \multimapinv b_i)$. ■

By exploiting the previous characterization, we are now able to prove the convergence of the least absolute completeness operator, as shown in the next theorem. Moreover, as a consequence of associativity of $\otimes$, we can prove that it converges in exactly three steps.

**Theorem 6.4.2** *Let $\langle C, \otimes\rangle$ be a quantale and $A \in uco(L)$. The most abstract solution to Eq.* (6.2.2) *is given by*

$$A \sqcap (C \overset{\curlywedge}{\rightarrowtriangle} A) \sqcap (A \overset{\curlywedge}{\leftarrowtriangle} C) \sqcap ((C \overset{\curlywedge}{\rightarrowtriangle} A) \overset{\curlywedge}{\leftarrowtriangle} C).$$

**Proof.** Let $B = A \sqcap C \overset{\curlywedge}{\rightarrowtriangle} A \sqcap A \overset{\curlywedge}{\leftarrowtriangle} C \sqcap (C \overset{\curlywedge}{\rightarrowtriangle} A) \overset{\curlywedge}{\leftarrowtriangle} C$. By Theorem 6.2.12, $B$ is complete if $l \rightarrowtriangle b, b \leftarrowtriangle l \in B$, for all $l \in C, b \in B$. Let us first prove that $l \rightarrowtriangle b \in B$. Since $\rightarrowtriangle$ is co-additive, we only need to show that it holds for all generators of $B$.

- Let $a \in A$. Clearly $l \rightarrowtriangle a \in B$ by construction.
- Let $c \rightarrowtriangle a$ be a generator of $C \overset{\curlywedge}{\rightarrowtriangle} A$. By Prop. 2.3.4 (vii), $l \rightarrowtriangle (c \rightarrowtriangle a) = (c \otimes l) \rightarrowtriangle a$ which belongs to $C \overset{\curlywedge}{\rightarrowtriangle} A$ and therefore to $B$.
- Let $a \leftarrowtriangle c$ be a generator of $A \overset{\curlywedge}{\leftarrowtriangle} C$. By Prop. 2.3.4 (ix), $l \rightarrowtriangle (a \leftarrowtriangle c) = (l \rightarrowtriangle a) \leftarrowtriangle c$ which belongs to $(C \overset{\curlywedge}{\rightarrowtriangle} A) \overset{\curlywedge}{\leftarrowtriangle} C$ and therefore to $B$.
- Let $(\bigwedge_{i\in I}(c_i \rightarrowtriangle a_i)) \leftarrowtriangle d$ be a generator of $(C \overset{\curlywedge}{\rightarrowtriangle} A) \overset{\curlywedge}{\leftarrowtriangle} C$. Then we have that

$$\begin{array}{lll} & l \rightarrowtriangle ((\bigwedge_{i\in I}(c_i \rightarrowtriangle a_i)) \leftarrowtriangle d) & \\ = & (l \rightarrowtriangle ((\bigwedge_{i\in I}(c_i \rightarrowtriangle a_i)) \leftarrowtriangle d & [\text{by Prop. 2.3.4 (ix)}] \\ = & ((\bigwedge_{i\in I}(l \rightarrowtriangle (c_i \rightarrowtriangle a_i)) \leftarrowtriangle d & [\text{by Prop. 2.3.4 (iii)}] \\ = & (\bigwedge_{i\in I}(c_i \otimes l) \rightarrowtriangle a_i) \leftarrowtriangle d & [\text{by Prop. 2.3.4 (vii)}] \end{array}$$

  Note that $\bigwedge_{i\in I}(c_i \otimes l) \rightarrowtriangle a_i \in C \overset{\curlywedge}{\rightarrowtriangle} A$ and therefore $(\bigwedge_{i\in I}(c_i \otimes l) \rightarrowtriangle a_i) \leftarrowtriangle d$ belongs to $(C \overset{\curlywedge}{\rightarrowtriangle} A) \overset{\curlywedge}{\leftarrowtriangle} C$ by construction.

Moreover, by Proposition 6.4.1, $((C \overset{\curlywedge}{\rightarrowtriangle} A) \overset{\curlywedge}{\leftarrowtriangle} C) = C \overset{\curlywedge}{\rightarrowtriangle} (A \overset{\curlywedge}{\leftarrowtriangle} C)$ and therefore $B = A \sqcap (C \overset{\curlywedge}{\rightarrowtriangle} A) \sqcap (A \overset{\curlywedge}{\leftarrowtriangle} C) \sqcap ((C \overset{\curlywedge}{\rightarrowtriangle} A) \overset{\curlywedge}{\leftarrowtriangle} C) \sqcap (C \overset{\curlywedge}{\rightarrowtriangle} (A \overset{\curlywedge}{\leftarrowtriangle} C))$ and thus, by duality, $b \leftarrowtriangle l \in B$.

Note that, by Proposition 6.2.13, $B$ is clearly the most abstract domain which is complete for $\otimes$ and contains $A$. ∎

**Example 6.4.3** Since $\langle \wp(list(\mathbb{N})), \otimes\rangle$ is a quantale (see Example 2.3.3), we are now able to solve the problem presented in Example 6.1.3 by exploiting Theorem 6.4.2. Recall that we look for the least absolute extension of $\eta = \{\, list(\mathbb{N}), Irr \,\}$ for list concatenation, i.e., the most abstract solution $\rho$ of $\rho \circ \otimes = \rho \circ \otimes \circ \langle \rho, \rho \rangle$, which is more concrete than $\eta$. For any $L \in \wp(list(\mathbb{N}))$, it holds:

$$L \multimap list(\mathbb{N}) = list(\mathbb{N}) \multimapinv L = list(\mathbb{N})$$

Moreover, observe that, whenever $L = list(N) \cap Irr$ for some $N \subseteq \mathbb{N}$, then $L \multimap Irr = Irr \multimapinv L = list(\mathbb{N} \smallsetminus N) \cap Irr$. By exploiting this property, it follows that:

$$\begin{array}{rcl}
\{\, L \multimap Irr \mid L \in \wp(list(\mathbb{N}))\,\} & = & \{\, Irr \multimapinv L \mid L \in \wp(list(\mathbb{N}))\,\} \\
& = & \{\, list(N) \cap Irr \mid N \subseteq \mathbb{N}\,\} \\[1em]
\{\, L \multimap (list(N) \cap Irr) \mid N \subseteq \mathbb{N}\,\} & = & \{\, (list(N) \cap Irr) \multimapinv L \mid N \subseteq \mathbb{N}\,\} \\
& = & \{\, list(N) \cap Irr \mid N \subseteq \mathbb{N}\,\}
\end{array}$$

Therefore, the solution is $\{\, list(N) \cap Irr \mid N \subseteq \mathbb{N}\,\}$. Note that, instead, the solution to the disjunctive completeness problem is simply $\eta \cup \{\,[]\,\}$, where $\{\,[]\,\}$ is obtained as $Irr \multimap Irr$. ■

Note that, when we deal with unital and commutative quantales, the above solutions to absolute completeness problems given by Theorem 6.4.2 can be significantly simplified. Moreover, it is not hard to check that for all $a \in C$, $\lambda c.(c \multimap a) \multimap a \in uco(C)$. This turns out to be the key observation in order to give a more compact form to solutions of absolute completeness problems on unital commutative quantales.

**Theorem 6.4.4** *Let $\langle C, \otimes \rangle$ be a unital commutative quantale and $A \in uco(C)$.*

1. *$C \overset{\curlywedge}{\multimap} A$ is the most abstract solution of Eq.* (6.2.2)

2. *$C \overset{\curlywedge}{\multimap} A = \lambda c \in C. \bigwedge_{a \in A}(c \multimap a) \multimap a$*

**Proof.**

1. By definition of commutativity, it holds that $C \overset{\curlywedge}{\multimap} A = A \overset{\curlywedge}{\multimapinv} C$. Therefore, $(C \overset{\curlywedge}{\multimap} A) \overset{\curlywedge}{\multimapinv} C = C \overset{\curlywedge}{\multimap} (C \overset{\curlywedge}{\multimap} A)$. Let us now prove that $C \overset{\curlywedge}{\multimap} (C \overset{\curlywedge}{\multimap} A) = C \overset{\curlywedge}{\multimap} A$. Let $c_i, d \in C$ and $a_i \in A$, for $i \in I$. We have that $d \multimap (\bigwedge_{i \in I}(c_i \multimap a_i)) = \bigwedge_{i \in I}(d \multimap (c_i \multimap a_i)) = \bigwedge_{i \in I}((c_i \otimes d) \multimap a_i) \in C \overset{\curlywedge}{\multimap} A$. Therefore, by Theorem 6.4.2, $C \overset{\curlywedge}{\multimap} A$ is the most abstract solution of Eq. (6.2.1).

2. Let us denote by $\rho$ the closure associated with $C \overset{\curlywedge}{\rightarrowtriangle} A$ and let $c \in C$. The following equalities hold.

$$
\begin{aligned}
\rho(c) &= \textstyle\bigwedge_{d \in C, a \in A} \{\, d \rightarrowtriangle a \mid c \leq d \rightarrowtriangle a \,\} \\
&= \textstyle\bigwedge_{d \in C, a \in A} \{\, d \rightarrowtriangle a \mid d \otimes c \leq a \,\} \\
&= \textstyle\bigwedge_{d \in C, a \in A} \{\, d \rightarrowtriangle a \mid c \otimes d \leq a \,\} && \text{[by commutativity]} \\
&= \textstyle\bigwedge_{d \in C, a \in A} \{\, d \rightarrowtriangle a \mid d \leq c \rightarrowtriangle a \,\} \\
&= \textstyle\bigwedge_{d \in C, a \in A} \{\, d \rightarrowtriangle a \mid d \leq ((c \rightarrowtriangle a) \rightarrowtriangle a) \rightarrowtriangle a \,\} && \text{[Prop. 2.3.5 (xiii)]} \\
&= \textstyle\bigwedge_{d \in C, a \in A} \{\, d \rightarrowtriangle a \mid ((c \rightarrowtriangle a) \rightarrowtriangle a) \otimes d \leq a \,\} \\
&= \textstyle\bigwedge_{d \in C, a \in A} \{\, d \rightarrowtriangle a \mid d \otimes ((c \rightarrowtriangle a) \rightarrowtriangle a) \leq a \,\} && \text{[by commutativity]} \\
&= \textstyle\bigwedge_{d \in C, a \in A} \{\, d \rightarrowtriangle a \mid (c \rightarrowtriangle a) \rightarrowtriangle a \leq d \rightarrowtriangle a \,\} \\
&= \textstyle\bigwedge_{a \in A} (c \rightarrowtriangle a) \rightarrowtriangle a
\end{aligned}
$$

∎

# Chapter 7

# Optimal semantics for logic programs

Abstract

In this chapter, we construct optimal semantics for logic programs. An optimal semantics can be thought of as a semantics which is neither too concrete nor too abstract for characterizing a given property of interest. By applying our results on completeness over quantales, we determine the least fixpoint extension of a generic abstract domain for a standard bottom-up semantics observing computed answer substitutions. This general result is then instantiated to the relevant case of groundness analysis.

The results presented in this chapter are a joint work with R. Giacobazzi and F. Ranzato and appeared in [Giacobazzi et al. 1998b].

## 7.1 From completeness to fixpoint completeness

The problem of determining "optimal" semantics for logic program analysis was first raised in [Giacobazzi 1996]. An optimal semantics can be thought of as a semantics which is neither too concrete nor too abstract for characterizing a given property of interest. Following the notation introduced in Section 3.1.2, a concrete interpretation is specified by a concrete domain $C$ and a concrete function $f_P : C \longmapsto C$, and the semantics $[\![P]\!]$ of a program $P$ is defined as $\mathit{lfp}(f_P)$. An abstract interpretation is given by an abstract domain $A$ and an abstract function $f_P^A : A \longmapsto A$. The abstract semantics $[\![P]\!]^A$ of a program $P$ is then defined as $[\![P]\!]^A = \mathit{lfp}(f_P^A)$. Note that, in this framework, we assume that concrete semantics are given as least fixpoint of some semantic operator and, more importantly, that this condition holds for abstract semantics too. This requirement ensures us that, even in the abstract context, abstract semantics can be constructively characterized. Moreover, we assume to deal with abstract functions which are induced by the concrete one, i.e., $f_P^A$ is the best correct approximation of $f_P$ in $A$. As a consequence, whenever the concrete semantic function $f_P$ is a recursive function (i.e., it is effectively given), then the abstract semantic function is recursive too, providing the abstraction function to be effectively defined. This ensures us that optimal semantics can be constructively characterized and, more importantly, that they can be effectively computed. Thus, given an abstract domain $\rho$ representing the property we are interested in, we look for the most abstract domain $A \in uco(C)$ such that the property $\rho$ is observable in the abstract semantics induced by $A$ without approximation errors, i.e., $\rho(lfp(f_P)) = \rho(lfp(\alpha_A \circ f_P))$. Therefore, the abstract domain $A$ exactly summarizes the least amount of information that we need in order to obtain precise abstract computations.

We proceed as follows: we first determine the least absolute extension of any logic program property w.r.t. a bottom-up semantics characterizing computed answer substitutions, which turns out to be equivalent to the well-known s-semantics [Falaschi et al. 1989]. Such extensions can be obtained by solving an absolute completeness problem relatively to the operation of unification, which is the fundamental computational step in any logic program semantics. Finally, we show that, under nonrestrictive hypotheses, least complete extension and least fixpoint extension do coincide.

## 7.2 Logic programming - Notation

Let $\mathcal{V}$ be an infinite, recursively enumerable (r.e.) set of variables, $\Sigma$ be a set of function symbols and $\pi$ be a set of predicate symbols, defining a r.e. first-order language $\mathcal{L}$. *Term* denotes the set of terms of $\mathcal{L}$. A *substitution* is a finite mappings from $\mathcal{V}$ to terms in $\mathcal{L}$. If $s$ is any syntactic object and $\sigma$ and $\theta$ are substitutions, then $\sigma(s)$ denotes the application of $\sigma$ to $s$, $\sigma \circ \theta$ denotes the standard composition of $\theta$ and $\sigma$ (i.e., $\sigma \circ \theta = \lambda x.\sigma(\theta(x))$), $vars(s)$ denotes the set of variables occurring in $s$, $dom(\sigma) \stackrel{\text{def}}{=} \{ v \in \mathcal{V} \mid \sigma(v) \neq v \}$, and $rng(\sigma) \stackrel{\text{def}}{=} \cup \{ vars(\sigma(x)) \mid x \in dom(\sigma) \}$. A term $t$ is ground if $vars(t) = \varnothing$.

The set of *idempotent* substitutions modulo renaming (i.e., $\theta \sim \sigma$ if and only if there exist $\beta$ and $\delta$ such that $\theta = \sigma \circ \beta$ and $\sigma = \theta \circ \delta$) on $\mathcal{L}$ is denoted by *Sub*. Both *Sub* and *Term* are partially ordered by instantiation, denoted by $\preceq$, i.e., $a \preceq b$ iff $\exists \theta \in Sub.\ a = b\theta$. By adding to *Sub* an extra object $\tau$ as least element, one gets a complete lattice $\langle Sub^{\tau}, \preceq, \vee, \wedge, \epsilon, \tau \rangle$, where $\vee$ is the least general anti-instance, $\wedge$ is the standard unification and $\epsilon$ is the empty substitution (see [Palamidessi 1990; Lassez et al. 1988] for more details). The set of most general atoms is given by $GAtom \stackrel{\text{def}}{=} \{ p(\bar{X}) \mid p \in \pi \}$, where $\bar{X}$ is a tuple of distinct variables.

We consider logic programs in normalized form, that is, a generic Horn clause is $p(\bar{X}) :- c, q_1(\bar{X}_1), \ldots, q_n(\bar{X}_n)$, where all the tuples of variables are distinct. The constraint $c$ is intended as a (finite) set of term equalities for variable binding (e.g., $X = f(Y), Z = g(Y)$). By abusing notation, we let $c$ denote also the set of most general unifiers which solve it.

## 7.3 $T_P$-completeness

Our basic semantic structure is the unital, commutative quantale $\langle \wp(Sub), \otimes \rangle$, where $\langle \wp(Sub), \subseteq \rangle$ is a complete lattice, $\otimes : \wp(Sub) \times \wp(Sub) \longmapsto \wp(Sub)$ is the obvious lifting of unification $\wedge$ to sets of substitutions, defined as:

$$X \otimes Y \stackrel{\text{def}}{=} \{ x \wedge y \mid x \wedge y \neq \tau, x \in X,\ y \in Y \}.$$

It is immediate to check that $\langle \wp(Sub), \otimes \rangle$ actually is a unital, commutative quantale, and $\{ \epsilon \} \in \wp(Sub)$ is the unit of $\otimes$. In the following, we will slightly abuse the notation by applying the operation $\otimes$ also to substitutions.

We define a concrete semantics $T_P$ which is a "simplified" and more concrete version of the immediate consequences operator $T_P^s$ of s-semantics. The $T_P$ operator is defined using only the operations of unification $\otimes$ and union of sets of idempotent substitutions and it does not take into account variable projection (the existential quantification over variables). This will simplify the technical development of this chapter. Note that, the s-semantics operator $T_P^s$ can be recovered from $T_P$ by a simple step of abstract interpretation considering the closure of sets of substitutions over variables projection. More importantly, we prove that the least fixpoint extension relatively to $T_P$, turns out to be the least fixpoint extension relatively to $T_P^s$.

We consider a concrete semantic domain $CInt \stackrel{\text{def}}{=} GAtom \longmapsto \wp(Sub)$ of functions, as usual called interpretations, which map most general atoms to sets of substitutions. $CInt$ is trivially a complete lattice, once ordered pointwise. Often, we will find convenient to denote an interpretation $I \in CInt$ by the set $\{ \langle p(\bar{X}), I(p(\bar{X})) \rangle \mid p \in \pi \}$. Then, for any program $P$, the immediate consequences operator $T_P : CInt \longmapsto CInt$ is defined as follows, for any $I \in CInt$:

$$T_P(I)(p(\bar{Y})) \stackrel{\text{def}}{=} \bigcup_{C \lll P} (c \otimes \bigotimes_{i=1..n} I(q_i(\bar{X}_i)) \otimes (\bar{Y} = \bar{X})),$$

where $C = p(\bar{X}) :- c, q_1(\bar{X}_1), \ldots, q_n(\bar{X}_n)$. Here, $C \lll P$ denotes that the clause $C$ of $P$ is renamed apart with fresh variables and, as usual, also $I(q_i(\bar{X}_i))$ is renamed apart. As before, $\bar{Y} = \bar{X}$ denotes the set of most general unifiers which solve it.

If $\rho \in uco(\wp(Sub))$ is any abstraction on sets of substitutions, a corresponding abstraction on interpretations $\wr\rho\wr \in uco(CInt)$ which acts accordingly to $\rho$ can be defined as follows, for any $I \in CInt$:

$$\wr\rho\wr(I) \stackrel{\text{def}}{=} \{ \langle p(\bar{X}), \rho(c)\rangle \mid \langle p(\bar{X}), c\rangle \in I \}.$$

Note that $\wr\cdot\wr$ is trivially monotone, i.e., for all $\rho, \eta \in uco(\wp(Sub))$, if $\rho \sqsubseteq \eta$ then $\wr\rho\wr \sqsubseteq \wr\eta\wr$.

Given a property of interest $\rho \in uco(\langle\wp(Sub), \subseteq\rangle)$, our goal is to find the most abstract domain which contains $\rho$ (more precisely, $\wr\rho\wr$) and is complete for any $T_P$ operator. Our strategy consists in characterizing the least complete extension of $\rho$ for the basic operations involved in the definition of $T_P$, and then in showing that, under reasonable hypotheses, this domain turns out to be the right one. Since every abstract domain is trivially always complete for l.u.b.'s, it turns out that union of sets of substitutions is troubleless, and therefore it is enough to concentrate on unification $\otimes$. Indeed, the following result shows that completeness for $\otimes$ implies completeness for any $T_P$.

**Theorem 7.3.1** *Let $\rho \in uco(\wp(Sub))$. If $\rho$ is (absolutely) complete for $\otimes$ then $\wr\rho\wr$ is (absolutely) complete for any $T_P$.*

**Proof.** We have to show that for each $I \in GAtom \longmapsto \wp(Sub)$ it holds:

$$\{ \langle p(\bar{X}), \rho(c)\rangle \mid \langle p(\bar{X}), c\rangle \in T_P(I) \} = \{ \langle p(\bar{X}), \rho(c)\rangle \mid \langle p(\bar{X}), c\rangle \in T_P(\wr\rho\wr(I)) \}.$$

Let $I \in GAtom \longmapsto \wp(Sub)$. Then, for all $C = p(\bar{X}) :\!- c, q_1(\bar{X}_1), \ldots, q_n(\bar{X}_n) \lll P$ and $\langle q_i(\bar{X}_i), c_i\rangle \in I$, for $i = 1..n$, the following equalities hold.

$$\begin{array}{ll}
& \rho(\bigcup_{C \lll P}(c \otimes \bigotimes_{i=1..n} \rho(c_i) \otimes (\bar{Y} = \bar{X}))) \\
= & \rho(\bigcup_{C \lll P} \rho(c \otimes \bigotimes_{i=1..n} \rho(c_i) \otimes (\bar{Y} = \bar{X}))) \\
= & \rho(\bigcup_{C \lll P}(\rho(c) \otimes \rho(\bigotimes_{i=1..n} \rho(c_i) \otimes \rho(\bar{Y} = \bar{X})))) \\
= & \rho(\bigcup_{C \lll P} \rho(c \otimes \bigotimes_{i=1..n} c_i \otimes (\bar{Y} = \bar{X}))) \\
= & \rho(\bigcup_{C \lll P}(c \otimes \bigotimes_{i=1..n} c_i \otimes (\bar{Y} = \bar{X}))).
\end{array}$$

■

By Theorem 6.4.4, given $\rho \in uco(\wp(Sub))$, the least complete extension of $\rho$ w.r.t. the quantale operation $\otimes$ exists and it is the domain $\wp(Sub) \stackrel{\curlywedge}{\multimap} \rho$ of linear implications from $\wp(Sub)$ to $\rho$. This complete domain $\wp(Sub) \stackrel{\curlywedge}{\multimap} \rho$ turns out to be the right one whenever the abstract domain $\rho$ satisfies the following weak decidability property.

**Definition 7.3.2** An abstract domain $\rho \in uco(\wp(Sub))$ is *decidable* if any $S \in \rho$ is a r.e. set. ■

It should be clear that decidability is a reasonable requirement for most abstract domains used in program analysis. In fact, for such a decidable abstraction (of sets) of substitutions, an effective procedure for checking whether a substitution belongs to (is approximated by) an abstract object is available. Moreover, the above notion of decidable abstract domain is much weaker than the standard notion of decidable property. While the latter one is a problem of computability (i.e., the whole abstract domain is r.e. itself), the first one is only a matter of representability (i.e., we can represent each abstract object by a r.e. set). It is worth noting that, most of the abstract domains used in logic program analysis (e.g., the basic domains for groundness and sharing properties) are decidable, while the corresponding abstract properties are not.

As announced, the following key result shows that least complete extensions w.r.t. unification $\otimes$ of decidable abstract domains actually are least fixpoint extensions for $T_P$ operators as well. Roughly, the proof is based on the fact that for any strict abstraction $\beta$ of $\wp(Sub)\stackrel{\curlywedge}{\multimap}\rho$, there exists a program $P$ such that $\beta$ is not fixpoint complete for $T_P$.

**Theorem 7.3.3** *Let $\rho \in uco(\wp(Sub))$ be a decidable abstract domain. The least fixpoint extension of $\wr\rho\wr$ for any $T_P$ is $\wr\wp(Sub)\stackrel{\curlywedge}{\multimap}\rho\wr$.*

**Proof.** By Theorem 6.4.4, $\wp(Sub)\stackrel{\curlywedge}{\multimap}\rho \sqsubseteq \rho$ and is complete for $\otimes$. Therefore, by Theorem 7.3.1 we have that:

$$\wr\rho\wr(\mathit{lfp}(T_P)) = \wr\rho\wr(\mathit{lfp}(\wr\wp(Sub)\stackrel{\curlywedge}{\multimap}\rho\wr \circ T_P)).$$

We show that $\wr\wp(Sub)\stackrel{\curlywedge}{\multimap}\rho\wr$ is the most abstract domain with this property, for any program $P$. Note that, by definition of $\stackrel{\curlywedge}{\multimap}$, a set of generators for $\wp(Sub)\stackrel{\curlywedge}{\multimap}\rho$ is clearly $\{\,\Theta \multimap a \mid \Theta \in \wp(Sub), a \in \rho\,\}$. Moreover, since $\wp(Sub)$ is completely distributive, it holds:

$$\Theta \multimap a = (\bigcup_{\theta\in\Theta}\{\,\theta\,\}) \multimap a = \bigcap_{\theta\in\Theta}(\{\,\theta\,\} \multimap a).$$

Therefore, a smaller generator set for $\wp(Sub)\stackrel{\curlywedge}{\multimap}\rho$ is simply

$$\{\,\{\,\theta\,\} \multimap a \mid \theta \in Sub, a \in \rho\,\}.$$

We show that, given any generator $\{\,\theta\,\} \multimap a$, there exist two programs $P$ and $Q$ whose semantics satisfy:

$$[\![P]\!](p(\bar{V})) = \{\,\theta\,\} \multimap a$$

$$[\![Q]\!](q(\bar{V})) = \{\, \theta \,\}$$

We can assume that no predicate is shared between $P$ and $Q$. We then consider a program $R$ defined as follows:

$$\{\, \mathtt{r(X)} :\!- \mathtt{p(X), q(X)} \,\} \cup \mathtt{P} \cup \mathtt{R}$$

Note that the semantics of $R$ satisfies

$$[\![R]\!](r(\bar{V})) = (\{\, \theta \,\} \multimap a) \otimes \{\, \theta \,\} \subseteq a.$$

Suppose, by contradiction, that there exists $\beta \in uco(GAtom \longmapsto \wp(Sub))$ such that $\wr\rho\wr(lfp(T_P)) = \wr\rho\wr(lfp(\beta \circ T_P))$ and $\beta([\![P]\!])(p(\bar{V})) > \{\, \theta \,\} \multimap a$. Note that $\{\, \theta \,\} \multimap a$ is the most abstract object which, when combined by $\otimes$ with $\{\, \theta \,\}$, is approximated by $a$. It follows that the semantics of $R$ satisfies $\beta([\![R]\!])(r(\bar{V})) > a$. Since $a \in \rho$, then $\beta$ is not complete. This proves that $\wr\wp(Sub) \overset{\curlywedge}{\multimap} \rho\wr$ is the least fixpoint extension of $\wr\rho\wr$.

Now we have to show that $P$ and $Q$ as above do exist. $Q$ is trivially $\{\, \mathtt{q}(\bar{\mathtt{V}}\theta). \,\}$, where $\bar{V} = dom(\theta)$. In order to show the existence of $P$ we note that, under the assumption of a r.e. term language, the set:

$$\{\, \theta \,\} \multimap a = \{\, \delta \in Sub \mid \text{if } \delta \wedge \theta \neq \tau \text{ then } \theta \wedge \delta \in a \,\}$$

is r.e.. This is because $Sub$ is r.e., unification is recursive, and testing for membership in an abstract object is semidecidable by the hypothesis that $\rho$ is decidable. Therefore, by Church's Thesis, there exists a program $P$ whose semantics is $\{\, \theta \,\} \multimap a$, for some predicate $p$. ∎

The previous theorem states that, for any decidable domain $\rho \in uco(\wp(Sub))$, $\wr\wp(Sub) \overset{\curlywedge}{\multimap} \rho\wr$ is the least fixpoint extension of $\wr\rho\wr$ for the immediate consequences operator $T_P$, for any program $P$. As already announced, when the s-semantics $T_P^s$ is adequate for observing the property $\rho$ we are interested in, that is $\rho(lfp(T_P)) = \rho(lfp(T_P^s))$, it turns out that $\wr\wp(Sub) \overset{\curlywedge}{\multimap} \rho\wr$ actually is the least fixpoint extension of $\wr\rho\wr$ for any immediate consequences operator $T_P^s$ of s-semantics. Note that, the standard s-semantics operator $T_P^s$ can be obtained by $T_P$ by projecting the nonrelevant variables, i.e., $T_P^s = \exists \circ T_P$ where

$$\exists(I) \overset{\text{def}}{=} \{\, \langle p(\bar{X}), \exists_{-X}(c)\rangle \mid \langle p(\bar{X}), c\rangle \in I \,\}$$

and $\exists_{-X}(c)$ is existential quantification (on substitutions) of all variables except those in $X$. Moreover, $\wr\wp(Sub) \overset{\curlywedge}{\multimap} \rho\wr$ clearly induces a semantic transformer less precise than $T_P^s$, i.e., $T_P^s \sqsubseteq \wr\wp(Sub) \overset{\curlywedge}{\multimap} \rho\wr \circ T_P$. Hence, by Theorem 7.3.3, we get the following desired consequence.

**Corollary 7.3.4** *Let $\rho \in uco(\wp(Sub))$ be a decidable abstract domain, such that $\rho(\mathit{lfp}(T_P)) = \rho(\mathit{lfp}(T_P^s))$, for any $P$. Then, $\wr\wp(Sub) \stackrel{\curlywedge}{\rightarrow} \rho\int$ is the least fixpoint extension of $\wr\rho\int$ for any $T_P^s$.*

**Proof.**

$$\begin{aligned}
\wr\rho\int(\mathit{lfp}(\exists \circ T_P)) &= \wr\rho\int(\mathit{lfp}(T_P^s)) && [\,\text{since } T_P^s = \exists \circ T_P\,]\\
&= \wr\rho\int(\mathit{lfp}(T_P)) && [\,\text{by hypothesis}\,]\\
&= \wr\rho\int(\mathit{lfp}(\wr\wp(Sub) \stackrel{\curlywedge}{\rightarrow} \rho\int \circ T_P)) && [\,\text{by Theorem 7.3.3}\,]
\end{aligned}$$

Therefore, by minimality of $\wr\wp(Sub) \stackrel{\curlywedge}{\rightarrow} \rho\int$, it holds that $\exists \sqsubseteq \wr\wp(Sub) \stackrel{\curlywedge}{\rightarrow} \rho\int$ and thus we have that $\wr\rho\int(\mathit{lfp}(\wr\wp(Sub) \stackrel{\curlywedge}{\rightarrow} \rho\int \circ T_P)) = \wr\rho\int(\mathit{lfp}(\wr\wp(Sub) \stackrel{\curlywedge}{\rightarrow} \rho\int \circ \exists \circ T_P)) = \wr\rho\int(\mathit{lfp}(\wr\wp(Sub) \stackrel{\curlywedge}{\rightarrow} \rho\int \circ T_P^s))$.

To prove that $\wr\wp(Sub) \stackrel{\curlywedge}{\rightarrow} \rho\int$ is the least abstraction $\alpha$ which satisfies $\rho(\mathit{lfp}(T_P^s)) = \rho(\mathit{lfp}(\alpha \circ T_P^s))$, note that $\rho(\mathit{lfp}(\alpha \circ T_P)) \leq \rho(\mathit{lfp}(\alpha \circ T_P^s)) = \rho(\mathit{lfp}(T_P^s)) = \rho(\mathit{lfp}(T_P))$ and therefore, by minimality of $\wr\wp(Sub) \stackrel{\curlywedge}{\rightarrow} \rho\int$ with respect to $T_P$, it follows that it is also the least fixpoint extension of $\wr\rho\int$ for any $T_P^s$. ∎

## 7.4 Complete semantics for groundness analysis

Groundness analysis is arguably one of the most important analysis for logic-based programming languages. Groundness analysis aims to statically detect whether variables will be bound to ground terms in successful derivations. This allows optimizing compilers to significantly speed up unification. By instantiating the results above, we are able to characterize the least fixpoint extension of the basic abstract domain representing plain groundness information w.r.t. any immediate consequences operator of s-semantics. The resulting semantics can therefore be interpreted as the "optimal" semantics for groundness analysis.

If $V \subseteq \mathcal{V}$ is a finite set of variables of interest, the simplest abstract domain for representing plain groundness information of variables in $V$ is $\mathcal{G}_V \stackrel{\text{def}}{=} \langle \wp(V), \supseteq \rangle$, as first put forward by Jones and Søndergaard [Jones and Søndergaard 1987]. The intuition is that each $W \in \mathcal{G}_V$ represents the set of substitutions which ground every variable in $W$. $\mathcal{G}_V$ is related to the concrete domain $\wp(Sub)$ by the following concretization map, for each $W \in \mathcal{G}$.

$$\gamma_{\mathcal{G}_V}(W) \stackrel{\text{def}}{=} \{\, \theta \in Sub \mid \forall v \in W.\ vars(\theta(v)) = \varnothing \,\}.$$

As usual, we shall abuse the notation by denoting with $\mathcal{G}_V \in uco(\wp(Sub))$ the corresponding isomorphic image $\gamma_{\mathcal{G}_V}(\mathcal{G}_V)$ in $\wp(Sub)$. Note that, the choice of an abstract domain $\mathcal{G}_V$ depends on the number of variables used during the analysis, and therefore on the specific program to analyze. A program independent abstract domain $\mathcal{G} \in uco(CInt)$ for representing plain groundness information can therefore be defined as follows:

$$\mathcal{G}(I) \stackrel{\text{def}}{=} \{ \langle p(\bar{X}), \mathcal{G}_{\bar{X}}(c) \rangle \mid \langle p(\bar{X}), c \rangle \in I \}.$$

It is easy to check that $\mathcal{G}$ actually is a closure on the concrete semantic domain *CInt*. Furthermore, $\mathcal{G}$ is clearly decidable and $T^s_P$ is adequate for observing groundness, i.e., $\mathcal{G}(\mathit{lfp}(T_P)) = \mathcal{G}(\mathit{lfp}(T^s_P))$ ([Giacobazzi 1996]). Thus, as an easy consequence of Corollary 7.3.4, we can prove the following result, where $V \subset_f \mathcal{V}$ means that $V$ is a finite subset of $\mathcal{V}$.

**Theorem 7.4.1** $\bigsqcap_{V \subset_f \mathcal{V}} \wr \wp(Sub) \overset{\curlywedge}{\rightarrow} \mathcal{G}_V \int$ *is the least fixpoint extension of* $\mathcal{G}$ *for any* $T^s_P$.

**Proof.** For each finite $V \subset_f \mathcal{V}$, the most abstract domain complete for $T_P$ which contains $\wr \mathcal{G}_V \int$ is $\wr \wp(Sub) \overset{\curlywedge}{\rightarrow} \mathcal{G}_V \int$. Therefore, the most abstract domain which contains all $\wr \mathcal{G}_V \int$ for each $V \in \mathcal{V}$ must contain $\bigsqcap_{V \subset_f \mathcal{V}} \wr \wp(Sub) \overset{\curlywedge}{\rightarrow} \mathcal{G}_V \int$. The thesis follows since the reduced product of complete domains is still complete. ∎

**Example 7.4.2** Consider the language $\mathcal{L} = \{ a/0, f/1 \} \cup \mathcal{V} \cup \pi$, where $\pi = \{ p/1 \}$ and $\mathcal{V} = \{ X_i \}_{i \in \mathbb{N}}$. Then, let us consider the following program $P$ built over $\mathcal{L}$:

$$\begin{array}{l} p(X_0) :\!- X_0 = a. \\ p(X_0) :\!- X_0 = f(X_1). \end{array}$$

Recall that the s-semantics $[\![P]\!]^s$ of a program $P$ is obtained from $T_P$ as $\mathit{lfp}(\exists \circ T_P)$. Moreover, the Clark semantics $[\![P]\!]^c$ (also known as c-semantics [Falaschi et al. 1989]) of a program $P$ characterizes the correct answer substitutions for $P$, and can be obtained from the s-semantics $[\![P]\!]^s$ by the closure under instantiation $\lambda X.\downarrow X$. Thus, in our notation, for any predicate symbol $p$, $[\![P]\!]^c(p(\bar{X})) = \downarrow([\![P]\!]^s(p(\bar{X})))$. It is then an easy task to check that, for any variable $X_i \in \mathcal{V}$,

$$\begin{array}{l} [\![P]\!]^s(p(X_i)) = \{ \{ X_i \leftarrow a \} \} \cup \{ \{ X_i \leftarrow f(X_j) \} \mid X_j \in \mathcal{V}, X_i \neq X_j \} \\ [\![P]\!]^c(p(X_i)) = \{ \sigma \in Sub \mid \forall X_j \in \mathcal{V}.\sigma(X_i) \neq X_j \}. \end{array}$$

In this case, the least fixpoint extension of $\mathcal{G}$ (which, by Theorem 7.4.1, is given by $\bigsqcap_{V \subset_f \mathcal{V}} \wr \wp(Sub) \overset{\curlywedge}{\rightarrow} \mathcal{G}_V \int$), when applied to s-semantics $[\![P]\!]^s$, differs from both s-semantics and c-semantics, and acts as follows:

$$\Big(\bigsqcap_{V \subset_f \mathcal{V}} \wr \wp(Sub) \overset{\curlywedge}{\rightarrow} \mathcal{G}_V \int\Big)([\![P]\!]^s) = \lambda\, p(X_i).Sub$$

Before proving this equality, let us informally justify it. Intuitively, a complete semantics including groundness information should be able to observe both groundness and failures: groundness, of course, because it is as concrete as the domain of observation; failures because, being complete for unification, it is able to discriminate between successful and failed computations. Moreover, if the set of computed answers, for a given predicate $p(X)$, includes all possible ground substitutions and some nonground substitutions, then any goal $p(t)$ can never fail (since the semantics contains at least one ground answer for the goal) and, at the same time, the set of computed answers for the most general atom $p(X)$ is not ground. Thus, the most abstract semantics including groundness information does not need to distinguish between such a program and one whose computed answers are all possible substitutions, since both such programs have the same (negative) result for groundness analysis and do not fail for any goal.

In order to prove the equality above, let us first show that, for all $V \subset_f \mathcal{V}$ and $g \in \mathcal{G}_V$, $[\![P]\!]^s(p(X_i)) \rightharpoonup g = g$. It suffices to show that $[\![P]\!]^s(p(X_i)) \rightharpoonup \varnothing = \varnothing$, and, for any $v \in \mathcal{V}$, $[\![P]\!]^s(p(X_i)) \rightharpoonup \{ v \} = \{ v \}$.

In the first case, $\delta \in [\![P]\!]^s(p(X_i)) \rightharpoonup \varnothing$ if and only if for all $\theta \in [\![P]\!]^s(p(X_i))$ it holds that $\delta \wedge \theta = \tau$. But note that, given any substitution $\delta$, there always exists a $\sigma \in [\![P]\!]^s(p(X_i))$ which unifies with $\delta$, and therefore $[\![P]\!]^s(p(X_i)) \rightharpoonup \varnothing = \varnothing$. On the other hand, for each $v \in \mathcal{V}$, $\delta \in [\![P]\!]^s(p(X_i)) \rightharpoonup \{ v \}$ if and only if $\forall\theta \in [\![P]\!]^s(p(X_i))$ such that $\delta \wedge \theta \neq \tau$ it holds that $\delta \wedge \theta \in \{ v \}$. We have the following cases, where $Z$ is a fresh variable not in $\delta$.

$$
\begin{array}{llllll}
\delta(X_i) = a & \Rightarrow & \delta \wedge \{ X_i \leftarrow a \} = \delta & & \Rightarrow & \delta \in \{ v \} \\
\delta(X_i) = X_j & \Rightarrow & \delta \wedge \{ X_i \leftarrow f(Z) \} \in \{ v \} \text{ iff } \delta \in \{ v \} & & \Rightarrow & \delta \in \{ v \}
\end{array}
$$

Therefore, any substitution $\delta \in [\![P]\!]^s(p(X_i)) \rightharpoonup \{ v \}$ belongs to $\{ v \}$. And thus, for any finite set of variables $V$, it holds that $[\![P]\!]^s(p(X_i)) \rightharpoonup V = V$. Hence, by Theorem 6.4.4 (2), we have that

$$
\begin{array}{ll}
& (\prod_{V \subset_f \mathcal{V}} \wr \wp(Sub) \overset{\curlywedge}{\rightharpoonup} \mathcal{G}_V \int)([\![P]\!]^s)(p(X_i)) \\
= & \bigwedge_{V \subset_f \mathcal{V}} \bigwedge_{g \in \mathcal{G}_V} ([\![P]\!]^s(p(X_i)) \rightharpoonup g) \rightharpoonup g \\
= & \bigwedge_{V \subset_f \mathcal{V}} \bigwedge_{g \in \mathcal{G}_V} g \rightharpoonup g \\
= & Sub.
\end{array}
$$

It is also worth noting that if we consider the equivalence relation on programs induced by the semantics $(\prod_{V \subset_f \mathcal{V}} \wr \wp(Sub) \overset{\curlywedge}{\rightharpoonup} \mathcal{G}_V \int)([\![P]\!]^s)$, the program $P$ is in the same equivalence class as the program $\{ p(X_0). \}$. ■

# Chapter 8

# Condensing in logic program analysis

Abstract

In this chapter we study condensing domains in analyses of (constraint) logic programs, i.e., domains where goal-independent analyses have the same precision as goal-driven ones. Because of this key property, condensation is known to be an important property of domains in order to apply goal-independent analysis to logic and constraint-based languages, yet providing the same precision as in a goal-directed one. We show that the solutions of disjunctive completeness problems are all condensing and, under suitable hypotheses, any condensing domain is a solution of some disjunctive completeness problem.

The results presented in this chapter are a joint work with R. Giacobazzi.

## 8.1 Condensing domains

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], may be less precise than goal-directed ones (cf. Marriott and Søndergaard [1993]). This is because goal-independent analysis can be thought of as 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. On the contrary, it may also happen that goal-independent analyses are more precise than goal-directed ones, for instance when failure properties are observed. Abstract domains where goal-independent and goal-directed analyses have the same precision are called *condensing* domains.

The notion of condensation has been first introduced by Langen [1990] in the context of dataflow analysis of logic programs. The original definition of condensing domain for $X \in uco(C)$ can be given for a generic abstract semantic function $f^\sharp : X \longmapsto X$, with respect to a binary operation $\otimes^\sharp : X \times X \longmapsto X$. A domain $X \in uco(C)$ is condensing if for any $x, y \in X$ we have that

$$x \otimes^\sharp f^\sharp(y) = f^\sharp(x \otimes^\sharp y).$$

Marriott and Søndergaard [1993] proposed a stronger notion of condensing, where the abstract function $f^\sharp$ and the abstract operation $\otimes^\sharp$ are supposed to be induced from a concrete semantic function $f : C \longmapsto C$ and a concrete operation $\otimes : C \times C \longmapsto C$, i.e., they consider the best correct approximations of the corresponding concrete operations. This is what makes it possible to consider condensing an abstract domain property. We follow [Marriott and Søndergaard 1993] and consider the following definition of condensing.

**Definition 8.1.1** An abstract domain $\rho \in uco(C)$ is *condensing* for $f : C \longmapsto C$ with respect to $\otimes : C \times C \longmapsto C$ if for any $x, y \in \rho$, we have that

$$\rho(x \otimes f^\rho(y)) = f^\rho(\rho(x \otimes y))$$

where $f^\rho : \rho \longmapsto \rho$ is the abstract semantics induced by $f$ on the domain $\rho$. ∎

The above property of condensation is particularly relevant in analysis of constraint logic programs, where $\otimes$ plays the rôle of unification (conjunction) of constraints [Langen 1990; Marriott and Søndergaard 1993]. More formally, if $true$ represents the top (weakest) constraint, representing any initial store, then in any condensing abstract domain, for any constraint $c \in \rho$, we have that $\rho(c \otimes f^\rho(\rho(true))) = f^\rho(c)$. The idea here is that computing the semantics of a query with initial constraint $c$, i.e., $f^\rho(c)$, is the same as unifying $c$ with the result of the semantics of the same query in the most general environment $true$, which is $f^\rho(\rho(true))$. 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. 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.

In this chapter we show that, the abstract domains which are solutions of the recursive domain equation $\rho = \rho \stackrel{\curlywedge}{\rightarrow} \rho$, 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 $\rho = \rho \overset{\curlywedge}{\rightarrow} \rho$. This property shows a surprising link between completeness in abstract interpretation and condensing domains and, more importantly, gives computational relevance in static program analysis to the notion of disjunctive completeness.

## 8.2 Goal-driven semantics

Consider a simple constraint logic programming (*clp*) language (cf. Jaffar and Maher [1994]). 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. The semantic domain is $\langle \wp(Store), \otimes \rangle$, which is assumed to be a unital, commutative quantale, with unity $true$. Moreover, we assume that $\otimes$ is idempotent, i.e., $c \otimes c = c$, for all $c \in \wp(Store)$. These are weak enough conditions to include most constraint systems developed in the literature (e.g., see Saraswat et al. [1991]). The (forward) semantics $[\![\langle p(\bar{x}), s \rangle]\!]_P$ of a process $p$, executed with initial store $s \in \wp(Store)$ in a program $P$, is defined as usual as the least fixpoint of the following recursive function $\mathcal{S}^s_{P,p(\bar{x})} : \wp(Store) \longmapsto \wp(Store)$[1], for any $\sigma \in \wp(Store)$.

$$\begin{array}{lcl} \mathcal{S}_c(\sigma) & = & s \otimes c \otimes \sigma \\ \mathcal{S}_{A_1 \wedge A_2}(\sigma) & = & \mathcal{S}_{A_1}(\sigma) \otimes \mathcal{S}_{A_2}(\sigma) \\ \mathcal{S}_{\sum_{i=1}^{n} A_i}(\sigma) & = & \bigvee_{i=1}^{n} \mathcal{S}_{A_i}(\sigma) \\ \mathcal{S}_{p(\bar{x})}(\sigma) & = & \mathcal{S}_A(\sigma \otimes \{\, \bar{x} = \bar{y} \,\}) \;\; \text{where } C = p(\bar{y}) \leftarrow A \lll P. \end{array}$$

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 $\lll$ selects a clause from $P$ whose variables are renamed apart from the rest of the computation. As before, we let a constraint $c$ denote the set of most general unifiers which solve it.

[1] When the program $P$ and the initial state $s$ are clear from the context, we simply write $\mathcal{S}_{p(\bar{x})}$.

The following result states that it is always possible to systematically design a condensing domain for a given basic property of interest $A \in uco(\wp(Store))$, by solving the recursive domain equation $\rho = A \sqcap \rho \stackrel{\curlywedge}{\multimap} \rho$. In order to prove this statement, we consider the following abstract semantics of processes, induced by the concrete one. An abstract semantics is associated with any abstract domain $\rho \in uco(\wp(Store))$, such that for any $\sigma \in \rho$ and initial state $s \in \rho$ we have that

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

For any process $p(\bar{x})$, abstract initial store $s \in \rho$, and program $P$, we define $[\![\langle p(\bar{x}), s\rangle]\!]^{\rho}_{P} = \mathit{lfp}(\mathcal{S}^{s,\rho}_{P,p(\bar{x})})$. Note that, in each equation above, $\mathcal{S}^{\rho}$ is recursively defined as the best correct approximation of $\mathcal{S}$ in $\rho$. The next example shows an abstract domain which is not condensing, where the concrete domain $\wp(Store)$ and $\otimes$ are, respectively, sets of substitutions and standard unification.

**Example 8.2.1** Consider the abstract domain $\mathcal{G}_V$ for groundness analysis of logic programs presented in Section 7.4, for $V = \{\, x, y \,\}$. Recall that the concrete domain is $\wp(Sub)$, the operation $\otimes$ is standard unification, and the unity $true$ is $\{\, \epsilon \,\}$, as defined in Section 7.3. We show that the domain $\mathcal{G}_V$ is not condensing. Consider the program $P$ defined as follows:

$$\mathtt{p(X,Y)} :- \mathtt{X} = \mathtt{Y}.$$

The semantics of the query $p(X,Y)$ with initial state $true$ is $[\![\langle p(X,Y), true\rangle]\!]^{\mathcal{G}_V}_{P} = \mathcal{G}_V(\{\, X = Y \,\}) = Sub$, and therefore $\mathcal{G}_V(\mathcal{G}_V(\{\, X = a \,\}) \otimes [\![\langle p(X,Y), true\rangle]\!]^{\mathcal{G}_V}_{P} = \mathcal{G}_V(\{\, X = a \,\}) = \{\, X \,\}$, which means that the first argument of $p$ is definitely ground. On the other hand, $[\![\langle p(X,Y), \mathcal{G}_V(\{\, X = a \,\})\rangle]\!]^{\mathcal{G}_V}_{P} = \mathcal{G}_V(\mathcal{G}_V(\{\, X = a \,\}) \otimes \{\, X = Y \,\}) = \{\, X, Y \,\}$, which states that both arguments are definitely ground. This shows that the domain $\mathcal{G}_V$ is not condensing. ∎

Before proving the main theorem, let us show the following simple property which will be frequently used in this chapter.

**Proposition 8.2.2** *Let $\langle C, \otimes\rangle$ be a unital commutative quantale, $\rho \in uco(C)$ and $c_1, .., c_n \in C$ for $n < \omega$. If $\rho$ satisfies Eq.* (6.3.1)*, then for all $i$, with $1 \leq i \leq n$, it holds:*

$$\rho(\rho(c_1) \otimes \ldots \rho(c_i) \cdots \otimes \rho(c_n)) = \rho(\rho(c_1) \otimes \ldots c_i \cdots \otimes \rho(c_n))$$

**Proof.** Straightforward from Eq. (6.3.1). ∎

**Theorem 8.2.3** *Let $P$ be a program, $\rho \in uco(\wp(Store))$, and $p(\bar{x})$ be a process. If $\rho = \rho \overset{\curlywedge}{\multimap} \rho$ then $\rho$ is condensing for $\lambda s \in \rho.\ [\![\langle p(\bar{x}), s\rangle]\!]^{\rho}_{P}$.*

**Proof.** Let $\rho$ be a solution of the recursive domain equation $\rho = \rho \overset{\curlywedge}{\multimap} \rho$. Let $\psi, \varphi \in \rho$. Since $\langle \wp(Store), \otimes\rangle$ is a unital, commutative quantale, it is sufficient to prove, by induction on the recursive calls to the function $\mathcal{S}^{\rho}$ in the definition of $[\![\langle p(\bar{x}), \psi \otimes \varphi\rangle]\!]^{\rho}_{P}$, that $\mathcal{S}^{\rho}$ is condensing, for any program $P$ and initial state $s \in \wp(Store)$, i.e., $\mathcal{S}^{\rho}(\rho(\psi \otimes \varphi)) = \rho(\psi \otimes \mathcal{S}^{\rho}(\varphi))$.

- Consider $\mathcal{S}^{\rho}_{c}$. By Proposition 8.2.2 we have that

$$\begin{array}{rcl}
\mathcal{S}^{\rho}_{c}(\rho(\psi \otimes \varphi)) & = & \rho(s \otimes c \otimes \rho(\psi \otimes \varphi)) \\
& = & \rho(s \otimes c \otimes \psi \otimes \varphi) \\
& = & \rho(\psi \otimes c \otimes s \otimes \varphi) \\
& = & \rho(\psi \otimes \rho(c \otimes s \otimes \varphi)) \\
& = & \rho(\psi \otimes \mathcal{S}^{\rho}_{c}(\rho(\varphi)))
\end{array}$$

- Consider $\mathcal{S}^{\rho}_{A_1 \wedge A_2}$. By Proposition 8.2.2 and inductive hypothesis we have that

$$\begin{array}{rcl}
\mathcal{S}^{\rho}_{A_1 \wedge A_2}(\rho(\psi \otimes \varphi)) & = & \rho(\mathcal{S}^{\rho}_{A_1}(\rho(\psi \otimes \varphi)) \otimes \mathcal{S}^{\rho}_{A_2}(\rho(\psi \otimes \varphi))) \\
& = & \rho(\psi \otimes \mathcal{S}^{\rho}_{A_1}(\varphi) \otimes \psi \otimes \mathcal{S}^{\rho}_{A_2}(\varphi)) \\
& = & \rho(\rho(\psi) \otimes \rho(\mathcal{S}^{\rho}_{A_1}(\varphi) \otimes \mathcal{S}^{\rho}_{A_2}(\varphi))) \\
& = & \rho(\rho(\psi) \otimes \mathcal{S}^{\rho}_{A_1 \wedge A_2}(\varphi)).
\end{array}$$

- Consider $\mathcal{S}^{\rho}_{\sum_{i=1}^{n} A_i}$. Recall that, by definition of quantale, the operation $\otimes$ is additive. Therefore, by inductive hypothesis and Proposition 8.2.2 we have that

$$\begin{array}{rcl}
\mathcal{S}^{\rho}_{\sum_{i=1}^{n} A_i}(\psi \otimes \varphi) & = & \rho(\bigvee_{i=1}^{n} \mathcal{S}^{\rho}_{A_i}(\psi \otimes \varphi)) \\
& = & \rho(\bigvee_{i=1}^{n} \psi \otimes \mathcal{S}^{\rho}_{A_i}(\varphi)) \\
& = & \rho(\psi \otimes \bigvee_{i=1}^{n} \mathcal{S}^{\rho}_{A_i}(\varphi)) \\
& = & \rho(\psi \otimes \rho(\bigvee_{i=1}^{n} \mathcal{S}^{\rho}_{A_i}(\varphi))) \\
& = & \rho(\psi \otimes \mathcal{S}^{\rho}_{\sum_{i=1}^{n} A_i}(\varphi)).
\end{array}$$

- Consider $\mathcal{S}^{\rho}_{p(\bar{x})}$. The thesis follows immediately by definition and inductive hypothesis. ■

The key point in the previous theorem is that the operation $\otimes$ is additive, i.e., it distributes over arbitrary l.u.b.'s. Thus, whenever the abstract domain contains all linear implications between any two abstract objects, no loss of precision is accumulated by distributing the abstract unification over nondeterministic computations, which are modeled by abstract disjunction.

In order to prove the converse of Theorem 8.2.3, i.e., that condensing domains are solutions of Eq. (6.3.1), we need some additional hypotheses. The next result, together with Theorem 8.2.3, 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 $\rho = A \sqcap \rho \overset{\curlywedge}{\multimap} \rho$.

**Theorem 8.2.4** *Let $P$ be a program, $p(\bar{x})$ be a process, and $\rho \in uco(\wp(Store))$. Assume that for any $x, y \in \rho$ there exist $\{ c_1, \dots c_n \} \subseteq Store$, $n < \omega$, such that the following conditions hold.*

- $\bigvee_{i=1}^{n} c_i \leq x \multimap y$
- $\rho(\bigvee_{i=1}^{n} c_i) = \rho(x \multimap y)$

*If $\rho$ is condensing for $\lambda s \in \rho.\ [\![\langle p(\bar{x}), s\rangle]\!]_P^{\rho}$ then $\rho = \rho \overset{\curlywedge}{\multimap} \rho$.*

**Proof.** Let $\rho$ be condensing. We have to prove that for any $a, b \in \rho$ we have that $\rho(a \multimap b) = a \multimap b$. Suppose, by contradiction, that there exist $a, b \in \rho$ such that $\rho(a \multimap b) > a \multimap b$. Note that, by definition, $a \multimap b$ is the most abstract object $c$ which satisfies $a \otimes c \leq b$. Therefore, $a \otimes \rho(a \multimap b) \not\leq b$. By hypothesis there exists $\{c_1, \dots c_n\} \subseteq Store$, $n < \omega$ such that the above conditions hold. Consider the program $P = \{ p :- \ \sum_{i=1}^{n} c_i \}$. Then, by hypothesis, we have that

$$
\begin{array}{rcl}
\rho(a \otimes [\![\langle p, true\rangle]\!]_P^{\rho}) & = & \rho(a \otimes \rho(\bigvee_{i=1}^{n} \rho(true \otimes c_i))) \\
& = & \rho(a \otimes \rho(\bigvee_{i=1}^{n} \rho(c_i))) \\
& = & \rho(a \otimes \rho(\bigvee_{i=1}^{n} c_i)) \\
& = & \rho(a \otimes \rho(a \multimap b)) \\
& \not\leq & b \\
& \geq & \rho(a \otimes (a \multimap b)) \\
& \geq & \rho(a \otimes \bigvee_{i=1}^{n} c_i) \\
& = & \rho(\bigvee_{i=1}^{n} a \otimes c_i) \\
& = & \rho(\bigvee_{i=1}^{n} \rho(a \otimes c_i)) \\
& = & [\![\langle p, a\rangle]\!]_P^{\rho}.
\end{array}
$$

Therefore $\rho$ would not be condensing, which is a contradiction. ∎

The hypothesis in Theorem 8.2.4 is not restrictive for most well-known domains in logic program analysis. Conditions (1) and (2) say that any implicational object $x \multimap y$ always approximates a finite disjunction of basic stores. This allows us, in Theorem 8.2.4, to construct, for any implicational object $x \multimap y$, a (finite) program having that object in the abstract semantics. The idea is that, in order to prove the converse of Theorem 8.2.3, any implicational object has to be the semantics of some well-defined program. The following characterization of condensing is therefore immediate by Theorems 8.2.3 and 8.2.4.

**Corollary 8.2.5** *Under the hypothesis of Theorem 8.2.4, $\rho$ is condensing for the semantic function $\lambda s \in \rho.\ [\![\langle p(\bar{x}), s\rangle]\!]^{\rho}_{P}$ if and only if $\rho = \rho \overset{\curlywedge}{\rightarrowtriangle} \rho$.*

# Chapter 9

# Relational analyses and reduced cardinal power

Abstract

In this chapter 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 in abstract interpretation. We show that Heyting completion can be characterized as a completeness problem and we study the algebraic properties of Heyting completion in relation with other well-known domain transformers, like reduced product and disjunctive completion.

The results presented in this chapter are a joint work with R. Giacobazzi and appeared in [Giacobazzi and Scozzari 1998].

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 presented in this chapter 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 relation $\varphi \to \psi$: given two properties $\varphi$ and $\psi$, the property $\varphi \to \psi$ 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 $A$ of basic properties to include the space of all implications. Therefore, for any pair of objects $a, b \in A$, a relational object, which is exactly the intuitionistic implication $a \to b$, is included in the enhancement of $A$. In order to ensure the existence of all intuitionistic implications, we consider as concrete domains complete Heyting algebras (see Section 2.3.2). They 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. Hence, we introduce a new operation for abstract domain refinement, called *Heyting completion*. The Heyting completion of a domain $A$ corresponds to enhancing $A$ with all the above-mentioned implicational objects. We first show that it provides a precise and clean logical model for reduced cardinal power of 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 $a \to b$ with $a \in A$ and $b \in B$. Therefore, any domain which is closed under reduced cardinal power with respect to itself, gets an immediate and clean logical interpretation as the collection of all intuitionistic formulas built in the fragment $\wedge, \to$ 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 9.2.3, 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.

We show that Heyting completion is, at the same time, a domain refinement operator and the solution of a completeness problem. It is easily seen that it is a refinement operator by construction and, in addition, we prove that Heyting completion can also be specified as a solution of a disjunctive completeness problem, where the hidden concrete function is exactly the meet operation of the concrete lattice. We shall exploit this property in order to study the algebraic properties of Heyting completion in relation with the remaining operators introduced in Cousot and Cousot [1979b], i.e., reduced product and disjunctive completion. The result is that all the operations introduced in Cousot and Cousot [1979b] 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 abstract domain equations 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 are exactly the solutions of a disjunctive completeness problem formulated by using as concrete function the meet operation of the concrete lattice. Therefore, they precisely define the family of abstract domains for which the abstraction function $\alpha$ satisfies one of the postulates 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. In Chapters 10 and 11, 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 captures 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.

## 9.1 Heyting completion domain refinement

The notion of *domain refinement* has been introduced in [Filé et al. 1996; 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 enhancement with respect to the standard ordering of precision, e.g., by adding information to 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 domains refinements include Cousot and Cousot's *reduced product*, *disjunctive completion*, and *reduced cardinal power* [1979b]; Nielson's *tensor product* [1985]; Giacobazzi and Ranzato's *dual-Moore-set completion* and *complete extension* [1995; 1997a], and the *open product* and *pattern completion* of Cortesi et al. [1994].

In this section we introduce the notion of *Heyting completion* which formalizes, in a unique operation on domains, the ideas described in the previous section. Heyting completion is obtained by lifting intuitionistic implication to sets: given two abstract domains $A$ and $B$, this is precisely given by the least Moore family (i.e., the most abstract domain) containing the space of all implications between objects of $A$ and objects of $B$.

**Definition 9.1.1** Let $L$ be a cHa and $A, B \in uco(L)$. The *Heyting completion of $A$ with respect to $B$*, denoted by $A \xrightarrow{\curlywedge} B$, is defined as $\bigcurlywedge(\{\, a \to b \mid a \in A, b \in B \,\})$. ■

First, let us show that Heyting completion can be characterized as solution of a completeness problem, as proved in the next theorem.

**Theorem 9.1.2** *Let $L$ be a cHa. $A \xrightarrow{\curlywedge} B = A \overset{\curlywedge}{\rightarrowtriangle} B$ in the quantale $\langle L, \wedge_L \rangle$.*

**Proof.** First note that, $L$ being a cHa, $\langle L, \wedge_L \rangle$ turns out to be a quantale, since cHa are completely meet-distributive. The result follows from the definition of $\overset{\curlywedge}{\rightarrowtriangle}$. ■

In view of this result, Heyting completion inherits all the properties of the domain constructor $\overset{\curlywedge}{\rightarrowtriangle}$, instantiated with $\otimes = \wedge_L$. Moreover, $\langle L, \wedge_L \rangle$ being a unital, commutative quantale with unity $\top_L$, Heyting completion inherits all the characterizations obtained for completeness problems in unital, commutative quantales. In particular, given $\rho, \mu, \eta \in uco(L)$, the following are equivalent, by Theorem 6.2.3.

- $\mu \sqsubseteq \rho \xrightarrow{\curlywedge} \eta$
- $\eta(\rho(x) \wedge_L y) = \eta(\rho(x) \wedge_L \mu(y))$, for all $x, y \in L$.
- $\langle \mu, \eta \rangle \in \Gamma(L, \lambda y. a \wedge_L y)$, for all $a \in \rho$.

In addition, Heyting completion is also reductive, and therefore $A \xrightarrow{\curlywedge} X$ is a refinement of the domain $X$, i.e., $A \xrightarrow{\curlywedge} X \sqsubseteq X$, as shown in the next proposition.

**Proposition 9.1.3** *Let $A \in uco(L)$. $\lambda X.\, A \xrightarrow{\curlywedge} X$ is reductive on $uco(L)$.*

**Proof.** Let $x \in X$. Since $A$ is a Moore family, then $\top_L \in A$ and therefore $x = \top_L \to x \in A \xrightarrow{\curlywedge} X$. ■

In the following, given an abstract domain $X$, $X \xrightarrow{\curlywedge} X$ is called the Heyting completion of $X$. This is a unary operation which is, by Propositions 6.2.1 and 9.1.3, monotone and reductive on $uco(L)$. Therefore, the operation of Heyting completion $\lambda X.\, X \xrightarrow{\curlywedge} X : uco(L) \longmapsto uco(L)$ is a domain refinement in the sense of [Filé et al. 1996; Giacobazzi and Ranzato 1997b]. Note that, while it is monotone and reductive, in general it is not idempotent, as shown in the next example.

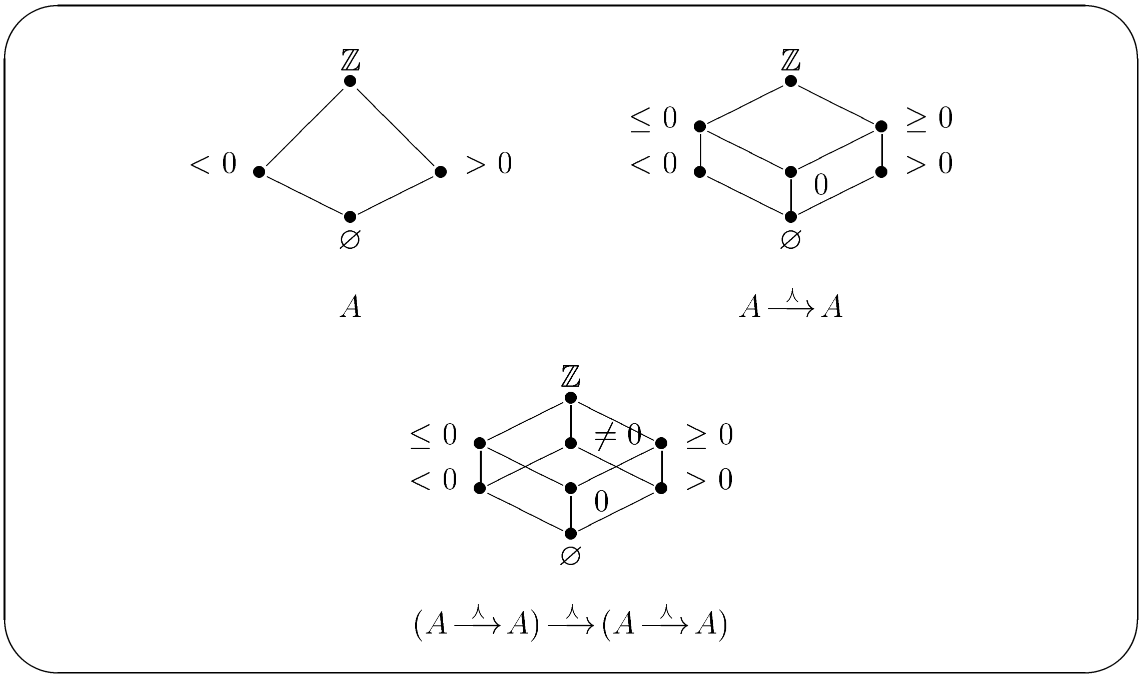



Figure 9.1: Examples of Heyting completion

**Example 9.1.4** Consider the abstract domain $A$ for sign analysis depicted in Figure 9.1, whose concrete domain is $\wp(\mathbb{Z})$ and whose objects have an obvious meaning as elements in $\wp(\mathbb{Z})$. Note that $A \xrightarrow{\curlywedge} A$ is strictly included in $(A \xrightarrow{\curlywedge} A) \xrightarrow{\curlywedge} (A \xrightarrow{\curlywedge} A)$ (both depicted in Figure 9.1). ■

## 9.2 Reduced cardinal power and Heyting completion

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 1979b]. 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 thesis, 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 underlying reduced cardinal power 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 [1979b, Section 10.2].

**Theorem 9.2.1 ([Cousot and Cousot 1979b])** *Let $X$ be a given set, and let $\langle \wp(X), \alpha_A, A, \gamma_A \rangle$ and $\langle \wp(X), \alpha_B, B, \gamma_B \rangle$ be G.i.'s. Then $(\wp(X), \alpha, A \xmapsto{m} B, \gamma)$ is a Galois connection 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 $A \xmapsto{m} B$, reduced with respect to concretization (cf. Cousot and Cousot [1979b]). 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 [1979b] 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 [1979b]. 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. Given $A, B \in uco(L)$, 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 9.2.1 above to cHa's and proves that $B^A$ refines $B$.

**Theorem 9.2.2 ([Giacobazzi and Ranzato 1998b])** *Let $L$ be a cHa and $A, B \in uco(L)$. 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[1] 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 intuitionistic implications.

**Theorem 9.2.3** *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 implications, for dependencies in $B^A$.

**Lemma 9.2.4** *Let $L$ be a cHa and $A, B \in uco(L)$. Then, for any $d \in L$ we have that*

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

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

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

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

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

$$
\begin{array}{ll}
\forall a \in A. & d \wedge a \leq \rho_B(d \wedge a) \\
\forall a \in A. & d \leq a \to \rho_B(d \wedge a) \\
 & d \leq \bigwedge_{a \in A}(a \to \rho_B(d \wedge_L a)) \\
 & d \wedge x \leq \bigwedge_{a \in A}(a \to \rho_B(d \wedge_L a)) \wedge x \\
 & \rho_B(d \wedge x) \leq \rho_B(\bigwedge_{a \in A}(a \to \rho_B(d \wedge_L a)) \wedge x).
\end{array}
$$

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

$$
\begin{array}{l}
\bigwedge_{a \in A}(a \to \rho_B(d \wedge_L a)) \leq x \to \rho_B(d \wedge x) \\
\bigwedge_{a \in A}(a \to \rho_B(d \wedge_L a)) \wedge x \leq (x \to \rho_B(d \wedge x)) \wedge x \leq \rho_B(d \wedge x) \\
\rho_B(\bigwedge_{a \in A}(a \to \rho_B(d \wedge_L a)) \wedge x) \leq \rho_B(d \wedge x)
\end{array}
$$

which closes the proof. ■

**Lemma 9.2.5** *Let $L$ be a cHa and $A, B \in uco(L)$. For any $d \in L$ we have that*

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

**Proof.** Let $b = \bigwedge_{a \in A}(a \to \rho_B(d \wedge_L a))$. By Lemma 9.2.4, $\lambda x \in A.\ \rho_B(d \wedge_L x) = \lambda x \in A.\ \rho_B(b \wedge_L x)$. Now we prove that $b$ is the most abstract object with this 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

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

which concludes the proof. ■

We are now in the position to give the proof of Theorem 9.2.3.

**Proof of Theorem 9.2.3.** Let $A, B \in uco(L)$. By Lemma 9.2.5 we know that

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

Hence $B^A \subseteq A \xrightarrow{\curlywedge} B$. We now prove that $A \xrightarrow{\curlywedge} B \subseteq B^A$. Let $a \to b$ be a generator of $A \xrightarrow{\curlywedge} B$. We show that $a \to b = \bigwedge_{x \in A}(x \to \rho_B(a \to b \wedge_L x))$, which implies, by Lemma 9.2.5, that $a \to b \in B^A$. By Lemma 9.2.5, we have that, for any $d \in L$, $d \leq \bigwedge_{x \in A}(x \to \rho_B(d \wedge_L x))$ (since $d \leq \vec{\gamma}(\vec{\alpha}(d))$). Thus $a \to b \leq \bigwedge_{x \in A}(x \to \rho_B(a \to b \wedge_L x))$. For the other part, for any $a \in A, b \in B$ we have that

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

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

$$\bigwedge_{x \in A}(x \to \rho_B(a \to b \wedge_L x)) \leq a \to \rho_B(a \to b \wedge a) \leq a \to b$$

and therefore $a \to b = \bigwedge_{x \in A}(x \to \rho_B(a \to b \wedge_L x))$. ■

## 9.3 Algebraic properties

In view of the results in the previous section, all the operations introduced in Cousot and Cousot [1979b] for systematic design of abstract domains are now characterized by a clean logical interpretation. In this section, we show some basic algebraic properties of Heyting completion with respect to the 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 1979b] of abstract domains, the operation for *disjunctive completion* ($\curlyvee$) [Cousot and Cousot 1979b; Cousot and Cousot 1994; Giacobazzi and Ranzato 1998a], and the above Heyting completion of abstract domains. As we will see, in most cases, the algebraic properties of Heyting completion are direct consequences of analogous logical properties of intuitionistic implication.

### 9.3.1 Heyting completion and reduced product

Reduced product was introduced in Cousot and Cousot [1979b] as the basic operation for combining domains by including conjunction of domain objects in the refined domain. From Proposition 6.2.1, we know that Heyting completion is co-additive in the second argument. The next proposition shows how to decompose reduced product in the first argument.

**Proposition 9.3.1** *Let $L$ be a cHa and $A, B, C \in uco(L)$.*

$$(A \sqcap B) \xrightarrow{\curlywedge} C = A \xrightarrow{\curlywedge} (B \xrightarrow{\curlywedge} C)$$

**Proof.** It suffices to note that, for each $a_i \in A$, $b, b_i \in B$ and $c, c_i \in C$, for $i \in I$, it holds that $(\bigwedge_{i \in I}(a_i \wedge b_i)) \to c = (\bigwedge_{i \in I} b_i) \to ((\bigwedge_{i \in I} a_i) \to c)$ and $b \to \bigwedge_{i \in I}(a_i \to c_i) = \bigwedge_{i \in I}((a_i \wedge b) \to c_i)$. ■

Moreover, Heyting completion can be decomposed as the reduced product of simpler Heyting completions where the second domain is always an atomic (i.e., two-points) abstract domain.

**Proposition 9.3.2** *Let $L$ be a cHa and $A \in uco(L)$ and $X \subseteq L$.*

$$A \xrightarrow{\curlywedge} \curlywedge(X) = \bigsqcap_{x \in X} A \xrightarrow{\curlywedge} \{\top_L, x\}$$

**Proof.** Note that $\curlywedge(X) = \bigsqcap_{x \in X} \{\top_L, x\}$. Therefore, the result follows from co-additivity (Proposition 6.2.1). ■

As observed above, any abstract interpretation $A$ of a concrete domain $L$, 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 $A \in uco(L)$. Recall that $A$ is generated by the set $B \subseteq A$ if $A = \curlywedge(B)$. Hence, by Proposition 9.3.2, Heyting completion can be constructed from atomic domains, 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 Proposition 9.3.2 that, if $B \in uco(L)$ is algebraic and $L$ is cHa, then the following statement holds:

$$A \xrightarrow{\curlywedge} B = \bigsqcap_{x \in MIrr(B)} A \xrightarrow{\curlywedge} \{\top_L, x\} \tag{9.3.1}$$

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

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

**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. (9.3.1). 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 that

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

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

## 9.3.2 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 [1979b] 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 [1979b] and Giacobazzi and Ranzato [1998a]): $\curlyvee(A) \stackrel{\text{def}}{=} \sqcup \{ X \in uco(L) \mid X \sqsubseteq A \text{ and } X \text{ is additive} \}$. 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 9.3.4** *If $L$ is completely distributive or $L$ is a cHa and $A$ is finite, then $\curlyvee(A) = \{ \bigvee_L X \mid X \subseteq A \}$.*

**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 $X, Y \subseteq A$, $(\bigvee_L X) \wedge (\bigvee_L Y) \in \curlyvee(A)$. Since $X$ and $Y$ are 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 9.3.5** *If $L$ is completely distributive or $L$ is a cHa and $A$ is finite, then $\curlyvee(A) \stackrel{\curlywedge}{\longrightarrow} B = A \stackrel{\curlywedge}{\longrightarrow} B$.*

**Proof.** Assume $L$ to be completely distributive or $L$ to be a cHa and $A$ to be finite. Let $a$ be a generator of $\curlyvee(A) \xrightarrow{\curlywedge} B$. By Proposition 9.3.4, 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 completely distributive then $\bigvee_L X \to b = \bigwedge_{x \in X}(x \to b)$ from which follows the thesis. ■

The inverse operation to disjunctive completion, which is called *least disjunctive basis*, was introduced in Giacobazzi and Ranzato [1998a] 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 [1998a] proved that $\Omega(A)$ exists under some hypotheses, e.g., when $L$ or $A$ are finite, or when $L^{\text{op}}$ 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 9.3.5, specifies the relation between Heyting completion and the least disjunctive basis operation.

**Corollary 9.3.6** *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 rôle of the most abstract domain generating $A$ by disjunctive completion. By combining the above result with Proposition 9.3.3 we obtain the following result:

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

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 9.3.7** *If $L$ is collecting or $L$ is a cHa and $A$ is finite, then*

$$A \xrightarrow{\curlywedge} A = \bigcurlywedge(A \cup \{ a \to b \mid a \in JIrr(A), b \in MIrr(A) \smallsetminus MIrr(L) \}).$$

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 an efficient representation for finite domains and their Heyting completion. As a consequence, in order to represent the domain $A \xrightarrow{\curlywedge} A$, it suffices to consider pairs in the concept representation of $A$, i.e., elements in $\{ a \to b \mid a \in JIrr(A), b \in MIrr(A) \smallsetminus MIrr(L) \}$. This representation turns out to be 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 $\{ a \to b \mid a, b \in A \}$ is not $|A|^2$, but $2|A|$, because $|\Omega(A)| = log(|A|)$, $A = \bigcurlywedge(\{ S \smallsetminus \{ x \} \mid x \in S \})$, and $|\{ S \smallsetminus \{ x \} \mid x \in S \}| = log(|A|)$.

## 9.4 Heyting completion closed domains

In Section 9.2, 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 1979b], 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 [1979b], 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? 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 is closed under $\xrightarrow{\curlywedge}$. This domain can be obtained as the greatest (most abstract) solution of the following recursive domain equation:

$$X = A \sqcap (X \xrightarrow{\curlywedge} X) \tag{9.4.1}$$

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

$$X = A \sqcap (X \xrightarrow{\curlywedge} X) \sqcap \bigvee(X) \tag{9.4.2}$$

Note that all the operators involved in the above equations are monotone, and therefore both equations always admits a greatest (most abstract) and least (most concrete) solution. The least fixpoint is the identity closure $\lambda x.x$. Therefore, we are interested in the greatest fixpoint in $uco(L)$, which exists and can be constructively obtained as greatest fixpoint of the corresponding refinement operators.

The following result characterizes the domains which are solutions of Eq. (9.4.1) in terms of their achieved precision. It turns out that such solutions are the domains which include $A$, and which are complete for the meet operation, whenever one of the arguments is an abstract objects. Intuitively, 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.

**Theorem 9.4.1** *Let $L$ be a cHa and $\rho, A \in uco(L)$. Then the following are equivalent.*

**a** $\rho = \rho \xrightarrow{\curlywedge} \rho$;

**b** *$\rho$ is complete for* $\{\, \lambda x \in L.\ \rho(c) \wedge x \,\}_{c \in L}$;

**c** $\rho(\rho(a) \wedge_L b) = \rho(a) \wedge_L \rho(b)$ *for any* $a, b \in L$.

*Moreover,* $gfp(\lambda X.A \sqcap (X \xrightarrow{\curlywedge} X)$ *is the most abstract solution of Eq.* (9.4.1).

**Proof.** First note that, by Proposition 9.1.3, $\rho \sqsubseteq \rho \xrightarrow{\curlywedge} \rho$ if and only if $\rho = \rho \xrightarrow{\curlywedge} \rho$. Therefore, $\langle L, \wedge_L \rangle$ being a unital, commutative quantale, by Theorem 6.3.1 and Corollary 6.3.2, applied with $\otimes = \wedge_L$, we obtain the results. ■

The operator for computing the greatest solution of Eq. (9.4.1) can be remarkably simplified, by observing that the greatest solution to the equation $X = A \sqcap (X \xrightarrow{\curlywedge} X)$ is exactly the greatest solution to the equation $Y = Y \xrightarrow{\curlywedge} A$, as proved in the next theorem.

**Theorem 9.4.2** *The following equations have the same greatest solution:*

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

**Proof.** Let $\lambda X.\ A \sqcap (X \xrightarrow{\curlywedge} X)$ and $\lambda Y.\ Y \xrightarrow{\curlywedge} A$ be the refinement operators associated with the equations. The greatest fixpoint of $\lambda X.\ A \sqcap (X \xrightarrow{\curlywedge} X)$ can be constructively obtained as the limit of the following transfinite sequence of domains $\{ X^\alpha \}_{\alpha \in Ord}$:

$$\begin{array}{lcl} X^0 & = & \top_{uco(L)} \\ X^{\alpha+1} & = & A \sqcap X^\alpha \xrightarrow{\curlywedge} X^\alpha \\ X^\beta & = & \bigsqcap_{\gamma<\beta} X^\gamma \ \text{ if } \beta \text{ is a limit ordinal} \end{array}$$

The greatest fixpoint of $\lambda Y.\ Y \xrightarrow{\curlywedge} A$ is the limit of the following transfinite sequence of domains $\{ Y^\alpha \}_{\alpha \in Ord}$:

$$\begin{array}{lcl} Y^0 & = & \top_{uco(L)} \\ Y^{\alpha+1} & = & Y^\alpha \xrightarrow{\curlywedge} A \\ Y^\beta & = & \bigsqcap_{\gamma<\beta} Y^\gamma \ \text{ if } \beta \text{ is a limit ordinal} \end{array}$$

We prove by transfinite induction that for each $\alpha \in Ord$ we have $X^\alpha = Y^\alpha$. First note that $X^0 = Y^0 = \top_{uco(L)}$ and for $\alpha \geq 1$ it holds that $X^\alpha \sqsubseteq A$ and $Y^\alpha \sqsubseteq A$.

$[\alpha = 0\ ]$ $X^0 = Y^0 = \top$.

$[\alpha = 1\ ]$ $X^1 = Y^1 = A$.

$[\alpha = 2\ ]$ $X^2 = Y^2 = A \xrightarrow{\curlywedge} A$.

$[\alpha = \beta + 2,\ \beta \neq 0\ ]$

$$\begin{array}{lll} X^\alpha & = A \sqcap (X^{\beta+1} \xrightarrow{\curlywedge} X^{\beta+1}) & [\ \beta \geq 1\ ] \\ & = X^{\beta+1} \xrightarrow{\curlywedge} X^{\beta+1} & [\ \text{by inductive hypothesis}\ ] \\ & = Y^{\beta+1} \xrightarrow{\curlywedge} Y^{\beta+1} & \\ & = Y^{\beta+1} \xrightarrow{\curlywedge} (Y^\beta \xrightarrow{\curlywedge} A) & [\ \text{by Prop. 9.3.1}\ ] \\ & = (Y^{\beta+1} \sqcap Y^\beta) \xrightarrow{\curlywedge} A & [\ \text{by monotonicity}\ ] \\ & = Y^{\beta+1} \xrightarrow{\curlywedge} A & \\ & = Y^{\beta+2} = Y^\alpha & \end{array}$$

[$\alpha = \beta + 1$, $\beta$ **limit ordinal** ]

$$
\begin{array}{lll}
X^\alpha & = A \sqcap (X^\beta \xrightarrow{\curlywedge} X^\beta) & [\ \beta \geq 1\ ] \\
& = X^\beta \xrightarrow{\curlywedge} X^\beta & [\ \text{by inductive hypothesis}\ ] \\
& = Y^\beta \xrightarrow{\curlywedge} Y^\beta & \\
& = Y^\beta \xrightarrow{\curlywedge} \bigsqcap_{\gamma<\beta} Y^\gamma & [\ \text{by definition of limit ordinal}\ ] \\
& = Y^\beta \xrightarrow{\curlywedge} \bigsqcap_{\gamma<\beta} Y^{\gamma+1} & [\ \text{by co-additivity in Prop. 6.2.1}\ ] \\
& = \bigsqcap_{\gamma<\beta} Y^\beta \xrightarrow{\curlywedge} Y^{\gamma+1} & \\
& = \bigsqcap_{\gamma<\beta} Y^\beta \xrightarrow{\curlywedge} (Y^\gamma \xrightarrow{\curlywedge} A) & [\ \text{by Prop. 9.3.1}\ ] \\
& = \bigsqcap_{\gamma<\beta} (Y^\beta \sqcap Y^\gamma) \xrightarrow{\curlywedge} A & [\ \text{since } \gamma < \beta\ ] \\
& = \bigsqcap_{\gamma<\beta} Y^\beta \xrightarrow{\curlywedge} A & \\
& = Y^\beta \xrightarrow{\curlywedge} A = Y^\alpha &
\end{array}
$$

[$\alpha$ **limit ordinal** ] Trivial by inductive hypothesis. ∎

It is worth noting that the operator $\lambda X.\ A \sqcap (X \xrightarrow{\curlywedge} X)$ is not, in general, co-continuous in $uco(L)$, as shown by the following example.

**Example 9.4.3** Consider the complete lattice $L = \langle \wp(\mathbb{N}), \mathbb{N}, \varnothing, \cap, \cup, \subseteq \rangle$ and the chain of domains $A_i = \curlywedge(\cup_{1 \leq j \leq i}\{\, \varnothing, \mathbb{N}, \mathbb{N} \setminus \{\, i\,\}, \{\, i\,\}\,\}) \in uco(\wp(\mathbb{N}))$, for $i \geq 1$. First note that, for each $i \geq 1$, it trivially holds $A_i \xrightarrow{\curlywedge} A_i = A_i$. Therefore, $\sqcap_{i \geq 1}(A_i \xrightarrow{\curlywedge} A_i) = \sqcap_{i \geq 1} A_i = \curlywedge(\cup_{i \geq 1} A_i)$. In particular, $\{\, 0\,\}, \varnothing \in \sqcap_{i \geq 1} A_i$, since $\{\, 0\,\} = \cap_{i \geq 1}(\mathbb{N} \setminus \{\, i\,\})$. On the contrary, the implication $\{\, 0\,\} \rightarrow \varnothing = \mathbb{N} \setminus \{\, 0\,\} \notin \sqcap_{i \in \mathbb{N}} A_i$, since, $\mathbb{N} \setminus \{\, 0\,\}$ being a co-atom, it is meet-irreducible. Thus, $\sqcap_{i \geq 1}(A_i \xrightarrow{\curlywedge} A_i)$ is not closed by $\xrightarrow{\curlywedge}$. ∎

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 9.4.4** *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 $\bigwedge_{j\in J'} b_j \in \bigcup_{i\in I} B_i$. ■

**Theorem 9.4.5** *If $L$ enjoys the descending chain property, then*

$$gfp(\lambda X.\ A \sqcap X \stackrel{\curlywedge}{\longrightarrow} X) = \prod_{n<\omega} (\lambda X. X \stackrel{\curlywedge}{\longrightarrow} A)^n(\top_{uco(L)}).$$

**Proof.** By Theorem 9.4.2, $gfp(\lambda X.\ A \sqcap X \stackrel{\curlywedge}{\longrightarrow} X) = gfp(\lambda X.\ X \stackrel{\curlywedge}{\longrightarrow} A)$. Moreover, we are in the hypotheses of Proposition 6.2.2, which ensures us that the operator $\lambda X.\ X \stackrel{\curlywedge}{\longrightarrow} A$ is co-additive in the first argument, and therefore co-continuous. ■

Let us now compare the solutions of Eq.'s (9.4.1) and (9.4.2). First note that the solutions to the equation $X = A \sqcap (X \stackrel{\curlywedge}{\longrightarrow} X) \sqcap \curlyvee(X)$ are all the domains which are closed with respect to both Heyting and disjunctive completion. Therefore, each solution of Eq. (9.4.2) is also a solution of Eq. (9.4.1).

**Corollary 9.4.6** *If $B$ is a solution of Eq.* (9.4.2) *then $B$ is a solution of Eq.* (9.4.1).

The next result gives a characterization of the solutions to Eq. (9.4.1), under the additional hypothesis that $L$ is a collecting domain and $\bot_L \in X$.

**Theorem 9.4.7** *Let $L = \wp(S)$ for some set $S$ and $X \in uco(L)$, with $\bot_L \in X$. Then $X = X \stackrel{\curlywedge}{\longrightarrow} X$ if and only if $X$ is disjunctive and complemented, i.e., $X = \curlyvee(X)$ and $X \stackrel{\curlywedge}{\longrightarrow} \{\top_L, \bot_L\} \subseteq X$*

**Proof.** Let $X = X \stackrel{\curlywedge}{\longrightarrow} X$. By monotonicity, $X = X \stackrel{\curlywedge}{\longrightarrow} X \sqsubseteq X \stackrel{\curlywedge}{\longrightarrow} \{\top, \bot_L\}$ and therefore $X \stackrel{\curlywedge}{\longrightarrow} \{\top_L, \bot_L\} \subseteq X$.

Let us now prove that $X$ is disjunctive. Let $\{a_i\}_{i\in I} \subseteq X$ for some set $I$. Since $X$ is complemented, then $\forall i \in I.\ \neg a_i \in X$, where $\neg x$ denotes $x \to \bot_L$. Therefore, $\bigwedge_{i\in I} \neg a_i \in X$ and thus $\neg(\bigwedge_{i\in I} \neg a_i) \in X$ because $X$ is complemented. $L$ being a cBa, $\neg(\bigwedge_{i\in I} \neg a_i) = \bigvee_{i\in I}(\neg(\neg a_i)) = \bigvee_{i\in I} a_i$. Therefore $\bigvee_{i\in I} a_i \in X$ and clearly $\bot_L \in X$. By Proposition 9.3.4, $X$ is a disjunctive domain or equivalently an additive closure.

Now assume $X$ to be complemented and disjunctive. By Proposition 9.1.3, $X \subseteq X \stackrel{\curlywedge}{\longrightarrow} X$. Let $a \to b$ be a generator of $X \stackrel{\curlywedge}{\longrightarrow} X$. Since $L$ is a Boolean algebra, it holds that $a \to b = \neg a \vee b = (a \to \bot_L) \vee b$. Hence $a \to b \in X$ because $X$ is complemented and disjunctive. Thus $X \stackrel{\curlywedge}{\longrightarrow} X \subseteq X$, and therefore $X = X \stackrel{\curlywedge}{\longrightarrow} X$. ■

By exploiting the previous result, we are now in the position to characterize the most abstract solutions to both Eq. (9.4.1) and (9.4.2) under the additional hypothesis that $L$ is a collecting domain and $\bot_L \in A$.

**Theorem 9.4.8** *Let* $L = \wp(S)$ *for some set* $S$ *and* $A \in uco(L)$*, with* $\bot_L \in A$*. Then* $\curlyvee(A \sqcap (A \xrightarrow{\curlywedge} \{\top_L, \bot_L\}))$ *is the most abstract solution to Eq.'s* (9.4.1) *and* (9.4.2).

**Proof.** We first prove that $\curlyvee(A \sqcap A \xrightarrow{\curlywedge} \{\top, \bot_L\})$ is a solution to Eq. (9.4.1). By construction, $\curlyvee(A \sqcap A \xrightarrow{\curlywedge} \{\top, \bot_L\}) \sqsubseteq A$, and it is trivially disjunctive. Therefore, by Theorem 9.4.7, we have only to show that it is complemented. Let $a \in \curlyvee(A \sqcap A \xrightarrow{\curlywedge} \{\top, \bot_L\})$. By construction, $a = \bigvee_{i \in I}(a_i \wedge \neg b_i)$ with $a_i, b_i \in A$, for some set $I$. Since $L$ is a cBa, we have that

$$\neg a = \neg(\bigvee_{i \in I}(a_i \wedge \neg b_i)) = \bigwedge_{i \in I}(\neg(a_i \wedge \neg b_i)) = \bigwedge_{i \in I}(\neg a_i \vee b_i).$$

Since $b_i \in A$ and $\neg a_i \in A \xrightarrow{\curlywedge} \{\top, \bot_L\}$ for all $i \in I$, then we have that $\neg a_i \vee b_i \in \curlyvee(A \sqcap A \xrightarrow{\curlywedge} \{\top, \bot_L\})$, and therefore $\neg a \in \curlyvee(A \sqcap A \xrightarrow{\curlywedge} \{\top, \bot_L\})$. Thus, $\curlyvee(A \sqcap A \xrightarrow{\curlywedge} \{\top, \bot_L\})$ is complemented. Being disjunctive, $\curlyvee(A \sqcap A \xrightarrow{\curlywedge} \{\top, \bot_L\})$ is a solution of both Eq.'s (9.4.1) and (9.4.2).

Now we prove that it is the most abstract solution. Let $S$ be a solution to Eq. (9.4.1), and therefore $S \sqsubseteq A$ and $S = S \xrightarrow{\curlywedge} S$. We first prove that $S \sqsubseteq \curlyvee(A \sqcap (A \xrightarrow{\curlywedge} \{\top, \bot_L\}))$. Note that $\bot_L \in S$, since $\bot_L \in A \subseteq S$. Hence, by the previous point, $S = \curlyvee(S)$. Moreover, $S = S \xrightarrow{\curlywedge} S \sqsubseteq S \xrightarrow{\curlywedge} \{\top, \bot_L\}$. It follows that $S = S \sqcap S \xrightarrow{\curlywedge} \{\top, \bot_L\}$ and $S = \curlyvee(S) = \curlyvee(S \sqcap S \xrightarrow{\curlywedge} \{\top, \bot_L\})$. Since $S \sqsubseteq A$, by monotonicity, it holds that $S = \curlyvee(S \sqcap S \xrightarrow{\curlywedge} \{\top, \bot_L\}) \sqsubseteq \curlyvee(A \sqcap (A \xrightarrow{\curlywedge} \{\top, \bot_L\}))$. ∎

We conclude this chapter by observing that, from a purely algebraic point of view, the solutions of Eq. (9.4.1), and therefore those of Eq. (9.4.2), have relevant properties that provide a strong link between abstraction functions and *algebraic quantifiers*. The above results, together with Point (c) in Theorem 9.4.1, show that the solutions of Eq. (9.4.1) 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. For instance, in the case of downward closed properties of logic programs, this fact implies that goal-directed and goal-independent analyses are equivalent in precision (the so-called condensation), as seen in Chapter 8.

# Chapter 10

# Groundness analysis

Abstract

In this chapter we apply our results to groundness analysis of logic programs. We consider the well-known domains for ground dependency analysis $\mathcal{D}ef$ and $\mathcal{P}os$, and show that they can be obtained by using the Heyting completion refinement operator. This allows us to characterize both domains in terms of the completeness problems they satisfy and therefore to easily compare their relative precision. Moreover, we show that the domain $\mathcal{P}os$ is an example of Heyting completion closed domain. Thus, $\mathcal{P}os$ inherits all our results for closed domain, in particular the condensing property. Finally, we propose and study a new domain for groundness analysis, which has the advantage of being independent from a specific program.

In the logic programming field, groundness is probably the most popular and also the most studied instance of static analysis. The aim of groundness analysis is to detect whether a variable is definitely ground, i.e., it is bound to a term which contains no variable. This allows optimizing compilers to significantly speed up unification – the prime computational step of any logic language. In the last ten years, much work has been devoted to develop and study abstract domains for groundness analysis of logic programs. Many domains have been proposed, from the simple domain $\mathcal{G}$ by Jones and Søndergaard [1987], to more complex ones, like $\mathcal{D}ef$ [Armstrong et al. 1994] and $\mathcal{P}os$ [Marriott and Søndergaard 1990; Cortesi et al. 1991; Armstrong et al. 1994]. The common basic idea is to use logical formulas as abstract objects. Basic groundness properties are expressed by variables and more structured formulas express relations among groundness of different variables. For instance, the abstract formula consisting of a single variable $X$ denotes the set of all the substitutions which ground the variable $X$.

Among them, $\mathcal{D}ef$ and $\mathcal{P}os$ are certainly the most studied and widely used domains for groundness analysis, since they are able to characterize both pure groundness, i.e., if a variable is instantiated to a ground term, and groundness relations between different variables, i.e., whether the groundness of a variable depends on the groundness of other variables. In addition $\mathcal{P}os$ is able to express disjunctive groundness properties between variables, like "$X$ is ground or $Y$ is ground".

In the standard literature, both the domain $\mathcal{D}ef$ and $\mathcal{P}os$ are built by considering suitable sets of propositional (classic) formulas (respectively *definite* and *positive* formulas). The abstract domains so obtained are then related to the concrete one (sets of substitutions closed by instantiation) through two suitable concretization functions, explicitly proving that they induce a Galois insertion. This method of constructing domains suffers from many drawbacks. The most important one is that the domain has to be "invented" in some way, through a procedure which is usually not formally related to the property we analyze. After defining the abstract domain, it has to be explicitly proved that the domain is an abstraction of the concrete one and, more importantly, that it is actually useful for the analysis. On the contrary, the spirit of this thesis is that abstract domains should be automatically set up by solving suitable completeness problems, starting from the simplest domain expressing the property to analyze and then refining it with respect to some operation involved in the semantics. As logic programs compute substitutions and groundness is a property closed by instantiation, the most natural choice for the concrete domain contains as elements sets of substitutions closed by instantiation, denoted by $\wp^{\downarrow}(Sub)$. It is well known that $\wp^{\downarrow}(Sub)$ is a cHa (see Section 2.3.2). In this context, the standard operation of unification is exactly the concrete meet operation [Palamidessi 1990]. Thus, we can exploit our result in Chapter 9 and reconstruct both domains as solutions of completeness problems for the operation of unification, by using the Heyting completion refinement operator.

The basic domain we start from is the simplest (most abstract) domain defining the property of groundness, which is $\mathcal{G} = \wp(Var)$ by Jones and Søndergaard [1987] where, intuitively, each variable $v$ denotes the set of substitutions which ground the variable $v$. We first show that the domain $\mathcal{D}ef$ is the Heyting completion of $\mathcal{G}$, i.e., $\mathcal{D}ef = \mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}$ and the domain $\mathcal{P}os$ is the Heyting completion of $\mathcal{D}ef$, and therefore, it can be obtained from $\mathcal{G}$ by applying twice the Heyting completion operator, i.e., $\mathcal{P}os = (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \xrightarrow{\curlywedge} \mathcal{G}$. Moreover, we show that $\mathcal{P}os$ is a Heyting completion closed domain, i.e., $\mathcal{P}os \xrightarrow{\curlywedge} \mathcal{P}os = \mathcal{P}os$. This implies that $\mathcal{P}os$ can not be further refined by exploiting the ground dependencies between variables. Our construction allows us to study in detail the relations between all these domains, to easily compare their expressive power and precision, and therefore to formally justify the optimality of $\mathcal{P}os$ from a computational point of view.

## 10.1 Properties of the concrete domain

Since logic programs compute substitutions and groundness is a property closed by instantiation, the most natural choice as concrete domain is $\wp^{\downarrow}(Sub)$. Most of the works about domains for groundness analysis and, in general, for properties closed by instantiation, use $\wp^{\downarrow}(Sub)$[1] as concrete domain. A formal explanation of this choice, with respect to the more general $\wp(Sub)$, can be found in [Giacobazzi 1996], where it is proved that $\wp^{\downarrow}(Sub)$ is absolutely (and therefore fixpoint) complete for a standard semantics of computed answer substitutions, when observing groundness analysis.

It is well known that $\langle \wp^{\downarrow}(Sub), \leq, \vee, \wedge, Sub, \varnothing \rangle$ is a cHa, where the ordering $\leq$ is set inclusion, the operations $\vee$ and $\wedge$ correspond to set union and set intersection and, for any $a, b \in \wp^{\downarrow}(Sub)$, the intuitionistic implication $a \to b = \bigvee \{ c \in \wp^{\downarrow}(Sub) \mid a \wedge c \leq b \}$ is also given by ([Birkhoff 1967])

$$a \to b = \{ \theta \in Sub \mid \forall \delta \leq \theta \ \delta \in a \ \Rightarrow \delta \in b \}.$$

**Example 10.1.1** Let $x, y \in \mathcal{V}$ be variables, $X = \{ \theta \in Sub \mid vars(\theta(x)) = \varnothing \}$ and $Y = \{ \theta \in Sub \mid vars(\theta(y)) = \varnothing \}$.

$$\begin{aligned} X \to Y &= \{ \theta \in Sub \mid \forall \delta \leq \theta \ \delta \in X \ \Rightarrow \delta \in Y \} \\ &= \{ \theta \in Sub \mid \forall \delta \leq \theta \ vars(\delta(x)) = \varnothing \ \Rightarrow vars(\delta(y)) = \varnothing \} \\ &= \{ \theta \in Sub \mid \forall \delta \leq \theta \ vars(\delta(x)) \subseteq vars(\delta(y)) \} \\ &= \{ \theta \in Sub \mid vars(\theta(x)) \subseteq vars(\theta(y)) \}. \end{aligned}$$

Note that, given a binary functor $f$, the substitution $\{ x \leftarrow f(y, w) \}$ belongs to $X \to Y$, while $\{ y \leftarrow f(x, w) \}$ does not. ■

By exploiting the above characterization of intuitionistic implication, we can state the following result on the decomposition of implications.

**Theorem 10.1.2** *Let $\sigma \in Sub$ and $b \in \wp^{\downarrow}(Sub)$.*

$$(\downarrow \sigma) \to b = \{ \theta \in Sub \mid \theta \wedge \sigma \in b \}^2$$

**Proof.** Let $\theta \in \downarrow \sigma \to b$. By definition of implication, $\forall \delta \leq \theta$ it holds that $\delta \leq \sigma \ \Rightarrow \delta \in b$. If $\theta$ and $\sigma$ do not unify, there is nothing to prove. Otherwise, it holds that $\theta \wedge \sigma \leq \theta$ and $\theta \wedge \sigma \leq \sigma$, therefore $\theta \wedge \sigma \in b$ since $\theta \in \downarrow \sigma \to b$.

Let $\theta$ such that $\theta \wedge \sigma \in b$ and let $\delta \leq \theta$. If $\delta \leq \sigma$ then $\delta \leq \theta \wedge \sigma$. Since $\theta \wedge \sigma \in b$ and $b$ is downward closed, also $\delta \in b$. ■

[1] More complex domains are used in [Cortesi et al. 1996] (a combination of $\wp^{\downarrow}(Sub)$ with sets of variables), and in [Marriot et al. 1994] (the so-called *ex-equations*).

[2] Note that unification $\wedge$ in $Sub$ is a partial operation. We shall abuse the notation and denote by $\theta \wedge \sigma \in b$ the statment *if $\theta \wedge \sigma$ is defined then $\theta \wedge \sigma \in b$.*

## 10.2 Groundness: $\mathcal{D}ef$ and $\mathcal{P}os$

Many domains have been proposed in order to study groundness of logic programs. If $Var$ is the (finite) set of variables of interest, the simplest one is $\mathcal{G} = \wp(Var)$, due to Jones and Søndergaard [1987], where each $V \subseteq Var$ denotes the set of substitutions which ground every variable in $V$. The domain $\mathcal{G}$ can be equivalently defined as a domain of formulas by considering the set of formulas built from $Var \cup \{ \top \}$ by using the connective $\wedge$, quotiented with respect to (classic) logical equivalence $(KL)$. Since each abstract domain must contain $\top$, in the following, for simplicity of notation, we shall identify $Var$ with $Var \cup \{ \top \}$. If we denote by $Form(A, \circ)$ the set of formulas built from $A$ by using the connective $\circ$, then:

$$\mathcal{G} = Form(Var, \wedge)_{/KL}$$

where a formula $v_1 \wedge \ldots \wedge v_n$ represents the set $\{ v_1, \ldots, v_n \}$ and $\top$ stands for the empty set. This domain is certainly the most intuitive one and represents exactly the property we want to analyze, since it is built from the definition of groundness itself. Unfortunately, $\mathcal{G}$ is not very useful for groundness analysis, since it fails in capturing the groundness relations between different variables.

Other domains, based on (classic) propositional logic, have been proposed in order to enrich $\mathcal{G}$. The domain $\mathcal{D}ef$ [Armstrong et al. 1994] is built by considering all the classic propositional formulas whose models are closed under intersection (the so-called *definite* formulas). $\mathcal{D}ef$ has been also characterized as the set of formulas which are conjunctions of definite clauses, always quotiented with respect to (classic) logical equivalence.

$$\mathcal{D}ef \stackrel{\text{def}}{=} \{ \bigwedge_{i \in I} (\bigwedge_{j \in J} x_i^j \Rightarrow y^j) \mid \forall i \in I, j \in J \; x_i^j, y^j \in Var \}_{/KL}$$

The most widely used domain for groundness analysis is the domain $\mathcal{P}os$ [Marriott and Søndergaard 1990; Cortesi et al. 1996; Armstrong et al. 1994]. $\mathcal{P}os$ 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 a clear example of relational analysis. $\mathcal{P}os$ is the set of (classic) propositional formulas built from $Var$, by using the connectives $\wedge, \vee, \Rightarrow$. $\mathcal{P}os$ can be defined in different ways [Cortesi et al. 1996; Armstrong et al. 1994]:

$$\mathcal{P}os \stackrel{\text{def}}{=} Form(Var, \wedge, \vee, \Rightarrow)_{/KL} = Form(Var, \wedge, \Rightarrow)_{/KL}$$

We can relate the three domains to the concrete domain $\wp^{\downarrow}(Sub)$ by using the same concretization function. 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 variables [Armstrong et al. 1994]. For the sake of simplicity, we write $\theta\,\texttt{grounds}\,\phi$ to denote $\{\, x \in Var \mid vars(\theta(x)) = \varnothing \,\} \models \phi$, for $\theta \in Sub$. The concretization function is [Cortesi et al. 1996]:

$$\gamma_{Sub}(\phi) = \{\, \theta \in Sub \mid \forall \sigma \leq \theta \;\sigma\,\texttt{grounds}\,\phi \,\}$$

As usual, we shall abuse the notation and call $\mathcal{G}$, $\mathcal{D}ef$ and $\mathcal{P}os$ the corresponding isomorphic images, subsets of $\wp^{\downarrow}(Sub)$: $\gamma_{Sub}(\mathcal{G})$, $\gamma_{Sub}(\mathcal{D}ef)$ and $\gamma_{Sub}(\mathcal{P}os)$.

## 10.3 Implicational groundness analysis

The first step toward the automatic construction of domains is to isolate the property to analyze and look for the simplest domain which describe the property. In the case of groundness analysis this is clear: the basic groundness property says if a variable is ground or not. If we fix a finite set $Var$ of variables of interest, big enough to analyze the program, the most abstract domain which describes the groundness of all variables in $Var$ is by definition $\curlywedge(Var)$. It contains $\top$ and all possible conjunctions for all variables in $Var$. This domain is exactly the domain $\mathcal{G} = \wp(Var)$ proposed by Jones and Søndergaard [1987]. Therefore, we shall use $\mathcal{G}$ as the basis to reconstruct the other domains.

On the side of domain reconstruction for groundness analysis, a first example has been shown by Giacobazzi and Ranzato [1995]. They proved that the abstract domain $\mathcal{D}ef$ is the reduced cardinal power of $\mathcal{G}$, i.e., $Def = \mathcal{G}^{\mathcal{G}}$. Therefore, by applying Theorem 9.2.3, we immediately obtain a characterization of $Def$ in terms of Heyting completion.

**Corollary 10.3.1** $Def = \mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}$.

**Proof.** Giacobazzi and Ranzato [1995] proved that $Def = \mathcal{G}^{\mathcal{G}}$. The thesis follows by Theorem 9.2.3. ■

In this section we show that also $\mathcal{P}os$ can be obtained by using the Heyting completion operator, starting from the domain $\mathcal{G}$. In particular, we prove that $\mathcal{P}os = (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \xrightarrow{\curlywedge} \mathcal{G}$. Therefore by Corollary 10.3.1, $\mathcal{P}os = \mathcal{D}ef \xrightarrow{\curlywedge} \mathcal{G}$. In order to prove it, we need to better understand the relations among the operations of $\mathcal{P}os$ and the operations of $\wp^{\downarrow}(Sub)$. Since $\gamma_{Sub}$ is co-additive, for every family of formulas $\{\, \phi_i \,\}_{i \in I} \subseteq \mathcal{P}os$ it clearly holds

$$\bigwedge_{i \in I} \gamma_{Sub}(\phi_i) = \gamma_{Sub}(\bigwedge_{i \in I} \phi_i) \tag{10.3.1}$$

As proved by Filé and Ranzato [1994], $\mathcal{P}os$ is not disjunctive, and thus $\gamma_{Sub}$ is not additive. Nevertheless, if we consider variables only, the join operation ($\vee$) on $\mathcal{P}os$ corresponds to the concrete join on $\wp^{\downarrow}(Sub)$.

**Lemma 10.3.2** *Let* $\{ y_j \}_{j\in J} \subseteq Var$.

$$\bigvee_{j\in J} \gamma_{Sub}(y_j) = \gamma_{Sub}(\bigvee_{j\in J} y_j)$$

**Proof.**
$$\begin{array}{rcl}
\gamma_{Sub}(\bigvee_{j\in J} y_j) & = & \{ \theta \in Sub \mid \forall\sigma \leq \theta.\ \sigma\,\texttt{grounds} \bigvee_{j\in J} y_j \} \\
& = & \{ \theta \in Sub \mid \theta\,\texttt{grounds} \bigvee_{j\in J} y_j \} \\
& = & \{ \theta \in Sub \mid \exists j \in J.\ \theta\,\texttt{grounds}\, y_j \} \\
& = & \bigvee_{j\in J} \{ \theta \in Sub \mid \theta\,\texttt{grounds}\, y_j \} \\
& = & \bigvee_{j\in J} \{ \theta \in Sub \mid \forall\sigma \leq \theta.\ \sigma\,\texttt{grounds}\, y_j \} \\
& = & \bigvee_{j\in J} \gamma_{Sub}(y_j).
\end{array}$$

■

A similar result holds also for the implication. In general, the classic implication $\Rightarrow$ does not coincide with the intuitionistic one. If we consider implications between a conjunction of variables and a disjunction of variables only, then they are isomorphic.

**Lemma 10.3.3** *Let* $X = \bigwedge_{i\in I} x_i$ *and* $Y = \bigvee_{j\in J} y_j$ *with* $x_i, y_j \in Var$.

$$\gamma_{Sub}(X \Rightarrow Y) = \gamma_{Sub}(X) \rightarrow \gamma_{Sub}(Y)$$

**Proof.**

$$\begin{array}{rl}
& \gamma_{Sub}(X \Rightarrow Y) \\
= & \{ \theta \in Sub \mid \forall\sigma \leq \theta.\ \sigma\,\texttt{grounds}\, X \Rightarrow Y \} \\
= & \{ \theta \in Sub \mid \forall\sigma \leq \theta.\ \sigma\,\texttt{grounds}\, X \Rightarrow \sigma\,\texttt{grounds}\, Y \} \\
= & \{ \theta \in Sub \mid \forall\sigma \leq \theta.\ (\forall i \in I\ \sigma\,\texttt{grounds}\, x_i) \Rightarrow (\exists j \in J\ \sigma\,\texttt{grounds}\, y_j) \} \\
= & \{ \theta \in Sub \mid \forall\sigma \leq \theta.\ (\forall i \in I\ \sigma \in \gamma_{Sub}(x_i)) \Rightarrow (\exists j \in J\ \sigma \in \gamma_{Sub}(y_j)) \} \\
= & \{ \theta \in Sub \mid \forall\sigma \leq \theta.\ \sigma \in \gamma_{Sub}(X) \Rightarrow \sigma \in \gamma_{Sub}(Y) \} \\
= & \gamma_{Sub}(X) \rightarrow \gamma_{Sub}(Y).
\end{array}$$

The last step holds because of the explicit representation for the element $a \rightarrow b$, i.e., $a \rightarrow b = \{ \theta \in Sub \mid \forall\sigma \leq \theta\ \sigma \in a \Rightarrow \sigma \in b \}$. ■

The idea is to find a normal form for elements in $\mathcal{P}os$ where disjunction is allowed between variables only, and (classic) implication is allowed between a conjunction and a disjunction of variables only. It is well known that every (classic) formula can be put in conjunctive normal form, i.e., it can be written as conjunction of disjunctions. Moreover, every disjunction is a clause. Therefore we can write it as a unique implication, by putting all the negative variables on the left and the positive variables on the right.

**Lemma 10.3.4** *Each formula $\phi \in \mathcal{P}os$ is equivalent to a formula*

$$\bigwedge_{k\in K}(\bigwedge_{i\in I} x_{i,k} \Rightarrow \bigvee_{j\in J} y_{j,k})$$

*in $\mathcal{P}os$ for suitable $x_{i,k}, y_{j,k} \in Var$.*

**Proof.** Since $\mathcal{P}os$ is a Boolean algebra, every formula can be put in disjunctive normal form (as conjunction of disjunctions) where every disjunction contains both variables and negated variables, i.e., formulas of kind $x \Rightarrow \bot$. Therefore we have only to prove that every disjunction can be transformed into our normal form. By exploiting the identity: $(\bigvee_{j\in J} y_j) \vee \bigvee_{i\in I}(x_i \Rightarrow \bot) = (\bigwedge_{i\in I} x_i) \Rightarrow (\bigvee_{j\in J} y_j)$ we obtain the result. ■

This result helps us to characterize the isomorphic image of $\mathcal{P}os$ in the concrete domain $\wp^{\downarrow}(Sub)$, i.e., $\gamma_{Sub}(\mathcal{P}os)$. The idea is to obtain the concretization of each abstract formula by simply replacing the abstract operations with the concrete ones. In some sense, this correspondence may seem counter-intuitive, since the concrete domain $\wp^{\downarrow}(Sub)$ is not a boolean algebra, as next example shows.

**Example 10.3.5** In a boolean algebra $(a \wedge b) \Rightarrow c = (a \Rightarrow c) \vee (b \Rightarrow c)$ holds. Let $X = \gamma_{Sub}(x)$, $Y = \gamma_{Sub}(y)$ and $Z = \gamma_{Sub}(z)$, where $x, y, z \in Var$. Consider in $\wp^{\downarrow}(Sub)$ the formula

$$(X \wedge Y) \rightarrow Z = \{\, \theta \in Sub \mid \forall \delta \leq \theta \;\, \delta \in X \wedge Y \;\Rightarrow \delta \in Z \,\}.$$

We show that $(X \rightarrow Z) \vee (Y \rightarrow Z)$ is strictly contained in $(X \wedge Y) \rightarrow Z$. In fact, the substitution $\{\, z \leftarrow f(x, y) \,\}$ belongs to $(X \wedge Y) \rightarrow Z$ but not to $(X \rightarrow Z)$ nor to $(Y \rightarrow Z)$. ■

The idea is to transform each formula in $\mathcal{P}os$ in normal form and then to exploit the isomorphism given by Lemmata 10.3.2 and 10.3.3, as shown in the next example.

**Example 10.3.6** Let $x \vee (x \Leftrightarrow y)$ be a formula in $\mathcal{P}os$. We look for its concretization $\gamma_{Sub}(x \vee (x \Leftrightarrow y))$. First we transform $x \vee (x \Leftrightarrow y)$ in normal form, which always exists by Lemma 10.3.4.

$$\begin{aligned} & x \vee (x \Leftrightarrow y) = x \vee ((x \Rightarrow y) \wedge (y \Rightarrow x)) = \\ = \; & (x \vee (x \Rightarrow y)) \wedge (x \vee (y \Rightarrow x)) = (x \vee \neg x \vee y) \wedge (x \vee \neg y \vee x) = \\ = \; & x \vee \neg y \vee x = y \Rightarrow x \end{aligned}$$

Next we transform $y \Rightarrow x$ by using Lemma 10.3.3 and obtain $\gamma_{Sub}(y \Rightarrow x) = \gamma_{Sub}(y) \rightarrow \gamma_{Sub}(x)$, which is a formula in $\wp^{\downarrow}(Sub)$. ■

Therefore, we have a constructive method to transform formulas in $\mathcal{P}os$ into formulas in $\wp^{\downarrow}(Sub)$ by simply replacing variables with their concretizations and abstract operations with the corresponding concrete ones. Let $\phi = \bigwedge_{k\in K}(\bigwedge_{i\in I} x_{i,k} \Rightarrow \bigvee_{j\in J} y_{j,k})$ be a generic element in $\mathcal{P}os$. The concretization of $\phi$, $\gamma_{Sub}(\phi)$ is precisely $\bigwedge_{k\in K}(\bigwedge_{i\in I}\gamma_{Sub}(x_{i,k}) \rightarrow \bigvee_{j\in J}\gamma_{Sub}(y_{j,k}))$, as proved by the following theorem.

**Theorem 10.3.7**

$$\gamma_{Sub}(\mathcal{P}os) = \{ \bigwedge_{k\in K}(\bigwedge_{i\in I} X_{i,k} \rightarrow \bigvee_{j\in J} Y_{j,k}) \mid X_{i,k}, Y_{j,k} \in \gamma_{Sub}(Var) \}^{3}.$$

**Proof.** By Lemma 10.3.4, we know that

$$\mathcal{P}os = \{ \bigwedge_{k\in K}(\bigwedge_{i\in I} x_{i,k} \Rightarrow \bigvee_{j\in J} y_{j,k}) \mid x_{i,k}, y_{j,k} \in Var \}.$$

$$\begin{array}{lll} & \gamma_{Sub}(\bigwedge_{k\in K}(\bigwedge_{i\in I} x_{i,k} \Rightarrow \bigvee_{j\in J} y_{j,k})) & \text{[by Eq. (10.3.1)]} \\ = & \bigwedge_{k\in K}\gamma_{Sub}(\bigwedge_{i\in I} x_{i,k} \Rightarrow \bigvee_{j\in J} y_{j,k}) & \text{[by Lemma 10.3.3]} \\ = & \bigwedge_{k\in K}(\gamma_{Sub}(\bigwedge_{i\in I} x_{i,k}) \rightarrow \gamma_{Sub}(\bigvee_{j\in J} y_{j,k})) & \text{[by Eq. (10.3.1) and Lemma 10.3.2]} \\ = & \bigwedge_{k\in K}(\bigwedge_{i\in I}\gamma_{Sub}(x_{i,k}) \rightarrow \bigvee_{j\in J}\gamma_{Sub}(y_{j,k})). & \end{array}$$

Hence we obtain:

$$\begin{array}{rl} \gamma_{Sub}(\mathcal{P}os) & = \{ \bigwedge_{k\in K}(\bigwedge_{i\in I}\gamma_{Sub}(x_{i,k}) \rightarrow \bigvee_{j\in J}\gamma_{Sub}(y_{j,k})) \mid x_{i,k}, y_{j,k} \in Var \} \\ & = \{ \bigwedge_{k\in K}(\bigwedge_{i\in I} X_{i,k} \rightarrow \bigvee_{j\in J} Y_{j,k}) \mid X_{i,k}, Y_{j,k} \in \gamma_{Sub}(Var) \} \end{array}$$

where $X_{i,k} = \gamma_{Sub}(x_{i,k})$ and $Y_{j,k} = \gamma_{Sub}(y_{j,k})$. ∎

From now on, since $\mathcal{P}os$ is isomorphic to $\gamma_{Sub}(\mathcal{P}os)$ and Theorem 10.3.7 allows us to describe concrete objects in $\gamma_{Sub}(\mathcal{P}os)$ as formulas, we identify $\mathcal{P}os$ with its concretization $\gamma_{Sub}(\mathcal{P}os)$.

Our aim is to describe the domain $\mathcal{P}os$ by using the Heyting completion refinement only. Therefore, we look for a normal form for concrete objects which involves meet and intuitionistic implication only and it does not contain any disjunction. Note that, in the concrete formulas of Theorem 10.3.7, disjunctions are allowed between variables only. The last step is then to find a representation in $\wp^{\downarrow}(Sub)$ for unions of variables in terms of intuitionistic implications. First of all, we prove that every concrete double implication can be rearranged as a disjunction and a (single) implication.

**Lemma 10.3.8** *Let $X = \bigwedge_{i\in I} x_i$, $Y = \bigvee_{j\in J} y_j$ and $Z = \bigvee_{k\in K} z_k$, where $x_i, y_j, z_k \in \gamma_{Sub}(Var)$.*

$$(X \rightarrow Y) \rightarrow Z = (X \vee Z) \wedge (Y \rightarrow Z)$$

[3] Note that $X_{i,k}, Y_{j,k}$ denote sets of substitutions.

**Proof.** Since $\wp^{\downarrow}(Sub)$ is a cHa, the following inclusions trivially hold.

- $(X \vee Z) \wedge (Y \to Z) \subseteq (X \to Y) \to Z$,
- $(X \to Y) \to Z \subseteq Y \to Z$.

We have to prove that $(X \to Y) \to Z \subseteq X \vee Z$. Let $\theta \in (X \to Y) \to Z$ and assume, for contradiction, that $\theta \notin X$ and $\theta \notin Z$. Therefore $\theta \notin Y$ (otherwise, from $\theta \in Y \to Z$ it would follow that $\theta \in Z$).

Let $V = \bigcup_{i \in I} vars(\theta(x_i)) \cup \bigcup_{j \in J} vars(\theta(y_j))$, where, by abusing the notation, we identify a concrete object $x \in \gamma_{Sub}(Var)$ with the corresponding variable in $Var$. Let $n$ be a new variable, not in $V$. Consider the substitution $\delta$ so defined: $\forall u \in Var$

$$\delta(u) = \begin{cases} n & \text{if } u \in V \setminus dom(\theta), \\ u & \text{otherwise.} \end{cases}$$

Note that $dom(\delta) \cap dom(\theta) = \varnothing$. Therefore the substitution $\theta \circ \delta$ is well defined.

- $\forall u \in Term$ $\delta(u)$ is ground iff $u$ is ground, since $\delta$ transforms variables to variables without affecting the groundness.
- $\forall v \in V \setminus dom(\theta)$, $vars(\delta(v)) = \{ n \}$. By definition of $\delta$.
- $\forall i \in I$, it holds:
  - if $vars(\theta(x_i)) = \varnothing$ then $vars(\theta \circ \delta(x_i)) = \varnothing$
  - if $vars(\theta(x_i)) \neq \varnothing$ then $vars(\theta \circ \delta(x_i)) = \{ n \}$
  - since $\theta \notin \bigwedge_{i \in I} x_i$, there exists $i \in I$ such that $vars(\theta(x_i)) \neq \varnothing$, and thus $vars(\theta \circ \delta(x_i)) = \{ n \}$.

  Therefore, $\bigcup_{i \in I} vars(\theta \circ \delta(x_i)) = \{ n \}$.
- since $\theta \notin \bigvee_{j \in J} y_j$, $\forall j \in J$ $vars(\theta(y_j)) \neq \varnothing$ and thus $vars(\theta \circ \delta(y_j)) = \{ n \}$.

We prove now that $\theta \circ \delta \in \bigwedge_{i \in I} x_i \to \bigvee_{j \in J} y_j$.

$$\begin{array}{lll} & & \theta \circ \delta \in \bigwedge_{i \in I} x_i \to \bigvee_{j \in J} y_j \\ \Leftrightarrow & \forall \sigma \leq \theta \circ \delta & \sigma \in \bigwedge_{i \in I} x_i \Rightarrow \sigma \in \bigvee_{j \in J} y_j \\ \Leftrightarrow & \forall \sigma \leq \theta \circ \delta & \theta \circ \delta \circ \sigma \in \bigwedge_{i \in I} x_i \Rightarrow \theta \circ \delta \circ \sigma \in \bigvee_{j \in J} y_j \quad [\theta \circ \delta \circ \sigma = \sigma] \\ \Leftrightarrow & \forall \sigma \leq \theta \circ \delta & \forall i \in I \; \theta \circ \delta \circ \sigma \in x_i \Rightarrow \exists j \in J \; \theta \circ \delta \circ \sigma \in y_j \end{array}$$

$$\begin{array}{lll}
\Leftrightarrow & \forall\sigma \leq \theta\circ\delta & \forall i \in I\ vars(\theta\circ\delta\circ\sigma(x_i)) = \varnothing \Rightarrow \\
 & & \exists j \in J\ vars(\theta\circ\delta\circ\sigma(y_j)) = \varnothing \\
\Leftrightarrow & \forall\sigma \leq \theta\circ\delta & (\forall i \in I\ \forall v \in vars(\theta\circ\delta(x_i))\ vars(\sigma(v)) = \varnothing) \Rightarrow \\
 & & \exists j \in J\ \forall v \in vars(\theta\circ\delta(y_j)) vars(\sigma(v)) = \varnothing \\
\Leftrightarrow & \forall\sigma \leq \theta\circ\delta & (\forall v \in \cup_{i\in I} vars(\theta\circ\delta(x_i))\ vars(\sigma(v)) = \varnothing) \Rightarrow \\
 & & \exists j \in J\ \forall v \in vars(\theta\circ\delta(y_j)) vars(\sigma(v)) = \varnothing \\
\Leftrightarrow & \forall\sigma \leq \theta\circ\delta & \forall v \in \{\, n \,\} vars(\sigma(v)) = \varnothing \Rightarrow \\
 & & \exists j \in J\ \forall v \in \{\, n \,\} vars(\sigma(v)) = \varnothing \quad [vars(\theta\circ\delta(y_j)) = \{\, n \,\}] \\
\Leftrightarrow & \forall\sigma \leq \theta\circ\delta & vars(\sigma(n)) = \varnothing \Rightarrow vars(\sigma(n)) = \varnothing
\end{array}$$

Therefore $\theta \circ \delta \in \bigwedge_{i\in I} x_i \rightarrow \bigvee_{j\in J} y_j$. Moreover $\theta \circ \delta \notin Z$ since $\theta \notin Z$ by hypothesis. Thus $\theta \circ \delta \notin (X \rightarrow Y) \rightarrow Z$, and therefore $\theta \notin (X \rightarrow Y) \rightarrow Z$, which is a contradiction. It follows that either $\theta \in X$ or $\theta \in Z$, i.e., $\theta \in X \vee Z$. ■

The previous lemma can be generalized to deal with conjunctions of implications. Such a result helps us to decompose complex formulas with disjunctions and implications into double implications and therefore to derive a transformation from formulas in $\mathcal{P}os$ into formulas in $\wp^{\downarrow}(Sub)$ which allows $\wedge$ and $\rightarrow$ only. In particular, note that implication is used with at most two levels of nesting. This suggests to construct $\mathcal{P}os$ as a double implication.

**Lemma 10.3.9** *Let $x_{i,k}, y_{j,k}, z_k \in \gamma_{Sub}(Var)$, $X_k = \bigwedge_{i\in I} x_{i,k}$, $Y_k = \bigvee_{j\in J} y_{j,k}$ and $Z = \bigvee_{k=1..n} z_k$. It holds that:*

$$(\bigwedge_{k=1}^{n}(X_k \rightarrow Y_k)) \rightarrow Z =$$
$$(\bigvee_{k=1}^{n} X_k \vee Z) \wedge ((\bigwedge_{k=1}^{n} Y_k) \rightarrow Z) \wedge \bigwedge_{G\subset\{\,1..n\,\}} ((\bigwedge_{k\in G} Y_k) \rightarrow (\bigvee_{k\in\overline{G}} X_k \vee Z))$$

*where $\overline{G} = \{\,1..n\,\} \setminus G$.*

**Proof.** The proof is by induction on $n$. For $n = 1$ the thesis boils down to proving $(X \rightarrow Y) \rightarrow Z = (X \vee Z) \wedge (Y \rightarrow Z)$, which is the result in Lemma 10.3.8. Let assume it for $n - 1$ and prove for $n$.

$$(\bigwedge_{k=1}^{n}(X_k \rightarrow Y_k)) \rightarrow Z = (X_n \rightarrow Y_n) \rightarrow (\bigwedge_{k=1}^{n-1}(X_k \rightarrow Y_k)) \rightarrow Z \quad \text{[by ind. hypothesis]}$$
$$=(X_n \rightarrow Y_n) \rightarrow \{(\bigvee_{k=1}^{n-1} X_k \vee Z) \wedge ((\bigwedge_{k=1}^{n-1} Y_k) \rightarrow Z) \wedge$$
$$\bigwedge_{G\subset\{\,1..n-1\,\}} ((\bigwedge_{k\in G} Y_k) \rightarrow (\bigvee_{k\in\overline{G}} X_k \vee Z))\}$$

$$
\begin{aligned}
=&((X_n \to Y_n) \to (\bigvee_{k=1}^{n-1} X_k \vee Z)) \wedge ((X_n \to Y_n) \to ((\bigwedge_{k=1}^{n-1} Y_k) \to Z)) \wedge \\
&((X_n \to Y_n) \to \bigwedge_{G \subset \{1..n-1\}} ((\bigwedge_{k \in G} Y_k) \to (\bigvee_{k \in \overline{G}} X_k \vee Z))) \qquad \text{[by Lemma 10.3.8]} \\
=&((X_n \vee \bigvee_{k=1}^{n-1} X_k \vee Z) \wedge (Y_n \to (\bigvee_{k=1}^{n-1} X_k \vee Z))) \\
&\wedge ((\bigwedge_{k=1}^{n-1} Y_k) \to ((X_n \vee Z) \wedge (Y_n \to Z))) \\
&\wedge \bigwedge_{G \subset \{1..n-1\}} ((X_n \to Y_n) \to ((\bigwedge_{k \in G} Y_k) \to (\bigvee_{k \in \overline{G}} X_k \vee Z))) \\
=&(\bigvee_{k=1}^{n} X_k \vee Z) \wedge (Y_n \to (\bigvee_{k=1}^{n-1} X_k \vee Z)) \wedge ((\bigwedge_{k=1}^{n-1} Y_k) \to (X_n \vee Z)) \wedge \\
&(\bigwedge_{k=1}^{n-1} Y_k \to (Y_n \to Z)) \wedge \bigwedge_{G \subset \{1..n-1\}} ((\bigwedge_{k \in G} Y_k) \to ((X_n \to Y_n) \to (\bigvee_{k \in \overline{G}} X_k \vee Z))) \\
=&(\bigvee_{k=1}^{n} X_k \vee Z) \wedge (Y_n \to (\bigvee_{k=1}^{n-1} X_k \vee Z)) \wedge ((\bigwedge_{k=1}^{n-1} Y_k) \to (X_n \vee Z)) \wedge ((\bigwedge_{k=1}^{n} Y_k) \to Z) \wedge \\
&\bigwedge_{G \subset \{1..n-1\}} ((\bigwedge_{k \in G} Y_k) \to ((X_n \vee \bigvee_{k \in \overline{G}} X_k \vee Z) \wedge (Y_n \to (\bigvee_{k \in \overline{G}} X_k \vee Z)))) \\
=&(\bigvee_{k=1}^{n} X_k \vee Z) \wedge (Y_n \to (\bigvee_{k=1}^{n-1} X_k \vee Z)) \wedge (\bigwedge_{k=1}^{n-1} Y_k \to (X_n \vee Z)) \wedge ((\bigwedge_{k=1}^{n} Y_k) \to Z) \wedge \\
&\bigwedge_{G \subset \{1..n-1\}} ((\bigwedge_{k \in G} Y_k \to (X_n \vee \bigvee_{k \in \overline{G}} X_k \vee Z)) \wedge (\bigwedge_{k \in G} Y_k \to (Y_n \to (\bigvee_{k \in \overline{G}} X_k \vee Z)))) \\
=&(\bigvee_{k=1}^{n} X_k \vee Z) \wedge (Y_n \to (\bigvee_{k=1}^{n-1} X_k \vee Z)) \wedge ((\bigwedge_{k=1}^{n-1} Y_k) \to (X_n \vee Z)) \wedge ((\bigwedge_{k=1}^{n} Y_k) \to Z) \wedge \\
&\bigwedge_{G \subset \{1..n-1\}} (((\bigwedge_{k \in G} Y_k) \to (\bigvee_{k \in \overline{G} \cup \{n\}} X_k \vee Z)) \wedge ((\bigwedge_{k \in G \cup \{n\}} Y_k) \to (\bigvee_{k \in \overline{G}} X_k \vee Z))) \\
=&(\bigvee_{k=1}^{n} X_k \vee Z) \wedge ((\bigwedge_{k=1}^{n} Y_k) \to Z) \wedge \bigwedge_{G \subset \{1..n\}} ((\bigwedge_{k \in G} Y_k) \to (\bigvee_{k \in \overline{G}} X_k \vee Z)).
\end{aligned}
$$

■

We first exploit this result to prove that $\mathcal{P}os$ is closed for the Heyting completion operator.

**Theorem 10.3.10** $\mathcal{P}os = \mathcal{P}os \xrightarrow{\curlywedge} \mathcal{P}os$

**Proof.** By Proposition 9.1.3, $\mathcal{P}os \xrightarrow{\curlywedge} \mathcal{P}os \sqsubseteq \mathcal{P}os$. For the other direction, by Theorem 10.3.7, every element in $\mathcal{P}os$ is of the form $\bigwedge_{k\in K}(X_k \to Y_k)$ where $X_k = \bigwedge_{i\in I} x_{i,k}$ and $Y_k = \bigvee_{j\in J} y_{j,k}$. By construction, any element in $\mathcal{P}os \xrightarrow{\curlywedge} \mathcal{P}os$ is of the form

$$\bigwedge_{m\in M}((\bigwedge_{k\in K}(X_k^m \to Y_k^m)) \to (\bigwedge_{a\in A}(W_a^m \to Z_a^m)))$$

where $W_a^m = \bigwedge_{b\in B} w_{b,a}^m$ and $Z_a^m = \bigvee_{c\in C} y_{c,a}^m$. $\mathcal{P}os$ being a Moore family, it suffices to prove that for each $m \in M$, every object $(\bigwedge_{k\in K}(X_k^m \to Y_k^m)) \to (\bigwedge_{a\in A}(W_a^m \to Z_a^m))$ is in $\mathcal{P}os$. Moreover:

$$\begin{array}{rl} & (\bigwedge_{k\in K}(X_k^m \to Y_k^m)) \to (\bigwedge_{a\in A}(W_a^m \to Z_a^m)) \\ = & \bigwedge_{a\in A}((\bigwedge_{k\in K}(X_k^m \to Y_k^m)) \to (W_a^m \to Z_a^m)) \\ = & \bigwedge_{a\in A}(\bigwedge_{k\in K}(X_k^m \to Y_k^m) \wedge W_a^m) \to Z_a^m \end{array}$$

Again, it suffices to prove that for each $a \in A$, every element of the form $(\bigwedge_{k\in K}(X_k^m \to Y_k^m) \wedge W_a^m) \to Z_a^m$ is in $\mathcal{P}os$. Moreover by writing $W_a^m = \bigwedge_{b\in B} w_{b,a}^m = \bigwedge_{b\in B}(\top \to w_{b,a}^m)$, we only need to prove that every formula of the form $\bigwedge_{k\in K}(X_k \to Y_k) \to Z$ is in $\mathcal{P}os$. By Lemma 10.3.9, it holds:

$$\bigwedge_{k\in K}(X_k \to Y_k) \to Z = (\bigvee_{k\in K} X_k \vee Z) \wedge ((\bigwedge_{k\in K} Y_k) \to Z) \wedge \bigwedge_{G\subset K}((\bigwedge_{k\in G} Y_k) \to (\bigvee_{k\in \overline{G}} X_k \vee Z)).$$

Note that, by distributivity of $\wp^{\downarrow}(Sub)$ and Theorem 10.3.7, all formulas $\bigvee_{k\in K} X_k \vee Z$, $(\bigwedge_{k\in K} Y_k) \to Z$ and $(\bigwedge_{k\in G} Y_k) \to (\bigvee_{k\in\overline{G}} X_k \vee Z)$ belong to $\mathcal{P}os$. Therefore, the conjunction also belongs to $\mathcal{P}os$. ∎

We are now in the position to prove the main theorem of this chapter, that is $\mathcal{P}os$ is the double Heyting completion of $\mathcal{G}$.

**Theorem 10.3.11** $\mathcal{P}os = (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \xrightarrow{\curlywedge} \mathcal{G}$

**Proof.** By Theorem 10.3.10, $\mathcal{P}os = (\mathcal{P}os \xrightarrow{\curlywedge} \mathcal{P}os) \xrightarrow{\curlywedge} \mathcal{P}os \sqsubseteq (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \xrightarrow{\curlywedge} \mathcal{G}$.

For the other direction, by Theorem 10.3.7, every element in $\mathcal{P}os$ is a conjunction of formulas of the form $\bigwedge_{i\in I} x_i \to \bigvee_{j\in J} y_j$, where $x_i, y_j \in \gamma_{Sub}(Var)$. By Lemma 10.3.9, it holds:

$$\begin{array}{rll} & (\bigwedge_{j\in J}(y_j \to \bot)) \to \bot & \\ = & (\bigwedge_{j\in J}(y_j \to \bigwedge_{v\in Var} v)) \to \bigwedge_{u\in Var} u) & \\ = & \bigwedge_{u\in Var}(\bigwedge_{j\in J} \bigwedge_{v\in Var}(y_j \to v)) \to u) & \text{[by Lemma 10.3.9]} \\ = & \bigwedge_{u\in Var}((\bigvee_{j\in J} y_j \vee u) \wedge (\bigwedge_{v\in Var} v \to \bigwedge_{u\in Var} u) \wedge & \\ & \bigwedge(\bigwedge_{v\in Var} v \to (\bigvee_{j\in J} y_j \vee u))) & \\ = & \bigwedge_{u\in Var}((\bigvee_{j\in J} y_j \vee u) \wedge (\bot \to \bot) \wedge \bigwedge(\bot \to (\bigvee_{j\in J} y_j \vee u))) & \\ = & \bigwedge_{u\in Var}(\bigvee_{j\in J} y_j \vee u) & \\ = & \bigvee_{j\in J} y_j \vee \bigwedge_{u\in Var} u & \\ = & \bigvee_{j\in J} y_j \vee \bot = \bigvee_{j\in J} y_j & \end{array}$$

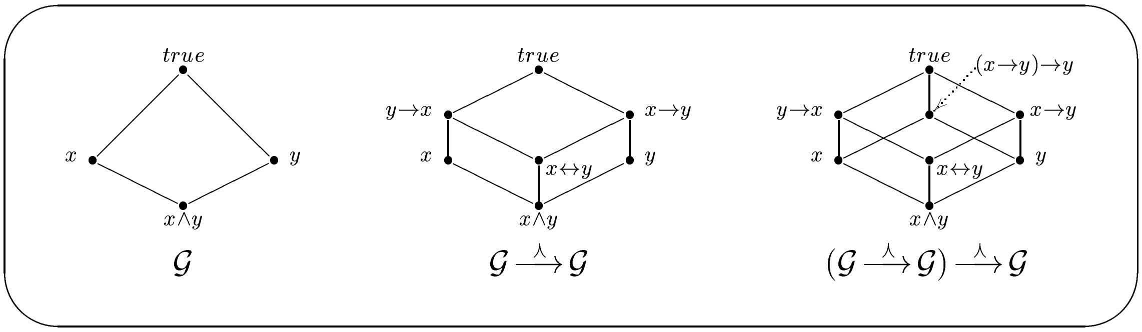



Figure 10.1: $\mathcal{G}$, $\mathcal{D}ef$ and $\mathcal{P}os$

It follows that

$$\begin{array}{rl} & \bigwedge_{i\in I} x_i \to \bigvee_{j\in J} y_j \\ = & \bigwedge_{i\in I} x_i \to ((\bigwedge_{j\in J}(y_j \to \bot)) \to \bot) \\ = & (\bigwedge_{i\in I} x_i \wedge \bigwedge_{j\in J}(y_j \to \bot)) \to \bot. \end{array}$$

By definition of Heyting completion, $\bigwedge_{i\in I} x_i \wedge \bigwedge_{j\in J}(y_j \to \bot) \in \mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}$ and $(\bigwedge_{i\in I} x_i \wedge (\bigwedge_{j\in J}(y_j \to \bot))) \to \bot \in (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \xrightarrow{\curlywedge} \mathcal{G}$. Thus $\mathcal{P}os \subseteq (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \xrightarrow{\curlywedge} \mathcal{G}$. ■

By Corollary 10.3.1, we can immediately derive the relations between the domains $\mathcal{P}os$ and $\mathcal{D}ef$.

**Corollary 10.3.12** $\mathcal{P}os = \mathcal{D}ef \xrightarrow{\curlywedge} \mathcal{G} = \mathcal{D}ef \xrightarrow{\curlywedge} \mathcal{D}ef$

**Proof.** By Corollary 10.3.1, we obtain $\mathcal{P}os = \mathcal{D}ef \xrightarrow{\curlywedge} \mathcal{G}$. For the other equation:

$$\begin{array}{rll} & \mathcal{D}ef \xrightarrow{\curlywedge} \mathcal{D}ef & \text{[by Corollary 10.3.1]} \\ = & (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \xrightarrow{\curlywedge} (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) & \text{[by Proposition 9.3.1]} \\ = & ((\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \sqcap \mathcal{G}) \xrightarrow{\curlywedge} \mathcal{G} & \text{[by Prop. 9.1.3]} \\ = & (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \xrightarrow{\curlywedge} \mathcal{G} & \end{array}$$

■

For $Var = \{x, y\}$, the domains $\mathcal{G}$, $\mathcal{D}ef$ and $\mathcal{P}os$ are depicted in Figure 10.1.

The first important consequence of Theorem 10.3.11 is that the domain $\mathcal{P}os$ is constructed by using the domain $\mathcal{G}$ only and by refining the operation of unification. We do not need to "invent" the domain, to prove that it is actually an abstraction of $\wp^{\downarrow}(Sub)$, nor to prove that it refines $\mathcal{G}$, since all these properties hold by construction. In our framework, $\mathcal{P}os$ arises as the natural refinement of $\mathcal{G}$ and $\mathcal{D}ef$. Moreover, we get a normal form for elements in $\gamma_{Sub}(\mathcal{P}os)$. This result allows us to see $\mathcal{P}os$ into $\wp^{\downarrow}(Sub)$ and to directly deal with $\gamma_{Sub}(\mathcal{P}os)$. In fact, when we use a formula in $\mathcal{P}os$, actually we use an equivalence class with respect to (classic) logic equivalence

$(KL)$. In our formalization, we do not need to use equivalence classes any longer. Moreover, the normal form of concrete formulas is indeed very natural and preserves the intuitive meaning of abstract formulas.

As a consequence of Theorems 10.3.10 and 10.3.11, it immediately follows that $\mathcal{P}os$ is the most abstract domain which contains $\mathcal{G}$ and which is closed with respect to Heyting completion.

**Corollary 10.3.13** *$\mathcal{P}os$ is the most abstract solution to the equation $X = \mathcal{G} \sqcap (X \xrightarrow{\curlywedge} X)$.*

**Proof.** Straightforward by Theorem 10.3.10 and 10.3.11. ∎

Therefore, it turns out that $Pos$ is the greatest solution to the disjunctive completeness problem, as defined in Theorem 9.4.1(c) and it 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, providing that at least one of the arguments belongs to $Pos$. Moreover, this corollary yields a representation result for elements in $\mathcal{P}os$. It precisely states that an element in $\wp^{\downarrow}(Sub)$ belongs to $\mathcal{P}os$ if and only if it can be written by using (concrete) meets and implications only, starting from objects in $\mathcal{G}$. Hence, it completely characterizes the concrete image of $\mathcal{P}os$, not only the abstract objects. It is worth noting that, differently from previous characterizations of $\mathcal{P}os$ [Armstrong et al. 1994; Cortesi et al. 1996], all these results hold on the concrete domain. In addition, our construction provides a characterization for the abstract disjunction. It implies that a disjunctive formula belongs to $\mathcal{P}os$ if and only if it can be written by using $\wedge$ and $\rightarrow$ only. It would be impossible to state a similar result with classic propositional formulas, since it is well known that each formula can be written by means of $\wedge$ and $\Leftrightarrow$ only.

This result allows us to answer the question "why $\mathcal{P}os$ is considered optimal without being disjunctive", also from an intuitive point of view. We know that a disjunctive formula belongs to $\mathcal{P}os$ if and only if it can be put in implicational form. From a logic point of view, the elements useful for the analysis are implications only (as they can be reduced by *modus ponens*). Since a good domain should contain all and only those joins which are indeed useful, we would only include those joins that can be reduced, which, in turn, are exactly the joins that $\mathcal{P}os$ contains. This explains the result in [Filé and Ranzato 1998] that $\mathcal{P}os$ is complete with respect to $\curlyvee(\mathcal{P}os)$, i.e., the domains $\mathcal{P}os$ and $\curlyvee(\mathcal{P}os)$ have the same precision with respect to groundness analysis.

### 10.3.1 Condensing of $Pos$

As proved in Corollary 10.3.13, $Pos$ is the greatest solution of the recursive domain equation $X = \mathcal{G} \sqcap X \overset{\curlywedge}{\rightarrow} X$. This confirms, in view of Theorem 8.2.3, a result by Marriott and Søndergaard [1993], which was derived from some results in Langen [1990], saying that $Pos$ is condensing. Moreover, $Pos$ clearly satisfies the hypothesis of Theorem 8.2.4. 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 8.2.3 and 8.2.4 is that the concrete domain is a cHa, and thus meet distributes over (possibly infinite) joins. Therefore, even if they are not disjunctive, still no loss of precision is accumulated by distributing the meet (abstract unification) over nondeterministic computations, which are modeled by abstract disjunction.

### 10.3.2 On disjunctive completion of $\mathcal{P}os$

We proved that we can not further refine $\mathcal{P}os$ by Heyting completion and [Filé and Ranzato 1998] showed that it is superfluous to use $\curlyvee(\mathcal{P}os)$ instead of $\mathcal{P}os$. We now show that, even combining disjunctive completion and Heyting completion, we still obtain the domain $\curlyvee(\mathcal{P}os)$, when starting from $\mathcal{G}$. Formally, we look for the solution of the equation $X = \curlyvee(\mathcal{P}os) \sqcap (X \overset{\curlywedge}{\longrightarrow} X)$. We show that also $\curlyvee(\mathcal{P}os)$ is closed with respect to Heyting completion. Before proving this result, we need a technical lemma to precisely describe the objects in $\mathcal{P}os$.

**Lemma 10.3.14** *Let $x_i, y_j \in \gamma_{Sub}(Var)$, for $i \in I$ and $j \in J$.*

$$\theta \in \bigwedge_{i\in I} x_i \to \bigvee_{j\in J} y_j \Leftrightarrow \exists j \in J \ vars(\theta(y_j)) \subseteq \cup_{i\in I} vars(\theta(x_i))$$

**Proof.** First note that, by definition, it holds:

$$\begin{array}{ll} & \theta \in \bigwedge_{i\in I} x_i \to \bigvee_{j\in J} y_j \\ \Leftrightarrow & \forall \delta \leq \theta \ \delta \in \bigwedge_{i\in I} x_i \Rightarrow \delta \in \bigvee_{j\in J} y_j \\ \Leftrightarrow & \forall \delta \leq \theta \ \cup_{i\in I} vars(\delta(x_i)) = \varnothing \Rightarrow \exists j \in J \ vars(\delta(y_j)) = \varnothing. \end{array}$$

For contradiction, assume that $\forall j \in J$ it holds $vars(\theta(y_j)) \not\subseteq \cup_{i\in I} vars(\theta(x_i))$. It follows that:

$$\begin{array}{ll} & \forall j \in J \ vars(\theta(y_j)) \smallsetminus \cup_{i\in I} vars(\theta(x_i)) \neq \varnothing \\ \Rightarrow & \forall j \in J \ \exists w_j \in vars(\theta(y_j)) \smallsetminus \cup_{i\in I} vars(\theta(x_i)) \end{array}$$

Let $\delta_1 = \theta \circ \{\, u \leftarrow a \mid u \in \cup_{i\in I} vars(\theta(x_i)) \,\}$ where $a$ is a constant of the language. Then, $\forall i \in I$ it holds that $\delta_1 \in x_i$. In fact:

$$\begin{aligned}\delta_1(x_i) &= \theta \circ \{\, u \leftarrow a \mid u \in \cup_{i\in I} vars(\theta(x_i)) \,\}(x_i)\\ &= \{\, u \leftarrow a \mid u \in \cup_{i\in I} vars(\theta(x_i)) \,\}(\theta(x_i))\end{aligned}$$

Therefore, $vars(\delta_1(x_i)) = \varnothing$ and $\delta_1 \in \bigwedge_{i\in I} x_i$. But $\forall j \in J$ we have that $\delta_1 \notin y_j$ since $vars(\delta_1(y_j)) = vars(\{\, u \leftarrow a \mid u \in \cup_{i\in I} vars(\theta(x_i)) \,\}(\theta(y_j)))$ and there always exists $w_j \in vars(\theta(y_j)) \setminus \cup_{i\in I} vars(\theta(x_i))$.

The other direction follows from the fact that if $vars(\theta(y_j)) \subseteq \cup_{i\in I} vars(\theta(x_i))$ then $\forall \delta \leq \theta\ vars(\delta(y_j)) \subseteq \cup_{i\in I} vars(\delta(x_i))$. ∎

**Theorem 10.3.15** $\curlyvee(\mathcal{P}os) = \curlyvee(\mathcal{P}os) \xrightarrow{\curlywedge} \curlyvee(\mathcal{P}os)$.

**Proof.** By Proposition 9.3.5, $\curlyvee(\mathcal{P}os) \xrightarrow{\curlywedge} \curlyvee(\mathcal{P}os) = \mathcal{P}os \xrightarrow{\curlywedge} \curlyvee(\mathcal{P}os)$. Moreover, by Proposition 9.3.4, each object in $\curlyvee(\mathcal{P}os)$ is a disjunction of elements in $\mathcal{P}os$. Therefore, it suffices to prove that for any $x_i, y_j, w_m^k, z_n^k \in \gamma_{Sub}(Var)$ it holds:

$$(\bigwedge_{i\in I} x_i \to \bigvee_{j\in J} y_j) \to \bigvee_{k\in K}(\bigwedge_{m\in M} w_m^k \to \bigvee_{n\in N} z_n^k) = \bigvee_{k\in K}((\bigwedge_{i\in I} x_i \to \bigvee_{j\in J} y_j) \to (\bigwedge_{m\in M} w_m^k \to \bigvee_{n\in N} z_n^k)).$$

By Lemma 10.3.14 it holds:

$$\begin{aligned}
& \theta \in (\textstyle\bigwedge_{i\in I} x_i \to \bigvee_{j\in J} y_j) \to \bigvee_{k\in K}(\bigwedge_{m\in M} w_m^k \to \bigvee_{n\in N} z_n^k)\\
\Leftrightarrow\ & \forall \delta \leq \theta\ (\exists j \in J\ vars(\delta(y_j)) \subseteq \cup_{i\in I} vars(\delta(x_i))) \Rightarrow\\
& \qquad (\exists k \in K\ \exists n \in N\ vars(\delta(z_n^k)) \subseteq \cup_{m\in M} vars(\delta(w_m^k)))\\
\Leftrightarrow\ & (\exists j \in J\ vars(\theta(y_j)) \subseteq \cup_{i\in I} vars(\theta(x_i))) \Rightarrow\\
& \qquad (\exists k \in K \exists n \in N\ vars(\theta(z_n^k)) \subseteq \cup_{m\in M} vars(\theta(w_m^k)))\\
\Leftrightarrow\ & \exists k \in K\ ((\exists j \in J\ vars(\theta(y_j)) \subseteq \cup_{i\in I} vars(\theta(x_i))) \Rightarrow\\
& \qquad (\exists n \in N\ vars(\theta(z_n^k)) \subseteq \cup_{m\in M} vars(\theta(w_m^k))))\\
\Leftrightarrow\ & \exists k \in K\ (\forall \delta \leq \theta\ (\exists j \in J\ vars(\delta(y_j)) \subseteq \cup_{i\in I} vars(\delta(x_i))) \Rightarrow\\
& \qquad (\exists n \in N\ vars(\delta(z_n^k)) \subseteq \cup_{m\in M} vars(\delta(w_m^k))))\\
\Leftrightarrow\ & \theta \in \textstyle\bigvee_{k\in K}((\bigwedge_{i\in I} x_i \to \bigvee_{j\in J} y_j) \to (\bigwedge_{m\in M} w_m^k \to \bigvee_{n\in N} z_n^k))
\end{aligned}$$

∎

Therefore, $\curlyvee(\mathcal{P}os)$ is greatest solution of the following equation.

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

Intuitively, it means that $\curlyvee(\mathcal{P}os)$ is the most concrete domain we can obtain by disjunctive and Heyting completion refinements starting from $\mathcal{G}$ and, in view of the result of completeness of $\mathcal{P}os$ with respect to $\curlyvee(\mathcal{P}os)$, it formally confirms that $\mathcal{P}os$ is definitely the best domain for groundness analysis.

## 10.4 Optimality in the literature

The result that $\mathcal{P}os$ is closed with respect to Heyting completion helps us to better understand the properties of abstract unification of $\mathcal{P}os$ with respect to $\mathcal{D}ef$'s. In the classic literature of abstract interpretation, many notions of optimality have been proposed in order to compare abstract domains (cf. [Cousot and Cousot 1979b; Cortesi et al. 1996; Giacobazzi and Ranzato 1997a]). In any abstract domain, abstract meet is always the best correct approximation of the concrete one by construction. So we do not need to distinguish between abstract and concrete meet. Moreover, since unification is exactly meet operation ([Palamidessi 1990]), it follows that abstract unification is always the best correct approximation of the concrete one, by simply choosing as abstract unification the meet operation (Corollary 6.6 in [Marriot et al. 1994]). Therefore, for both domains $\mathcal{P}os$ and $\mathcal{D}ef$, abstract unification is the best correct approximation of concrete unification, i.e., for each $A, B \in \wp^{\downarrow}(Sub)$ it holds:

$$\begin{array}{rcl} \alpha_{\mathcal{P}os}(A) \wedge \alpha_{\mathcal{P}os}(B) & = & \alpha_{\mathcal{P}os}(\alpha_{\mathcal{P}os}(A) \wedge \alpha_{\mathcal{P}os}(B)) \\ \alpha_{\mathcal{D}ef}(A) \wedge \alpha_{\mathcal{D}ef}(B) & = & \alpha_{\mathcal{D}ef}(\alpha_{\mathcal{D}ef}(A) \wedge \alpha_{\mathcal{D}ef}(B)). \end{array}$$

On the contrary, $\mathcal{P}os$ and $\mathcal{D}ef$ are not complete for unification, i.e., it does not hold neither $\alpha_{\mathcal{P}os}(A) \wedge \alpha_{\mathcal{P}os}(B) = \alpha_{\mathcal{P}os}(A \wedge B)$ nor $\alpha_{\mathcal{D}ef}(A) \wedge \alpha_{\mathcal{D}ef}(B) = \alpha_{\mathcal{D}ef}(A \wedge B)$[4], as shown in the next example.

**Example 10.4.1** Let $\theta = \{ X \leftarrow f(Y, a) \}$ and $\sigma = \{ X \leftarrow f(a, Y) \}$. It holds:

$$\alpha_{\mathcal{P}os}(\downarrow \theta \wedge \downarrow \sigma) = \alpha_{\mathcal{D}ef}(\downarrow \theta \wedge \downarrow \sigma) = X \wedge Y$$

$$\alpha_{\mathcal{P}os}(\downarrow \theta) \wedge \alpha_{\mathcal{P}os}(\downarrow \sigma) = \alpha_{\mathcal{D}ef}(\downarrow \theta) \wedge \alpha_{\mathcal{D}ef}(\downarrow \sigma) = X \leftrightarrow Y.$$

■

Since $\mathcal{D}ef = \mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}$, by Theorem 6.2.3, $\mathcal{D}ef$ enjoys the following property.

$$\alpha_{\mathcal{G}}(\alpha_{\mathcal{G}}(A) \wedge B) = \alpha_{\mathcal{G}}(\alpha_{\mathcal{G}}(A) \wedge \alpha_{\mathcal{D}ef}(B))$$

Analogously, since $\mathcal{P}os = \mathcal{D}ef \xrightarrow{\curlywedge} \mathcal{D}ef$, it holds:

$$\alpha_{\mathcal{D}ef}(\alpha_{\mathcal{D}ef}(A) \wedge B) = \alpha_{\mathcal{D}ef}(\alpha_{\mathcal{D}ef}(A) \wedge \alpha_{\mathcal{P}os}(B))$$

In particular, they are the most abstract domains which satisfy the above equalities. In addition, since $\mathcal{P}os$ is closed for Heyting completion, by Theorem 9.4.1(c), it is a

[4] A similar (negative) result for $\mathcal{P}os \cup \{ \varnothing \}$ already appears in [Cortesi et al. 1996]

solution of a disjunctive completeness problem and therefore the following equality holds.

$$\alpha_{\mathcal{P}os}(\alpha_{\mathcal{P}os}(A) \wedge B) = \alpha_{\mathcal{P}os}(A) \wedge \alpha_{\mathcal{P}os}(B).$$

Note that the domain $\mathcal{D}ef$ does not satisfy this property, as proved in the next example.

**Example 10.4.2** Let $\theta = \{ X \leftarrow a \}$, $\sigma = \{ Y \leftarrow a \}$ and $\eta = \{ X \leftarrow Y \}$. It holds:

$$\alpha_{\mathcal{D}ef}((\downarrow \theta \vee \downarrow \sigma) \wedge (\downarrow \eta)) = \alpha_{\mathcal{D}ef}(X \wedge Y) = X \wedge Y$$

$$\alpha_{\mathcal{D}ef}(\alpha_{\mathcal{D}ef}(\downarrow \theta \vee \downarrow \sigma) \wedge (\downarrow \eta)) = \alpha_{\mathcal{D}ef}(Sub \wedge (\downarrow \eta)) = \alpha_{\mathcal{D}ef}(\downarrow \eta) = X \Leftrightarrow Y.$$

■

It is worth noting that $\mathcal{P}os$ is the most abstract domain which contains $\mathcal{G}$ and satisfies the above property. In this way we obtain a precise characterization of $\mathcal{P}os$, which depends only on the domain $\mathcal{G}$. The next theorem shows this result.

**Corollary 10.4.3** *$\mathcal{P}os$ is the most abstract domain which contains $\mathcal{G}$ and enjoys the following property:* $\forall A, B \in \wp^{\downarrow}(Sub)$

$$\alpha_{\mathcal{P}os}(\alpha_{\mathcal{P}os}(A) \wedge B) = \alpha_{\mathcal{P}os}(A) \wedge \alpha_{\mathcal{P}os}(B)$$

**Proof.** By Theorem 10.3.10 and Theorem 9.4.1, it follows that $\alpha_{\mathcal{P}os}(\alpha_{\mathcal{P}os}(A) \wedge B) = \alpha_{\mathcal{P}os}(A) \wedge \alpha_{\mathcal{P}os}(B)$. Moreover, by Theorem 10.3.11, $\mathcal{P}os$ is the most abstract domain which contains $\mathcal{G}$ and is closed with respect to Heyting completion. Therefore, it is the most abstract domain which contains $\mathcal{G}$ and satisfies the above condition. ■

The importance of this characterization is that it allows us to discriminate between the precision of abstract unification in the domains $\mathcal{P}os$ and $\mathcal{D}ef$. In fact, for both domains, abstract unification is the best correct approximation of the concrete unification and, in addition, neither $\mathcal{P}os$ nor $\mathcal{D}ef$ are complete for unification. The disjunctive completeness problem precisely captures the threshold between the two domains. Moreover, this is not only a result of differentiation, but a complete characterization of $\mathcal{P}os$ since it turns out that $\mathcal{P}os$ is the most abstract domain which is a solution of a disjunctive completeness problem, which could be taken as an alternative definition of $\mathcal{P}os$.

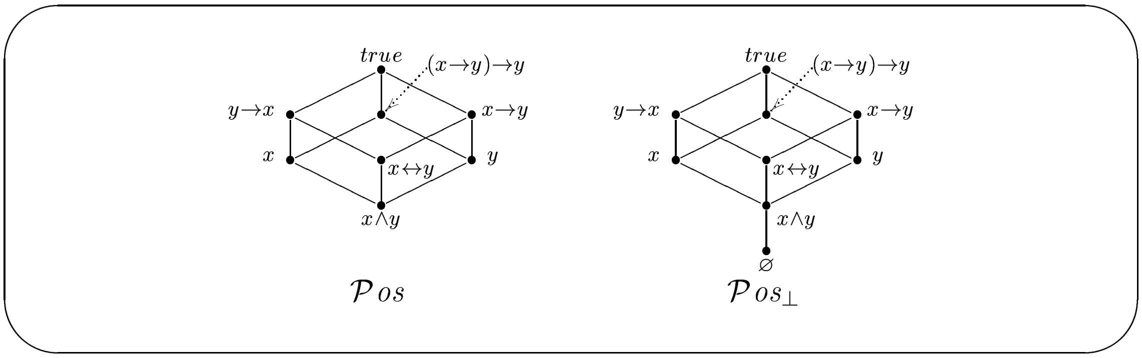



Figure 10.2: $\mathcal{P}os$ and $\mathcal{P}os_\perp$

## 10.5 Reachability analysis

Some authors (cf. [Cortesi et al. 1996; Marriott and Søndergaard 1993; Armstrong et al. 1994]) consider a slightly different definition of the domain $\mathcal{P}os$, by including also the empty set of substitutions as least element. Given a domain $A$, we denote by $A_\perp$ the domain $A \cup \{ \varnothing \}$, which is always a Moore family. For $Var = \{ x, y \}$, the domain $\mathcal{P}os$ and $\mathcal{P}os_\perp$ are depicted in Figure 10.2.

The domain $\mathcal{P}os_\perp$ is used to perform reachability analysis. When a fixpoint semantics is computed, the starting value is usually the least element of the domain. In case the semantics of a program is empty, the concrete semantics is $\varnothing$, while the corresponding abstract element in $\mathcal{P}os$ is the conjunction of all variables of interest. The abstract domain can not distinguish between programs with empty semantics and programs whose answer solutions are all ground. In order to make this distinction clear and, therefore, to add an extra precision to the domain, $\mathcal{P}os_\perp$ is used. When the abstract semantics is $\varnothing$, we conclude that no predicate has a solution. In some sense, no solution is "reached" during the computation.

We show that our reconstruction of $\mathcal{P}os$ applies to $\mathcal{P}os_\perp$ as well, by considering as starting domain the domain $\mathcal{G}_\perp = \mathcal{G} \cup \{ \varnothing \}$. This domain can be seen as the reduced product of $\mathcal{G}$ and the two-points domain $\{ \top, \varnothing \}$, which is the basic domain for reachability analysis. Note that $\mathcal{G}_\perp$ is the most abstract domain which encodes groundness and reachability analysis. The next theorem shows that $\mathcal{P}os_\perp$ can be obtained by $\mathcal{G}_\perp$ with three steps of Heyting completion operator and that $\mathcal{P}os_\perp$ is still closed.

**Theorem 10.5.1** *The following equations hold.*

- $\mathcal{G}_\perp = \curlywedge(Var \cup \{ \varnothing \})$
- $\mathcal{D}ef_\perp = \mathcal{G}_\perp \xrightarrow{\curlywedge} \mathcal{G}_\perp$

- $\mathcal{P}os_\perp = (\mathcal{G}_\perp \xrightarrow{\curlywedge} \mathcal{G}_\perp) \xrightarrow{\curlywedge} \mathcal{G}_\perp$
- $\mathcal{P}os_\perp = \mathcal{P}os_\perp \xrightarrow{\curlywedge} \mathcal{P}os_\perp$

**Proof.** $\mathcal{G}_\perp = \curlywedge(Var \cup \{\varnothing\})$ trivially holds by definition of $\mathcal{G}_\perp$.

Recall that the least element $\perp$ of $\mathcal{P}os$ is $\bigwedge Var$. We first show that $\perp \to \varnothing = \varnothing$. The proof is by contradiction. Suppose there exists a substitution $\theta \in \perp \to \varnothing$. By definition, $\forall \delta \leq \theta \;\; \delta \in \perp \;\Rightarrow\; \delta \in \varnothing$, that is $\forall \delta \leq \theta \;\; \delta \notin \perp$. But given any substitution $\theta$, there always exists an instance which grounds all the variables in $Var$, since $Var$ is finite. Therefore $\perp \to \varnothing = \varnothing$. It follows that, for each $a \in \mathcal{P}os$, $a \to \varnothing \leq \perp \to \varnothing = \varnothing$, that is $a \to \varnothing = \varnothing$. Moreover for each $a \in \mathcal{P}os$, $\varnothing \to a = \top$. It follows that

- $\mathcal{G}_\perp \xrightarrow{\curlywedge} \mathcal{G}_\perp \subseteq \mathcal{D}ef_\perp$
- $(\mathcal{G}_\perp \xrightarrow{\curlywedge} \mathcal{G}_\perp) \xrightarrow{\curlywedge} \mathcal{G}_\perp \subseteq \mathcal{P}os_\perp$
- $\mathcal{P}os_\perp \xrightarrow{\curlywedge} \mathcal{P}os_\perp \subseteq \mathcal{P}os_\perp$

We prove the other inclusions by monotonicity.

- $\mathcal{P}os_\perp \subseteq \mathcal{P}os_\perp \xrightarrow{\curlywedge} \mathcal{P}os_\perp$ trivially holds.
- $\mathcal{D}ef_\perp = \mathcal{D}ef \cup \{\varnothing\} = (\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \cup \{\varnothing\} \subseteq (\mathcal{G}_\perp \xrightarrow{\curlywedge} \mathcal{G}_\perp) \cup \{\varnothing\} = \mathcal{G}_\perp \xrightarrow{\curlywedge} \mathcal{G}_\perp$
- $\mathcal{P}os_\perp = \mathcal{P}os \cup \{\varnothing\} = ((\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}) \xrightarrow{\curlywedge} \mathcal{G}) \cup \{\varnothing\} \subseteq ((\mathcal{G}_\perp \xrightarrow{\curlywedge} \mathcal{G}_\perp) \xrightarrow{\curlywedge} \mathcal{G}_\perp) \cup \{\varnothing\} = (\mathcal{G}_\perp \xrightarrow{\curlywedge} \mathcal{G}_\perp) \xrightarrow{\curlywedge} \mathcal{G}_\perp$ ∎

As a consequence, an analogous result of Corollary 10.4.3 holds for the domain $\mathcal{P}os_\perp$, as shown in next Corollary.

**Corollary 10.5.2** *$\mathcal{P}os_\perp$ is the most abstract domain which contains $\mathcal{G}_\perp$ and enjoys the following property: $\forall A, B \in \wp^\downarrow(Sub)$*

$$\alpha_{\mathcal{P}os_\perp}(\alpha_{\mathcal{P}os_\perp}(A) \wedge B) = \alpha_{\mathcal{P}os_\perp}(A) \wedge \alpha_{\mathcal{P}os_\perp}(B)$$

**Proof.** The proof is analogous to that one for Corollary 10.4.3. ∎

## 10.6 Program independent domains

In the previous sections, we have fixed a finite set of variables $V \subset_f \mathcal{V}$, where $\mathcal{V}$ is denumerable, and shown how to automatically build the domains ${\mathcal{G}_\perp}^V$, ${\mathcal{D}ef_\perp}^V$ and ${\mathcal{P}os_\perp}^V$. This naturally induces an ordering on the abstract domains for groundness analysis built from different sets of variables. A question which naturally arises is if it is possible to construct a $\mathcal{P}os$-like abstract domain which is independent from the fixed variables. We show that the answer is positive and this domain is exactly the reduced product of all domains ${\mathcal{P}os_\perp}^V$ for each finite set $V$. Since all operators used in the construction process are monotone, the ordering is simply the set theoretical one on underlying sets of variables.

**Proposition 10.6.1** *Let $V \subseteq W \subset_f \mathcal{V}$.*

1. ${\mathcal{G}_\perp}^W \sqsubseteq {\mathcal{G}_\perp}^V$;
2. ${\mathcal{D}ef_\perp}^W \sqsubseteq {\mathcal{D}ef_\perp}^V$;
3. ${\mathcal{P}os_\perp}^W \sqsubseteq {\mathcal{P}os_\perp}^V$.

**Proof.** Straightforwardly from the fact that all operators used in the definitions are monotone. ∎

Moreover, for the basic case of the domain $\mathcal{G}$, the following property holds.

**Proposition 10.6.2** *Let $V, W \subset_f \mathcal{V}$. ${\mathcal{G}_\perp}^V \sqcap {\mathcal{G}_\perp}^W = {\mathcal{G}_\perp}^{V \cup W}$;*

**Proof.**
$$\begin{aligned}
{\mathcal{G}_\perp}^V \sqcap {\mathcal{G}_\perp}^W &= \curlywedge(V \cup \{\varnothing\}) \sqcap \curlywedge(W \cup \{\varnothing\}) \\
&= \curlywedge(V \cup W \cup \{\varnothing\}) \\
&= {\mathcal{G}_\perp}^{V \cup W}.
\end{aligned}$$
∎

The idea is to exploit the ordering on these domains to construct program independent domains, which are suitable for each program of any dimension and are still based on the same concrete domain $\wp^{\downarrow}(Sub)$. In this section we show that this is always possible and that the new domains still enjoy the same properties of the corresponding finite domains.

The basic idea is to consider the reduced product of all domains for any finite set of variables. Clearly, such domains are able to deal with any variable in $\mathcal{V}$. We define the domain $\mathcal{G}^\omega$, $\mathcal{D}ef^\omega$ and $\mathcal{P}os^\omega$ as follows:

$$\begin{aligned}
\mathcal{G}^\omega &\stackrel{\text{def}}{=} \sqcap_{V \subset_f \mathcal{V}} \mathcal{G}^V \\
\mathcal{D}ef^\omega &\stackrel{\text{def}}{=} \sqcap_{V \subset_f \mathcal{V}} \mathcal{D}ef^V \\
\mathcal{P}os^\omega &\stackrel{\text{def}}{=} \sqcap_{V \subset_f \mathcal{V}} \mathcal{P}os^V
\end{aligned}$$

We show that all the relations proved in the finite case still hold. We start, as before, from the basic domain $\mathcal{G}^\omega$ and reconstruct the other two as Heyting completion and double Heyting completion. As before, we let a variable $v$ denote the set of substitutions which ground $v$. First note that, all domains contain the element $\perp$ obtained as the infinite conjunction of all variables in $\mathcal{V}$.

**Proposition 10.6.3** $\bigwedge \mathcal{V} = \varnothing$.

**Proof.** $\bigwedge \mathcal{V} = \{\, \theta \in Sub \mid \forall v \in \mathcal{V}\ \theta \in v \,\} = \{\, \theta \in Sub \mid \forall v \in V\ vars(\theta(v)) = \varnothing \,\}$. By definition, there exists no substitution which grounds an infinite set of variables. Since $|\mathcal{V}|$ is infinite, it implies that $\bigwedge \mathcal{V} = \varnothing$. ■

**Corollary 10.6.4** *The following properties hold.*

1. $\mathcal{G}^\omega = \sqcap_{V \subset_f \mathcal{V}} {\mathcal{G}_\perp}^V$
2. $\mathcal{D}ef^\omega = \sqcap_{V \subset_f \mathcal{V}} {\mathcal{D}ef_\perp}^V$
3. $\mathcal{P}os^\omega = \sqcap_{V \subset_f \mathcal{V}} {\mathcal{P}os_\perp}^V$

**Proof.** By definition, all domains contain all variables $v \in \mathcal{V}$ and are closed by possibly infinite meets. Therefore they contain $\bigwedge \mathcal{V}$. ■

Before proving the properties and relations among the new domains, we show that they do not contain any infinite conjunction different from $\perp$, that is any possibly infinite conjunction either reduces to a finite one or is equivalent to $\perp$. This allows us to keep on using a finitary propositional logic instead of an infinitary one. We first prove that $\mathcal{G}^\omega$ can be built by using only finite formulas and the empty set.

**Proposition 10.6.5** $\mathcal{G}^\omega = \{\, \bigwedge V \mid V \subset_f \mathcal{V} \,\} \cup \{\, \varnothing \,\}$

**Proof.** One direction is trivial, that is $\mathcal{G}^\omega \supseteq \{\, \bigwedge V \mid V \subset_f \mathcal{V} \,\} \cup \{\, \varnothing \,\}$. Let $x \in \mathcal{G}^\omega$. Then there exists a family of variables $v_i \in \mathcal{V}$ for $i \in I$ such that $x = \bigwedge_{i \in I} v_i$. If $|\{\, v_i \,\}_{i \in I}|$ is finite, then there exists $W \subset_f \mathcal{V}$ such that $\bigwedge_{i \in I} v_i = \bigwedge_{v_i \in W} v_i$ and the thesis trivially follows. Let $|\{\, v_i \,\}_{i \in I}|$ be infinite. Then $\bigwedge_{i \in I} v_i = \{\, \theta \in Sub \mid \forall i \in I\ \theta \in v_i \,\} = \{\, \theta \in Sub \mid \forall i \in I\ vars(\theta(v_i)) = \varnothing \,\}$ Since $|\{\, v_i \,\}_{i \in I}|$ is infinite, by definition of substitution, there exists no substitution which grounds an infinite set of variables. Therefore $x = \varnothing$. ■

This result allows us to establish an isomorphism between the domain $\mathcal{G}^\omega$ and $\wp_f(\mathcal{V})$, analogously to the finite case. As suggested by Proposition 10.6.5, in order to obtain the isomorphism, we only need to add to $\wp_f(\mathcal{V})$ a least element $\perp$, by extending the ordering in the obvious way.

**Corollary 10.6.6** *$\mathcal{G}^\omega$ is isomorphic to $\wp_f(\mathcal{V}) \cup \{\perp\}$.*

**Proof.** Straightforward from Proposition 10.6.5 by simply representing subsets as conjunctions and mapping the empty set of substitutions to $\perp$. ■

It is worth noting that $\mathcal{G}^\omega$ is not isomorphic to $\wp(\mathcal{V})$, since the latter one contains distinct infinite sets of variables, all of which should be reduced to the empty set of substitutions. Therefore we obtain a Galois connection with the concrete domain, and not a Galois insertion. As a consequence, both domains $\mathcal{G}^\omega$, $\mathcal{D}ef^\omega$ and $\mathcal{P}os^\omega$ can be reconstructed from the corresponding finite ones without introducing infinite conjunctions nor infinite implications.

**Proposition 10.6.7** *The following properties hold.*

1. $\mathcal{G}^\omega = \cup_{V \subset_f \mathcal{V}} \mathcal{G}_\perp{}^V$
2. $\mathcal{D}ef^\omega = \cup_{V \subset_f \mathcal{V}} \mathcal{D}ef_\perp{}^V$
3. $\mathcal{P}os^\omega = \cup_{V \subset_f \mathcal{V}} \mathcal{P}os_\perp{}^V$

**Proof (Sketch).**

1. $\mathcal{G}^\omega = \cup_{V \subset_f \mathcal{V}} \mathcal{G}_\perp{}^V$ is straightforward by Proposition 10.6.5.
2. Let $a_i \in \cup_{V \subset_f \mathcal{V}} \mathcal{D}ef_\perp{}^\mathcal{V}$ for $i \in I$. We prove that $\bigwedge_{i \in I} a_i \in \cup_{V \subset_f \mathcal{V}} \mathcal{D}ef_\perp{}^\mathcal{V}$. If $\bigwedge_{i \in I} a_i$ is equivalent to either a finite conjunction of implications or $\top$ we have nothing to prove. Otherwise, we claim that $\bigwedge_{i \in I} a_i = \perp$. In fact, by definition, a substitution can involve only a finite set of variables. For contradiction, if there exists a substitution $\theta \in \bigwedge_{i \in I} a_i$, it would follows that either $\theta$ binds a variable to a term which contains infinite variables, i.e., an infinite term, or $\theta$ grounds an infinite number of variables. This is because $\bigwedge_{i \in I} a_i$ is not equivalent to a finite formula. But $\theta$ would not be a substitution, by definition. Therefore $\mathcal{D}ef^\omega = \cup_{V \subset_f \mathcal{V}} \mathcal{D}ef_\perp{}^V$ holds.
3. Analogous to the previous point. ■

The last characterization says that when constructing $\mathcal{G}^\omega$ starting from $\mathcal{G}_\perp{}^V$, we can simply take the union of all such domains and this is actually a Moore family, i.e., we do not need to add infinite conjunctions in order to obtain an abstract domain. This proves that when dealing with $\mathcal{G}^\omega$, $\mathcal{D}ef^\omega$ and $\mathcal{P}os^\omega$, we do not have to worry about infinite formulas and we can use a finitary logic to describe abstract objects. We now reconstruct the relations between the new domains.

**Theorem 10.6.8** *The following properties hold.*

*1.* $\mathcal{G}^\omega = \curlywedge(\mathcal{V})$

*2.* $\mathcal{D}ef^\omega = \mathcal{G}^\omega \xrightarrow{\curlywedge} \mathcal{G}^\omega$

*3.* $\mathcal{P}os^\omega = (\mathcal{G}^\omega \xrightarrow{\curlywedge} \mathcal{G}^\omega) \xrightarrow{\curlywedge} \mathcal{G}^\omega$

*4.* $\mathcal{P}os^\omega = \mathcal{P}os^\omega \xrightarrow{\curlywedge} \mathcal{P}os^\omega$

**Proof.** 1.
$$\begin{aligned}
\mathcal{G}^\omega &= \sqcap_{V \subset_f \mathcal{V}} {\mathcal{G}_\perp}^V \\
&= \sqcap_{V \subset_f \mathcal{V}} \curlywedge(V \cup \{ \varnothing \}) \\
&= \curlywedge(\cup_{V \subset_f \mathcal{V}} V \cup \{ \varnothing \}) \\
&= \curlywedge(\mathcal{V} \cup \{ \varnothing \}) && \text{[by Proposition 10.6.3]} \\
&= \curlywedge(\mathcal{V})
\end{aligned}$$

2. $\mathcal{D}ef^\omega = \sqcap_{V \subset_f \mathcal{V}} \mathcal{D}ef^V = \sqcap_{V \subset_f \mathcal{V}} (\mathcal{G}^V \xrightarrow{\curlywedge} \mathcal{G}^V) \subseteq \sqcap_{V \subset_f \mathcal{V}} (\mathcal{G}^\omega \xrightarrow{\curlywedge} \mathcal{G}^\omega) = \mathcal{G}^\omega \xrightarrow{\curlywedge} \mathcal{G}^\omega$

   By monotonicity of Heyting completion, it holds:

$$\begin{aligned}
\mathcal{G}^\omega \xrightarrow{\curlywedge} \mathcal{G}^\omega &= (\sqcap_{V \subset_f \mathcal{V}} \mathcal{G}^V) \xrightarrow{\curlywedge} (\sqcap_{W \subset_f \mathcal{V}} \mathcal{G}^W) \\
&= \sqcap_{W \subset_f \mathcal{V}} ((\sqcap_{V \subset_f \mathcal{V}} \mathcal{G}^V) \xrightarrow{\curlywedge} \mathcal{G}^W) \\
&= \sqcap_{W \subset_f \mathcal{V}} \sqcap_{V \subset_f \mathcal{V}} (\mathcal{G}^V \xrightarrow{\curlywedge} \mathcal{G}^W) \\
&\subseteq \sqcap_{W \subset_f \mathcal{V}} \sqcap_{V \subset_f \mathcal{V}} ((\mathcal{G}^V \sqcap \mathcal{G}^W) \xrightarrow{\curlywedge} (\mathcal{G}^W \sqcap \mathcal{G}^V)) \\
&\subseteq \sqcap_{W \subset_f \mathcal{V}} \sqcap_{V \subset_f \mathcal{V}} (\mathcal{G}^{V \cup W} \xrightarrow{\curlywedge} \mathcal{G}^{V \cup W}) \\
&\subseteq \sqcap_{W \subset_f \mathcal{V}} \sqcap_{V \subset_f \mathcal{V}} (\mathcal{D}ef^{V \cup W}) \\
&= \sqcap_{V \subset_f \mathcal{V}} (\mathcal{D}ef^V).
\end{aligned}$$

3. analogous to point (2) above.

4. by monotonicity of Heyting completion, $\mathcal{P}os^\omega \subseteq \mathcal{P}os^\omega \xrightarrow{\curlywedge} \mathcal{P}os^\omega$ holds. For the other inclusion:

$$\begin{aligned}
\mathcal{P}os^\omega \xrightarrow{\curlywedge} \mathcal{P}os^\omega &= (\sqcap_{V \subset_f \mathcal{V}} \mathcal{P}os^V) \xrightarrow{\curlywedge} (\sqcap_{W \subset_f \mathcal{V}} \mathcal{P}os^W) \\
&= \sqcap_{W \subset_f \mathcal{V}} ((\sqcap_{V \subset_f \mathcal{V}} \mathcal{P}os^V) \xrightarrow{\curlywedge} \mathcal{P}os^W) \\
&= \sqcap_{W \subset_f \mathcal{V}} \sqcap_{V \subset_f \mathcal{V}} (\mathcal{P}os^V \xrightarrow{\curlywedge} \mathcal{P}os^W) \\
&\subseteq \sqcap_{W \subset_f \mathcal{V}} \sqcap_{V \subset_f \mathcal{V}} (\mathcal{P}os^{V \cup W} \xrightarrow{\curlywedge} \mathcal{P}os^{V \cup W}) \\
&= \sqcap_{W \subset_f \mathcal{V}} \sqcap_{V \subset_f \mathcal{V}} (\mathcal{P}os^{V \cup W}) \\
&= \sqcap_{V \subset_f \mathcal{V}} (\mathcal{P}os^V).
\end{aligned}$$

■

Our approach in constructing a unique, program independent domain differs from [Cortesi et al. 1996] and [Marriot et al. 1994] mainly in the choice of the concrete domain. In [Cortesi et al. 1996], the authors consider a different concrete domain: $(\wp(Sub) \times \wp_f(\mathcal{V})) \cup \{ \top_c, \bot_c \}$. Our results show that, even keeping the simple concrete domain $\wp^{\downarrow}(Sub)$, we can construct program independent domains. Moreover, our approach allows us to compare different abstract domains for different sets of variables, while they are incomparable in the other approach. In [Marriot et al. 1994], a still different concrete domain is used, i.e., sets of ex-equations. Intuitively, ex-equations are existentially quantified conjunctions of basic term equations. The whole SLD resolution theory is then lifted to ex-equations, by extending the usual definitions of unification and semantics to this domain. It is known from [Palamidessi 1990] that the lattice of idempotent substitutions is isomorphic to the lattice of term equations. By definition, ex-equations are more expressive, and therefore more concrete, than simple equations. As a consequence, sets of ex-equations are more concrete than $\wp(Sub)$, and thus than $\wp^{\downarrow}(Sub)$. But, as shown by our construction, $\wp^{\downarrow}(Sub)$ is concrete enough to formalize program independent groundness analysis.

# Chapter 11

# Directionality in logic program analysis

Abstract

In this chapter, we apply Heyting completion to characterize directionality in analysis of downward closed properties of logic programs. We show that Heyting completion surfaces in most domains and applications aimed to detect directional information in type and groundness analysis.

The results presented in this chapter are a joint work with R. Giacobazzi and appeared in [Giacobazzi and Scozzari 1998].

## 11.1 Directionality in type inference

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

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

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 *Term* 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 by $p \in \langle I, O\rangle$, 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, for any abstract domain of types $\mathcal{T} \in uco(\wp^{\downarrow}(Term))$ and types $\tau_1, \tau_2 \in \mathcal{T}$, the implicational type $\tau_1 \to \tau_2 \in \mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$ exactly models the above notion of directional type $\langle \tau_1, \tau_2\rangle$.

**Theorem 11.1.1** *Let $P$ be a program, $\mathcal{T} \in uco(\wp^{\downarrow}(Term))$, $\tau_1, \tau_2 \in \mathcal{T}$, and $p$ be a predicate. Then the following equivalence holds.*

$$(\forall\theta.\ p(\bar{X}) \xrightarrow{\theta}{}^{*}_{P} \Box \Rightarrow \theta(\bar{X}) \in \tau_1 \to \tau_2) \iff (\forall \bar{t} \in \tau_1. \forall\sigma.\ p(\bar{t}) \xrightarrow{\sigma}{}^{*}_{P} \Box \Rightarrow \sigma(\bar{t}) \in \tau_2)$$

**Proof.** Assume that $\forall\theta.\ p(\bar{X}) \xrightarrow{\theta}{}^{*}_{P} \Box \Rightarrow \theta(\bar{X}) \in \tau_1 \to \tau_2$. 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 $\sigma(\bar{t})$. Let $\delta$ such that $\sigma(\bar{t}) = \delta(\bar{X})$ (it exists since $\bar{X}$ are new variables). By definition of $\longrightarrow^{*}_{P}$, there exists a substitution $\beta$ such that $\delta \leq \beta$ and $p(\bar{X}) \xrightarrow{\beta}{}^{*}_{P} \Box$. By hypothesis, $\beta(\bar{X}) \in \tau_1 \to \tau_2$. Since $\bar{t} \in \tau_1$ and $\tau_1$ is closed under instantiation, it follows that $\sigma(\bar{t}) \in \tau_1$. Moreover $\sigma(\bar{t}) = \delta(\bar{X}) \leq \beta(\bar{X})$. Therefore, from $\beta(\bar{X}) \in \tau_1 \to \tau_2$, it follows that $\sigma(\bar{t}) \in \tau_2$.

For the other direction, assume that $\forall \bar{t} \in \tau_1. \forall\sigma.\ p(\bar{t}) \xrightarrow{\sigma}{}^{*}_{P} \Box \Rightarrow \sigma(\bar{t}) \in \tau_2$, and let $\theta \in Sub$ such that $p(\bar{X}) \xrightarrow{\theta}{}^{*}_{P} \Box$. In order to prove that $\theta(\bar{X}) \in \tau_1 \to \tau_2$, we have to show that for every instance $\bar{t}$ of $\theta(\bar{X})$ 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, $\bar{t} \in \tau_2$. ∎

**Corollary 11.1.2** *Under the hypothesis of Theorem 11.1.1, $p$ 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, directional types in $\mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$, that satisfy the theorem above, can be obtained by abstract interpretation, 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]), with the domain $\mathcal{T} \xrightarrow{\curlywedge} \mathcal{T}$. The next example shows the behaviour of a simple type inference in a basic domain including integer and Boolean types, when considering directional types.

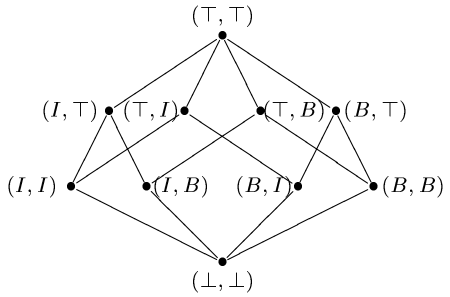



Figure 11.1: A simple domain of basic types $\mathcal{T}$

**Example 11.1.3** Let us consider the program $P = \{\, p(X,X).,\, r(2,Y) :\!-p(2,Y).\,\}$ and the simple domain of basic types $\mathcal{T}$ depicted in Figure 11.1, where symbols I (integer) and B (Boolean) have the obvious meaning as fixpoints of a closure operator on $\wp^{\downarrow}(\mathit{Term})$. 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 the arguments of the predicate $p$ have the same type. Instead, this information can be expressed in the refined domain $\mathcal{T} \stackrel{\curlywedge}{\longrightarrow} \mathcal{T}$.

Let us denote by $p(\tau,\tau)$ the conjunction $\bigwedge_{(\tau_1,\tau_2)\in\mathcal{T}} p(\tau_1,\tau_2) \to p(\tau_1\cap\tau_2, \tau_1\cap\tau_2)$, i.e., $p(I,\top) \to p(I,I) \wedge p(\top,I) \to p(I,I) \wedge p(B,\top) \to p(B,B) \wedge p(\top,B) \to p(B,B) \wedge p(I,B) \to p(\bot,\bot) \wedge p(B,I) \to p(\bot,\bot)$. 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 $\mathcal{T} \stackrel{\curlywedge}{\longrightarrow} \mathcal{T}$, we therefore obtain the more precise type patterns: $\{\, p(\tau,\tau), r(I,I)\,\}$. ■

## 11.2 From types to groundness

In view of the observation that groundness is a downward closed property, the abstract domain $\mathcal{G}$ can be viewed as some basic type definition, and ground dependencies in $\mathit{Def}$ can be interpreted as directional types. The key point here, as well as in directional types, is that any concrete element is a set of objects closed under instantiation and the concrete domain is, in general, a non-Boolean cHa. It is worth noting that a different concrete domain would completely change the interpretation of directionality. Consider the Boolean concrete domain $\wp(\mathit{Sub})$. For

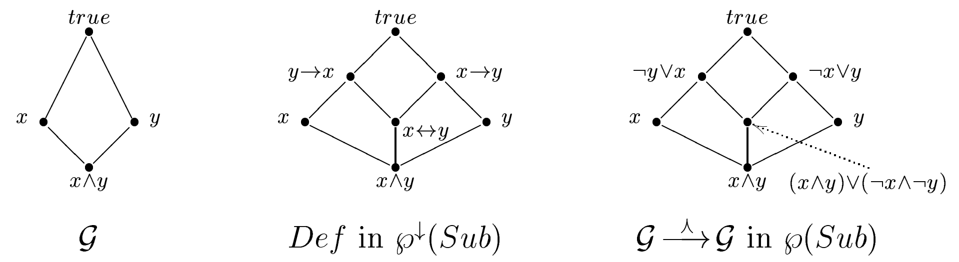



Figure 11.2: Heyting completion in $\wp(Sub)$

$Var = \{x, y\}$, $\mathcal{G}$, $\mathcal{D}ef$, and the Heyting completion of $\mathcal{G}$ in $\wp(Sub)$ are depicted in Figure 11.2. While $\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}$ 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 implications introduced by Heyting completion provide negation of properties. This is because, in Boolean lattices, $a \to b = \neg a \vee b$ and therefore, for $x, y \in \wp(Sub)$, $\neg x \vee y = \overline{\gamma_{Def}(x)} \cup \gamma_{Def}(y)$, while this is not the case in complete Heyting algebras. For instance, the empty substitution belongs to $\neg x \vee y$, as well as any substitution where $x$ is not ground, but they provide no groundness information. Clearly, because the set-theoretic complement of a set closed under instantiation is not, in general, closed under instantiation, then $\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}$ above is not an abstraction of $\wp^{\downarrow}(Sub)$. In this sense, when the concrete domain is $\wp(Sub)$, the analysis obtained from $\mathcal{G} \xrightarrow{\curlywedge} \mathcal{G}$ cannot be regarded as a type inference. From a computational point of view, the reason for this behaviour is that Heyting completion solves a disjunctive completeness problem for the concrete meet operation, which is set intersection, but, when considering $\wp(Sub)$ as concrete domain, standard unification is no longer set intersection (cf. Chapter 7).

## 11.3 Domains for relational program analysis in the literature

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], Codish and Demoen [1994], Dart [1991], Giacobazzi and Ranzato [1995], 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 [1979b], 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]. The authors used the directional information obtainable for groundness analysis in $Pos$ to efficiently implement a variety of type inference algorithms by abstract compilation. In Chapters 10 and 11 we have shown that both $Pos$ and directional types can be obtained by Heyting completion. Because Heyting completion precisely captures the common structure in all these applications to directionality, 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.

The use of (recursive) domain equations for specifying domains in static analysis was already considered in Benton [1992]. 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 as solution of abstract domain equations (e.g., see the characterization of $Pos$ in Corollary 10.3.13). However, these two methods differ in 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 completeness problem for some operation involved in the concrete semantics. In fact, completeness problems may be not related to any concrete data-type used in the considered programming language. This makes our approach applicable to deriving domains for any (possibly untyped) programming language. For instance, in the case of Heyting completion, the greatest solution to Eq. (9.4.1) 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 to allow these properties to be propagated on domain's objects by conjunction (see Theorem 9.4.1). 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.

# BIBLIOGRAPHY


AIKEN, A. AND LAKSHMAN, T. 1994. Directional type checking of logic programs. 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, 43–60.

AMATO, G. AND LEVI, G. 1997. Properties of the lattice of observables in logic programming. In *Proc. of the Italian-Portuguese-Spanish Joint Conf. on Declarative Programming, (APPIA-GULP-PRODE '97)*, M. Falaschi, M. Navarro, and A. Policriti, Eds. 175–187.

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.

BAGNARA, R., HILL, P., AND ZAFFANELLA, E. 1997. Set-sharing is redundant for pair-sharing. 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, 53–67.

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.

BENSALEM, S., LAKHNECH, Y., AND OWRE, S. 1998. InVeSt: A tool for the verification of invariants. In *Proc. of the 10th International Conference on Computer Aided Verification (CAV '98)*, A. Hu and M. Vardi, Eds. Lecture Notes in Computer Science, vol. 1427. Springer-Verlag, Berlin.

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

Berry, G. 1978. Stable models of typed lambda-calculi. In *Proceedings of the 5th International Colloquium on Automata, Languages and Programming* (*ICALP '78*). Lecture Notes in Computer Science, vol. 62. Springer-Verlag, Berlin, 375–387.

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.

Burn, G., Hankin, C., and Abramsky, S. 1986. Strictness analysis for higher-order functions. *Sci. Comput. Program. 7*, 249–278.

Clarke, E. M., Grumberg, O., and Long, D. E. 1994. Model checking and abstraction. *ACM Trans. Program. Lang. Syst. 16,* 5, 1512–1542.

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. Le 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.

Colby, C. 1997. Accumulated imprecision in abstract interpretation. In *Proc. of the 1st ACM Workshop on Automatic Analysis of Software* (*AAS '97*), R. Cleaveland and D. Jackson, Eds. ACM Press, New York, 77–89.

Comini, M. and Levi, G. 1994. An algebraic theory of observables. In *Proc. of the 1994 Int'l Logic Programming Symposium (ILPS '94)*, M. Bruynooghe, Ed. The MIT Press, Cambridge, Mass., 172–186.

Comini, M. and Meo, M. C. 1999. Compositionality properties of *SLD*-derivations. *Theoretical Computer Science 211,* 1-2, 275–309.

Cortesi, A., Filé, G., , and Winsborough, W. 1998. The quotient of an abstract interpretation. *Theor. Comput. Sci. 202,* 1-2, 163–192.

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., Filè, G., and Winsborough, W. 1991. *Prop* revisited: Propositional formula as abstract domain for groundness analysis. In *Proc. Sixth IEEE Symp. on Logic In Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., 322–327.

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. 1995. Completeness in abstract interpretation (invited talk). In *Proc. 1995 Joint Italian-Spanish Conference on Declarative Programming*. 37–38.

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

Cousot, P. 1997a. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation (Invited Paper). In *Proc. of the 13th Internat. Symp. on Mathematical Foundations of Programming Semantics* (*MFPS '97*), S. Brookes and M. Mislove, Eds. Electronic Notes in Theoretical Computer Science, vol. 6. Elsevier, Amsterdam.

Cousot, P. 1997b. Types as abstract interpretations (Invited Paper). In *Conference Record of the 24th ACM Symp. on Principles of Programming Languages* (*POPL '97*). ACM Press, New York, 316–331.

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. 1979a. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics 82,* 1, 43–57.

Cousot, P. and Cousot, R. 1979b. 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. 1992a. Abstract interpretation and application to logic programs. *J. Logic Program. 13,* 2-3, 103–179.

Cousot, P. and Cousot, R. 1992b. Inductive definitions, semantics and abstract interpretation. In *Conference Record of the 19th ACM Symp. on Principles of Programming Languages* (*POPL '92*). ACM Press, New York, 83–94.

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.

Cousot, P. and Cousot, R. 1997. Abstract interpretation of algebraic polynomial systems. In *Proc. 6th AMAST Conf.* Lecture Notes in Computer Science, vol. 1349. Springer-Verlag, Berlin, 138–154.

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.

Davis, K. 1992. A note on the choice of domains for projection-based program analysis. In *Proc. of the 1991 Glasgow Funct. Progr. Workshop.* Springer Workshops in Computing.

Davis, K. 1993. Analysing functions by projection-based backward abstraction. In *Proc. of the 1992 Glasgow Funct. Progr. Workshop.* Springer Workshops in Computing.

Debray, S. K. 1995. On the complexity of dataflow analysis of logic programs. *ACM Trans. Program. Lang. Syst. 17,* 2, 331–365.

Deutsch, A. 1997. On the complexity of escape analysis. In *Conference Record of the 24th ACM Symp. on Principles of Programming Languages* (*POPL '97*). ACM Press, New York, 358–371.

Falaschi, M., Levi, G., Martelli, M., and Palamidessi, C. 1989. Declarative modeling of the operational behavior of logic languages. *Theoretical Computer Science 69,* 3, 289–318.

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. *Theoretical Computer Science*. To appear. 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. 1996. "Optimal" collecting semantics for analysis in a hierarchy of logic program semantics. In *Proc. of the 13th Int'l Symposium on Theoretical Aspects of Computer Science (STACS '96)*, C. Puech and R. Reischuk, Eds. Lecture Notes in Computer Science, vol. 1046. Springer-Verlag, Berlin, 503–514.

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. 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. 1998a. Optimal domains for disjunctive abstract interpretation. *Sci. Comput. Program.* To appear. Preliminary version in *Proc. ESOP '96*, LNCS 1058, pages 141–155, Springer-Verlag, Berlin.

GIACOBAZZI, R. AND RANZATO, F. 1998b. The reduced relative power operation of abstract domains. *Theoretical Computer Science*. To appear. 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 *Static Analysis, Proceedings of the Fifth International Static Analysis Symposium SAS 98*, G. Levi, Ed. Lecture Notes in Computer Science, vol. 1503. Springer-Verlag, Berlin, 215–229.

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.

GIACOBAZZI, R. AND SCOZZARI, F. 1998. A logical model for relational abstract domains. *ACM Transactions on Programming Languages and Systems 20,* 5, 1067–1109.

GIERZ, G., HOFMANN, K., KEIMEL, K., LAWSON, J., MISLOVE, M., , AND SCOTT, D. 1980. *A Compendium of Continuous Lattices*. Springer-Verlag, Berlin.

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.

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 MAHER, M. J. 1994. Constraint logic programming: A survey. *J. Logic Programm. 19-20*, 503–581.

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.

Lassez, J.-L., Maher, M. J., and Marriott, K. 1988. Unification revisited. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, Los Altos, 587–625.

Le Charlier, B. and Van Hentenryck, P. 1994. Experimental evaluation of a generic abstract interpretation algorithm for Prolog. *ACM Transactions on Programming Languages and Systems 16,* 1, 35–101.

Marriot, K., Søndergaard, H., and Jones, N. D. 1994. Denotational abstract interpretation of logic programs. *ACM TOPLAS 16,* 3, 607–648.

Marriott, K. and Søndergaard, H. 1990. Abstract interpretation of logic programs: the denotational approach. In *Proc. GULP '90*, A. Bossi, Ed. Padova, 399–425.

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. *PortugaliæMathematica 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.

Muthukumar, K. and Hermenegildo, M. 1992. Compile-time derivation of variable dependency using abstract interpretation. *Journal of Logic Programming 13,* 2-3, 315–347.

Mycroft, A. 1980. The theory and practice of transforming call-by-need into call-by-value. In *Proc. of the 4th Int'l Symposium on Programming*, B. Robinet, Ed. Lecture Notes in Computer Science, vol. 83. Springer-Verlag, Berlin, 270–281.

Mycroft, A. 1981. Abstract Interpretation and Optimising Transformations for Applicative Programs. Ph.D. thesis, University of Edinburgh. CST-15/81.

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.

Palamidessi, C. 1990. Algebraic properties of idempotent substitutions. In *Proc. of the 17th International Colloquium on Automata, Languages and Programming*, M. S. Paterson, Ed. Lecture Notes in Computer Science, vol. 443. Springer-Verlag, Berlin, 386–399.

Park, Y. and Goldberg, B. 1992. Escape analysis on lists. In *Proc. of the 1992 ACM Conf. on Programming Language Design and Implementation* (*PLDI '92*). ACM Press, New York, 116–127.

Ranzato, F. 1998. Closures on cpos form complete lattices. Tech. rep., Math. Dept., Univ. of Padova, Italy. Manuscript.

Reddy, U. and Kamin, S. 1993. On the power of abstract interpretation. *Computer Languages 19,* 2, 79–89.

Rosenthal, K. 1990. *Quantales and their Applications.* Longman Scientific & Technical, Harlow, Essex, U.K.

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.

Sekar, R., Mishra, P., and Ramakrishnan, I. 1997. On the power and limitation of strictness analysis. *J. ACM 44,* 3, 505–525.

Søndergaard, H. 1986. An application of abstract interpretation of logic programs: occur check reduction. In *Proc. ESOP '86.* Lecture Notes in Computer Science, vol. 213. Springer-Verlag, Berlin, 327–338.

Steffen, B. 1989. Optimal data flow analysis via observational equivalence. In *Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science* (*MFCS '89*). Lecture Notes in Computer Science, vol. 379. Springer-Verlag, Berlin, 492–502.

Steffen, B., Jay, C. B., and Mendler, M. 1992. Compositional characterization of observable program properties. *Informatique Theéorique et Applications* (*AFCET*) *26*, 403–424.

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.

Tarski, A. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics 5*, 285–309.

van Dalen, D. 1994. *Logic and Structure.* 3rd ed. Springer-Verlag, Berlin.

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

Wadler, P. and Hughes, R. 1987. Projections for strictness analysis. In *Proc. of the Conference on Functional Programming Languages and Computer Architecture (FPCA '87)*, G. Kahn, Ed. Lecture Notes in Computer Science, vol. 274. Springer-Verlag, Berlin, 385–407.

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

Yetter, D. 1990. Quantales and (noncommutative) linear logic. *Journal of Symbolic Logic 55,* 1, 41–64.

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.

# Index