Mathematical Structures in the Theory of Programming Languages

An annotated selection of publications

Dem Fachbereich Mathematik

der Technischen Hochschule Darmstadt

vorgelegte Habilitationsschrift

von

Dr. Achim Jung

Darmstadt, 2. Februar 1994

Contents

Preface **3**

Introduction **5**

1 The simply typed lambda calculus **9**

1.1 A New Characterization of Lambda Definability 11

2 A Typed Lambda Calculus with Recursion **25**

2.1 The Fully Abstract Model of PCF 27

2.2 The Classification of Continuous Domains 45

3 The polymorphic lambda calculus **55**

3.1 The Dependent Product Construction 58

3.2 Coherence and Consistency in Domains 65

4 Types for database languages **87**

4.1 Powerdomains to Generalize Relational Databases 90

4.2 Decomposition of Domains 129

Bibliography **147**

Preface

In this work I have collected seven publications, which were written during the years 1989–93, some of them under joint authorship. The order, in which they appear here, is not the chronological one. Rather, I have tried to give some perspective on my work by grouping them into four chapters according to the computational paradigm with which my co-authors and I were concerned at the time. Each chapter begins with an explanatory text which has been written for the present purpose. It contains some additional information on how the papers fit into the general plan laid out in the Introduction. Furthermore, I try to give an honest account of the respective contribution in the case of joint authorship as it is required by the "Habilitationsordnung" of "Technische Hochschule Darmstadt". Finally, I give an overview of further developments in the area inasmuch as they are related to my own research.

The papers appear here almost exactly as they have appeared in the various proceedings and journals but some changes seemed necessary. These are the following. Firstly, I have tried to unify the use of mathematical symbols to some extent. Secondly, the references all appear in a single bibliography at the end of the text. Where possible, I have also updated the bibliographical entries by giving precise coordinates where previously the work had been cited as "to appear". Finally, information about authorship has been moved into the explanatory text with which each chapter begins. That seemed also the right place for the various acknowledgements. Surely, more pruning could have been done and there remains a certain amount of redundancy in the introductory parts of the papers. For that I have to beg the readers' forbearance.

This is a proper occasion to express my gratitude to a number of people without whose support this work would not have been possible. I begin by thanking my colleagues Jiří Adámek, Bard Bloom, Peter Buneman, Carl Gunter, Michael Mislove,

Maurice Nivat, Jan Paseka, Frank Pfenning, Andy Pitts, Phil Scott, Allen Stoughton, Paul Taylor, and Glynn Winskel, at whose invitation I could meet and talk to the leading researchers in my field. I am grateful for their hospitality and initiative.

I thank Samson Abramsky for offering me a research position at Imperial College. The knowledge I gained during those nine month – mostly in discussion with him – has proven to be a solid and reliable foundation for understanding developments in the theory of programming as well as for conducting original research myself. I am grateful for his invitation to participate in the ESPRIT project "Categorical Logic in Computer Science II" which allows me to keep in contact with the very front of theoretical computer science research.

I thank my home department at the "Technische Hochschule Darmstadt" for supporting my activities by allowing me to invite guest professors (two papers in this collection were initiated during such visits) and by encouraging me to offer regular courses.

I thank Karl Heinrich Hofmann and Klaus Keimel for cooperating in the seminar series of the "Arbeitsgruppe Domains" (running now for more than five years) and for giving me a free hand to select topics and supervise students.

This alone does not give a true account of my intellectual and scientific debts to Klaus Keimel. I wished I could claim for myself just a fraction of his clear judgement, his sincerity, and his unfailing supportiveness, from which I have profitted so much.

I thank my co-authors Peter Buneman, Carl Gunter, Atsushi Ohori, Hermann Puhlmann, Allen Stoughton, and Jerzy Tiuryn for their enthusiasm, without which some of these papers would never have been written at all.

And I will not forget to thank Ingrid, Tanis, Elena, and Anne. They keep reminding me that life has more to offer than Syntax and Semantics.

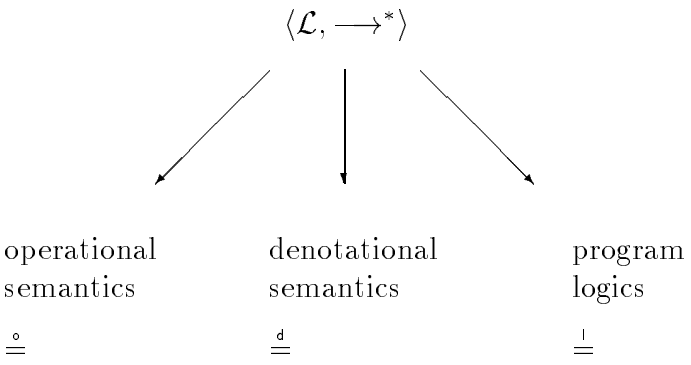
Introduction

We begin by reviewing the general set-up for semantics of programming languages. In each case, the object of interest is a particular formal language $\mathcal{L}$ together with an evaluation mechanism which we denote by $\longrightarrow^*$. Indeed, the purpose of the formal language is to write programs which we then expect to be evaluated by a machine. There are many different paradigms to which $\mathcal{L}$ can belong, we only mention imperative, functional, and logical programming style. Of these, the functional paradigm is probably the most basic one, because any programming language allows subroutines to be written which can be invoked later on and, in a sense, functional languages consist of nothing else than this mechanism. In other words, in order to understand a programming language it is a prerequisite to understand its functional component.

On the practical side it has evolved that the λ-calculus is a convenient base language for many semantical studies. Its syntax is of utmost simplicity, yet, its theory exhibits all phenomena of functional languages. Furthermore, we can add to the λ-calculus other features, such as various type systems, first order constants and recursion, notions of state and assignment, dynamic storage allocation, to name a few. The point is that often these features can be added to the λ-calculus one at a time and thus can be studied in isolation; we do not have to deal with a fully fledged programming language from the start. It is according to the language that I have grouped the papers in this collection.

Now suppose a functional language and an evaluation mechanism have been chosen and we ask what the *meaning* of a piece of code is supposed to be. The answer seems straightforward: we apply the evaluation mechanism and wait for the result. But observe that this is only appropriate for complete programs in the sense that routines *and* data are present. If the data is missing then the evaluation will just stall as soon as input is requested. This leads to the fundamental distinction between

programs – which can be executed and which return a value upon termination – and *incomplete code*. The meaning of a program is clear; it is either the value returned upon termination or undefined otherwise. The undecidability of the halting problem bars us from making any progress beyond this. Hence the object of our interest is the other half, the incomplete code. There are three ways to make sense of its meaning, which we lay out in the following diagram.



We describe each of these in turn.

In *operational semantics* the approach is a pragmatic one. In order to see the effect of an expression M it has to be embedded into a complete program. Hence one defines a notion of context $C[\ ]$, which is nothing else than an expression of the language under consideration containing occurrences of a place holder $[\ ]$ into which other code may be inserted. The equivalence relation on the set of expressions induced by this may be described as follows: We say that M is *operationally equivalent* to N, written as $M \stackrel{\circ}{=} N$, if for every context $C[\ ]$ such that $C[M]$ and $C[N]$ are syntactically correct and complete programs, we have that $C[M]$ evaluates to a value c if and only if $C[N]$ evaluates to value c. (Further conditions may be necessary to ensure that the set of applicable contexts is non-empty.) With the provisos mentioned we may write

$$M \stackrel{\circ}{=} N \quad \text{if} \quad \forall C[\ ].\, (C[M] \longrightarrow^* c) \iff (C[N] \longrightarrow^* c)\,.$$

Obviously, it is this notion of interchangeability of code, in which – from a practical viewpoint – we are ultimately interested in.

In *denotational semantics* the approach is more fundamental. Here the goal is to assign a meaning $[\![M]\!]$ to each syntactical expression M in some mathematical universe $\mathcal{D}$. If this universe is based on Set Theory, then we want to take advantage

of the set-theoretical, that is: extensional, notion of equality. Hence we define

$$M \stackrel{d}{=} N \quad \text{if} \quad [\![M]\!] = [\![N]\!] \,.$$

The construction of suitable universes is by no means trivial. In fact, we may note that there is a fundamental mismatch between the dynamics of computation and the static world of Set Theory. It is probably fair to say that whenever this mismatch can be overcome, this constitutes a definite achievement likely to increase our insight on both sides.

The first breakthrough in this enterprise was achieved by Dana Scott in 1969/70 when he found that type-free computation could be modelled in the rigidly typed world of mathematics. This marked the birth of *Domain Theory* (documented in [Sco93, Sco72]). It may be defined – in the narrow sense – as the study of certain non-Hausdorff topological spaces and – in the wider sense – as the search for mathematical structures suited to the interpretation of programming languages. The main references on Domain Theory are [GHK+80, Sco81a, Plo81, AJ94]. Mostly via the special class of continuous lattices there is also a strong connection to more mainstream mathematics as laid out in [GHK+80, BH81, HH87].

Despite its success and, as we see more and more clearly, its fundamental place in the Mathematics of Computation even beyond Semantics, it must not be forgotten that the richness of programming paradigms continues to pose a formidable challenge to mathematical ingenuity. And, indeed, quite a few researchers in the field have expressed the need to widen the ontological fundament of Mathematics, that is, Set Theory. We mention non-well-founded sets, [Acz88], various type theories, [ML84], and categorical foundations, [Hyl82, LS86, LM92, Abr93].

Program logics, the third component in the semantics set-up, is the study of logical systems suited to the description of programming languages. Usually the formulas of these logics take the form $\phi(M)$, meaning: 'Property ϕ holds of M', or $\phi M \psi$: 'Assuming that ϕ holds and that the execution of M terminates, then afterwards ψ will hold'. In the first formulation we may define a logical equivalence between expressions by setting

$$M \stackrel{\perp}{=} N \quad \text{if} \quad \forall \phi.\, \phi(M) \iff \phi(N) \,.$$

This approach is very flexible and may be tailored to the needs of the application as we may be interested in a restricted set of properties only. If we seek a full

characterization of operational equivalence then it has been shown in the work of Samson Abramsky, [Abr91], that it is advantageous to establish the correspondence with the denotational model first, because here we can obtain valuable guidance from Stone duality.

All in all, the ultimate goal is to establish the equivalence between the three equalities $\stackrel{\circ}{=}$, $\stackrel{d}{=}$, and $\stackrel{\perp}{=}$. It has turned out that the relation between $\stackrel{\circ}{=}$ and $\stackrel{d}{=}$ is particularly intriguing. While the implication $M \stackrel{\circ}{=} N \Longrightarrow M \stackrel{d}{=} N$ can be proven relatively easy, its converse (for the case of PCF) has resisted full clarification so far (but essential progress has been made just recently, [AJM, HO]). More information on this so-called *Full Abstraction Problem* is given in Chapter 2. For other languages we haven't even reached the point where we would worry about full abstraction. There we still struggle with the very construction of a denotational model itself. Examples are the polymorphic lambda calculus (see Chapter 3), polymorphic languages with subtyping (see [CW85, Ghe90]), and languages which deal with a set-like data type (see Chapter 4).

Chapter 1

The simply typed lambda calculus

This chapter contains the paper *A new characterization of lambda definability* written by Jerzy Tiuryn and myself during 1992 and published as [JT93].

It is quite self-contained except for a definition of the simply typed λ-calculus, which I therefore include here. The set $\mathbb{T}$ of *types* is generated by the following rules in a free fashion.

T1 $\iota \in \mathbb{T}$ (the base type).

T2 If $\sigma \in \mathbb{T}$ and $\tau \in \mathbb{T}$ then $(\sigma \to \tau) \in \mathbb{T}$.

To construct the terms we assume that we are given a disjoint family of countably infinite sets of variables Var_σ, one for each type $\sigma \in \mathbb{T}$. We write the variables as x^σ to indicate their type. The following rules then generate the *terms* of the simply typed λ-calculus and at the same time determine a type for each of them.

L1 Each variable x^σ is a term of type σ.

L2 If M is a term of type $\sigma \to \tau$ and if N is a term of type σ then (MN) is a term of type τ.

L3 If M is a term of type τ and if x^σ is a variable then $\lambda x^\sigma.M$ is a term of type $\sigma \to \tau$.

My motivation to study the lambda definability problem was very strong because I felt (and still feel) that it is at the very heart of the Full Abstraction Problem for PCF as I have laid out in Section 2.1.4 below. Jerzy Tiuryn, when he visited Darmstadt in February 1992, agreed that this was a worthwhile problem and we both

studied the paper [Plo80] by Gordon Plotkin and the (then very new) paper by Kurt Sieber [Sie92] which offered a novel view of logical relations. Jerzy Tiuryn observed rather quickly that by combining the two approaches of Plotkin and Sieber and by adding the feature of "varying arity" one can get a characterization theorem. The result in Section 1.1.2 is essentially due to him. We then tried quite hard to employ this characterization for a decidability result but all we came up with was a clear view of why our approach cannot succeed (Section 1.1.3). I took up the question again in September of the same year while visiting Allen Stoughton. I could show that a more involved notion of logical relation allows for a finer analysis of lambda definability and indeed I could show decidability in the case where only variables from a fixed finite set are allowed to enter the terms (Section 1.1.4).

The actual writing was done by myself. We submitted the paper to the conference *Typed Lambda Calculi and Applications*, where it was accepted and presented by me in March 1993.

It got enough attention so that several people began to look at the problem again, which had been dormant for many years, and, indeed, in June 1993 Ralph Loader showed the undecidability of lambda definability in the general case, [Loa93]. So, in a sense, our paper contains the most one can hope for in direction of a positive solution. A consequence of Loader's theorem is that there are functionals at rank 3 which are invariant under all logical relations of finite arity. We have no idea what they are. This illustrates how poor our knowledge and our intuitions about higher order functions still are. In my view, Loader's negative answer to the lambda definability problem is a first step towards a delimiting result regarding the Full Abstraction Problem for PCF. It would be worthwhile to see if the definability problem for finitary PCF is undecidable, too. By "finitary PCF" I mean the simply typed λ-calculus over base type bool plus constants true, false of type bool, a branching construct such as $\mathsf{if\text{-}then\text{-}else} : \mathsf{bool}^3 \to \mathsf{bool}$ and a constant Ω of type bool at the encounter of which evaluation will stall. It seems that Loader's techniques are not easily adapted to solve this case, too.

1.1 A New Characterization of Lambda Definability

Introduction

An *applicative structure* consists of a family $(A_\sigma)_{\sigma\in\mathbb{T}}$ of sets, one for each type σ, together with a family $(app_{\sigma,\tau})_{\sigma,\tau\in\mathbb{T}}$ of application functions, where $app_{\sigma,\tau}$ maps $A_{\sigma\to\tau}\times A_\sigma$ into A_τ. For an applicative structure to be a model of the simply typed lambda calculus (in which case we call it a *Henkin model*, following [Mit90]), one requires two more conditions to hold. It must be *extensional* which means that the elements of $A_{\sigma\to\tau}$ are uniquely determined by their behavior under $app_{\sigma,\tau}$, or, more intuitively, that $A_{\sigma\to\tau}$ can be thought of as a set of functions from A_σ to A_τ. Secondly, the applicative structure must be rich enough to interpret every lambda term. (This requirement can be formalized using either the combinatory or the environment model definition, see Sect. 1.1.1 below.) The simplest examples for Henkin models are derived if one takes a set A_ι for the base type ι (more base types could be accommodated in the same way) and then defines $A_{\sigma\to\tau}$ to be the set of all functions from A_σ to A_τ. The application functions are in this case just set-theoretic application of a function to an argument. These models are sometimes called the *full type hierarchy over* A_ι.

Simple as this construction is, there remains a nagging open question. Suppose A_ι is finite (in which case every A_σ is finite), is there an algorithm which, given an element of some A_σ, decides whether it is the denotation of a closed lambda term? We could also ask for an algorithm which works uniformly for all finite sets A_ι, but the essential difficulty seems to arise with the first question. The assumption that a positive solution exists is known under the name *lambda definability conjecture*. We shall speak of the *lambda definability problem* instead. Besides this being an intriguing question in itself, there are also connections to other open problems, such as the *higher order matching problem* (cf. [Sta82a], and also [SD92]) and the *full abstraction problem for PCF* (cf. [JS93]).

Let us quickly review the existing literature on the question. A first attempt to characterize lambda definable elements in the full type hierarchy was made by H.Läuchli [Läu70]. He showed that lambda definable elements are invariant under

permutations of the ground set A_ι which is a not too surprising result as there are no means by which the lambda calculus could speak about particular elements of A_ι. He also observed that permutation invariance was too weak a property for full characterization at all types. This line of thought was taken up by G.Plotkin in [Plo80] (a precursor of this is [Plo73]). He replaced permutation invariance by invariance under logical relations and proved that for infinite ground sets this characterizes lambda definability at types of rank less than three. Using more complicated logical relations defined over a quasi-ordered set he could remove the restriction on the rank. The restriction on the size of A_ι, however, remained. In both cases the proof is by coding the theory of lambda terms into the ground set. The problem is also discussed in papers by R.Statman (cf. [Sta82a, Sta82b, Sta83, Sta85]). In [Sta85] a characterization is stated (without proof) which is applicable to all Henkin models and which employs logical relations on a free extension of the given model by infinitely many variables. More recently, K.Sieber [Sie92] used logical relations in a novel fashion to tackle the full abstraction problem for PCF. His logical relations have large arity and are reminiscent of value tables. It was this paper from which we got the initial idea for the results presented here. By looking at logical relations which are defined over an ordered set (as in [Plo80]) but which in addition increase their arity as we pass to later "worlds", we derive a characterization theorem which works for all ground sets A_ι and, in fact, every Henkin model, which again contrasts to the characterization in [Plo80], which can not be generalized to arbitrary Henkin models. (A counterexample is given in [Sta85].) Furthermore, our characterization theorem has a very straightforward proof. Indeed, the proof is so simple that it suggests a positive solution to the lambda definability problem. Even though we do not achieve this, at least we can make the obstacles very clear. These lie in the fact that higher order terms (even when they are in normal form) can contain arbitrarily many auxiliary variables. For a restricted set of variables one would expect a decidability result. This can be achieved as we show in Sect. 1.1.4, but the proof becomes somewhat technical.

Our definition of logical relation will still make sense if we replace the ordered set by a small category and, in fact, it reduces to a logical predicate on the presheaf model built from the initial Henkin model (for details, see [LS86] or [MM91]). A bit of this generality indeed simplifies our presentation of Kripke logical relations with varying

arity in the next section. The characterization theorem in Sect. 1.1.2, however, works with a very simple fixed ordered set.

1.1.1 Kripke Logical Relations with Varying Arity

Suppose A_ι is a set and we are studying the semantics of the simply typed lambda calculus in the full type hierarchy over A_ι. (We could take an arbitrary Henkin model instead but would have to write out the application functions explicitly in every instance.) Let $\mathbf{C}$ be a small category of sets. We want to build a logical relation over each object w of $\mathbf{C}$, taking the cardinality of w as the arity of the relation at w. Thus elements of the relations are tuples indexed by elements of w. It makes no difference whether w is finite or infinite.

We start with ground relations $R_\iota^w \subseteq A_\iota^w$ which have the following compatibility property: Whenever $f: v \to w$ is a map in $\mathbf{C}$ and $(x_j)_{j\in w}$ is an element of R_ι^w then $(x_{f(i)})_{i\in v}$ is an element of R_ι^v (note the contravariance). The ground relations are extended to higher types as usual. For a function type $\sigma \to \tau$ let

$$R^w_{\sigma\to\tau} = \{(g_j)_{j\in w} \mid \forall j \in w.g_j \in A_{\sigma\to\tau} \wedge \forall f: v \to w \forall (x_i)_{i\in v} \in R^v_\sigma. \\ (g_{f(i)}(x_i))_{i\in v} \in R^v_\tau\}.$$

(A tuple of functions at w must have the defining property of logical relations at all v reachable - via a map in $\mathbf{C}$ - from w.) Relations $(R^w_\sigma)^{w\in Obj(\mathbf{C})}_{\sigma\in\mathbb{T}}$ constructed this way we shall call *Kripke logical relations with varying arity.* Ordinary logical relations are subsumed by this concept - just take a one object one morphism category $\mathbf{C}$ - as well as Plotkin's "I-relations": take a category all of whose objects have the same cardinality and all of whose morphisms are bijections such that the category is isomorphic to a quasi-ordered set.

We observe that for each type σ we have the compatibility with morphisms of $\mathbf{C}$ we required at ground level:

Lemma 1.1.1 *Let $(R^w_\sigma)^{w\in Obj(\mathbf{C})}_{\sigma\in\mathbb{T}}$ be a Kripke logical relation with varying arity. For all types σ, objects v, w of $\mathbf{C}$, morphisms $f: v \to w$, and tuples $(x_j)_{j\in w}$ in R^w_σ, the tuple $(x_{f(i)})_{i\in v}$ is in R^v_σ.*

Proof. By induction on types. For ι it is part of the definition. If $\sigma \to \tau$ is a function type we have to show that $(g_j)_{j\in w} \in R^w_{\sigma\to\tau}$ implies $(g_{f(i)})_{i\in v} \in R^v_{\sigma\to\tau}$. By definition we

have to supply arguments $(x_l)_{l\in u} \in R^u_\sigma$, for $h\colon u \to v$ to the functions. The resulting tuple has the form $(g_{f(h(l))}(x_l))_{l\in u}$ which belongs to R^u_τ because $f \circ h\colon u \to w$ is also a map in $\mathbf{C}$ and was taken account of in the definition of $R^w_{\sigma\to\tau}$. ▮

Our logical relations have the usual "un-Currying" property.

Lemma 1.1.2 *Let $(R^w_\sigma)^{w\in Obj(\mathbf{C})}_{\sigma\in\mathbb{T}}$ be a Kripke logical relation with varying arity. For any type $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \iota$ and any object w, a tuple $(g_j)_{j\in w}$ is in R^w_σ if and only if for every chain of maps $v_n \xrightarrow{f_n} \ldots \xrightarrow{f_2} v_1 \xrightarrow{f_1} w$ and tuples $(x^k_i)_{i\in v_k} \in R^{v_k}_{\sigma_k}$, $k = 1, \ldots, n$, the result of applying the functions coordinatewise to all n arguments is in $R^{v_n}_\iota$.*

Proof. Easy induction on the length of the unfolded types $\sigma_1 \to \ldots \to \sigma_n \to \iota$. ▮

In order to prove the "Fundamental Theorem of Logical Relations" (in the words of [Sta85]) let us recall how the simply typed lambda calculus is interpreted over A_σ. Free variables are assigned values by environments $\rho\colon \mathsf{Var} \to \bigcup_{\sigma\in\mathbb{T}} A_\sigma$ (where a variable x^σ of type σ is mapped to A_σ) and the denotation of a lambda term M is then defined with respect to environments as follows:

$M \equiv x^\sigma$: $[\![x^\sigma]\!]\rho = \rho(x^\sigma)$.

$M \equiv M_1M_2$: $[\![M_1M_2]\!]\rho = [\![M_1]\!]\rho([\![M_2]\!]\rho)$.

$M \equiv \lambda x^\sigma.M_1$: $[\![\lambda x^\sigma.M_1]\!]\rho =$ the map which assigns to $a \in A_\sigma$ the value $[\![M_1]\!]\rho[x^\sigma \mapsto a]$. (In a general extensional applicative structure there need not be a representative in $A_{\sigma\to\tau}$ for this map. This is the "richness" of Henkin models we referred to in the Introduction.)

Theorem 1.1.3 *For every Kripke logical relation with varying arity $(R^w_\sigma)^{w\in Obj(\mathbf{C})}_{\sigma\in\mathbb{T}}$, object w of $\mathbf{C}$, and closed term M of type σ the constant tuple $([\![M]\!])_{j\in w}$ is in R^w_σ.*

Proof. The proof is for all objects of $\mathbf{C}$ simultaneously by induction on the term structure. Hence we must also take open terms into account. For $w \in Obj(\mathbf{C})$ let $(\rho_j)_{j\in w}$ be a tuple of environments such that for every free variable x^σ of M the tuple $(\rho_j(x^\sigma))_{j\in w}$ is in R^w_σ. We show that under this condition the tuple $([\![M]\!]\rho_j)_{j\in w}$ is in R^w_σ. We check the three cases in the definition of $[\![\cdot]\!]$:

$M \equiv x^\sigma$: $([\![x^\sigma]\!]\rho_j)_{j\in w} = (\rho_j(x^\sigma))_{j\in w} \in R^w_\sigma$ by assumption.

$M \equiv M_1 M_2$: $([\![M_1 M_2]\!]\rho_j)_{j\in w} = ([\![M_1]\!]\rho_j([\![M_2]\!]\rho_j))_{j\in w}$. By induction hypothesis $([\![M_1]\!]\rho_j)_{j\in w}$ is in $R^w_{\sigma\to\tau}$ and $([\![M_2]\!]\rho_j)_{j\in w}$ is in R^w_σ. Because a category contains an identity morphism for every object, we get that the tuple resulting from pointwise application is in R^w_τ.

$M \equiv \lambda x^\sigma.M_1$: $([\![\lambda x^\sigma.M_1]\!]\rho_j)_{j\in w}$ is a tuple of functions from A_σ to A_τ. To check that it is in relation we to apply to it a tuple $(a_i)_{i\in v}$ of arguments from R^v_σ for an object v and a morphism $f: v \to w$. We get the tuple $([\![M_1]\!]\rho_{f(i)}[x^\sigma \mapsto a_i])_{i\in v}$. From Lemma 1.1.1 we know that each of the tuples $(\rho_{f(i)}(y))_{i\in v}$, y a variable, is in relation at v. Updating the environments at x^σ to $(a_i)_{i\in v}$ retains this property. So we can conclude from the induction hypothesis that $([\![M_1]\!]\rho_{f(i)}[x^\sigma \mapsto a_i])_{i\in v}$ is in R^v_τ. ∎

Let us emphasize again that the preceding theorem is neither a surprise nor a generalization over already established results. Our Kripke logical relation with varying arity is nothing more than a logical predicate over a particular Henkin model in the cartesian closed functor category $\mathcal{S}et^{\mathbf{C}^{op}}$. The point is that we want to look at a complicated logical relation over a simple Henkin model in order to characterize lambda definability in the latter. We included the proof of the Fundamental Theorem in order to acquaint the reader with the technical apparatus.

1.1.2 A Characterization of Lambda Definability

We will now characterize lambda definability in the full type hierarchy over some ground set A_ι. (The proof for an arbitrary Henkin model is the same but involves more notational overhead.) From A_ι we construct a concrete category $\mathbf{A}$ as follows. Objects are finite products $A_{\sigma_1\ldots\sigma_n} = A_{\sigma_1} \times \ldots \times A_{\sigma_n}$ of our denotational domains, one for each sequence $\sigma_1 \ldots \sigma_n$ of types. The empty sequence ϵ is represented by an arbitrary one-point set A_ϵ. If $\sigma_1 \ldots \sigma_n$ is a prefix of the sequence $\sigma_1 \ldots \sigma_n\tau_1 \ldots \tau_m$ then our category contains the projection morphism from $A_{\sigma_1} \times \ldots \times A_{\sigma_n} \times A_{\tau_1} \times \ldots \times A_{\tau_m}$ to $A_{\sigma_1} \times \ldots \times A_{\sigma_n}$. So $\mathbf{A}$ is really an ordered set, namely, the dual of $\mathbb{T}^*$ with the prefix ordering. The logical relation $(T^w_\sigma)^{w\in Obj(\mathbf{A})}_{\sigma\in\mathbb{T}}$ which will give us the characterization, has arity $|A_{\sigma_1} \times \ldots \times A_{\sigma_n}|$ at the object $w = A_{\sigma_1} \times \ldots \times A_{\sigma_n}$. A tuple from T^w_σ is therefore indexed by tuples $\vec{a} = (a_1, \ldots, a_n) \in A_{\sigma_1} \times \ldots \times A_{\sigma_n}$. At ground level we take those tuples $(x_{\vec{a}})_{\vec{a}\in w}$ into T^w_ι for which there is a closed lambda term M of type $\sigma_1 \to \ldots \to \sigma_n \to \iota$ such that for each $\vec{a} = (a_1, \ldots, a_n)$ in $A_{\sigma_1} \times \ldots \times A_{\sigma_n}$ we

have $x_{\vec{a}} = [\![M]\!](a_1)\ldots(a_n)$. Intuitively, we take only those tuples which are "value tables" of lambda definable functions. This idea is taken directly from [Sie92]. These relations have the compatibility property with morphisms in $\mathbf{A}$. Indeed, if M defines the tuple $(x_{\vec{a}})_{\vec{a}\in w}$ at $w = A_{\sigma_1} \times \ldots \times A_{\sigma_n}$ then $\lambda x_1^{\sigma_1} \ldots x_n^{\sigma_n} y_1^{\tau_1} \ldots y_m^{\tau_m}.M x_1^{\sigma_1} \ldots x_n^{\sigma_n}$ defines the corresponding tuple at $v = A_{\sigma_1} \times \ldots \times A_{\sigma_n} \times A_{\tau_1} \times \ldots \times A_{\tau_m}$. The following lemma asserts that the lambda definable functions can be read off at A_ϵ, the one element object in $\mathbf{A}$.

Lemma 1.1.4 *A one-element tuple (x) is in $T_\sigma^{A_\epsilon}$ if and only if x is the denotation of a closed lambda term of type σ.*

Proof. We prove by induction on types (simultaneously for all objects $w = A_{\sigma_1} \times \ldots \times A_{\sigma_n}$ of $\mathbf{A}$) that T_σ^w only contains tuples which are definable by closed lambda terms of type $\sigma_1 \to \ldots \to \sigma_n \to \sigma$. For $\sigma = \iota$ this is the definition of T_ι^w, so let us look at a function type $\sigma \to \tau$.

If M is a closed term of type $\sigma_1 \to \ldots \to \sigma_n \to \sigma \to \tau$ which defines the tuple $(f_{\vec{a}})_{\vec{a}\in w}$ we want to assert that it is in relation at w. To this end we supply an argument tuple $(x_{\vec{b}})_{\vec{b}\in v}$ for an object $v = A_{\sigma_1} \times \ldots \times A_{\sigma_n} \times A_{\tau_1} \times \ldots \times A_{\tau_m}$. By induction hypothesis, this tuple is represented by a closed term N of type $\sigma_1 \to \ldots \to \sigma_n \to \tau_1 \to \ldots \to \tau_m \to \sigma$. The resulting tuple $(f_{\pi(\vec{b})}(x_{\vec{b}}))_{\vec{b}\in v}$ is represented by the term $\lambda x_1^{\sigma_1} \ldots x_n^{\sigma_n} y_1^{\tau_1} \ldots y_m^{\tau_m}.(M x_1^{\sigma_1} \ldots x_n^{\sigma_n})(N x_1^{\sigma_1} \ldots x_n^{\sigma_n} y_1^{\tau_1} \ldots y_m^{\tau_m})$ and so, by induction hypothesis, is contained in T_τ^v.

Conversely, assume that the tuple $(f_{\vec{a}})_{\vec{a}\in w}$ belongs to $T_{\sigma\to\tau}^w$. We supply it with the argument tuple over the object $v = A_{\sigma_1} \times \ldots \times A_{\sigma_n} \times A_\sigma$ which is given by the term $\lambda x_1^{\sigma_1} \ldots x_n^{\sigma_n} x^\sigma.x^\sigma$. By induction hypothesis it is contained in T_σ^v. The resulting tuple $(f_{\vec{a}}(a))_{\vec{a}a\in v}$ is in T_τ^v and, again by induction hypothesis, there is a closed term N of type $\sigma_1 \to \ldots \to \sigma_n \to \sigma \to \tau$ representing it. We claim that N also represents $(f_{\vec{a}})_{\vec{a}\in w}$: Using the denotation of N we get a tuple $(g_{\vec{a}})_{\vec{a}\in w}$ of functions of type $\sigma \to \tau$ where $g_{(a_1,\ldots,a_n)} = [\![N]\!](a_1)\ldots(a_n)$. In order to see that such a function is equal to the corresponding $f_{\vec{a}}$ we supply a generic argument a from A_σ. We get $f_{\vec{a}}(a) = [\![N]\!](a_1)\ldots(a_n)(a) = g_{\vec{a}}(a)$, which completes our argument. ∎

Theorem 1.1.3 and Lemma 1.1.4 together give our main result:

Theorem 1.1.5 *An element of a Henkin model is lambda definable if and only if it is invariant under all Kripke logical relations with varying arity.*

Somewhat slicker but maybe less transparent is the following description of the relations T^w_σ. We replace sequences of types by finite sets of variables. The objects of $\mathbf{A}$ remain almost the same, $\{x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\}$ corresponds to the set $Env\{x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\}$ of finite environments over this collection of variables. The tuples should now be labeled by our symbol for environments ρ. For $w = Env\{x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\}$ we take a tuple $(x_\rho)_{\rho \in w}$ into T^w_ι if there is a lambda term M whose free variables are contained in $\{x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\}$ such that for all $\rho \in w$ we have $x_\rho = [\![M]\!]\rho$. The proof of Lemma 1.1.4 can be changed accordingly.

1.1.3 The Lambda Definability Problem

We return to the problem of finding an effective characterization of lambda definability for hereditarily finite Henkin models. Indeed, studying the Definability Lemma 1.1.4 one gets the impression that for a particular type ϕ only a finite piece of the category $\mathbf{A}$ is used. More formally, we can precisely define the objects from $\mathbf{A}$ that occur in the proof of Lemma 1.1.4. Fix a type ϕ and define two relations $\vdash_\phi$ and $\Vdash_\phi$ between strings of types and types as follows:

(i) $\epsilon \vdash_\phi \phi$,

(ii) if $s \vdash_\phi \sigma \to \tau$ then $s\sigma \vdash_\phi \tau$ and $s\sigma \Vdash_\phi \sigma$,

(iii) if $s \Vdash_\phi \sigma_1 \to \ldots \to \sigma_n \to \iota$ and if for strings $s_1 \leq \ldots \leq s_n$ there are types $\xi_1, \ldots, \xi_n$ such that for all $k = 1, \ldots, n$, $s_k \vdash_\phi \xi_k$ then for all $k = 1, \ldots, n$, $s_k \vdash_\phi \sigma_k$.

Now let $\mathbf{F}_\phi$ be the full sub-category of $\mathbf{A}$ whose objects are given by $\{A_s \in \mathbf{A} \mid \exists \xi \in \mathbb{T}. s \vdash_\phi \xi\}$. (Note that the strings occurring on the left hand side of the relation $\Vdash_\phi$ all occur on the left hand side of $\vdash_\phi$ already.) We show that the proof of Lemma 1.1.4 for a particular type ϕ can be based on $\mathbf{F}_\phi$. At ground type we start with the same logical relation $(T^w_\sigma)^{w \in Obj(\mathbf{F}_\phi)}_{\sigma \in \mathbb{T}}$ as before.

Lemma 1.1.6 *Given a type $\phi \in \mathbb{T}$ the following is true for all $\sigma \in \mathbb{T}$ and $s \in \mathbb{T}^*$:*

(i) If $s \vdash_\phi \sigma$ then every element of $T^{A_s}_\sigma$ is lambda definable.

(ii) If $s \Vdash_\phi \sigma$ then every lambda definable tuple is in $T^{A_s}_\sigma$.

Proof. By induction on σ. If σ is the ground type ι then both statements follow from the definition of T^w_ι. The proof of (i) for a function type $\sigma \to \tau$ works as in Lemma 1.1.4: Assume $s = \sigma_1 \dots \sigma_n \vdash_\phi \sigma \to \tau$ and $(f_j)_{j \in A_s} \in T^{A_s}_{\sigma \to \tau}$ (where we have identified w with A_s). We have $s\sigma \Vdash_\phi \sigma$ and so by induction hypothesis we can apply the tuple given by the term $\lambda x_1^{\sigma_1} \dots x_n^{\sigma_n} x^\sigma . x^\sigma$ to it. The result is in $T^{A_s \times A_\sigma}_\tau$ and since $s\sigma \vdash_\phi \tau$ it is given by a term N. As before we see easily that N also defines $(f_j)_{j \in A_s}$.

To prove part (ii) we have to un-Curry completely: $\sigma \to \tau = \sigma_1 \to \dots \to \sigma_n \to \iota$ (we have re-named σ to σ_1). Assume that the tuple $(f_j)_{j \in A_s}$ is given by a term M. By Lemma 1.1.2 we have to apply the functions to argument tuples from $T^{A_{s_k}}_{\sigma_k}$, $k = 1, \dots, n$ for strings $s \leq s_1 \leq \dots \leq s_n$ from $\mathbf{F}_\phi$. By our rule (iii) we have for each k, $s_k \vdash_\phi \sigma_k$. Hence we can use the induction hypothesis and conclude that all argument tuples are lambda definable. The application of $(f_j)_{j \in A_s}$ to these argument tuples results in a tuple which again is lambda definable and of type ι. But at ground type lambda definable tuples are in relation and we are done. ∎

Theorem 1.1.7 *An element of type ϕ of a Henkin model is lambda definable if and only if it is invariant under all logical relations based on $\mathbf{F}_\phi$.*

If we are looking at a hereditarily finite Henkin model, for example the full type hierarchy over a finite ground set, and if for some type ϕ the category $\mathbf{F}_\phi$ happens to have only finitely many objects then we can effectively determine the lambda definable elements of A_ϕ by simply checking the finitely many Kripke logical relations with varying arity over $\mathbf{F}_\phi$. Unfortunately, this approach can only succeed for types of rank less than 3:

Lemma 1.1.8 *For every type ϕ of rank at least 3 the category $\mathbf{F}_\phi$ has infinitely many objects.*

Proof. We illustrate the idea for the type $\phi = ((\iota \to \iota) \to \iota) \to \iota$. The general proof is exactly the same but involves a lot of indices. Using rules (i)–(iii) above, we get

(1) $\epsilon \vdash_\phi \phi$ by (i).

(2) $(\iota \to \iota) \to \iota \vdash_\phi \iota$ and $(\iota \to \iota) \to \iota \Vdash_\phi (\iota \to \iota) \to \iota$ by (1) and (ii).

(3) $(\iota \to \iota) \to \iota \vdash_\phi \iota \to \iota$ by (2) and (iii).

(4) $\langle(\iota \to \iota) \to \iota\rangle\langle\iota\rangle \vdash_\phi \iota$ by (3) and (ii).

(5) $\langle(\iota \to \iota) \to \iota\rangle\langle\iota\rangle \vdash_\phi \iota \to \iota$ by (2), (4), and (iii).

The last two steps can be repeated forever. ▮

Behind this proof is the observation that from rank 3 on we can no longer bound the number of variables occurring in a normal form. What happens if we do impose a bound is the topic of the next section.

1.1.4 Lambda Definability with Fixed Sets of Variables

Two-layered logical relations

We proceed by further refining the notion of logical relation and we begin by studying this refinement for ordinary logical relations, letting the varying arity and the Kripke universe at the side for the moment.

Observe that the definition of the extension of a logical relation to a type $\sigma \to \tau$ falls naturally into two halves:

(1) If $f: A_\sigma \to A_\tau$ belongs to $R_{\sigma\to\tau}$ then it maps each element of R_σ into R_τ.

(2) If $f: A_\sigma \to A_\tau$ maps each element of R_σ into R_τ then it belongs $R_{\sigma\to\tau}$.

In the proof of the Fundamental Theorem the first condition is needed in order to show that an application remains invariant if the constituents are, and the second is needed for abstraction. Of course, the power of logical relations resides in the fact that the two properties are fulfilled simultaneously. Nevertheless, we shall study these two conditions separately and thus tie up our logical relations more closely with the structure of lambda terms. To this end we define the following two-layer type system $(\mathbb{T}_0, \mathbb{T}_1)$ (over a single ground type ι and over the set Var of variables):

- $\iota \in \mathbb{T}_0$
- $\sigma, \tau \in \mathbb{T}_0 \Longrightarrow \sigma \to \tau \in \mathbb{T}_0$
- $B \in \mathsf{Var}^*, \tau \in \mathbb{T}_0 \Longrightarrow B \to \tau \in \mathbb{T}_1$

Note that $\mathbb{T}_0$ may be viewed as a subset of $\mathbb{T}_1$ by virtue of the empty string in Var^*. We will also need the forgetful map $e\colon \mathbb{T}_1 \to \mathbb{T}_0$ which maps $x_1^{\sigma_1} \ldots x_n^{\sigma_n} \to \tau$ to $\sigma_1 \to \ldots \to \sigma_n \to \tau$.

Now let $R_\iota \subseteq A_\iota$ be an arbitrary relation (for simplicity we let the arity be 1). It is extended to the types of $\mathbb{T}_0$ and $\mathbb{T}_1$ as follows. For $\sigma \to \tau \in \mathbb{T}_0$ let $R_{\sigma\to\tau}$ be *any subset* of

$$\{f \in A_{\sigma\to\tau} \mid \forall \Sigma \in \mathbb{T}_1.(e(\Sigma) = \sigma \Longrightarrow \forall a \in R_\Sigma.f(a) \in R_\tau)\}$$

and for $n \geq 1, B = x_1^{\sigma_1} \ldots x_n^{\sigma_n}, B \to \tau \in \mathbb{T}_1$ let $R_{B\to\tau}$ be *any superset* of

$$\{f \in A_{e(B\to\tau)} \mid \forall a_1 \in R_{\sigma_1} \ldots \forall a_n \in R_{\sigma_n}.f(a_1)\ldots(a_n) \in R_\tau\} \ .$$

Obviously, such two-layered relations are no longer determined by their value at ground type. But starting from some R_ι we can always construct a two-layered logical relation. The Fundamental Theorem now reads as follows:

Theorem 1.1.9 *Let $M \equiv \lambda x_1^{\sigma_1} \ldots x_n^{\sigma_n}.N$ be a lambda term in normal form and of type $\sigma_1 \to \ldots \to \sigma_n \to \tau$ such that N is not an abstraction and let ρ be an environment which maps each free variable y^σ of M into R_σ. Then $[\![M]\!]\rho \in R_{x_1^{\sigma_1}\ldots x_n^{\sigma_n}\to\tau}$.*

Proof. We have to argue more carefully, but the proof is essentially as usual. Variables can't cause any problems. In the case that M is an application M_1M_2, we employ the assumption that M is in normal form, hence the denotation of M_1 under ρ is in $R_{\sigma\to\tau}$ where $\sigma \to \tau$ is an ordinary type. The denotation of M_2 under ρ is in some R_Σ where $e(\Sigma) = \sigma$. So the composed term is in R_τ as required.

The case that M is an abstraction is characterized by the fact that $n \geq 1$. Unlike in the usual proof we have to unwind all leading lambdas at one stroke. We want $[\![M]\!]\rho$ to be in $R_{x_1^{\sigma_1}\ldots x_n^{\sigma_n}\to\tau}$ and to check this we have to apply it to arguments a_i from R_{σ_i}, $i = 1, \ldots, n$, and see whether the result is in R_τ. This is indeed the case, as $[\![M]\!]\rho(a_1)\ldots(a_n) = [\![N]\!]\rho[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]$ and the induction hypothesis applies to N and the updated environment. (The updating must be read from left to right. This way the lemma remains true also for sequences $x_1^{\sigma_1} \ldots x_n^{\sigma_n}$ which contain some variables more than once.) ∎

Two-layered Kripke logical relations with varying arity

Let us now combine the techniques of Sect. 1.1.1 with these two-layered logical relations. We use the presentation of Kripke logical relations with varying arity via environments as briefly described at the end of Sect. 1.1.1.

So let $V \subseteq \mathsf{Var}$ be a set of variables. It is our goal to characterize all functionals which are definable by lambda terms containing variables (free or bound) only from V. Our base category $\mathbf{V}$ is the set of all subsets of V together with inclusion morphisms. There is a contravariant equivalence between $\mathbf{V}$ and the category $\mathbf{E}$ of environments $\mathsf{Env}(F)$ over sets F of variables contained in V with restriction maps. It no longer helps to think of $\mathbf{E}$ as a concrete example of a general category, as we make use of its particular structure. In other words, we have so far no abstract concept for a two-layered Kripke logical relation with varying arity.

For each object in $\mathbf{V}$, that is, for each set F of variables contained in V, we want a two-layered logical relation $(R^F_\Sigma)_\Sigma$ of arity $|\prod_{x^\sigma \in F} A_\sigma|$. (Elements from the set $\prod_{x^\sigma \in F} A_\sigma$ serve a double purpose. We use them to index elements from the relations and we use them as environments. From now on, we will always use the letter μ to denote them.) Since we have restricted the set of variables available we cannot allow arbitrary types Σ to occur, only those $\Sigma = B \to \sigma$ for which the sequence $B = x_1^{\sigma_1} \ldots x_n^{\sigma_n}$ contains each variable at most once and all variables are contained in V. Let us call such sequences and types built from them *non-repeating* and let $\mathbb{T}_1(V)$ stand for the collection of all non-repeating types over V. We will also allow ourselves to treat B as a set sometimes, just to keep the complexity of our formulas within manageable range.

Definition 1.1.10 *Let $(R^F_\Sigma)^{F \subseteq V}_{\Sigma \in \mathbb{T}_1(V)}$ be a family of relations such that the following conditions are satisfied:*

(1) $\forall \sigma \to \tau \in \mathbb{T}_0 . R^F_{\sigma \to \tau} \subseteq \{(f_\mu)_{\mu \in \mathsf{Env}(F)} \in A^{\mathsf{Env}(F)}_{\sigma \to \tau} \mid \forall \Sigma \in \mathbb{T}_1(V).(e(\Sigma) = \sigma \Longrightarrow \forall (x_\mu)_{\mu \in \mathsf{Env}(F)} \in R^F_\Sigma . (f_\mu(x_\mu))_{\mu \in \mathsf{Env}(F)} \in R^F_\tau)\}$,

(2) $\forall \Sigma \in \mathbb{T}_1(V)$ *where* $\Sigma = B \to \tau$ *and* $B = x_1^{\sigma_1} \ldots x_n^{\sigma_n}, n \geq 1$ *it is the case that* $R^F_\Sigma \supseteq \{(f_\mu)_{\mu \in \mathsf{Env}(F)} \in A^{\mathsf{Env}(F)}_{e(\Sigma)} \mid f_\mu = f_{\mu'}$ *if* $\mu|_{F \setminus B} = \mu'|_{F \setminus B}$ *and* $\forall (a^1_\mu)_{\mu \in \mathsf{Env}(F \cup B)} \in R^{F \cup B}_{\sigma_1}, \ldots, \forall (a^n_\mu)_{\mu \in \mathsf{Env}(F \cup B)} \in R^{F \cup B}_{\sigma_n}$.

$$(f_{\mu|_F}(a_\mu^1)\dots(a_\mu^n))_{\mu\in\mathsf{Env}(F\cup B)} \in R_\tau^{F\cup B}\},$$

(3) $\forall\sigma\in\mathbb{T}_0\forall F\subseteq F'\subseteq V.(x_\mu)_{\mu\in\mathsf{Env}(F)}\in R_\sigma^F \Longrightarrow (x_{\mu'|_F})_{\mu'\in\mathsf{Env}(F')}\in R_\sigma^{F'}$.

If these three conditions are satisfied then we call the family $(R_\Sigma^F)_{\Sigma\in\mathbb{T}_1(V)}^{F\subseteq V}$ *a* two-layered Kripke logical relation with varying arity over V.

We need to check carefully whether the Fundamental Theorem remains valid:

Theorem 1.1.11 *Let* $M\equiv\lambda x_1^{\sigma_1}\dots x_n^{\sigma_n}.N$ *be a lambda term in normal form and of type* $\sigma_1\to\dots\to\sigma_n\to\tau$ *such that* N *is not an abstraction, let* $F\subseteq V\subseteq\mathsf{Var}$ *be sets of variables such that all variables occurring in* M *are contained in* V *and such that all its free variables are contained in* F*, let* $(R_\Sigma^F)_{\Sigma\in\mathbb{T}_1(V)}^{F\subseteq V}$ *be a two-layered Kripke logical relation with varying arity over* V *and, finally, let* $(\rho_\mu)_{\mu\in\mathsf{Env}(F)}$ *be a family of environments such that for all* $x^\sigma\in FV(M)$, $(\rho_\mu(x^\sigma))_{\mu\in\mathsf{Env}(F)}$ *is in* R_σ^F*. Then the tuple* $([\![M]\!]\rho_\mu)_{\mu\in\mathsf{Env}(F)}$ *is in* $R^F_{x_1^{\sigma_1}\dots x_n^{\sigma_n}\to\tau}$.

Proof. The proof is by induction on the complexity of M, simultaneously for all appropriate F,V, and $(\rho_\mu)_{\mu\in\mathsf{Env}(F)}$. The situation is trivial as usual for variables. If $M\equiv M_1M_2$ is an application then because M is in normal form, M_1 is not an abstraction. The free variables of M_1 and M_2 are also contained in F. We can therefore apply the induction hypothesis and get that $([\![M_1]\!]\rho_\mu)_{\mu\in\mathsf{Env}(F)}$ is in $R^F_{\sigma\to\tau}$ and $([\![M_2]\!]\rho_\mu)_{\mu\in\mathsf{Env}(F)}$ is in R_Σ^F where $e(\Sigma)=\sigma$. So $([\![M]\!]\rho_\mu)_{\mu\in\mathsf{Env}(F)}$ is in R_τ^F by part (1) of the definition.

Let now $M\equiv\lambda x_1^{\sigma_1}\dots x_n^{\sigma_n}.N$ be an abstraction, that is, $n\geq 1$. We want to see that $([\![M]\!]\rho_\mu)_{\mu\in\mathsf{Env}(F)}$ is in $R^F_{B\to\tau}$ where we have introduced B as an abbreviation for $x_1^{\sigma_1}\dots x_n^{\sigma_n}$. Let F' stand for $F\cup B$. By part (2) of our definition we have to supply the functions $[\![M]\!]\rho_\mu$ with arguments $(a_\mu^i)_{\mu\in\mathsf{Env}(F')}$ from $R^{F'}_{\sigma_i}$, $i=1,\dots,n$. But since $([\![M]\!]\rho_{\mu|_F}(a_\mu^1)\dots(a_\mu^n))_{\mu\in\mathsf{Env}(F')}$ equals $([\![N]\!]\rho_{\mu|_F}[x_1\mapsto a_\mu^1,\dots,x_n\mapsto a_\mu^n])_{\mu\in\mathsf{Env}(F')}$ we may apply the induction hypothesis to N,F', and $(\rho_{\mu|_F}[x_1\mapsto a_\mu^1,\dots,x_n\mapsto a_\mu^n])_{\mu\in\mathsf{Env}(F')}$. That the new family of environments meets the requirements of the theorem is a consequence of the persistency part (3) of our definition. ∎

The term relation

As usual, a term construction will give us completeness of the characterization. So fix a set V of variables and let $(T^F_\Sigma)^{F\subseteq V}_{\Sigma\in\mathbb{T}_1(V)}$ be the family of relations for which each T^F_Σ is the collection of all $(f_\mu)_{\mu\in\mathsf{Env}(F)} \in A_\Sigma^{\mathsf{Env}(F)}$ definable by lambda terms, i.e., $(f_\mu)_{\mu\in\mathsf{Env}(F)}$ is in T^F_Σ if there exists a lambda term $M \equiv \lambda x_1^{\sigma_1}\ldots x_n^{\sigma_n}.N$ (N not an abstraction) in normal form all of whose variables belong to V, all of whose free variables belong to F, for which $x_1^{\sigma_1}\ldots x_n^{\sigma_n} = B$ is non-repeating and $B\to\tau = \Sigma$ such that $\forall\mu\in\mathsf{Env}(F)$ we have $f_\mu = [\![M]\!]\mu$. We have to check that we get a valid relation this way.

Lemma 1.1.12 *$(T^F_\Sigma)^{F\subseteq V}_{\Sigma\in\mathbb{T}_1(V)}$ is a two-layered Kripke logical relation with varying arity over V.*

Proof. (1) Let $(f_\mu)_{\mu\in\mathsf{Env}(F)}$ be in $T^F_{\sigma\to\tau}$. Then this tuple is given by a term M of type $\sigma\to\tau$ which is not an abstraction. Let further N be term which defines a tuple $(x_\mu)_{\mu\in\mathsf{Env}(F)}$ from T^F_Σ where $e(\Sigma)=\sigma$. Then MN is in normal form and defines $(f_\mu(x_\mu))_{\mu\in\mathsf{Env}(F)}$.

(2) Let $(f_\mu)_{\mu\in\mathsf{Env}(F)}$ be an element from the right hand side of (2) in Definition 1.1.10. We can apply it to the tuples defined by $x_1^{\sigma_1},\ldots,x_n^{\sigma_n}$ and the result $(f_{\mu\restriction_F}([\![x_1]\!]\mu)\ldots([\![x_n]\!]\mu))_{\mu\in\mathsf{Env}(F')}$ will be in $T^{F'}_\tau$, hence given by a lambda term M which is not an abstraction. We claim that $(f_\mu)_{\mu\in\mathsf{Env}(F)}$ is given by $\lambda x_1^{\sigma_1}\ldots x_n^{\sigma_n}.M$. Indeed, if $a_1\in A_{\sigma_1},\ldots,a_n\in A_{\sigma_n}$ are arguments for the function f_μ then

$$\begin{aligned} f_\mu(a_1)\ldots(a_n) &= f_{\mu'\restriction_F}([\![x_1]\!]\mu')\ldots([\![x_n]\!]\mu') \\ &= [\![M]\!]\mu' \\ &= [\![\lambda x_1^{\sigma_1}\ldots x_n^{\sigma_n}.M]\!]\mu(a_1)\ldots(a_n) \end{aligned}$$

where $\mu'(y) = \begin{cases} a_i & \text{if } y\equiv x_i; \\ \mu(y) & \text{otherwise.} \end{cases}$ Here we have used the fact that $f_\mu = f_{\mu'\restriction_F}$ because μ and $\mu'\restriction_F$ differ only at variables from $F\cap\{x_1,\ldots,x_n\}$. Also the fact that $x_1^{\sigma_1}\ldots x_n^{\sigma_n}$ is non-repeating is crucial here.

(3) Persistency is clear as the denotation of a term only depends on its free variables. ∎

Theorem 1.1.13 *A functional is definable from a fixed set V of variables if and only if it is invariant under all two-layered Kripke logical relations with varying arity over V.*

Decidability

We are now ready to harvest the fruit from our hard labor in this section. Unlike for full definability, the notion of definability from a fixed set of variables becomes decidable if we restrict to finite ground sets A_σ and finite sets V of variables. The reason for this simply is that there are only finitely many relations to check. Thus we have:

Theorem 1.1.14 *The problem whether a given functional from a hereditarily finite Henkin model is lambda definable by a term over a fixed finite set of variables is decidable.*

Although two-layered Kripke logical relations with varying arity over some set of variables may seem complicated, there is nevertheless a fairly simple underlying idea. The relations can be thought of as value tables for functionals where the new types $\Sigma \in \mathbb{T}_1$ and the restrictions to subsets F of V are just a way of keeping track of free and bound variables in defining terms. (Note that a variable may be re-used several times, that is, may occur both bound and free.) Gordon Plotkin has suggested to us that one may obtain Theorem 1.1.14 by working backwards from the given value table for a functional in search for a defining term. At each stage, one determines the *set* of value tables which, applied in the right order, give a value table in the set of sought after tables. Each branch of the search stops if either a projection (which corresponds to a variable) can satisfy the requirements or if only tables occur which we are looking for already. Since the set of variables is restricted the tables are finite objects and the search must eventually end. Bookkeeping over free and bound variables is also necessary in this approach and while we haven't formally carried through this approach, we think that it will amount to a scheme with probably the same complexity as ours.

Chapter 2

A Typed Lambda Calculus with Recursion

We have already noted in the introduction to this collection that the language PCF and its domain theoretic denotational model constitute the archetypical example of a mathematical treatment of a programming language. Introduced by Dana Scott in 1969 (published as [Sco93]) and studied in close detail by Gordon Plotkin in [Plo77] and Robin Milner in [Mil77] it stimulated a large number of deep and excellent research. We refer to [BCL85] for an overview of results up to 1985. More recent work includes [Mul87, Sto88, Blo90, Sie92, Cur93, CCF93, AJM, HO, Ehr93].

The first paper below, entitled *Studying the Fully Abstract Model of PCF within its Continuous Function Model* (joint work with Allen Stoughton) achieves the following. We give a simple construction of the fully abstract (order-extensional) model of PCF (which, by the work of Robin Milner, we know to exist uniquely) along the following lines.

1. Select the PCF-definable elements in the usual Scott-model based on flat domains and continuous functions.

2. Add limit points so as to get dcpo's at every type. The resulting structure is not extensional. Hence:

3. Take the extensional collapse by factoring with the binary logical relation based on identity for the ground types.

4. We prove that for each type there is a least element in each equivalence class. These elements represent the fully abstract model.

The programme up to Step 3 was laid out in [Sto88] already. During Allen Stoughton's visit to Darmstadt in September 1991 we discussed the problem and Allen conjectured that 4 could be true. This conjecture I confirmed and furthermore showed that the map (called norm in the paper) from a denotation to the least element equivalent to it, is also Scott-continuous. Once that was settled, many questions about the properties of norm arose. Allen established some, others remain open.

The paper was written up by Allen Stoughton, except for Section 2.1.4, where I laid out my speculations about the the connection between Full Abstraction and Definability. We submitted the paper to the conference *Typed Lambda Calculi and Applications* and Allen presented it in March 1993. It appeared as [JS93] in the proceedings of that conference.

The second paper *The Classification of Continuous Domains* was written by myself in the fall of 1989. It was accepted for the conference *Logic in Computer Science 1990* and appeared as [Jun90b].

The problem of finding the maximal cartesian closed categories of continuous domains was addressed in my doctoral thesis [Jun89] already but the definition of FS-domains was still missing. A complete classification could therefore not be found at the time.

Admittedly, this result has little bearing on the Full Abstraction Problem, as for a solution of the latter we must move to more specialized classes of domains. It is included in this chapter just the same because it delineates the arena in which a Scott-style denotational model must live, where by "Scott-style denotational model" I mean a structure which is a dcpo – needed for the interpretation of recursion – and which satisfies a certain separability axiom which we need to tie up operational and denotational semantics (see [Mil77]). Furthermore, if we add a probabilistic choice operator to PCF then the denotational model must at least accommodate the unit interval of real numbers and we find ourselves in the realm of continuous domains (see [SD80, Jon90, Kir93]).

2.1 Studying the Fully Abstract Model of PCF within its Continuous Function Model

Introduction

As is well known, the continuous function model $\mathcal{E}$ of the applied typed lambda calculus PCF fails to be inequationally fully abstract [Plo77], but PCF has a unique inequationally fully abstract, order-extensional model $\mathcal{F}$ [Mil77, Sto90], where models are required to interpret the ground type ι as the flat dcpo of natural numbers. Two attempts at finding connections between $\mathcal{E}$ and $\mathcal{F}$ have been made in the literature.

Mulmuley's idea was to connect complete lattice versions of $\mathcal{E}$ and $\mathcal{F}$ [Mul87]. Using a syntactically defined inductive (inclusive) predicate, he defines an inequationally fully abstract, order-extensional model $\mathcal{F}'$ as the image of a continuous closure (retraction that is greater than the identity function) of the complete lattices version $\mathcal{E}'$ of the continuous function model. The use of complete lattices is essential in this construction, and, e.g., parallel or is mapped to $\top$. Very pleasingly, $\mathcal{F}'$ inherits both its ordering relation and function application operation from $\mathcal{E}'$. Thus some of PCF's operations must be sent by the closure to strictly greater functions. Although the closure isn't a homomorphism between $\mathcal{E}'$ and $\mathcal{F}'$ (since it doesn't preserve application in general), it does preserve the meaning of terms. $\mathcal{F}'$ isn't a combinatory algebra, since all functions of $\mathcal{F}'$ preserve $\top$, and thus the usual axiom for the K combinator cannot hold. Finally, Mulmuley is able to recover $\mathcal{F}$ from $\mathcal{F}'$ simply by removing $\top$ at all types.

A more algebraic connection between $\mathcal{E}$ and $\mathcal{F}$ was subsequently developed by the second author [Sto88]. Here one begins by forming the inductively reachable subalgebra $R(\mathcal{E})$ of $\mathcal{E}$, which in this case simply consists of those elements of $\mathcal{E}$ that are lub's of directed sets of denotable elements. $R(\mathcal{E})$ is then continuously quotiented by a syntactically defined inductive pre-ordering, producing $\mathcal{F}$. Furthermore, in contrast to the situation with $\mathcal{E}'$ and $\mathcal{F}'$ above, there is a continuous homomorphism from $R(\mathcal{E})$ to $\mathcal{F}$.

The purpose of this paper is to give a concrete presentation of this construction of $\mathcal{F}$ from $R(\mathcal{E})$. We define a model $N(\mathcal{E})$ as the image of a syntactically defined continuous projection over $R(\mathcal{E})$, and show that $N(\mathcal{E})$ is inequationally fully abstract and

order-extensional, and is thus order-isomorphic to $\mathcal{F}$. As in Mulmuley's construction, the ordering relation of $N(\mathcal{E})$ is inherited from $\mathcal{E}$ (and $R(\mathcal{E})$). On the other hand we prove that the application operation of $N(\mathcal{E})$ cannot be inherited from $\mathcal{E}$. There is, of course, a continuous homomorphism from $R(\mathcal{E})$ to $N(\mathcal{E})$.

In the final section of the paper, we consider the relationship between the full abstraction problem, lambda definability and our presentation of $\mathcal{F}$, and propose a minimal condition that any "solution" to the full abstraction problem should satisfy.

2.1.1 Background

The reader is assumed to be familiar with such standard domain-theoretic concepts as (directed) complete partial orders (dcpo's), (directed) continuous functions, and ω-algebraic, strongly algebraic (SFP) and consistently complete dcpo's.

If X is a subset of a poset P, then we write $\bigsqcup X$ and $\sqcap X$ for the lub and glb, respectively, of X in P, when they exist. We abbreviate $\bigsqcup\{x, y\}$ and $\sqcap\{x, y\}$ to $x \sqcup y$ and $x \sqcap y$, respectively. We write $\omega_\perp$ for the flat dcpo of natural numbers. Given dcpo's P and Q, we write $P \stackrel{c}{\to} Q$ for the dcpo of all continuous functions from P to Q, ordered pointwise. A dcpo P is a *sub-dcpo* of a dcpo Q iff $P \subseteq Q$, $\sqsubseteq_P$ is the restriction of $\sqsubseteq_Q$ to P, $\perp_P = \perp_Q$ and $\bigsqcup_P D = \bigsqcup_Q D$ for all directed $D \subseteq P$. A pre-ordering $\leq$ over a dcpo $\langle P, \sqsubseteq_P \rangle$ is *inductive* iff $\sqsubseteq_P \subseteq \leq$ and, whenever D is a directed set in $\langle P, \sqsubseteq_P \rangle$ and $D \leq d$, $\bigsqcup D \leq d$.

In the remainder of this section, we briefly recall the definitions and results from [Sto88] that will be required in the sequel.

The reader is assumed to be familiar with many-sorted signatures Σ over sets of sorts S, as well as algebras over such signatures, i.e., Σ-algebras. Signatures are assumed to contain distinguished constants Ω_s at each sort s, which intuitively stand for divergence. Many operations and concepts extend naturally from sets to S-indexed families of sets, in a pointwise manner. For example, if A and B are S-indexed families of sets, then a function $f\colon A \to B$ is an S-indexed family of functions $f_s\colon A_s \to B_s$. We will make use of this and other such extensions without explicit comment. We use uppercase script letters ($\mathcal{A}$, $\mathcal{B}$, etc.) to denote algebras and the corresponding italic letters (A, B, etc.) to stand for their carriers.

We write $\mathcal{T}_\Sigma$ (or just $\mathcal{T}$) for the initial (term) algebra, so that T_s is the set of terms of sort s. Given an algebra $\mathcal{A}$ and a term t of sort s, $[\![t]\!]_\mathcal{A}$ (or just $[\![t]\!]$) is the

meaning of t in A_s, i.e., the image of t under the unique homomorphism from $\mathcal{T}$ to $\mathcal{A}$. Sometimes we write $t_{\mathcal{A}}$ (or even just t) for $[\![t]\!]_{\mathcal{A}}$.

An algebra is *reachable* iff all of its elements are denotable (definable) by terms. A pre-ordering over an algebra is *substitutive* iff it is respected by all of the operations of that algebra. Substitutive equivalence relations are called *congruences*, as usual. The congruence over $\mathcal{T}$ that is induced by an algebra $\mathcal{A}$ is called $\approx_{\mathcal{A}}$: two terms are congruent when they are mapped to the same element of A. When we say that $c[v_1, \ldots, v_n]$ is a *derived operator* of type $s_1 \times \cdots \times s_n \to s'$, this means that c is a context of sort s' over context variables v_i of sort s_i. We write $c_{\mathcal{A}}$ for the corresponding *derived operation* over an algebra $\mathcal{A}$.

The reader is also assumed to be familiar with *ordered algebras*, i.e., algebras $\mathcal{A}$ whose carriers are S-indexed families of posets $A_s = \langle A_s, \sqsubseteq_s \rangle$ with least elements $\bot_s$ denoted by the Ω_s constants, and whose operations are monotone functions. Such an algebra is called *complete* when its carrier is a dcpo and operations are continuous. A homomorphism over complete ordered algebras is called *continuous* when it is continuous on the underlying dcpo's. We write $\mathcal{OT}_\Sigma$ (or just $\mathcal{OT}$) for the initial ordered algebra, which consists of $\mathcal{T}$ with the "Ω-match" ordering: one term is less than another when the second can be formed by replacing occurrences of Ω in the first by terms. The substitutive pre-ordering over $\mathcal{T}$ that is induced by an ordered algebra $\mathcal{A}$ is called $\preceq_{\mathcal{A}}$: one term is less than another when the meaning of the first is less than that of the second in A.

Given complete ordered algebras $\mathcal{A}$ and $\mathcal{B}$, we say that $\mathcal{A}$ is an *inductive subalgebra* of $\mathcal{B}$ (written $\mathcal{A} \preceq \mathcal{B}$) iff $\mathcal{A}$ is a subalgebra of $\mathcal{B}$ and A is a sub-dcpo of B. Given a complete ordered algebra $\mathcal{A}$, we write $R(\mathcal{A})$ for the $\preceq$-least inductive subalgebra of $\mathcal{A}$. Its carrier $R(A)$ contains all of the elements reached by the transfinite process that starts with the denotable elements and closes under lub's of directed sets, and thus we are able to carry out proofs by induction on $R(A)$. A complete ordered algebra $\mathcal{A}$ is *inductively reachable* iff $\mathcal{A} = R(\mathcal{A})$. Complete ordered algebras whose carriers are ω-algebraic are inductively reachable iff all of their isolated elements are denotable. It is easy to see that $R(\mathcal{A})$ itself is inductively reachable.

If $\mathcal{A}$ is an algebra and R is a pre-ordering over A, then R is *unary-substitutive* iff all unary-derived operations respect R: for all derived operators $c[v]$ of type $s \to s'$ and $a, a' \in A_s$, if $a R_s a'$, then $c\langle a \rangle R_{s'} c\langle a' \rangle$. Unary-substitutive pre-orderings can fail

to be substitutive; see Lemma 2.2.27 of [Sto88] and Counterexample 2.1.22.

If $P \subseteq S$, $\mathcal{A}$ is an algebra and R is a pre-ordering over $A|P$ then R^c, the *contextualization* of R, is the relation over A defined by: $a\, R^c_s\, a'$ iff $c\langle a\rangle\, R_p\, c\langle a'\rangle$, for all derived operators $c[v]$ of type $s \to p$, $p \in P$.

Lemma 2.1.1 *If $P \subseteq S$, $\mathcal{A}$ is an algebra and R is a pre-ordering (respectively, equivalence relation) over $A|P$ then R^c is the greatest unary-substitutive pre-ordering (respectively, equivalence relation) over $\mathcal{A}$ whose restriction to P is included in R.*

Proof. See Lemma 2.2.25 of [Sto88].

Lemma 2.1.2 *If $\mathcal{A}$ is a complete ordered algebra and $\leq$ is an inductive pre-ordering over $A|P$, for $P \subseteq S$, then $\leq^c$ is a unary-substitutive, inductive pre-ordering over $\mathcal{A}$.*

Proof. See Lemma 2.3.14 of [Sto88].

Lemma 2.1.3 *(i) Unary substitutive pre-orderings over reachable algebras are substitutive.*

(ii) Unary substitutive, inductive pre-orderings over inductively reachable, complete ordered algebras are substitutive.

Proof. See Lemmas 2.2.29 and 2.3.35 of [Sto88].

2.1.2 Syntax and Semantics of PCF

For technical simplicity, we have chosen to work with a combinatory logic version of PCF with a single ground type ι, whose intended interpretation is the natural numbers. From the viewpoint of the conditional operations, non-zero and zero are interpreted as true and false, respectively.

The syntax of PCF is specified by a signature, the sorts of which consist of PCF's types. The set of sorts S is least such that

(i) $\iota \in S$, and

(ii) $s_1 \to s_2 \in S$ if $s_1 \in S$ and $s_2 \in S$.

As usual, we let $\to$ associate to the right. Define s^n, for $n \in \omega$, by: $s^0 = s$ and $s^{n+1} = s \to s^n$. The signature Σ over S has binary (application) operators $\cdot_{s_1,s_2}$ of

type $(s_1 \to s_2) \times s_1 \to s_2$ for all $s_1, s_2 \in S$, as well as the following constants (nullary operators), for all $s_1, s_2, s_3 \in S$:

(i) Ω_s of sort s,

(ii) K_{s_1,s_2} of sort $s_1 \to s_2 \to s_1$,

(iii) S_{s_1,s_2,s_3} of sort $(s_1 \to s_2 \to s_3) \to (s_1 \to s_2) \to s_1 \to s_3$,

(iv) Y_s of sort $s^1 \to s$,

(v) n of sort ι, for $n \in \omega$,

(vi) Succ and Pred of sort $\iota \to \iota$, and

(vii) If_s of sort $\iota \to s^2$.

We usually abbreviate $x \cdot y$ to $x\,y$, and let application associate to the left.

A *model* $\mathcal{A}$ is a complete ordered algebra such that the following conditions hold:

(i) $A_\iota = \{\bot_\iota, 0_\mathcal{A}, 1_\mathcal{A}, \ldots\}$, where $\bot_\iota \sqsubseteq n_\mathcal{A}$ for all $n \in \omega$ and $n_\mathcal{A}$ and $m_\mathcal{A}$ are incomparable whenever $n \neq m$ (we often confuse A_ι with $\omega_\bot$ below);

(ii) For all $a_1 \in A_{s_1}$ and $a_2 \in A_{s_2}$, $K_{s_1,s_2}\,a_1\,a_2 = a_1$;

(iii) For all $a_1 \in A_{s_1 \to s_2 \to s_3}$, $a_2 \in A_{s_1 \to s_2}$ and $a_3 \in A_{s_1}$, $S_{s_1,s_2,s_3}\,a_1\,a_2\,a_3 = a_1\,a_3\,(a_2\,a_3)$;

(iv) For all $a \in A_{s^1}$, $Y_s\,a$ is the least fixed point of the continuous function over A_s that a represents;

(v) For all $a \in A_\iota$, $\mathsf{Succ}\,a$ is equal to $\bot$, if $a = \bot$, and is equal to $a + 1$, if $a \in \omega$;

(vi) For all $a \in A_\iota$, $\mathsf{Pred}\,a$ is equal to $\bot$, if $a = \bot$, is equal to 0, if $a = 0$, and is equal to $a - 1$, if $a \in \omega - \{0\}$;

(vii) For all $a_1 \in A_\iota$ and $a_2, a_3 \in A_s$, $\mathsf{If}_s\,a_1\,a_2\,a_3$ is equal to $\bot$, if $a_1 = \bot$, is equal to a_2, if $a_1 \in \omega - \{0\}$, and is equal to a_3, if $a_1 = 0$;

A model $\mathcal{A}$ is *extensional* iff, for all $a_1, a_2 \in A_{s_1 \to s_2}$, if $a_1\,a = a_2\,a$ for all $a \in A_{s_1}$, then $a_1 = a_2$, and *order-extensional* iff, for all $a_1, a_2 \in A_{s_1 \to s_2}$, if $a_1\,a \sqsubseteq a_2\,a$ for all $a \in A_{s_1}$, then $a_1 \sqsubseteq a_2$. Finally, *morphisms* between models are simply continuous homomorphisms between the complete ordered algebras.

Application is left-strict in all models $\mathcal{A}$ since $\bot_{s_1 \to s_2} \sqsubseteq_{s_1 \to s_2} K_{s_2,s_1}\,\bot_{s_2}$, and thus $\bot_{s_1 \to s_2}\,a \sqsubseteq_{s_2} K_{s_2,s_1}\,\bot_{s_2}\,a = \bot_{s_2}$, for all $a \in A_{s_1}$.

The *continuous function model* $\mathcal{E}$ is the unique model $\mathcal{E}$ such that $E_\iota = \omega_\bot$, $E_{s_1 \to s_2} = E_{s_1} \stackrel{c}{\to} E_{s_2}$ for all $s_1, s_2 \in S$, application is function application and $n_\mathcal{A} = n$ for all $n \in \omega$. $\mathcal{E}$ is clearly order-extensional. The "parallel or" operation $\mathsf{por} \in E_{\iota^2}$ is defined by: $\mathsf{por}\,x\,y = 1$, if $x \in \omega - \{0\}$ or $y \in \omega - \{0\}$, $\mathsf{por}\,x\,y = 0$, if $x = 0$, and

$\mathsf{por}\, x\, y = \bot$, otherwise.

Lemma 2.1.4 *If $\mathcal{A}$ is a model, then so is $R(\mathcal{A})$.*

Proof. Follows easily from the fact that $R(\mathcal{A})$ is an inductive subalgebra of $\mathcal{A}$.

For $s \in S$, we write I_s for the term $S_{s,s^1,s}\, K_{s,s^1}\, K_{s,s}$ of sort s^1. I is the identity operation in all models. We code lambda abstractions in terms of the S, K and I combinators, in the standard way.

For $s \in S$, define approximations Y_s^n to Y_s of sort $s^1 \to s$ by $Y_s^0 = \Omega_{s^1 \to s}$ and $Y_s^{n+1} = S_{s^1,s,s}\, I_{s^1}\, Y_s^n$, so that Y_s^n is an ω-chain in the initial ordered algebra, and thus in all models.

Following [Mil77, BCL85], we can define syntactic projections Ψ_s^n of sort s^1, for all $n \in \omega$ and $s \in S$, by $\Psi_\iota^n = Y_{\iota^1}^n\, F$ and $\Psi_{s_1 \to s_2}^n = \lambda xy.\, \Psi_{s_2}^n (x(\Psi_{s_1}^n\, y))$, where F of sort $\iota^1 \to \iota^1$ is $\lambda xy.\, \mathsf{If}\, y\, (\mathsf{Succ}(x(\mathsf{Pred}\, y)))\, 0$. Expanding the abstractions, one can see that the Ψ_s^n form an ω-chain in the initial ordered algebra, and thus in all models. Given a model $\mathcal{A}$, we write A_s^n for the sub-poset of A_s whose elements are $\{\, \Psi_s^n\, a \mid a \in A_s \,\}$. Clearly $A_\iota^n = \{\bot, 0, 1, \ldots, n-1\}$ and $A_s^n \subseteq A_s^m$ if $n \leq m$.

Lemma 2.1.5 (Milner/Berry) *Suppose $\mathcal{A}$ is an extensional model, and let $s \in S$. The Ψ_s^n represent an ω-chain of continuous projections with finite image over A_s whose lub is the identify function. Hence A_s is a strongly algebraic dcpo whose set of isolated elements is $\bigcup_{n \in \omega} A_s^n$.*

Proof. The Ψ_s^n obviously represent an ω-chain of continuous functions. Inductions on S suffice to show that they are retractions, have finite image and that their lub is the identity function. But then each Ψ_s^n is less than the identity function. The rest follows easily.

Lemma 2.1.6 *Suppose $\mathcal{A}$ is an extensional model, and let $s \in S$. The Ψ_s^n also represent an ω-chain of continuous projections with finite image over $R(A)_s$ whose lub is the identity function. Hence $R(A)_s$ is a strongly algebraic dcpo and, for all $a \in A_s$,*

(i) x is isolated in $R(A)_s$ iff x is isolated in A_s and denotable.

(ii) $x \in R(A)_s$ iff $\Psi^n\, x$ is denotable for all $n \in \omega$.

Proof. Follows from Lemma 2.1.5 and the fact that $R(\mathcal{A})$ is the least inductive inductive subalgebra of $\mathcal{A}$.

We write por^2 for $\Psi^2\,\mathsf{por}$. It is easy to see that por^2 and por are interdefinable elements of E_{ι^2}.

Let the equality test Eq of sort ι^2 be

$$Y(\lambda zxy.\,\mathsf{If}\,x\,(\mathsf{If}\,y\,(z(\mathsf{Pred}\,x)(\mathsf{Pred}\,y))\,0)\,(\mathsf{Not}\,y)),$$

where Not of sort ι^1 is $\lambda x.\,\mathsf{If}\,x\,0\,1$.

For $n \in \omega$, define operators And_n of sort ι^n by: $\mathsf{And}_0 = 1$ and

$$\mathsf{And}_{n+1} = \lambda xy_1 \cdots y_n.\,\mathsf{If}\,x\,(\mathsf{And}_n\,y_1\ \cdots\ y_n)\,0.$$

Also following [Mil77, BCL85], define glb operators Inf^n_s of sort s^n, for $n \geq 1$, by:

$$\begin{aligned}\mathsf{Inf}^n_\iota &= \lambda y_1 \cdots y_n.\,\mathsf{If}\,(\mathsf{And}_{n-1}\,(\mathsf{Eq}\,y_1\,y_2)\,\cdots\,(\mathsf{Eq}\,y_1\,y_n))\,y_1\,\Omega\\ \mathsf{Inf}^n_{s_1\to s_2} &= \lambda y_1 \cdots y_n z.\,\mathsf{Inf}^n_{s_2}\,(y_1\,z)\,\cdots\,(y_n\,z).\end{aligned}$$

Lemma 2.1.7 (Milner) *If $\mathcal{A}$ is an order-extensional model, then $\mathsf{Inf}^n\,x_1 \cdots x_n$ is the glb of $\{x_1, \ldots, x_n\}$ in A_s, for all $x_1, \ldots, x_n \in A_s$, $n \geq 1$ and $s \in S$.*

Proof. By induction on S.

Lemma 2.1.8 *Suppose $\mathcal{A}$ is an order-extensional model, and let $s \in S$. Then, for all nonempty $X \subseteq A_s$ (respectively, $X \subseteq R(A)_s$), $\bigsqcup_{n\in\omega}(\sqcap(\Psi^n\,X))$ is the glb of X in A_s (respectively, $R(A)_s$). Thus A_s and $R(A)_s$ are consistently complete, ω-algebraic dcpo's.*

Proof. Suppose $X \subseteq A_s$ is nonempty, and let $x \in X$. Then $\sqcap(\Psi^n\,X) \sqsubseteq x$ for all $n \in \omega$, and thus $\bigsqcup_{n\in\omega}(\sqcap(\Psi^n\,X)) \sqsubseteq x$. Now, let y be a lb of X. Then $\Psi^n\,y \sqsubseteq \sqcap(\Psi^n\,X)$ for all $n \in \omega$, so that $y = \bigsqcup_{n\in\omega}(\Psi^n\,y) \sqsubseteq \bigsqcup_{n\in\omega}(\sqcap(\Psi^n\,X))$, completing the proof that $\bigsqcup_{n\in\omega}(\sqcap(\Psi^n\,X))$ is the glb of X in A_s. But, if $X \subseteq R(A)_s$, then each $\sqcap(\Psi^n\,X) \subseteq R(A)_s$ by Lemma 2.1.7, so that $\bigsqcup_{n\in\omega}(\sqcap(\Psi^n\,X)) \in R(A)_s$, as required. The rest follows by Lemmas 2.1.5 and 2.1.6.

Lemma 2.1.9 *If $\mathcal{A}$ is an order-extensional model, then $\Psi^n(\sqcap X) = \sqcap(\Psi^n\,X)$, for all $n \in \omega$ and finite, nonempty $X \subseteq A_s$.*

Proof. By induction on S, using the fact (Lemma 2.1.7) that finite, nonempty glb's are determined pointwise.

Since glb's of infinite subsets of E are not always determined pointwise, it is somewhat surprising that we have an infinitary version of the preceding lemma.

Lemma 2.1.10 *If $\mathcal{A}$ is an order-extensional model, then $\Psi^n(\sqcap X) = \sqcap(\Psi^n X)$, for all $n \in \omega$ and nonempty $X \subseteq A_s$.*

Proof. For all $x \in X$, we have that $\Psi^n(\sqcap X) \sqsubseteq \Psi^n x$. Thus $\Psi^n(\sqcap X) \sqsubseteq \sqcap(\Psi^n X)$. For the other direction, $\sqcap(\Psi^n X) = \sqcap(\Psi^n(\Psi^n X)) = \Psi^n(\sqcap(\Psi^n X)) \sqsubseteq \Psi^n(\sqcap X)$ by Lemma 2.1.9 and the fact that $\sqcap(\Psi^n X) \sqsubseteq \sqcap X$.

Following [Plo80], we say that an n-ary *logical relation* L over a model $\mathcal{A}$, for $n \in \omega$, is an n-ary relation over A such that $\langle x_1, \ldots, x_n \rangle \in L_{s_1 \to s_2}$ iff $\langle x_1\, y_1, \ldots, x_n\, y_n \rangle \in L_{s_2}$ for all $\langle y_1, \ldots, y_n \rangle \in L_{s_1}$. Given such an L and $\mathcal{A}$, we say that an element $a \in A_s$ *satisfies* L iff $\langle a, \ldots, a \rangle \in L_s$.

Lemma 2.1.11 *Suppose L is an n-ary logical relation over a model $\mathcal{A}$, $s \in S$ and $D_1, \ldots, D_n \subseteq A_s$ are directed sets such that, for all $x_i \in D_i$, $1 \leq i \leq n$, there are $y_i \in D_i$, $1 \leq i \leq n$, such that $x_i \sqsubseteq y_i$ for all i and $\langle y_1, \ldots, y_n \rangle \in L_s$. Then $\langle \bigsqcup D_1, \ldots, \bigsqcup D_n \rangle \in L_s$.*

Proof. By induction on S.

Lemma 2.1.12 *Suppose L is an n-ary logical relation over a model $\mathcal{A}$. If L is satisfied by Ω_ι, n, for all $n \in \omega$, Succ, Pred and If, then all elements of $R(A)$ satisfy L.*

Proof. First we must show that the remaining constants satisfy L. The satisfaction of L by K and S at all sorts follows as usual. One shows that Ω satisfies L at all sorts by induction on S, using the fact that application is strict in its left argument. The proof that If satisfies L at all sorts also proceeds by induction on S, using the fact that $\mathsf{If}_{s_1 \to s_2}\, x\, y\, z\, w = \mathsf{If}_{s_2}\, x\, (y\, w)\, (z\, w)$ for all $x \in A_\iota$, $y, z \in A_{s_1 \to s_2}$ and $w \in A_{s_1}$. Finally, the satisfaction of L by Y at all sorts follows using Lemma 2.1.11.

A simple induction on T then shows that all denotable elements of A satisfy L, following which we use Lemma 2.1.11 again to show, by induction on $R(A)$, that L is satisfied by all elements of $R(A)$.

Lemma 2.1.13 (Plotkin) *There is no $f \in R(E)_{\iota^2}$ such that $f \sqsupseteq \mathsf{por}^2$.*

Proof. Following [Sie92], let L be the ternary logical relation over $\mathcal{E}$ such that $\langle x_1, x_2, x_3\rangle \in L_\iota$ iff either $x_i = \bot$ for some i or $x_1 = x_2 = x_3$. It is easy to see that L satisfies the hypotheses of Lemma 2.1.12, and thus all elements of $R(E)$ satisfy L. Clearly, $\langle 1, \bot, 0\rangle \in L_\iota$ and $\langle \bot, 1, 0\rangle \in L_\iota$. Thus, if there were such an f, then we would have that $\langle x_1, x_2, x_3\rangle \in L_\iota$, where $x_1 = f\,1\,\bot$, $x_2 = f\,\bot\,1$ and $x_3 = f\,0\,0$. But $x_1 = 1$, $x_2 = 1$ and $x_3 = 0$—contradicting the definition of L.

The following theorem allows us to define the meaning $[\![M]\!] \in \omega_\bot$ of a term M of sort ι to be $[\![M]\!]_\mathcal{A}$, for an arbitrary model $\mathcal{A}$.

Theorem 2.1.14 (Plotkin) *For all models $\mathcal{A}$ and $\mathcal{B}$ and terms M of sort ι, $[\![M]\!]_\mathcal{A} = [\![M]\!]_\mathcal{B}$.*

Proof. See Theorem 3.1 of [Plo77].

We now define notions of program ordering and equivalence for PCF. Define a pre-ordering $\underset{\sim}{\sqsubseteq}$ over $T|\{\iota\}$ by $M \underset{\sim}{\sqsubseteq}_\iota N$ iff $[\![M]\!] \sqsubseteq [\![N]\!]$, and let $\approx$ be the equivalence relation over $T|\{\iota\}$ induced by $\underset{\sim}{\sqsubseteq}$. By Lemmas 2.1.1 and 2.1.3 (i), $\underset{\sim}{\sqsubseteq}^c$ is a substitutive pre-ordering over $\mathcal{T}$ and $\approx^c$ is a congruence over $\mathcal{T}$. It is easy to see that $\underset{\sim}{\sqsubseteq}^c$ induces $\approx^c$. We say that a model $\mathcal{A}$ is *inequationally fully abstract* iff $\preceq_\mathcal{A} = \underset{\sim}{\sqsubseteq}^c$. From [Plo77], we know that $\mathcal{E}$ is not inequationally fully abstract. On the other hand, by [Mil77], there exists a unique (up to order-isomorphism) inequationally fully abstract, order-extensional model.

Finally, we recall Milner's important result concerning the order-extensional nature of $\underset{\sim}{\sqsubseteq}^c$ and the extensional nature of $\approx^c$ [Mil77].

Lemma 2.1.15 (Milner) *(i) $\underset{\sim}{\sqsubseteq}^c_\iota = \underset{\sim}{\sqsubseteq}_\iota$ and $\approx^c_\iota = \approx_\iota$.*
(ii) If $M_1\,N \underset{\sim}{\sqsubseteq}^c_{s_2} M_2\,N$ for all $N \in T_{s_1}$, then $M_1 \underset{\sim}{\sqsubseteq}^c_{s_1\to s_2} M_2$.
(iii) If $M_1\,N \approx^c_{s_2} M_2\,N$ for all $N \in T_{s_1}$, then $M_1 \approx^c_{s_1\to s_2} M_2$.

Proof. See Lemma 4.1.11 of [Cur86].

From Lemma 2.1.15 (i), we know that, for all terms M of sort ι, either $M \approx^c_\iota \Omega$ or $M \approx^c_\iota n$ for some $n \in \omega$.

2.1.3 Normalization of $R(\mathcal{E})$

In this section, we focus on $\mathcal{E}$. Apart from the counterexamples, however, we could just as well work with any other order-extensional model, such as the bidomains model [BCL85]. We begin by defining semantic analogues of $\sqsubseteq^c_\sim$ and $\approx^c$.

Definition 2.1.16 *Define an inductive pre-ordering $\leq$ over $E|\{\iota\}$ by $x \leq_\iota y$ iff $x \sqsubseteq y$, and let $\equiv$ be the equivalence relation over $E|\{\iota\}$ induced by $\leq$.*

Clearly, $\equiv_\iota$ is just the identity relation over E_ι.

Lemma 2.1.17 *(i) $\leq^c$ is a unary-substitutive, inductive pre-ordering over $\mathcal{E}$.*

(ii) $\equiv^c$ is the unary-substitutive equivalence relation over $\mathcal{E}$ induced by $\leq^c$.

(iii) For all $M, N \in T_s$, $M \sqsubseteq^c_{\sim s} N$ iff $[\![M]\!] \leq^c_s [\![N]\!]$.

(iv) For all $M, N \in T_s$, $M \approx^c_s N$ iff $[\![M]\!] \equiv^c_s [\![N]\!]$.

Proof. (i) and (ii) follow from Lemmas 2.1.1 and 2.1.2, (iii) can be shown by a simple calculation, and (iv) follows from (iii).

Lemma 2.1.18 *(i) The restriction of $\leq^c$ to $R(E)$ is a substitutive, inductive pre-ordering over $R(\mathcal{E})$.*

(ii) The restriction of $\equiv^c$ to $R(E)$ is a congruence over $R(\mathcal{E})$.

Proof. (i) follows by Lemma 2.1.3 (ii), and (ii) follows from (i).

Lemma 2.1.19 *(i) $\leq^c_\iota = \leq_\iota$ and $\equiv^c_\iota = \equiv_\iota$.*

(ii) For all $x_1, x_2 \in R(E)_{s_1 \to s_2}$, if $x_1\, y \leq^c x_2\, y$ for all $y \in R(E)_{s_1}$, then $x_1 \leq^c x_2$.

(iii) For all $x_1, x_2 \in R(E)_{s_1 \to s_2}$, if $x_1\, y \equiv^c x_2\, y$ for all $y \in R(E)_{s_1}$, then $x_1 \equiv^c x_2$.

Proof. (i) follows from Lemma 2.1.15 (i). For (ii), it suffices to show that $\Psi^n\, x_1 \leq^c \Psi^n\, x_2$ for all $n \in \omega$, since $\leq^c$ is inductive. But isolated elements of $R(E)$ are denotable, and thus, by Lemma 2.1.15 (ii), it is sufficient to show that $\Psi^n\, x_1\, y \leq^c \Psi^n\, x_2\, y$ for all isolated $y \in R(E)_{s_1}$. But $\Psi^n(x_1(\Psi^n\, y)) \leq^c \Psi^n(x_2(\Psi^n\, y))$ follows from the hypothesis and Lemma 2.1.18, completing the proof of (ii). Finally, (iii) follows from (ii).

The following term features prominently below and is a generalization of the parallel or tester introduced in [Plo77].

Definition 2.1.20 *Let the term* Test *of sort* $\iota \to \iota \to \iota^2 \to \iota$ *be*

$$\begin{aligned}\lambda xyf.\, &\mathsf{If}\,(\mathsf{Eq}\,(f\,1\,y)\,1)\\ &(\mathsf{If}\,(\mathsf{Eq}\,(f\,x\,1)\,1)\\ &\quad(\mathsf{If}\,(f\,0\,0)\,\Omega\,0)\\ &\quad\Omega)\\ &\Omega.\end{aligned}$$

Lemma 2.1.21 *For all* $f \in E_{\iota^2}$, $\mathsf{Test}\,\bot\,\bot\,f$ *is* 0, *if* $f \sqsupseteq \mathsf{por}^2$, *and* $\bot$, *otherwise.* $\square$

The following is a counterexample to $\equiv^c$ (and thus $\leq^c$) being substitutive.

Counterexample 2.1.22 $\mathsf{Test}\,\bot\,\bot \equiv^c \bot$, *but* $\mathsf{Test}\,\bot\,\bot\,\mathsf{por} \not\equiv^c \bot\,\mathsf{por}$.

Proof. By Lemmas 2.1.19, 2.1.21 and 2.1.13, we have $\mathsf{Test}\,\bot\,\bot \equiv^c \bot$. But $\mathsf{Test}\,\bot\,\bot\,\mathsf{por} = 0$, and thus $\mathsf{Test}\,\bot\,\bot\,\mathsf{por} \not\equiv^c \bot\,\mathsf{por}$.

We do, however, have:

Lemma 2.1.23 *(i) For all* $x_1, x_2 \in E_{s_1 \to s_2}$ *and* $y \in R(E)_{s_1}$, *if* $x_1 \leq^c x_2$, *then* $x_1\,y \leq^c x_2\,y$.

(ii) For all $x \in R(E)_{s_1 \to s_2}$ *and* $y_1, y_2 \in E_{s_2}$, *if* $y_1 \leq^c y_2$, *then* $x\,y_1 \leq^c x\,y_2$.

Proof. For (i), since application is continuous and $\leq^c$ is inductive, it suffices to show $x_1\,y \leq^c x_2\,y$ when y is isolated. But this follows since all isolated elements of $R(E)_{s_1}$ are denotable and $\leq^c$ is unary-substitutive. (ii) follows similarly.

The following result shows that we cannot allow x_1, x_2 to range over $E_{s_1 \to s_2}$ in parts (ii) and (iii) of Lemma 2.1.19. This raises the question (which we leave unanswered) of when nondenotable elements are related by $\leq^c$ and $\equiv^c$.

Counterexample 2.1.24 *Define* $G_1, G_2 \in E_{\iota^2 \to \iota^2}$ *by*

$$G_1 = \lambda f.\mathsf{If}\,(f\,0\,0)\,\mathsf{por}\,(\lambda xy.\,\mathsf{Test}\,\Omega\,\Omega\,f), \qquad G_2 = \lambda f.\mathsf{If}\,(f\,0\,0)\,\mathsf{por}\,\Omega.$$

Then $G_1\,f \equiv^c G_2\,f$ *for all* $f \in R(E)_{\iota^2}$, *but* $G_1 \not\equiv^c G_2$.

Proof. It is easy to see that $G_1\, f \equiv^c G_2\, f$ for all $f \in R(E)_{\iota^2}$. But $c\langle G_1\rangle = 0$ and $c\langle G_2\rangle = \bot$, where the derived operator $c[v]$ of type $(\iota^2 \to \iota^2) \to \iota$ is $v\,(v(\lambda xy.\,1))\,\Omega\,\Omega$. Thus $G_1 \not\equiv^c G_2$.

We are now ready to define our continuous projection over $R(E)$.

Definition 2.1.25 *The function norm:* $R(E) \to R(E)$ *is defined by*

$$\mathsf{norm}_s\, x = \sqcap\{\, x' \in R(E)_s \mid x' \equiv^c x \,\}.$$

By Lemma 2.1.10, $\Psi^n(\mathsf{norm}\, x) = \sqcap\{\, \Psi^n\, x' \mid x' \equiv^c x \text{ and } x' \in R(E)_s \,\}$ for all $n \in \omega$ and $x \in R(E)_s$, $s \in S$. We write $x \sqsubseteq\!\equiv^c y$ for $x \sqsubseteq y$ and $x \equiv^c y$.

Lemma 2.1.26 *If X is a finite subset of $R(E)_s$ and $x' \in X$ is such that $x' \leq^c x$ for all $x \in X$, then $x' \equiv^c \sqcap X$.*

Proof. By induction on S.

Lemma 2.1.27 *Let $x, y \in R(E)_s$, $s \in S$, and $n \in \omega$.*

(i) $\mathsf{norm}\, x \sqsubseteq x$.

(ii) $\mathsf{norm}\, x \equiv^c x$.

(iii) If $x \sqsubseteq\!\equiv^c \mathsf{norm}\, y$, *then* $x = \mathsf{norm}\, y$.

(iv) $x \leq^c y$ *iff* $\mathsf{norm}\, x \sqsubseteq \mathsf{norm}\, y$.

(v) $x \equiv^c y$ *iff* $\mathsf{norm}\, x = \mathsf{norm}\, y$.

(vi) If $x \sqsubseteq y$, *then* $\mathsf{norm}\, x \sqsubseteq \mathsf{norm}\, y$.

(vii) $\mathsf{norm}(\mathsf{norm}\, x) = \mathsf{norm}\, x$.

(viii) $\mathsf{norm}(\Psi^n\, x) \sqsubseteq\!\equiv^c \Psi^n(\mathsf{norm}\, x)$.

(ix) $\Psi^n(\mathsf{norm}(\Psi^n\, x)) = \mathsf{norm}(\Psi^n\, x)$.

(x) $\mathsf{norm}\, x = \bigsqcup_{n\in\omega} \mathsf{norm}(\Psi^n\, x)$.

(xi) norm_s *is continuous.*

Proof. (i) Immediate from the reflexivity of $\equiv^c$.

(ii) By Lemma 2.1.26, we have that $\Psi^n\, x \equiv^c \sqcap\{\, \Psi^n\, x' \mid x' \equiv^c x \text{ and } x' \in R(E)_s \,\} \sqsubseteq \mathsf{norm}\, x$, for all $n \in \omega$. Thus $x \leq^c \mathsf{norm}\, x$, since $\leq^c$ is inductive. The result then follows by (i).

(iii) If $x \sqsubseteq\!\equiv^c \mathsf{norm}\, y$, then $x \equiv^c \mathsf{norm}\, y \equiv^c y$ by (ii), so that $\mathsf{norm}\, y \sqsubseteq x$. But then $x = \mathsf{norm}\, y$, since $x \sqsubseteq \mathsf{norm}\, y$.

(iv) The "if" direction follows from (ii) and the fact that $\sqsubseteq_s \subseteq \leq^c_s$. For the "only if" direction, suppose that $x \leq^c y$. Let $y' \in R(E)_s$ be such that $y' \equiv^c y$. Then $x \leq^c y'$, so that $x \sqcap y' \equiv^c x$ by Lemma 2.1.26. But then $\mathsf{norm}\, x \sqsubseteq x \sqcap y' \sqsubseteq y'$. Thus $\mathsf{norm}\, x \sqsubseteq \mathsf{norm}\, y$.

(v) Immediate from (iv).

(vi) Follows from (iv), since $\sqsubseteq_s \subseteq \leq^c_s$.

(vii) Follows by (i)–(iii).

(viii) Follows by (i), (ii) and (v).

(ix) Since $\mathsf{norm}(\Psi^n\, x) \equiv^c \Psi^n\, x$, we have $\Psi^n(\mathsf{norm}(\Psi^n\, x)) \equiv^c \Psi^n(\Psi^n\, x) = \Psi^n\, x \equiv^c \mathsf{norm}(\Psi^n\, x)$, and thus $\Psi^n(\mathsf{norm}(\Psi^n\, x)) \equiv^c \mathsf{norm}(\Psi^n\, x)$. The result then follows by (iii), since $\Psi^n(\mathsf{norm}(\Psi^n\, x)) \sqsubseteq \mathsf{norm}(\Psi^n\, x)$.

(x) By (vi) and (viii), $\mathsf{norm}(\Psi^n\, x) \sqsubseteq \mathsf{norm}\, x$ and $\bigsqcup_{n\in\omega} \mathsf{norm}(\Psi^n\, x) \geq^c \Psi^n(\mathsf{norm}\, x)$, for all $n \in \omega$. Thus $\bigsqcup_{n\in\omega} \mathsf{norm}(\Psi^n\, x) \sqsubseteq\equiv^c \mathsf{norm}\, x$, since $\leq^c$ is inductive. The result then follows by (iii).

(xi) Follows from (x).

Lemma 2.1.28 $\mathsf{norm}(\mathsf{Test}\, \bot\, \bot) = \bot$.

Proof. Follows from Counterexample 2.1.22.

Lemma 2.1.29 norm_ι *is the identity function on* $R(E)_\iota$.

Proof. Immediate by Lemma 2.1.19 (i).

The following counterexample shows that Lemma 2.1.27 (viii) cannot be strengthened to an identity.

Counterexample 2.1.30 $\mathsf{norm}(\Psi^2(\mathsf{Test}\, 2\, 2)) \neq \Psi^2(\mathsf{norm}(\mathsf{Test}\, 2\, 2))$.

Proof. Let the term A of sort ι^2 be

$$\begin{array}{l}\lambda xy.\, \mathsf{If}\,(\mathsf{And}_2\,(\mathsf{Eq}\, x\, 1)\,(\mathsf{Eq}\, y\, 2)) \\ \qquad 1 \\ \qquad (\mathsf{If}\,(\mathsf{And}_2\,(\mathsf{Eq}\, x\, 2)\,(\mathsf{Eq}\, y\, 1)) \\ \qquad\quad 1 \\ \qquad\quad (\mathsf{If}\,(\mathsf{And}_2\,(\mathsf{Eq}\, x\, 0)\,(\mathsf{Eq}\, y\, 0))\, 0\, \Omega)),\end{array}$$

so that $A \sqsubseteq \mathsf{por}^2$. Since $\mathsf{Test}\,2\,2\,A = 0$, it follows that $(\mathsf{norm}(\mathsf{Test}\,2\,2))A = 0$, and thus that $(\mathsf{norm}(\mathsf{Test}\,2\,2))\mathsf{por}^2 = 0$. But then

$$\Psi^2\,(\mathsf{norm}(\mathsf{Test}\,2\,2))\,\mathsf{por} = \Psi^2((\mathsf{norm}(\mathsf{Test}\,2\,2))\mathsf{por}^2) = \Psi^2\,0 = 0,$$

showing that $\Psi^2(\mathsf{norm}(\mathsf{Test}\,2\,2)) \neq \bot$. On the other hand, it is easy to show that $\Psi^2(\mathsf{Test}\,2\,2) = \mathsf{Test}\,\bot\,\bot$, and thus $\mathsf{norm}(\Psi^2(\mathsf{Test}\,2\,2)) = \bot$ by Lemma 2.1.28.

In preparation for three key counterexamples, we now define the following or operations of sort ι^2, where the "L", "R" and "D" stand for "Left", "Right" and "Divergent", respectively:

$$\begin{aligned}
\mathsf{LOr} &= \lambda xy.\,\mathsf{If}\,x\,1\,(\mathsf{If}\,y\,1\,0)\\
\mathsf{ROr} &= \lambda xy.\,\mathsf{If}\,y\,1\,(\mathsf{If}\,x\,1\,0)\\
\mathsf{DOr} &= \lambda xy.\,\mathsf{If}\,x\,(\mathsf{If}\,y\,\Omega\,1)\,(\mathsf{If}\,y\,1\,0).
\end{aligned}$$

Lemma 2.1.31 *There is no $h \in R(E)_{\iota\to\iota\to\iota^2\to\iota}$ such that*

$$h\,\bot\,\bot\,\mathsf{por} = \bot, \qquad h\,0\,0\,\mathsf{DOr} = 0, \qquad h\,0\,\bot\,\mathsf{LOr} = 0, \qquad h\,\bot\,0\,\mathsf{ROr} = 0.$$

Proof. Suppose, toward a contradiction, that such an h does exist.

Let L be the 4-ary logical relation over $\mathcal{E}$ such that $\langle x_1, x_2, x_3, x_4\rangle \in L_\iota$ iff $\{x_1, x_2, x_3, x_4\} \subseteq \{\bot, n\}$ for some $n \in \omega$ and, if $x_1 = \bot$, then one of x_2, x_3, x_4 is also $\bot$. Clearly, Ω_ι and all $n \in \omega$ satisfy L. Furthermore, Succ and Pred satisfy L since it is satisfied by all elements of E_{ι^1}. Finally, it is easy to show that L is satisfied by If. Hence h satisfies L, by Lemma 2.1.12.

Next, we show that $\langle \mathsf{por}, \mathsf{DOr}, \mathsf{LOr}, \mathsf{ROr}\rangle \in L_{\iota^2}$. Suppose that $\langle x_1, x_2, x_3, x_4\rangle \in L_\iota$ and $\langle y_1, y_2, y_3, y_4\rangle \in L_\iota$. We must show that $\langle z_1, z_2, z_3, z_4\rangle \in L_\iota$, where $z_1 = \mathsf{por}\,x_1\,y_1$, $z_2 = \mathsf{DOr}\,x_2\,y_2$, $z_3 = \mathsf{LOr}\,x_3\,y_3$ and $z_4 = \mathsf{ROr}\,x_4\,y_4$. Clearly, each $z_i \in \{\bot, 0, 1\}$. Furthermore, if $z_i = 0$ for some i, then both x_i and y_i must be 0, so that no x_j or y_j is a nonzero element of ω, and thus no $z_j = 1$. Now, suppose that $z_1 = \bot$. We must show that one of z_2, z_3, z_4 is $\bot$. Either x_1 or y_1 must be $\bot$, and we consider the case when $x_1 = \bot$, the other case being dual. Since DOr and LOr are strict in their first arguments, if $x_i = \bot$ for some $i \in \{2, 3\}$, then $z_i = \bot$. Otherwise, we must have that $x_4 = \bot$ and $x_2 = x_3 \neq \bot$. Now, if $y_4 \in \{\bot, 0\}$, then $z_4 = \bot$. Otherwise, $y_4 \in \omega - \{0\}$ and $y_1, y_2, y_3 \in \{\bot, y_4\}$. But then $y_1 = \bot$ (otherwise $z_1 = 1$), and thus either $y_2 = \bot$

or $y_3 = \bot$. Since DOr is also strict in its second argument, if $y_2 = \bot$, then $z_2 = \bot$. Otherwise, $y_3 = \bot$ and $y_2 = y_4$. Now, if $x_3 = 0$, then $z_3 = \bot$. Otherwise, we have that $x_2 = x_3 \in \omega - \{0\}$. But then $z_2 = \bot$, since both $x_2, y_2 \in \omega - \{0\}$.

Summarizing, we have that h satisfies L and $\langle \mathsf{por}, \mathsf{DOr}, \mathsf{LOr}, \mathsf{ROr} \rangle \in L_{\iota^2}$. Furthermore, $\langle \bot, 0, 0, \bot \rangle \in L_\iota$ and $\langle \bot, 0, \bot, 0 \rangle \in L_\iota$, so that $\langle z_1, z_2, z_3, z_4 \rangle \in L_\iota$, where $z_1 = h \bot \bot \mathsf{por}$, $z_2 = h\, 0\, 0\, \mathsf{DOr}$, $z_3 = h\, 0 \bot \mathsf{LOr}$ and $z_4 = h \bot 0\, \mathsf{ROr}$. But $z_1 = \bot$ and $z_2 = z_3 = z_4 = 0$, contradicting the definition of L.

The following counterexample shows that application is not preserved by norm.

Counterexample 2.1.32 $\mathsf{norm}(\mathsf{Test}\,\bot) \neq (\mathsf{norm}\,\mathsf{Test})(\mathsf{norm}\,\bot)$.

Proof. By Lemma 2.1.19 (iii) and Counterexample 2.1.22, we have that

$$\mathsf{Test}\,\bot \equiv^c \lambda y.\, \mathsf{If}\, y\, (\mathsf{Test}\,\bot\, y)\, (\mathsf{Test}\,\bot\, y),$$

so that $(\mathsf{norm}(\mathsf{Test}\,\bot))\,\bot\,\mathsf{por} = \bot$. Since $\mathsf{norm}\,\bot = \bot$, it is thus sufficient to show that $h \bot \bot \mathsf{por} \neq \bot$, where $h = \mathsf{norm}\,\mathsf{Test}$. But

$$h\, 0\, 0\, \mathsf{DOr} = 0, \qquad h\, 0 \bot \mathsf{LOr} = 0, \qquad h \bot 0\, \mathsf{ROr} = 0,$$

since $h \equiv^c \mathsf{Test}$, and thus $h \bot \bot \mathsf{por} \neq \bot$ by Lemma 2.1.31.

Since $\mathsf{norm}((\mathsf{norm}\,\mathsf{Test})(\mathsf{norm}\,\bot)) = \mathsf{norm}(\mathsf{Test}\,\bot)$, it follows from the preceding counterexample that the image of norm is not closed under application.

Counterexample 2.1.33 *There is no* $\mathsf{norm}' \in R(E)_{(\iota \to \iota^2 \to \iota)^1}$ *such that* $\mathsf{norm}'\, x = \mathsf{norm}\, x$ *for all* $x \in R(E)_s$.

Proof. Suppose, toward a contradiction, that such a norm' does exist. Then, for all $y \in R(E)_\iota$,

$$\mathsf{Test}\, y \equiv^c \mathsf{norm}(\mathsf{Test}\, y) = \mathsf{norm}'(\mathsf{Test}\, y) = (\lambda y.\, \mathsf{norm}'(\mathsf{Test}\, y))\, y,$$

so that $\mathsf{Test} \equiv^c \lambda y.\, \mathsf{norm}'(\mathsf{Test}\, y)$. Then,

$$\begin{aligned}
(\mathsf{norm}\,\mathsf{Test})(\mathsf{norm}\,\bot) &= (\mathsf{norm}(\lambda y.\, \mathsf{norm}'(\mathsf{Test}\, y)))\,\bot \\
&\sqsubseteq\equiv^c (\lambda y.\, \mathsf{norm}'(\mathsf{Test}\, y))\,\bot \\
&= \mathsf{norm}'(\mathsf{Test}\,\bot) \\
&= \mathsf{norm}(\mathsf{Test}\,\bot).
\end{aligned}$$

But then $(\mathsf{norm}\,\mathsf{Test})(\mathsf{norm}\,\bot) = \mathsf{norm}(\mathsf{Test}\,\bot)$, contradicting Counterexample 2.1.32.

The following counterexample shows that denotable elements can be contextually equivalent to nondenotable ones.

Counterexample 2.1.34 *$h \equiv^c \mathsf{Test}$ does not imply that $h \in R(E)$.*

Proof. Let $h \in E_{\iota\to\iota\to\iota^2\to\iota}$ be $\lambda xy.\,\mathsf{If}\,(\mathsf{pcon}\,x\,y)\,(\mathsf{Test}\,x\,y)\,\Omega$, where the "parallel convergence" operation $\mathsf{pcon} \in E_{\iota^2}$ is defined by: $\mathsf{pcon}\,x\,y = 1$, if $x \neq \bot$ or $y \neq \bot$, and $\mathsf{pcon}\,x\,y = \bot$, otherwise. Then $h \notin R(E)$, by Lemma 2.1.31. It remains to show that $h \equiv^c \mathsf{Test}$.

In the remainder of the proof, we work in the result of adding to PCF a constant PCon of sort ι^2 whose interpretation is pcon. All of the results preceding Lemma 2.1.31 hold for the extended language, with the exception of Lemma 2.1.12. This lemma can be repaired, however, by adding PCon to the list of constants in its hypothesis. The logical relation defined in the proof of Lemma 2.1.13 is also satisfied by PCon and thus this lemma is true for the extended language. (The original proof that parallel or is not definable from parallel convergence can be found in [Abr90].)

It is sufficient to show $h \equiv^c \mathsf{Test}$, and, since $h \in R(E)$, this will be a consequence of showing that $h\,x\,y \equiv^c \mathsf{Test}\,x\,y$ for all $x, y \in R(E)_\iota$. If $x \neq \bot$ or $y \neq \bot$, then $h\,x\,y = \mathsf{Test}\,x\,y$. But $\mathsf{Test}\,\bot\,\bot \equiv^c \bot$ was shown in Counterexample 2.1.22.

Although we were able to solve negatively the question of whether norm preserves application, the following problem is still open.

Open Problem 2.1.35 *Is $\mathsf{norm}\,\sigma = \sigma$ for all constants $\sigma \in \Sigma$? In particular, is $(\mathsf{norm}\,K)\,x\,y$ ever strictly less than x?*

Now, we are able to show how the unique inequationally fully abstract, order-extensional model lives inside the continuous function model.

Definition 2.1.36 *We define the ordered algebra $N(\mathcal{E})$ as follows. For all $s \in S$, $N(E)_s$ consists of $\mathsf{norm}\,R(E)_s$, ordered by the restriction of $\sqsubseteq_{E_s}$ to $\mathsf{norm}\,R(E)_s$. For all $x \in N(E)_{s_1\to s_2}$ and $y \in N(E)_{s_1}$, $x \cdot_{N(\mathcal{E})} y = \mathsf{norm}(x \cdot_{\mathcal{E}} y)$. For all constants σ, $\sigma_{N(\mathcal{E})} = \mathsf{norm}\,\sigma_{\mathcal{E}}$.*

$N(E)$ is a sub-dcpo of $R(E)$ and $N(\mathcal{E})$ is well-defined, since norm is strict and continuous.

Theorem 2.1.37 *$N(\mathcal{E})$ is an order-extensional model and* norm *is a surjective morphism from $R(\mathcal{E})$ to $N(\mathcal{E})$.*

Proof. $N(\mathcal{E})$ is a complete ordered algebra by the preceding remark and the continuity of norm. Condition (i) of the definition of model holds by Lemma 2.1.29, and the remaining conditions can be shown using Lemma 2.1.27 (ii) and (v) and (for condition (iv)) the continuity of norm. For the order-extensionality of $N(\mathcal{E})$, suppose that $x_1, x_2 \in N(E)_{s_1 \to s_2}$ are such that $x_1 \cdot_{N(\mathcal{E})} y \sqsubseteq x_2 \cdot_{N(\mathcal{E})} y$ for all $y \in N(E)_{s_1}$. Then, for all $y \in R(E)_{s_1}$,

$$\mathsf{norm}(x_1 \cdot_{\mathcal{E}} y) = x_1 \cdot_{N(\mathcal{E})} \mathsf{norm}\, y \sqsubseteq x_2 \cdot_{N(\mathcal{E})} \mathsf{norm}\, y = \mathsf{norm}(x_2 \cdot_{\mathcal{E}} y),$$

and thus $x_1 \cdot_{\mathcal{E}} y \leq^c x_2 \cdot_{\mathcal{E}} y$. But then $x_1 \leq^c x_2$ by Lemma 2.1.19 (ii), so that $x_1 = \mathsf{norm}\, x_1 \sqsubseteq \mathsf{norm}\, x_2 = x_2$. Finally, norm is a surjective morphism from $R(\mathcal{E})$ to $N(\mathcal{E})$ because of the way $N(\mathcal{E})$ was defined.

By Lemma 2.1.8, we know that $N(E)$ is a consistently complete, ω-algebraic dcpo.

Lemma 2.1.38 *For all term M, $[\![M]\!]_{N(\mathcal{E})} = \mathsf{norm}[\![M]\!]_{\mathcal{E}}$.*

Proof. A consequence of norm being a morphism from $R(\mathcal{E})$ to $N(\mathcal{E})$.

Theorem 2.1.39 *$N(\mathcal{E})$ is inequationally fully abstract.*

Proof. Follows from Lemmas 2.1.17 (iii) and 2.1.38.

2.1.4 Full Abstraction and Lambda Definability

There appears to be no clear definition of what the "full abstraction problem" for PCF really is. By Milner's construction [Mil77] we know that there is a unique inequationally fully abstract, order-extensional model $\mathcal{F}$ (which we refer to below as *the fully abstract model*) that is made up out of Scott-domains of continuous (set-theoretic) functions. Why are we not satisfied? The answer to this question, as one often reads, is that Milner's model is "syntactic in nature". The same words are used against Mulmuley's description [Mul87] of the fully abstract model. What people vaguely imagine is that there ought to be a description of $\mathcal{F}$ using dcpo's enriched with some additional structure (order-theoretic, topological, etc.) which

allows the domains of the fully abstract model to be constructed without recourse to the syntax of PCF. Of course, nobody can specify what this additional structure will be or should be. Stated this way, there is no chance to falsify this research programme, in the sense that there is no way one can prove a result saying that there is no "semantic" presentation of $\mathcal{F}$.

We would therefore like to give a weak but precise minimal condition that a semantic solution of the full abstraction problem should satisfy. Namely, it should allow us to *effectively* construct the finite domains F_s of the fully abstract model $\mathcal{F}$ of *finitary PCF*, i.e., the variant of PCF in which the sort ι is interpreted as the booleans ($\{\bot, 0, 1\}$) rather than the natural numbers. (The result of this paper can be trivially adapted to finitary PCF.) Clearly, neither Milner's nor Mulmuley's constructions achieve this. On the other hand, even if we can find such an algorithm for presenting $\mathcal{F}$, we may still be unsatisfied with it as a semantic description.

The results of this paper give one of the simplest descriptions of the fully abstract model to date. In order to satisfy the above condition, all one needs to find is an algorithm that decides whether an element of E is denotable, since then one will be able to effectively present $R(E)$ and thus $N(E)$.

The problem of deciding which elements of a model are definable in the case of the typed lambda calculus (without constants) and the full set-theoretic type hierarchy based on a finite set is known as "Plotkin's conjecture". (It seems that the term was coined by Statman in his 1982 paper [Sta82a]. We do not know whether Plotkin ever considered the question nor whether he ever conjectured anything.) The "conjecture" is that the problem is decidable. We prefer to call it the "lambda definability problem" (cf. [JT93]). This problem can be studied in all kinds of contexts, and certainly it makes sense to ask whether it is decidable which elements of E are denotable.

Since a positive solution to the lambda definability problem for PCF will mean that $N(\mathcal{E})$ and thus $\mathcal{F}$ are effectively presentable, it is natural to ask whether the converse is also true. We conjecture that it is.

2.2 The Classification of Continuous Domains

Introduction

The first spaces suitable for the interpretation of programming language constructs were *continuous lattices* discovered by Dana Scott in the late sixties. Continuous lattices turned out to have numerous connections to other fields of mathematics such as algebra, topology, and convex analysis. An indication of this is the voluminous *Bibliography of Continuous Lattices* contained in [HH87].

In Computer Science, however, it was soon recognized that the subclass of *algebraic lattices* is fully sufficient for the purposes of semantics. Indeed, the basic concept of finite pieces of information corresponds nicely to the idea of compact elements in these structures. Generality was sought in a different direction, namely, in the way the least upper bound of pieces of information was to be formed. This led to a variety of different classes of *domains*: Lattices, meet-semilattices (= Scott-domains), SFP-objects, to name a few.

It is our belief that continuous domains do have a similar importance for computer science in areas largely still to be developed. One application is the analysis of probabilistic algorithms. Here the central domain is clearly the unit interval, a non-algebraic but continuous lattice. Some work in this direction has been carried out in [Gra88, JP89].

Looking at all those different definitions of domains the novice in the field will naturally ask for some orientation. And indeed, it is possible to give a rather complete overview once the basic assumption is shared that a collection of domains should form a cartesian closed category. Michael Smyth [Smy83] showed 1983 that there is a largest cartesian closed full subcategory in the class of all countably based algebraic dcpo's with least element. In his doctoral thesis [Jun89] the present author completely described all categories of algebraic domains with respect to that criterion of cartesian closedness. It is the purpose of the present note to do the same for continuous domains.

It is an easy exercise to show that any Scott-continuous retract of an algebraic dcpo is a continuous dcpo and it is equally simple to see that the class of all such retracts is cartesian closed if one starts with a cartesian closed category. This immediately gives us a class of continuous domains for each class of algebraic domains.

It is then an obvious question whether this will give us the whole variety on the continuous side. As Smyth notes at the end of his paper [Smy83] "(this result) does not come out by manipulating retractions". It turned out to be a very hard problem, indeed. The solution, which was partly provided in [Jun89] and is completed here, involves the definition of two new classes of domains: *L-domains* and *FS-domains*. We will show below that each cartesian closed category of continuous domains with least element consists of continuous L-domains or of FS-domains. The special question, whether the retracts of SFP-objects form a maximal class we leave unanswered. They are FS-domains but we do not know whether this containment is proper.

FS-domains do have a distinctive advantage over SFP-retracts: They are easy to discover. This is illustrated below by showing that the collection of all closed discs in the plane together with the plane itself (the ordering being reversed inclusion) forms a countably based FS-domain. Even for this well-structured concrete example it appears to be extremely hard to decide whether it is an SFP-retract.

2.2.1 Background

Our notation will be fairly standard. We call directed-complete partial orders *dcpo's* and do not generally assume that they have a least element. If a dcpo does have a bottom element then we call it *pointed*. A dcpo is *continuous* if every element is the directed supremum of elements way-below it, where an element x is *way-below* an element y ($x \ll y$) if whenever the sup of a directed set is above y then some element of the directed set is above x. A subset B is a *basis* if every element x is the directed sup of base elements way-below x. A dcpo is *countably based* or *ω-continuous* if it has a countable basis.

Our functions are *Scott-continuous* that is, they preserve directed sups. Dcpo's together with Scott-continuous maps form a cartesian closed category **DCPO**. The full subcategories **CONT** and $\mathbf{CONT}_\perp$ of continuous dcpo's (with bottom) are not cartesian closed. It is the purpose of this note to describe all maximal cartesian closed full subcategories of $\mathbf{CONT}_\perp$.

The basic properties of the way-below relation are summarized in the following lemma.

Lemma 2.2.1 *If D is a continuous dcpo then the following holds for all*

$x, x', y, y' \in D$:

(i) $x \ll y \Longrightarrow x \leq y$.

(ii) $x' \leq x \ll y \leq y' \Longrightarrow x' \ll y'$.

(iii) $x \ll y \Longrightarrow \exists z.x \ll z \ll y$. ∎

Given a pointed dcpo E and a dcpo D and given elements $e \in E$ and $d \in D$ we can define the *step function* $(d \searrow e)$ as follows:

$$(d \searrow e)(x) = \begin{cases} e, & \text{if } d \ll x; \\ \bot, & \text{otherwise.} \end{cases}$$

A step function is always Scott-continuous. If x' is way-below x in D then the step function $(x \searrow x')$ is way below the identity function id_D on D. In a continuous dcpo we can interpolate between x' and x with elements y' and y: $x' \ll y' \ll y \ll x$. It is then easy to check that the step function $(x \searrow x')$ is way-below the step function $(y \searrow y')$ in the dcpo $[D \longrightarrow D]$.

2.2.2 Continuous L-domains

Definition 2.2.2 *A dcpo D is an* L-domain *if it is pointed and if every principal ideal in D is a complete lattice. The category of continuous L-domains is denoted by* **cL**.

L-domains were discovered by the present author and by T.Coquand [Coq88, Coq89] independently. A thorough treatment of their main properties can be found in [Jun89], where it was shown already that they form a maximal cartesian closed full subcategory of $\mathbf{CONT}_\bot$ ('Theorem 4.25').

Continuous L-domains occur in 'nature': Given a compact connected and locally connected space X the collection of all closed connected nonempty subsets of X ordered by reversed inclusion forms a continuous L-domain. This example is due to Klaus Keimel and Jimmie Lawson.

2.2.3 FS-domains

It was generally conjectured (see for example [Smy83, KT84]) that the retracts of SFP-objects (or rather: bifinite domains, see [Jun89, Tay87b]) form another maximal cartesian closed full subcategory of $\mathbf{CONT}_\perp$ and that there are no other. In what follows we shall characterize the second maximal class and show that every cartesian closed full subcategory of $\mathbf{CONT}_\perp$ is contained in one of the two. This second class will consist of FS-domains, which are introduced here for the first time. They contain the retracts of SFP-objects, but it is open whether this inclusion is strict. However, we hope to convince the reader that FS-domains are preferable to SFP-retracts anyway.

Definition 2.2.3 *Let $f, g: D \to E$ be functions from a set D to a dcpo E. We say that f is* finitely separated *from g if there exists a finite subset M of E such that for every $x \in D$ there is some $m \in M$ such that $f(x) \leq m \leq g(x)$ holds. The function f is* strongly finitely separated *from g if there exists a finite set M of pairs $(m', m) \in E \times E$ with $m' \ll m$ such that for every $x \in D$ there is some pair from M between $f(x)$ and $g(x)$. We will mostly need functions $f: D \to D$ separated from the identity id_D.*

Lemma 2.2.4 *Let $f: D \to D$ be a Scott-continuous function on a dcpo D finitely separated from id_D. Then $f(x) \ll x$ holds for every $x \in D$, $f \circ f$ is strongly finitely separated from id_D and way-below id_D in $[D \longrightarrow D]$.* ∎

Definition 2.2.5 *A pointed dcpo D is called an* FS-domain *if there exists a directed family $(f_i)_{i \in I}$ of Scott-continuous functions, each finitely separated from id_D, with supremum id_D. The category of FS-domains with Scott-continuous functions as arrows is denoted by* **FS**.

By the preceding lemma it is obvious that FS-domains are continuous. Considering the characterization ('Theorem 4.1') in [Jun89], it is also clear that **FS** contains all retracts of bifinite domains. In fact, **FS** has all closure properties one usually expects from a category of domains:

Theorem 2.2.6 *Any product of FS-domains is an FS-domain and the inverse limit of FS-domains is an FS-domain. Also,* **FS** *is a cartesian closed subcategory of* $\mathbf{CONT}_\perp$.

Proof. We show that the function space $[D \longrightarrow E]$ for FS-domains D and E is again an FS-domain. Let $f: D \to D$ be finitely separated from id_D and $g: E \to E$ be finitely separated from id_E. We show that the function $F: [D \longrightarrow E] \to [D \longrightarrow E]$, defined by $\lambda h.g \circ g \circ h \circ f \circ f$, is finitely separated from $id_{[D \to E]}$. Let M_f, M_g be finite separating sets for f and g, respectively. Define an equivalence relation on $[D \longrightarrow E]$ by

$$h_1 \sim h_2 \iff \forall m \in M_f.\ \uparrow g \circ h_1(m) \cap M_g = \uparrow g \circ h_2(m) \cap M_g.$$

Obviously there are only finitely many equivalence classes on $[D \longrightarrow E]$. Let $\widetilde{M_F}$ be a set of representatives from each class. We show that $M_F = g \circ \widetilde{M_F} \circ f$ is a separating set for F. Given $h \in [D \longrightarrow E]$ let $\bar{h}$ be the corresponding representative in $\widetilde{M_F}$. We calculate for an $x \in D$:

$$\begin{array}{lll} h(x) \geq h(m_f) & \text{;for some } m_f \in M_f \text{ with } f(x) \leq m_f \leq x \\ \geq m_g & \text{;for some } m_g \in M_g \text{ with } g(h(m_f)) \leq m_g \leq h(m_f) \\ \geq g(\bar{h}(m_f)) & \text{;because } g(h(m_f)) \leq m_g \text{ and } h \sim \bar{h} \\ \geq g(\bar{h}(f(x))) & \text{;because } f(x) \leq m_f. \end{array}$$

By symmetry we also have $\bar{h}(x) \geq g(h(f(x)))$ and hence $g \circ \bar{h} \circ f \geq g \circ g \circ h \circ f \circ f$. So indeed: $h \geq g \circ \bar{h} \circ f \geq F(h)$. ∎

Definition 2.2.7 *For a continuous dcpo D the* Lawson-topology λ_D *is generated by the subbasic open sets $\Uparrow x, x \in D$ and $D \setminus \uparrow x, x \in D$.*

On continuous dcpo's the Lawson-topology will always be Hausdorff. With the results in [Jun89] or in [Law88] it is easy to see that FS-domains are Lawson-compact. In fact, the Lawson-topology is closely connected to the function space:

Theorem 2.2.8 *Let D be an FS-domain.*

(i) For each $f \ll id_D$ we define an entourage $\mathcal{U}_f = \{(x, y) \in D \times D \mid f(x) \leq y \wedge f(y) \leq x\}$ resulting in a basis $(\mathcal{U}_f)_{f \ll id_D}$ for a uniformity on D. The corresponding topology equals λ_D.

(ii) A function $f: D \to D$ is way-below id_D if and only if it is strongly finitely separated from id_D. ∎

We finish this section with the discussion of a concrete example of an FS-domain. (It was suggested to me by Jimmie Lawson.) Let Disc be the collection of all closed discs in the plane plus the plane itself, ordered by reversed inclusion. One checks that the filtered intersection of discs is again a disc, so Disc is a dcpo. A disc d_1 is way-below a disc d_2 if and only if d_1 is a neighborhood of d_2. This proves that Disc is continuous. For every $\epsilon > 0$ we define a map f_ϵ on Disc as follows. All discs inside the open disc with radius $\frac{1}{\epsilon}$ are mapped to their closed ϵ-neighborhood, all other discs are mapped to the plane which is the bottom element of Disc. Because the closed discs contained in some compact set form a compact space under the Hausdorff subspace topology, these functions are finitely separated from the identity map. This proves that Disc is a countably based FS-domain. We do not know whether this domain is a retract of an SFP-object.

2.2.4 The classification

The following lemma, which we cite from [Jun89] is the starting point for our classification:

Lemma 2.2.9 *Let D and E be continuous pointed dcpo's with property m. If $[D \longrightarrow E]$ is continuous then E is an L-domain or D is Lawson-compact.* ∎

Lawson-compact dcpo's do not form a cartesian closed category. Indeed, we are now going to show that FS-domains are the largest cartesian closed full subcategory of $\mathbf{CONT}_\perp$ which consists of Lawson-compact domains only.

Definition 2.2.10 *For any dcpo D and any $d \in D$ the retraction $r_d: D \to D$ is defined by $r_d(x) = x$ if $x \leq d$ and $r_d(x) = d$ otherwise.*

Lemma 2.2.11 *If a function $f: D \to D$ is below r_x and r_y then $f(x) \leq x, f(x) \leq y, f(y) \leq x$, and $f(y) \leq y$.* ∎

The following Lemma appears also in [Law88]

Lemma 2.2.12 *If D is a dcpo with continuous and Lawson-compact function space $[D \longrightarrow D]$ and if $f \ll id_D$ holds then there exist pairs $x'_1 \ll x_1, \ldots, x'_n \ll x_n$ such that every upper bound of the step functions $(x_i \searrow x'_i)$, $i = 1, \ldots, n$, is above f.*

Proof. $\Uparrow f$ is a Lawson-neighborhood of $\uparrow id_D$. Since $[D \longrightarrow D]$ is Lawson-compact, each of the sets $\uparrow(x_1 \searrow x'_1) \cap \ldots \cap \uparrow(x_m \searrow x'_m)$, for any finite set of pairs $x'_1 \ll x_1, \ldots, x'_m \ll x_m$, is Lawson-compact. The intersection of all these sets is filtered and equals $\uparrow id_D$. Therefore one of them is already contained in $\Uparrow f$. ∎

Theorem 2.2.13 *If D and $[D \longrightarrow D]$ are continuous and Lawson-compact and if $f \ll id_D$ then f is finitely separated from id_D.*

Proof. Let $g \ll id_D$ be such that $f \leq g \circ g$ and let $X_1 = (x_1 \searrow x'_1), \ldots, X_n = (x_n \searrow x'_n)$ be step functions such that any upper bound of them is above g according to Lemma 2.2.12. For each $i \in I = \{1, \ldots, n\}$ interpolate between x'_i and x_i to get y'_i, y_i such that $x'_i \ll y'_i \ll y_i \ll x_i$ and let Y_i be the step function $(y_i \searrow y'_i)$. We noted above that $X_i \ll Y_i$ holds in $[D \longrightarrow D]$. Also note that for each $x \in D \setminus O$ — where $O = \Uparrow y_1 \cup \ldots \cup \Uparrow y_n$ — $g(x) = \bot$ holds. That is because the function which maps each element of O onto itself and everything else onto bottom is above all Y_i and hence above g.

For each $x \in O$ consider the retraction r_x. The element x is way-above some of the y_i but not necessarily above all of them. Call the subset of I for which $y_i \ll x$, I_x. Then r_x is above all Y_i with $i \in I_x$, because $r_x(e) = x \geq y_i \geq y'_i \geq Y_i(e)$ for $e \not\leq x$ and $r_x(e) = e = id_D(e) \geq Y_i(e)$ otherwise.

Claim: *If $h\colon D \to D$ is below r_x and above all X_i with $i \in I_x$, then $h\,|_{\downarrow x} \geq g\,|_{\downarrow x}$.*

Define $h'\colon D \to D$ by $h'(e) = e$ if $e \not\leq x$ and $h'(e) = h(e)$ otherwise. This is continuous because $h'\,|_{\downarrow x} = h\,|_{\downarrow x} \leq r_x\,|_{\downarrow x} = id_D\,|_{\downarrow x}$ and $h'\,\big|_{D \setminus \downarrow x} = id_D\,\big|_{D \setminus \downarrow x}$. The map h' is above all step functions X_i:

Case 1: $e \not\leq x : h'(e) = e = id_D(e) \geq X_i(e), i \in I$.

Case 2a: $e \leq x, i \in I_x : h'(e) = h(e) \geq X_i(e)$ by assumption.

Case 2b: $e \leq x, i \in I \setminus I_x : e \notin \Uparrow x_i \subseteq \Uparrow y_i$ by the definition of I_x, so $h'(e) = h(e) \geq \bot = X_i(e)$.

So h' is above g and hence $h\,|_{\downarrow x} = h'\,|_{\downarrow x} \geq g\,|_{\downarrow x}$. This proves our claim.

Now let J be some subset of I. Since $[D \longrightarrow D]$ is Lawson-compact there exists a finite set M_J contained in $\bigcap\{\uparrow X_i \mid i \in J\}$ such that every upper bound of $\{Y_i \mid i \in J\}$ is above some $h \in M_J$, that is

$$\bigcap_{i \in J} \uparrow Y_i \subseteq \bigcup_{h \in M_J} \uparrow h.$$

In particular, for a given $x \in D$, there is $h \in M_{I_x}$ with $h \leq r_x$. We now take all h from each M_J that we need, that is:

$$FM = \{h \in \textstyle\bigcup_{J \subseteq I} M_J \mid \quad \exists x \in D. I_x = J \wedge \\ \wedge h \leq r_x \wedge h \in M_J\}.$$

A function in FM will in general be below many r_x with $J = I_x$. We select just one x_h for each $h \in FM$ and define

$$M = \{h(x_h) \mid h \in FM\}.$$

It remains to show that M separates f from the identity on D. To this end, let x be some arbitrary but fixed element in D and let $h \in M_{I_x}$ be such that $r_x \geq h$. x is not necessarily equal to x_h but we have $h \leq r_x, r_{x_h}$ and we can apply Lemma 2.2.11: $h(x_h) \leq x$ and $h(x) \leq x_h$, also, $h \geq X_i$ for all $i \in I_x$ by construction. Hence by the 'Claim' above, $h\,|_{\downarrow x} \geq g\,|_{\downarrow x}$ and $h\,|_{\downarrow x_h} \geq g\,|_{\downarrow x_h}$. So we can calculate:

$$\begin{array}{lcll} x & \geq & h(x_h) & \text{Lemma 2.2.11} \\ & \geq & g(x_h) & \text{'Claim'} \\ & \geq & g(h(x)) & \text{Lemma 2.2.11} \\ & \geq & g(g(x)) & \text{'Claim'} \\ & \geq & f(x) & \text{by construction.} \end{array}$$

Thus with $m = h(x_h)$ we have found a separating element in M between x and $f(x)$. ∎

Corollary 2.2.14 *If D and $[D \longrightarrow D]$ are Lawson-compact and continuous then D is an FS-domain.* ∎

Corollary 2.2.15 *Every cartesian closed full subcategory of* $\mathbf{CONT}_\perp$ *is contained in* **cL** *or in* **FS**. ∎

If we restrict our attention to continuous domains with a countable basis, then we must have Lawson-compactness. This was shown in [Jun89]. So we also have the following continuous analogue to Smyth's Theorem for continuous domains:

Theorem 2.2.16 *The class* ω-$\mathbf{CONT}_\perp$ *of pointed continuous countably based dcpo's contains a largest cartesian closed full subcategory, the class of all countably based FS-domains.* ▮

It is possible to extend the results of this paper to dcpo's without bottom element. Most of the work for this was done in [Jun89] already. One gets four maximal cartesian closed full subcategories of **CONT**.

Chapter 3

The polymorphic lambda calculus

The polymorphic λ-calculus was introduced by Jean-Yves Girard in 1971 [Gir71, Gir72] and independently by John Reynolds in 1974 [Rey74]. The simple idea behind this language is that for certain routines the type of arguments does not play a role, the routine will behave uniformly (the technical expression generally used is *parametric*) for all types. The papers in this chapter do not refer to the syntax of the polymorphic λ-calculus directly, so it is probably admissible to leave out a formal definition in this overview.

As a formal system, the polymorphic λ-calculus exhibits striking features; we just mention a few.

1. It is strongly normalizing and satisfies the Church-Rosser property, hence every term has a unique normal form.

2. By encoding $\mathbb{N}$ into the calculus as so-called Church numerals we can ask what number-theoretic functions are definable. The answer is: all functions that can be proved total in second-order arithmetic.

3. The term algebra for any signature may be represented as the set of closed terms of a certain type.

Building a denotational model for the polymorphic λ-calculus proved to be an extraordinary challenge. While its type system and strong normalization suggest that an interpretation in the category of sets should be possible (similar to the one for the simply typed λ-calculus), it was in fact shown in [Rey84] that under some minimal

conditions on the interpretation such a model does not exist (unless one works in a constructive universe of sets [Pit87]).

Domain-theoretic models were constructed in [Gir86] and [CGW89]. While Girard's model is based on stable functions – a refinement of Scott-continuity – the latter model lives in the category of Scott-domains and continuous functions. At the heart of this construction is the so-called dependent product (explained in Section 3.1 below) but a formidable multitude of details has to be checked in order to be sure of correctness. An advantage of this model is that the usual programming constants can be added to the language (just as we have added them to the simply typed λ-calculus in order to obtain PCF) and the interpretation stays essentially the same. The disadvantage is that the model is rather inflated; so we cannot hope to prove a close connection between denotational and operational semantics and therefore it is questionable whether any nice semantic properties can be found and transferred from the model to the syntax.

Worse, this kind of construction can not be carried through in the preferred category of *bifinite domains*. I made this observation in 1988 and it appeared as a note in *Theoretical Computer Science* in 1991, [Jun91]. It is reprinted in the first section of this chapter.

From these negative results it is justified to look for an alternative formulation of domain-theoretic models for the polymorphic λ-calculus. Such an approach does indeed exist in the form of *retraction models*. The idea (due to Dana Scott [Sco76]) is to reduce types to certain retractions on a universal domain for the class under consideration, where a domain U is *universal* if every other domain (from that class) can be embedded into U. If we restrict to finitary retractions on U (those which have an algebraic image) then it can be shown [McC79, Hut90, Ber91, Rot91] that they together form yet another domain of the same kind, also embeddable into U. By this procedure, types become finitary retractions become points of U, and inverse limits of types become directed suprema. The dependent product, too, can be calculated in U by a simple formula. For details I refer to [Gun92].

It is then natural to ask which classes of domains contain a universal object. For some classes the answer is given in [Sco76, Plo78, Gun87]. After I had found the new class of L-domains [Jun89], Carl Gunter tried to apply his techniques from [Gun87], which had given him a universal domain for the bifinites, to the class of

bifinite L-domains. The task seemed hopelessly complicated at the beginning but I found that with an alternative definition a solution could be given. Furthermore, the adjustment allowed a uniform treatment of all main classes of domains and yielded universal domains with an additional homogeneity property.

We presented our results to the conference *Logic in Computer Science* in 1988 and submitted a full paper to the *Journal of Pure and Applied Algebra*, where it appeared as [GJ89]. The writing in this case was done by Carl Gunter (except for Section 3.2.4). In particular, the category theoretic set-up is due to him.

I may add that this approach to universal domains stimulated further research by Manfred Droste and Rüdiger Göbel [DG90, DG91, Dro92, DG93] in the course of which they unearthed the true significance of our homogeneity property and established the connection with the standard model theoretic concept of *amalgamation*. The presentation in [DG93] could hardly be improved but I may be allowed the comment that the technical content of that paper differs very little from Section 3.2.4 below.

Finally, I quote a very recent result of J. B. Wells who showed that the type reconstruction problem for the polymorphic λ-calculus is undecidable (an earlier result in this direction is [Pfe93]). This makes this language rather unattractive as a practical programming language but I believe that due to its concise and pure form it will remain the object and basis of further semantical studies.

3.1 The Dependent Product Construction in Various Categories of Domains

Introduction

In their recent paper "Domain Theoretic Models of Polymorphism", [CGW89], Coquand, Gunter, and Winskel show how to use the class **S** of all (countably based) Scott domains as a model for the polymorphic lambda calculus. In particular, a type which contains n free type variables is interpreted as a continuous functor $F\colon \mathbf{S}^n \to \mathbf{S}$. The category of continuous sections of the corresponding Grothendieck fibration is shown to be isomorphic to a Scott domain. We show that this construction does not work for bifinite domains or algebraic L-domains.

3.1.1 Notation

All partially ordered sets considered in this note have a least element and allow to form joins of directed subsets. We use the abbreviation *dcpo = directed-complete partial order* in the following. The set of compact elements of a dcpo D is denoted by $\mathsf{K}(D)$. The categories we consider are equipped with embedding-projection pairs as arrows. This is expressed by the superscript ep to the name of these classes. It is well known from the theory of domains that the class $\mathbf{DCPO}^{ep}$ possesses limits for directed systems. These limits can be calculated either as directed colimits in the category $\mathbf{DCPO}^{e}$ or as codirected limits in the category $\mathbf{DCPO}^{p}$. Since the two are isomorphic, we call it the *bilimit* of the system. If D is the bilimit of finite posets then we call it a *bifinite domain*. The class of all bifinite domains we denote by **B**. We say that a functor $F\colon \mathbf{D} \to \mathbf{E}$ is *continuous* if it preserves bilimits.

Definition. Let **C** and **D** be categories of dcpo's and let $F\colon \mathbf{C}^{ep} \to \mathbf{D}^{ep}$ be a continuous functor. A *continuous section* is a class of elements $(t_X)_{X\in\mathbf{C}}$ such that

(i) $\forall X \in \mathbf{C} : t_X \in F(X)$,

(ii) $\forall f^E\colon X \to Y : F(f^E)(t_X) \leq t_Y$,

(iii) $t_{lim X_i} = \bigsqcup^{\uparrow} F(d_i^E)(t_{X_i})$ for directed systems (D_i, d_{ij}) of domains and limiting morphisms d_i.

Of course, we can compare two continuous sections pointwise but we run into foundational problems if we try to form an ordered set from all continuous sections of a functor $F: \mathbf{C}^{ep} \to \mathbf{D}^{ep}$. The problem is that $\mathbf{C}$ may be a class. If $\mathbf{C}$ consists of countably based algebraic dcpo's only, then there are clearly only set-many objects in $\mathbf{C}$ up to isomorphism and any continuous section is determined by its values on some set of representatives.

In the following we will need the concept of *L-domains*. We cite from [Jun89] some basic results:

Definition. A dcpo D is an *L-domain* if every principal ideal $\downarrow x, x \in D$, is a complete lattice.

Theorem 3.1.1 ([Jun89]) *The category* $\mathbf{L}$ *of algebraic L-domains is cartesian closed.*

Theorem 3.1.2 ([Jun89]) *Any cartesian closed full subcategory of the class of algebraic domains is contained in either the class* $\mathbf{L}$ *of algebraic L-domains or in the class* $\mathbf{B}$ *of bifinite domains.*

The proof is based on the following crucial lemma:

Lemma 3.1.3 *Let D and E be algebraic dcpo's with property m such that the space $[D \longrightarrow E]$ of Scott-continuous functions is algebraic. Then either $\mathsf{K}(D)$ has property M or E is an L-domain.*

(Reminder:

property m = for each upper bound x of a set A there is a minimal upper bound of A below x.

property M = property m and each finite set has only finitely many minimal upper bounds.)

We now show that the class of all algebraic L-domains is too big in the sense that there are class-many sections of a functor $F: \mathbf{L}^{ep} \to \mathbf{L}^{ep}$.

Example. For any cardinal κ, let A_κ be the algebraic L-domain consisting of a least element $\perp$, two upper neighbors x, y of $\perp$ and κ-many minimal upper bounds of $\{x, y\}$. Figure 1 shows A_2 and A_3.

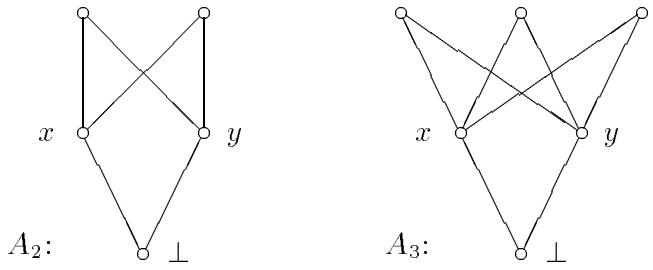


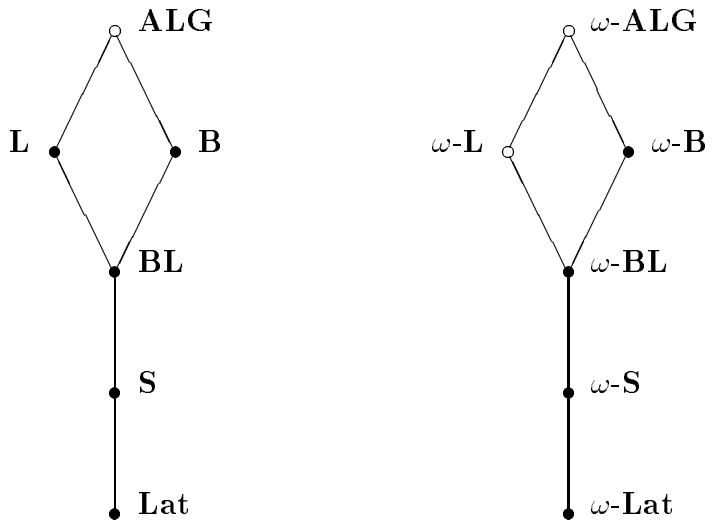
Figure 3.1: Two L-domains.

Note that no A_κ is embedded in any $A_{\kappa'}$ if $\kappa \neq \kappa'$. Now let $F{:}\mathbf{L}^{ep} \to \mathbf{L}^{ep}$ be the functor which maps each object onto A_1 and each arrow onto the identity on A_1. For any $\kappa \in \mathbf{CARD}$, we can define a continuous section $S^\kappa = (t^\kappa_X)_{X \in \mathbf{L}^{ep}}$ where

$$t^\kappa_X = \begin{cases} x, & \text{if there is an embedding from } A_\kappa \text{ to } X; \\ \bot, & \text{otherwise.} \end{cases}$$

These are continuous sections since each A_κ, which is embedded in a bilimit, must be embedded in one of the limiting domains as embeddings preserve minimal upper bounds where they exist. Obviously, there are class-many sections S^κ.

In the following we restrict our attention to subcategories of the category ω-$\mathbf{B}$ of countably based bifinite domains. It is probably helpful to have a picture of the hierarchy of domains here:



The filled dots indicate cartesian closed categories and the denotations are:

ALG: algebraic dcpo's,

L: algebraic L-domains,

B: bifinite domains,

BL: bifinite L-domains,

S: Scott domains,

Lat: algebraic lattices.

Theorem 3.1.4 *The classes* ω-$\mathbf{B}^{ep}$, ω-$\mathbf{BL}^{ep}$, ω-$\mathbf{S}^{ep}$, *and* ω-$\mathbf{Lat}^{ep}$ *are closed under the formation of bilimits and each domain contained in one of these classes is a bilimit of finite posets from the respective class.*

From this theorem we infer that any continuous section defined over one of these categories is determined by its values on the finite objects already. So for any continuous functor we get an ordered set of continuous sections. Following the notation in [CGW89], we denote it by ΠF.

Theorem 3.1.5 ([CGW89]) *If* $F{:}\omega$-$\mathbf{S}^{ep} \to \omega$-$\mathbf{S}^{ep}$ *is a continuous functor then* ΠF *is a Scott domain.*

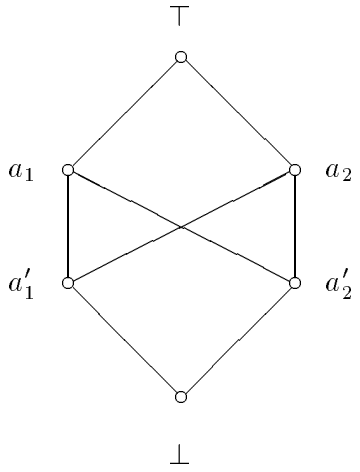


Figure 3.2: NL: the smallest pointed poset which is not an L-domain.

We show that the corresponding theorem for the class ω-$\mathbf{B}^{ep}$ does not hold. **Example.** Let NL be the poset depicted in Figure 2. Consider the constant functor $F{:}\,\omega\text{-}\mathbf{B}^{ep} \to \omega\text{-}\mathbf{B}^{ep}$ which maps each object onto NL, each morphism onto id_{NL}. Note that we can order the finite posets in ω-$\mathbf{B}^{ep}$ by setting $D \leq E$ if there is an embedding from D into E. Call this ordered set S. The definition of ΠF then reduces to $\{(t_X)_{X\in S} \mid t_X \leq t_Y \text{ if } X \leq Y\}$ and we see that ΠF is indeed isomorphic to the space of monotone functions from S into NL. This in turn is isomorphic to the space $[\mathsf{Idl}(S) \longrightarrow NL]$ of Scott-continuous functions from the ideal completion of S into NL. We apply Lemma 3 and find that $\mathsf{K}(\mathsf{Idl}(S)) = S$ should have property M if ΠF is assumed to be algebraic. But this is not the case. The posets A_2 and A_3, see Figure 1, are embedded in each of the posets B_n of which Figure 3 shows B_4 and B_5. The image of an embedding is closed under the formation of minimal upper bounds and this shows that for $n \geq 4$, there are only the obvious embeddings of A_2 and A_3 into B_n. Furthermore, there is no poset strictly between $\{A_2, A_3\}$ and B_n. So S does not have property M and therefore $[\mathsf{Idl}(S) \longrightarrow NL] = \Pi F$ is not algebraic.

Repeating the argument from [Jun90a], this can be seen directly as follows:

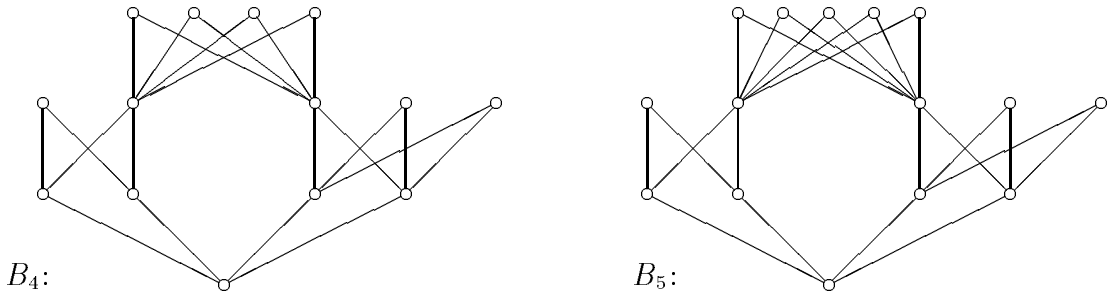


Figure 3.3: L-domains which are upper bounds for $\{A_2, A_3\}$.

Define a section $t = (t_X)_{X \in S}$ of F by

$$t_X = \begin{cases} a_1', & \text{if } A_2 \leq X, A_3 \not\leq X, \\ a_2', & \text{if } A_2 \not\leq X, A_3 \leq X, \\ a_2, & \text{if } A_2, A_3 \leq X, \\ \bot, & \text{otherwise.} \end{cases}$$

This should be a compact element of ΠF if this is an algebraic dcpo. On the other hand, we can define a section t^M for each finite subset M of minimal upper bounds of $\{A_2, A_3\}$ in S as follows:

$$t^M_X = \begin{cases} a_1', & \text{if } A_2 \leq X, A_3 \not\leq X, \\ a_2', & \text{if } A_2 \not\leq X, A_3 \leq X, \\ \top, & \text{if } X \in \uparrow M, \\ a_1, & \text{if } X \in \uparrow A_2 \cap \uparrow A_3 \setminus \uparrow M, \\ \bot, & \text{otherwise.} \end{cases}$$

The supremum of all t^M is clearly above t but no t^M itself is greater or equal to t.

3.1.2 Discussion

To our knowledge, the dependent product is the first domain theoretic construction under which the class of all bifinites is not closed. On the other hand, the convex powerdomain construction is closed only on **B**! Since there is a universal domain for ω-**B**, see [Gun87], there is at least some way to model polymorphism with bifinite do-

mains, following the ideas of [ABL86]. This dichotomy should help us to understand better the internal structure of the two models.

One wonders whether the class ω-**BL** of bifinite L-domains is closed under forming dependent products. We conjecture that this is not the case. The method of this note — taking constant functors — will not decide the problem.

More generally, we would like to ask the following question: Is there a largest cartesian closed category which allows to form dependent products?

3.2 Coherence and Consistency in Domains

Introduction.

The first structures used as a mathematical foundation for the denotational semantics of programming languages were lattices. With lattices it was possible to solve the necessary recursive equations and an elegant mathematical theory could be developed using the familiar category of (countably based) algebraic lattices [Sco76] (although it was necessary to take some care to choose the right notion of morphism). As experience with denotational semantics grew, deeper computational intuitions were developed and new categories were introduced in attempts to match these intuitions to the mathematical constructs. For example, it was desirable to have a class of domains which included such structures as the partial functions from natural numbers to natural numbers which—under their usual ordering—do not form a lattice. Such theories were proposed by Plotkin [Plo78], Berry [Ber78] and also Scott [Sco81b, Sco82a, Sco82b].

The category which Scott proposed was very similar to the algebraic lattices: a dcpo D is said to be a *Scott domain* (or bounded complete domain) if the dcpo $D^\top$ obtained by adding a top to D is an algebraic lattice (with a countable basis). The arrows of the category are continuous functions, *i.e.* monotone functions which preserve joins of directed collections of elements. The category of Scott domains is easy to work with and has an intuitive logical character which has been the subject of several investigations (see, in particular, [Sco82a, Abr91]). One central feature of these treatments is the concept of *consistency* of data. One may think of a Scott domain as a collection of propositions or data elements under an ordering of *partial information.* An element x is ordered below an element y in a domain D if x is "more partial" than y. The element x is a kind of partial description of y. Now, given two data elements x_1 and x_2, there may or may not be a third element y which they describe. If there is such a y, then x_1 and x_2 are said to be *consistent,* otherwise they are *inconsistent.* A crucial feature of a Scott domain is the following fact: *if two elements of a Scott domain D are consistent, then they have a join in D.* This property is commonly referred to as *consistent completeness.*

The use of consistent complete domains for modeling the semantics of types in programming languages has become the general practice. However, we would like

to note in this paper that *it is not the only reasonable direction the theory could have taken* at the point that consistency was recognized as a central concept. Up until the time we are writing this paper, almost all of the categories of domains that have been proposed as a possible foundation for the semantics of programming languages have been (essentially equivalent to) dcpo's which satisfy the consistent completeness condition. This includes those categories which use stable continuous functions [Ber78, Gir86] as well as categories related to the Scott domains (such as the continuous lattices).[1] The one noteworthy exception is the category of ω-bifinite domains which was introduced by Plotkin [Plo76] (where it is called **SFP**). These will be discussed below.

One might apply the following line of reasoning in an attempt to deal with the concept of consistency of data. A domain is a collection of propositions providing partial descriptions of elements (which may also be propositions describing further elements); a given element dominates a collection of data elements which provide partial descriptions of it. We propose the following condition on the structure of the partial descriptions of an element: *the partial descriptions of an element must form an algebraic lattice.* Let us refer to this condition as *local algebraicity.* But a locally algebraic dcpo (with a countable basis) is just a Scott domain right? No, not at all! Aside from the fact that such a domain need not have a least element (an infinite discrete domain is locally algebraic for example) it is even possible that a consistent pair of elements have no join! (See Figure 3.4.) One can show, however, that almost all of the essential features needed to provide semantics for programming languages are satisfied by locally algebraic domains.

The concept of a locally algebraic domain was formulated by the second author who came across the concept in the course of his investigations into extensions of Smyth's Theorem [Jun90a, Jun88]. We refer to locally algebraic domains as *L-domains* to keep the terminology short. They were independently discovered by Thierry Coquand as a special instance of his categories of embeddings [Coq89]. We will discuss some basic properties of L-domains in the next section—for a more detailed discussion, the reader can examine [Coq89, Jun90a, Jun88]. The bulk of the paper will focus on the properties of a subcategory of the L-domains which were

[1] We omit from discussion categories of dcpo's with no assumptions about the existence of a basis.

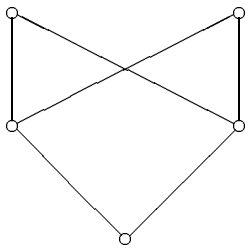


Figure 3.4: A locally algebraic domain which is not consistent complete.

introduced in the first author's doctoral dissertation [Gun85]. The category which was investigated there (the objects were called *short* domains) consisted of those L-domains which were ω-bifinite. It was observed at that time that such domains formed a cartesian closed category in which one could solve recursive domain equations. However, we would like to demonstrate a further fact about them below. Namely, that *there is a "universal" domain in this category.*

Our construction is similar to that which appears in [Gun87] for the ω-bifinite domains, but a more subtle ordering is needed to make things work properly. We prove a lemma expressed in categorical terms which aids one in demonstrating the existence of a universal domain by demonstrating the existence of what we call a *finite relative saturation.* This lemma is sufficiently general that it applies not only to our construction of a universal ω-bifinite L-domain and the construction of a universal ω-bifinite domain as in [Gun87], but also to consistent complete domains and even countably based algebraic lattices! The universal domains so constructed are characterized by a property very similar to what model theories call *countable saturation* [CK73]. We prove that a model with this property is unique up to isomorphism. We can apply this result to show that Scott's universal domain for the consistent completes [Sco81b, Sco82a, Sco82b] is *not* saturated.

The paper is divided into six sections which we overview briefly. Section two provides some definitions and establishes notation. A few basic propositions are also remarked. The third section discusses the coherence condition on the topology of a domain. We show how this condition translates into an order-theoretic one and

discuss some important properties of domains with coherent topologies. The fourth section discusses the universal domain construction. Since this construction seems to have a general significance, we have attempted to provide a categorical treatment of it. This categorical treatment makes it possible to see the construction in this paper and the one that was presented in [Gun87] as instances of a more general theory which may have applications in other cases. In the fifth section we instantiate the general theory for the classes ω-**Lat** of algebraic lattices, ω-**S** of Scott domains, ω-**BL** of ω-bifinite L-domains and ω-**B** of ω-bifinite domains. The universal domains which we thus construct are saturated. We prove in Section 6 that any saturated object in a subclass of ω-$\mathbf{B}^{ep}$ is universal and that there is at most one such object (up to isomorphism).

3.2.1 Basic definitions and facts.

For the purposes of this paper a dcpo (complete poset) is a poset $(D, \sqsubseteq)$ with least element and with joins $\bigsqcup M$ for all directed subsets M. A function $f\colon D \to E$ between dcpo's D and E is *continuous* if it is monotone and preserves joins of directed subsets of D. An element x of a dcpo D is said to be *compact* if, whenever M is a directed subset of D and $x \sqsubseteq \bigsqcup M$, then there is a $y \in M$ such that $x \sqsubseteq y$. Let $\mathsf{K}(D)$ be the collection of compact elements of a dcpo D. A dcpo D is said to be *algebraic* if every element of D is the join of a directed collection of compact elements. D is said to be ω-algebraic if it is algebraic and $\mathsf{K}(D)$ is countable. An *algebraic lattice* is an algebraic dcpo which is a lattice.

Definition 3.2.1 *A dcpo D is* locally algebraic *if, for every $x \in D$, the principal ideal*

$$\downarrow x = \{y \in D \mid y \sqsubseteq x\}$$

generated by x is an algebraic lattice.

Proposition 3.2.2 *If D is locally algebraic, then it is algebraic.*

Proof. Suppose c is a compact element in $\downarrow x$ and $(e_i)_{i\in I}$ is a directed collection of elements with supremum e above c. The principal ideal $\downarrow e$ is by assumption an algebraic dcpo, so in particular the element c is the supremum of a directed collection

$(c_j)_{j\in J}$ of compact elements in the $\downarrow e$-sense. All these elements belong to $\downarrow x$ as well and since c is compact there, one of the elements c_j must be equal to c. Going back to $\downarrow e$ we learn that c is equal to a compact element in this ideal, so some e_i must be above c. This proves that any locally compact element is also globally compact and hence D is algebraic. ∎

To keep the terminology short, we will refer to locally algebraic dcpo's as *L-domains.* The category of L-domains properly contains the class of Scott-domains: Figure 3.4 shows an example. The difference between the two concepts is illustrated by the following characterizations:

Proposition 3.2.3 *Let D be an algebraic dcpo.*

- *D is a Scott-domain, if and only if every nonempty subset has a meet in D.*
- *D is an L-domain, if and only if every bounded nonempty subset has a meet in D.*

(For a proof see [Jun90a].)

The difference may seem a slight one but it has some important consequences. The basis of the function space of a Scott-domain D has always the same cardinality as $\mathsf{K}(D)$, whereas the cardinality may increase if D is an L-domain. However, the following (which was found independently by Thierry Coquand) remains true:

Theorem 3.2.4 *The category of L-domains and continuous functions is cartesian closed.*

In [Jun90a] it is proved that, in the category of algebraic dcpo's with least element, there are exactly two maximal cartesian closed subcategories: the category of L-domains and the category of bifinite domains, which we now proceed to define.

A continuous function $f^L\colon D \to E$ between dcpo's D and E is said to be an *embedding* if there is a continuous function $f^R\colon E \to D$ such that $f^R \circ f^L = id_D$ and $f^L \circ f^R \sqsubseteq id_E$ where id_D and id_E are the identity functions on D and E respectively. If there is such a function f^R, then it is uniquely determined by f^L and is said to be the *projection* corresponding to f^L. Pairs $f = \langle f^L, f^R\rangle\colon D \to E$, where f^L is

an embedding and f^R the corresponding projection, form the arrows of a category $\mathbf{DCPO}^{ep}$ which has dcpo's as its objects. Composition is given by

$$\langle f^L, f^R \rangle \circ \langle g^L, g^R \rangle = \langle f^L \circ g^L, g^R \circ f^R \rangle.$$

It is a basic fact in the theory of domains that $\mathbf{DCPO}^{ep}$ has directed colimits, which we call *bilimits* since they can be gotten either from the directed system of embeddings or from the codirected system of projections.

Theorem 3.2.5 *The category of L-domains and embedding-projection pairs has bilimits.*

If a dcpo is a bilimit in $\mathbf{DCPO}^{ep}$ of a family of finite posets with least element, then it is said to be a *bifinite domain.* It is possible to show that bifinite domains must be algebraic. Let $\mathbf{B}$ and $\mathbf{B}^{ep}$ be the categories of bifinite domains with continuous functions and embedding-projection pairs respectively. It is possible to show that $\mathbf{B}$ is a cartesian closed category and $\mathbf{B}^{ep}$ has bilimits of directed families [Gun85, Gun87]. Bifinite domains with a countable basis and least element are the "**SFP**-objects" of Plotkin [Plo76]. We will follow Smyth's terminology [Smy83] and refer to them as ω-*bifinite* domains. We write ω-$\mathbf{B}$ for the category with continuous functions and ω-$\mathbf{B}^{ep}$ for the category with embedding-projection pairs. It is not hard to see that ω-$\mathbf{B}$ is a cartesian closed category and ω-$\mathbf{B}^{ep}$ has bilimits for countable directed families.

3.2.2 Coherence.

In order to get a satisfactory class of spaces as domains for denotational semantics it is desirable to impose a more restrictive condition than local algebraicity. Suppose one wished to define a notion of *computability* on L-domains. It might be possible to do this for the L-domains with a countable basis. So why not restrict oneself to these? The problem is that the L-domains with countable basis are not closed under the exponential! Consider the poset K pictured in Figure 3.5. This is an L-domain with a countable basis but $[K \longrightarrow K]$ has a basis with continuum many members.

Since M. Smyth [Smy83] has proved that any domain which has an ω-algebraic function space is in fact bifinite, it is reasonable to investigate the category ω-$\mathbf{BL}$ of bifinite L-domains which have countable bases and least elements, *i.e.* the ω-*bifinite*

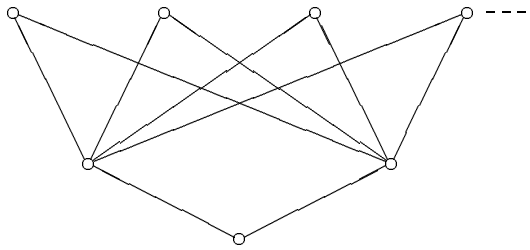


Figure 3.5: K has a countable basis, but $[K \longrightarrow K]$ does not.

L-domains. The poset in Figure 3.5 is a typical example of an L-domain that fails to be bifinite.

An unfortunate drawback to the bifiniteness condition is the fact that it is not very easy to understand. Although intrinsic descriptions are possible and these do help in reasoning about bifinite domains, it would still be nice to work with a simpler class of structures. However, it turns out that the ω-bifinite domains which are L-domains may be somewhat more easily characterized than ω-bifinite domains in general. In particular, they may be identified as those L-domains which have a "nice" Scott topology.

We will follow the definitions and notation in Johnstone [Joh82]. A dcpo D can be given a topology as follows. The open subsets of the topology are those which satisfy:

1. whenever $x \in U$ and $x \sqsubseteq y$, then $y \in U$, and
2. whenever $M \subseteq D$ is directed and $\bigsqcup M \in U$, then $M \wedge U \neq \emptyset$.

This is usually called the *Scott topology* on D and it will be denoted ΣD. It is possible to show that a function $f: D \to E$ between dcpo's D and E is continuous in the sense that $f(\bigsqcup M) = \bigsqcup f(M)$, for any directed $M \subseteq D$, if and only if it is continuous in the usual topological sense—with respect to the Scott topology.

Definition 3.2.6 *Let D be an algebraic dcpo. The topology ΣD is said to be* coherent *if the quasicompact open subsets of D are closed under finite intersections.*

We would like to make two brief remarks about this terminology. First, to keep things simple, we have restricted the definition to algebraic dcpo's; the definition above would not correspond to the usual notion of a coherent topology if D were allowed to be an arbitrary dcpo. Second, we would like to comment that the meaning for the term "coherent" which we have given should not be confused with other meanings from the domain theory literature. In particular, a poset is sometimes said to be coherent if any pairwise consistent set has a least upper bound. This condition is *stronger* than consistent completeness and certainly does not correspond to the condition we are using here!

Coherence is an elegant condition on the topology of a domain D which has an important significance for the order structure of D. Let us say that a poset P has the *strong minimal upper bounds property* (or *property M* for short) if, for every finite subset $A \subseteq P$, the set $\mathsf{mub}(A)$ of minimal upper bounds of A satisfies the following properties:

1. $\mathsf{mub}(A)$ has only finitely many elements and
2. $\mathsf{mub}(A)$ is complete in the sense that for every $p \in P$, if $x \sqsubseteq p$ for every $x \in A$, then $y \sqsubseteq p$ for some $y \in \mathsf{mub}(A)$.

We have the following:

Proposition 3.2.7 *Let D be an algebraic dcpo. Then ΣD is coherent if and only if the basis $\mathsf{K}(D)$ of D has property M.*

Proof. Since the sets of the form $\uparrow c$, with c a compact element of D, form a basis of the Scott topology, a set A is quasicompact open if and only if it is a finite union of such principal filters.

So let A and A' be upper sets generated by finite sets $M, M' \subseteq \mathsf{K}(D)$, respectively. Each element of $A \wedge A'$ is above some element of M and above some element of M'. So $A \wedge A'$ is generated by the finite set $\bigcap_{m \in M, m' \in M'} \mathsf{mub}(m, m')$ and hence itself quasicompact.

For the converse let $m \subseteq \mathsf{K}(D)$ be a finite set. Each set $\uparrow m$, $m \in M$ is quasicompact open and, by coherence, so is $\bigcap_{m \in M} \uparrow m$. The latter set is therefore covered by finitely many principal open filters and hence generated by finitely many compact elements. This proves that $\mathsf{K}(D)$ has property M. ∎

The central theorem of this section states that a bifinite L-domain may be characterized using the coherence condition:

Theorem 3.2.8 *Let D be an L-domain. Then ΣD is coherent if and only if D is bifinite.*

Proof. It is well known (see [Plo76], for example) that the basis of a bifinite domain has property M, so by the previous proposition the 'only if'-part is taken care of.

For the converse we know that D is an L-domain and that $\mathsf{K}(D)$ has property M. Given any finite set A of compact elements and any element x of D there is a supremum of the set $\downarrow x \wedge A$ in the principal ideal generated by x. Mapping each element onto this supremum is a continuous function, since A consists of compact elements only and suprema of compact elements are again compact in a lattice. The image of this function is finite by property M. This shows that D is isomorphic to a bilimit of finite posets. (A more detailed account of this well known fact can be found in any of the following sources [Plo76, Gun85, Jun88]. ∎

Since the bifinite L-domains lie at the intersection of two "nice" categories, they inherit some of that niceness themselves:

Proposition 3.2.9 *The category of bifinite L-domains and continuous functions is a cartesian closed category.*

Proposition 3.2.10 *The category of bifinite L-domains and embedding-projection pairs has bilimits for directed collections.*

3.2.3 Building universal domains.

The concept of a "universal domain" dates back at least to Scott's paper [Sco76] on $P\omega$ and is widely used in the current literature. The term "universal domain" is somewhat vaguely defined, however. We see basically two uses as being the most common. The easiest of these to understand is what one might call a "poor man's universal domain". Typically it is a domain which satisfies an isomorphism

$$V \cong (V \to V) + F_1(V) + \cdots + F_n(V) \tag{3.1}$$

where $F_1, \ldots, F_n$ are operators over which domain equations must be solved. One often sees such universal domains being used in the type theory literature [MPS86, Car84]. The theory of domains provides us with all of the mathematical tools generally needed for solving equations like (3.1) so that we may employ such definitions quite freely and confidently. On the other hand, the poor man's universal domain depends on the choice of the functors F_i and it would be nice to know more facts about the order structure of the solution than the existence result for the solution tells us. It is therefore appealing to have a *single* universal domain $\mathcal{U}$ which has *all* domains of interest as retracts. Of course, this is subject to one's interpretation of "domains of interest", but it is not dependent on a commitment to some finite list of functors. We refer the reader to Taylor [Tay87b] for a full discussion of universal domains (which he calls "saturated domains"). For the purpose of clarity, let us propose a definition of "universal domain" which will give the reader some idea what we are after.

Definition 3.2.11 *Let* $\mathbf{C}$ *be a category. An object* $\mathcal{U}$ *is* universal *in* $\mathbf{C}$ *if it is weakly terminal,* i.e. *for every object* A *of* $\mathbf{C}$*, there is a (not necessarily unique) arrow* $f: A \to \mathcal{U}$*.*

The term "universal domain" probably comes from the model theoretic notion of a "universal model" which has a similar definition [CK73]. Universal models can be built using the concept of *saturation* first presented in [Vau61] and it will be our goal below to convert this model-theoretic technique to domain-theoretic ends. Of course, any category that has a terminal object has a universal domain. However, one typically has it in mind that the arrows of the category $\mathbf{C}$ are monics. In particular, we show that the category ω-$\mathbf{BL}^{ep}$ of ω-bifinite L-domains with embedding-projection pairs has a universal domain.

The proof uses techniques from Gunter [Gun87]. However, naively mimicking the construction which appears there will not work. We therefore begin by devising a general theory which can be applied to obtain a universal domain for both ω-$\mathbf{B}^{ep}$ (as described in [Gun87]) and ω-$\mathbf{BL}^{ep}$. We also derive universal domains for ω-$\mathbf{S}^{ep}$ (the category of Scott domains) and ω-$\mathbf{Lat}^{ep}$ (the category of algebraic lattices), which differ from the ones given by Scott in [Sco76, Sco81b].

In particular, we provide a categorical treatment of the essential ingredients that make the universal domain construction work. The construction is reminiscent of

one from general model theory. For example, fix a first order theory T in a countable language and suppose that T has a countable homogeneous model A. One can show that A is elementarily embedded in a countable model of T as follows. It is easy to see that A is elementarily embedded in a countable model A_1 which is homogeneous with respect to finite sequences taken from A. One can use a similar construction to build a sequence of models A_i such that, for each $j < i$, the model A_i is homogeneous with respect to finite sequences of elements from A_j and A_j is elementarily embedded in A_i. The colimit of this chain will be the desired homogeneous extension of A. The reader can find many constructions that use this basic idea in a standard book on model theory such as [CK73].

We begin with the following concept:

Definition 3.2.12 *An arrow $f: A \to B$ is an* increment *if, whenever $f = h \circ g$, then either h or g is an isomorphism.*

Perhaps the simplest example of an increment is the inclusion map $f: S \to T$ between finite sets S and T, such that $S = T \cup \{x\}$ for some x. If $\mathbf{C}$ is a poset (considered as a category), then an arrow $x \sqsubseteq y$ is an increment if and only if there is no element of $\mathbf{C}$ between x and y. If we consider the category of L-domains with embedding-projection pairs, then an arrow $s: A \to A'$ from a finite L-domain A into an L-domain A' is an increment if and only if A' has at most one more point than A. Figure 3.6 indicates a typical increment in this category. The increment embeds a four element poset into a poset with five elements; the closed circle indicates the "new" element.

An *ω-chain* in a category $\mathbf{C}$ is a functor $F: \omega \to \mathbf{C}$ from the ordinal ω (considered as a category) into $\mathbf{C}$. In essence, an ω-chain is a sequence of objects A_i where $i < \omega$ and a collection of arrows $a_{ji}: A_i \to A_j$ where $i \leq j < \omega$. For each i, the arrow a_{ii} is the identity on A_i and, for any $i \leq j \leq k$, one has $a_{kj} \circ a_{ji} = a_{ki}$.

Definition 3.2.13 *A concrete category $\mathbf{C}$ is* incremental *if*

1. *$\mathbf{C}$ has an initial object,*
2. *$\mathbf{C}$ has bilimits of ω-chains,*
3. *every object A of $\mathbf{C}$ is a bilimit of an ω-chain (A_i, a_{ij}) where A_0 is initial, each A_i is finite (in the category $\mathbf{C}$) and each arrow $a_{i+1,i}: A_i \to A_{i+1}$ is an increment.*

Figure 3.6: A typical increment in ω-$\mathbf{BL}^{ep}$. The poset on the left is embedded in the poset on the right. The open circles show the image of the embedding.

For example, the category of countable sets and injections is incremental. However, we are interested in a more subtle example:

Theorem 3.2.14 *The category ω-$\mathbf{B}^{ep}$ of ω-bifinite domains and embedding-projection pairs is incremental.*

Proof. This is Theorem 22 (the Enumeration Theorem) of [Gun87].

Corollary 3.2.15 *The categories ω-$\mathbf{BL}^{ep}$, ω-$\mathbf{S}^{ep}$, and ω-$\mathbf{Lat}^{ep}$ are incremental.*

Proof. Let D be a ω-bifinite L-domain (Scott domain, Lattice) and let (D_i, d_{ij}) be an ω-chain of increments with bilimit D in ω-$\mathbf{B}^{ep}$. By definition, each D_i is embedded in D and must therefore itself be a ω-bifinite L-domain (Scott domain, Lattice). ∎

Let $\mathbf{C}$ be an incremental category and let A be an object of $\mathbf{C}$. An object A^+ and arrow $s: A \to A^+$ is a *relative saturation* of A (or just a *saturation* for short) if, for every increment $f: B \to B'$ and arrow $g: B \to A$, there is an arrow h which makes the following diagram commute:

$$\begin{array}{ccc} B & \xrightarrow{f} & B' \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ A & \xrightarrow[s]{} & A^+ \end{array}$$

Let us say that an incremental category $\mathbf{C}$ *has finite saturations* if, for every finite object A of $\mathbf{C}$, there is a saturation $s: A \to A^+$ where A^+ is finite.

Theorem 3.2.16 *If an incremental category has finite saturations, then it has a universal object.*

Proof. Suppose $\mathbf{C}$ is an incremental category with finite saturations. Let S_0 be any initial object of $\mathbf{C}$. Build the chain $S_0, S_1 = S_0^+, ..., S_{i+1} = S_i^+, ...$ where $s_{i+1,i}$ is a saturation for each i. Let $\mathcal{U}$ be a bilimit for this chain. We claim that $\mathcal{U}$ is universal. To see this, suppose A is any object of $\mathbf{C}$ and we will demonstrate an arrow $f: A \to \mathcal{U}$. Since $\mathbf{C}$ is incremental, A is the bilimit of a chain (A_i, a_{ij}) of finite objects where A_0 is initial and each arrow $a_{i+1,i}: A_i \to A_{i+1}$ is an increment. Now, there is an arrow $f_0: A_0 \to S_0$ since A_0 is initial. Suppose an arrow $f_i: A_i \to S_i$ is given. Since $a_{i+1,i}$ is an increment and $s_{i+1,i}$ is a saturation, there is an arrow f_{i+1} such that the following diagram commutes:

$$\begin{array}{ccc} A_i & \xrightarrow{a_{i+1,i}} & A_{i+1} \\ {\scriptstyle f_i}\downarrow & & \downarrow{\scriptstyle f_{i+1}} \\ S_i & \xrightarrow{s_{i+1,i}} & S_{i+1} \end{array}$$

This collection of arrows f_i gives rise to a cocone with vertex $\mathcal{U}$ over the chain (A_i, a_{ij}) whose vertex is $\mathcal{U}$. Since A is a bilimit of this chain, there must consequently be a mediating arrow $f: A \to \mathcal{U}$ as desired. ∎

Thus, to prove that there is a universal object in the category of ω-bifinite domains (as was done in [Gun87]) or that of ω-bifinite L-domains, it suffices to demonstrate that the category in question has finite saturations. The fact that ω-$\mathbf{B}^{ep}$ has finite saturations is proved in [Gun87]. We show how to derive this result for ω-$\mathbf{B}^{ep}$, ω-$\mathbf{BL}^{ep}$, ω-$\mathbf{S}^{ep}$, and ω-$\mathbf{Lat}^{ep}$ in the next section. By Theorem 3.2.16 this will prove:

Theorem 3.2.17 *The following categories have universal domains:*

1. ω-$\mathbf{B}^{ep}$
2. ω-$\mathbf{BL}^{ep}$
3. ω-$\mathbf{S}^{ep}$
4. ω-$\mathbf{Lat}^{ep}$

3.2.4 Constructing saturations.

In this section we will construct finite saturations for ω-$\mathbf{B}^{ep}$, ω-$\mathbf{BL}^{ep}$, ω-$\mathbf{S}^{ep}$, and ω-$\mathbf{Lat}^{ep}$. We saw earlier that if D is a finite poset then an increment $f\colon D \to D'$ adds at most one point x_f to D. The idea for constructing a saturation D^+ is to take all points which may be added by an increment.

Since each increment $f\colon D \to D'$ corresponds to a unique projection $g\colon D' \to D$, there is some element $u_f \in D$ onto which x_f is mapped by g. In fact, $f(u_f)$ is the largest element of $f(D)$ below x_f. Similarly, the set $\uparrow x_f \wedge f(D)$ corresponds to an upper set U_f in D. This suggests the following definition for a finite poset $D \in \omega$-$\mathbf{B}^{ep}$:

$$D^+ = \{(u, U) \mid u \in D, u \sqsubseteq U = \uparrow U \subseteq D\},$$

with the intended meaning that (u, U) stands for a new element x_f just above $u = u_f$ and below all elements of $U = U_f$. Obviously there cannot be any new element between u and $\uparrow u$, so the pairs $(d, \uparrow d), d \in D$ represent D inside D^+.

We have to be a little bit more careful in defining D^+ for L-domains, however. Recall that D is an L-domain if and only if each bounded nonempty subset of D has a global meet. A new element added by an increment must not destroy this property. This implies that if x_f is a new element added to D by an increment in ω-$\mathbf{BL}^{ep}$ and if $d, d' \sqsubseteq b$ are contained in U_f then x_f is a lower bound for $\{d, d'\}$ and must consequently be below or directly above $d \wedge d'$. This says that $d \wedge d'$ must belong to U_f or it must be equal to u_f. We add this property to the definition of D^+ for finite L-domains D:

$$\begin{aligned} D \in \omega\text{-}\mathbf{BL}^{ep} : D^+ \quad = \quad & \{(u, U) \mid D \ni u \sqsubseteq U = \uparrow U \subseteq D \\ & \text{and } \{u\} \cup U \text{ is closed under bounded} \\ & \text{nonempty meets.}\}. \end{aligned}$$

Similarly for the two remaining categories:

$$\begin{aligned} D \in \omega\text{-}\mathbf{S}^{ep} : D^+ \quad = \quad & \{(u, U) \mid D \ni u \sqsubseteq U = \uparrow U \subseteq D \\ & \text{and } \{u\} \cup U \text{ is closed under nonempty meets.}\}, \\ D \in \omega\text{-}\mathbf{Lat}^{ep} : D^+ \quad = \quad & \{(u, U) \mid D \ni u \sqsubseteq U = \uparrow U \subseteq D \\ & \text{and } \{u\} \cup U \text{ is closed under meets.}\}. \end{aligned}$$

The order on D^+ is defined uniformly by

$$(u, U) \leq (v, V) \Leftrightarrow v \in U \text{ or } (v = u \text{ and } V \subseteq U).$$

Note that $(u, U) \leq (v, V)$ implies $u \sqsubseteq v$ and $V \subseteq U$, so $\leq$ is indeed a partial order on D^+. It is also helpful to recognize that for a given $u \in D$ the set of all $U \subseteq D$ such that $(u, U) \in D^+$, is a lattice. This follows from the observation that $(u, \uparrow u) \in D^+$ and that if $(u, U_1), (u, U_2) \in D^+$ then $(u, U_1 \wedge U_2) \in D^+$. We denote the smallest set U which contains a set $X \subseteq D$ and for which (u, U) belongs to D^+ by $\langle X \rangle_u$.

Lemma 3.2.18 *If D is a finite L-domain (bounded-complete domain, lattice) then so is D^+.*

Proof. We have to show that D^+ has infima for bounded sets. So let $(u, U), (u', U') \leq (v, V)$ be three elements in D^+. Since $\{u, u'\}$ is bounded by v, the infimum $u \wedge u'$ exists in D. The corresponding upper set U'' must at least contain U and U' but depending on whether $u \wedge u'$ is contained in $\{u, u'\}$ or not it may be necessary to include also u and/or u'. We can express this as follows: $U'' = \langle U \cup U' \cup (\{u, u'\} \setminus \{u \wedge u'\}) \rangle_{u \wedge u'}$. If (w, W) is any other lower bound then either $w = u \wedge u'$ or $w < u \wedge u'$. In the first case W must contain U'' as we took U'' as small as possible. In the second case, W must contain u and u' and hence also $u \wedge u'$.

The proof for Scott-domains is the same with the single difference that $\{(u, U), (u', U')\}$ is not necessarily bounded. In order to show that D^+ is a lattice if D belongs to ω-$\mathbf{Lat}^{ep}$ it suffices to note that $(\top, \phi)$ is the largest element of D^+. ∎

Lemma 3.2.19 *If D is a finite poset (L-domain, Scott-domain, lattice) then D^+ is a saturation for D in the respective category.*

Proof. We indicated above that D is embedded in D^+ via the mapping $d \mapsto (d, \uparrow d)$. The corresponding projection is given by $(u, U) \mapsto u$.

Let $f: D \to D'$ be an increment and let $u_f \in D$ and $U_f \subseteq D$ be defined as above. In the definition of D^+ we already argued that (u_f, U_f) belongs to D^+ in all four cases. It therefore remains to show that D' is embedded in D^+. We identify D' with the subset $\{(d, \uparrow d) \mid d \in D\} \cup \{(u_f, U_f)\}$ of D^+. For each $(u, U) \in D^+$ there is a

largest element of D' below it: if $(u_f, U_f) \leq (u, U)$ then either $u = u_f$, in which case (u_f, U_f) is the largest element of $\downarrow(u, U) \wedge D'$, or u is contained in U_f. In the latter case we have that $(u_f, U_f) \leq (u, \uparrow u) \leq (u, U)$ and $(u, \uparrow u)$ is the largest element of D' below (u, U). Hence there is a projection from D^+ onto D'. ∎

An illustration of the four different constructions can be found in Figure 3.7 at the end of the paper. The reader is challenged to check that the figure labelled A^+ in ω-$\mathbf{B}^{ep}$ is, in fact, not an L-domain whereas the figure labelled A^+ in ω-$\mathbf{BL}^{ep}$ is one. Similarly, the figure labelled B^+ in ω-$\mathbf{BL}^{ep}$ is not a Scott domain although the figure to its right is a Scott domain. The third trio of examples is a similar illustration for algebraic lattices.

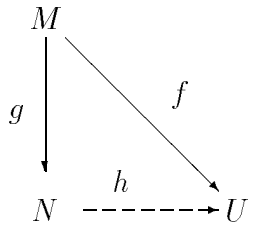
3.2.5 Saturated domains.

We hope that the reader can now appreciate how Theorem 3.2.16 can be used to demonstrate the existence of a universal object. In the proof of that theorem, there is a construction of a universal domain using the saturations that exist in the category. Since a given finite object may have many non-isomorphic saturations, it is possible that the construction used there may give different universal domains if one uses different saturations. In this section we demonstrate that this is not the case in a category of ω-bifinite domains: *regardless of the choice of saturations, the construction in Theorem 3.2.16 is unique up to isomorphisms.* In particular, we will define the notion of a *saturated* domain by analogy with the concept of a saturated model of a first order theory [CK73]. We then show, as one shows the corresponding model-theoretic result, that there is a unique saturated domain up to isomorphism. It is then shown that the universal domain constructed in Theorem 3.2.16 is, in fact, saturated. This shows that there is a "canonical" choice of universal domain for many of the categories of domains used in denotational semantics [GS90]. It is remarked that the bounded complete universal domain of Scott [Sco81b, Sco82a, Sco82b] is *not* saturated and is therefore not isomorphic to the universal bounded complete domain constructed in the previous section.

As an abbreviation, let us refer to an incremental full sub-category $\mathbf{C} \subseteq \omega$-$\mathbf{B}^{ep}$ as a *category of domains* if it is closed under embeddings: *i.e.* if $E \in \mathbf{C}$, $D \in \omega$-$\mathbf{B}$ and there is an embedding-projection pair $f : D \to E$, then D is in $\mathbf{C}$. The key concept

of this section is given in the following:

Definition 3.2.20 *Let* $\mathbf{C}$ *be a category of domains. An object* $U \in \mathbf{C}$ *is* fully saturated in $\mathbf{C}$ *(or* saturated*, for short) if, for every pair of finite domains* M, N *and embedding-projection pairs,* $f: M \to U$ *and* $g: M \to N$*, there is a (not necessarily unique) embedding-projection pair* h *which completes the following diagram:*



Theorem 3.2.21 *Let* U *be a fully saturated object in a category* $\mathbf{C}$ *of domains. Then* U *is universal for* $\mathbf{C}$*.*

Proof. Each ω-bifinite domain D is the bilimit of an ω-chain (A_i, a_{ij}) of finite posets. We may assume that $A_0 = \{\bot\}$. Clearly, A_0 is embedded in $U \in \mathbf{C}$ and so by definition $A_1, A_2, \ldots$ are embedded in U. This cocone over U gives rise to an embedding $g: D \to U$. ∎

To prove the desired results about saturated domains, it is useful to introduce a few notations and facts which are useful in dealing with categories of domains. If $f: D \to E$ is an embedding-projection pair and f^L is an inclusion map then we write $D \lhd E$. The following lemma is easy to prove and will be used implicitly in the proof of the theorem below:

Lemma 3.2.22 *1. If* M *is a finite poset such that* $M \lhd U$*, then* $M \subseteq \mathsf{K}(U)$*.*

2. If D *is* ω*-bifinite and* $S \subseteq \mathsf{K}(D)$ *is finite, then there is a finite* $N \lhd D$ *such that* $S \subseteq N$*.*

3. If $M \lhd D$*,* $N \lhd D$ *and* $M \subseteq N$*, then* $M \lhd N$*.*

Lemma 3.2.23 *Let* U *be an object in a category of domains. If* U *is saturated, then for every finite* $M \lhd U$ *and embedding-projection pair* $f: M \to N$ *into a finite poset* N*, there is a poset* $N' \lhd U$ *such that* $N \cong N'$*.*

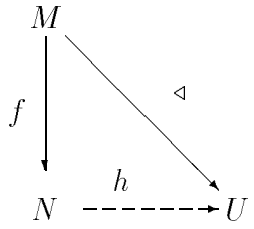
Proof. Let N' be the image under the embedding h whose existence is guaranteed by definition. ∎

Theorem 3.2.24 *If a category of domains has a saturated object, then it is unique up to isomorphism.*

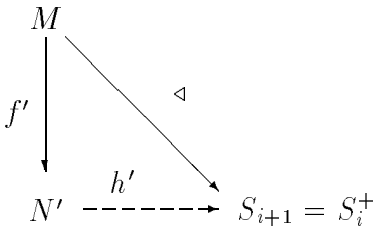
Proof. Let $\mathbf{C}$ be a category of domains and suppose that U and V are saturated objects of $\mathbf{C}$. Let $u_0, u_1, \ldots$ and $v_0, v_1, \ldots$ be enumerations of the bases of U and V respectively. Assume that $u_0 = \bot_U$ and $v_0 = \bot_V$. We construct an isomorphism between $\mathsf{K}(U)$ and $\mathsf{K}(V)$ by a "back and forth" construction. The first partial isomorphism is the unique arrow $f_0 : \{u_0\} \cong \{v_0\}$. Suppose now that we have finite posets $L \triangleleft U$ and $L' \triangleleft V$ such that there is an isomorphism $f_{n-1} : L \cong L'$. Suppose further that $\{u_0, \ldots, u_{n-1}\} \subseteq L$ and $\{v_0, \ldots, v_{n-1}\} \subseteq L'$. We wish to extend the isomorphism f_{n-1} to an isomorphism $f_n : M \cong M'$ where $M \triangleleft U$ and $M' \triangleleft V$ are finite and $u_n \in M$ and $v_n \in M'$. Now, we know that there is a finite poset $N \triangleleft U$ with $L \cup \{u_n\} \subseteq N$. From the inverse of the isomorphism f_{n-1} we can build an embedding-projection pair $f : L' \to N$. Since V is saturated, there is a poset $N' \triangleleft V$ and an isomorphism $g : N' \cong N$. To complete the argument, we add $\{v_n\}$ to N' and find a subset $M' \subseteq V$ such that $\{v_n\} \cup N' \subseteq M'$. Since U is saturated we find an isomorphic copy M of M' inside U, containing L, such that the isomorphism $g^{-1} : N \cong N'$ is extended to an isomorphism $f_n : M \cong M'$. In this way we obtain a sequence $f_0, f_1, \ldots$ of isomorphisms whose union is an isomorphism between $\mathsf{K}(U)$ and $\mathsf{K}(V)$. This isomorphism extends to an isomorphism between U and V. ∎

Theorem 3.2.25 *If an incremental category of domains has finite saturations, then it has a saturated object.*

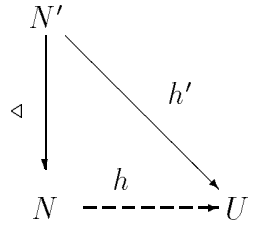
Proof. Recall the construction in the proof of Theorem 3.2.16. Suppose $\mathbf{C}$ is an incremental category with finite saturations. Let S_0 be any initial object of $\mathbf{C}$. Build the chain $S_0, S_1 = S_0^+, ..., S_{i+1} = S_i^+, ...$ where $s_{i+1,i}$ is a saturation for each i. Let $\mathcal{U}$ be a bilimit for this chain. It will simplify matters to assume that each of these saturations is an inclusion by replacing each S_i by its embedded image in $\mathcal{U}$. Suppose $M \triangleleft U$ and there is an embedding-projection pair $f : M \to N$ for some finite $N \in \mathbf{C}$. We must show that there is is an h such that



The proof is by induction on the number n of elements of N not in the image of f. If $n = 0$, then f is an isomorphism so we may take the coextension of f^{-1} to U as h. If $n \geq 1$, then it is possible to find an increment $f' : M \to N'$ such that f' extends f and $N' \triangleleft N$ and there is exactly one element in N' which is not in the image of f'. Since M is finite, there is an i such that $M \subseteq S_i$. Since f' is an increment, there is an h' such that



We can now apply our inductive hypothesis to find an h such that



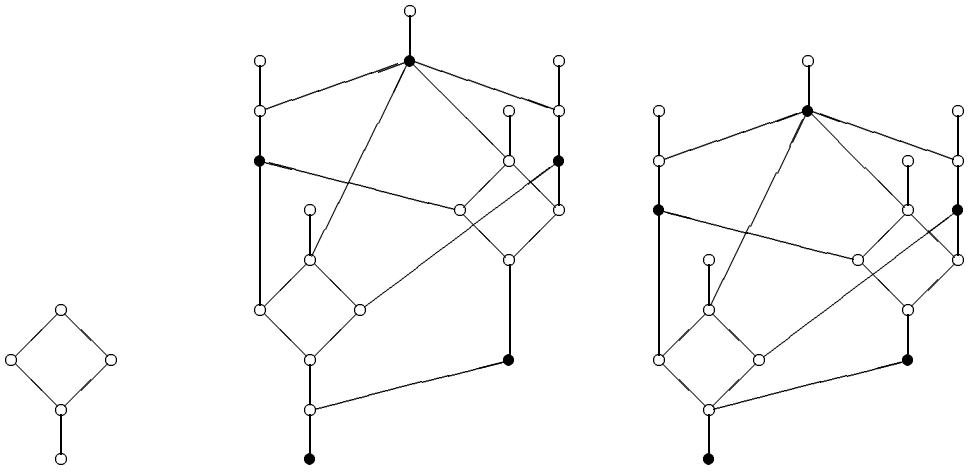
By putting these last two diagrams together we see that h has the desired properties. ∎

Corollary 3.2.26 *There are saturated objects in each of the following categories:*

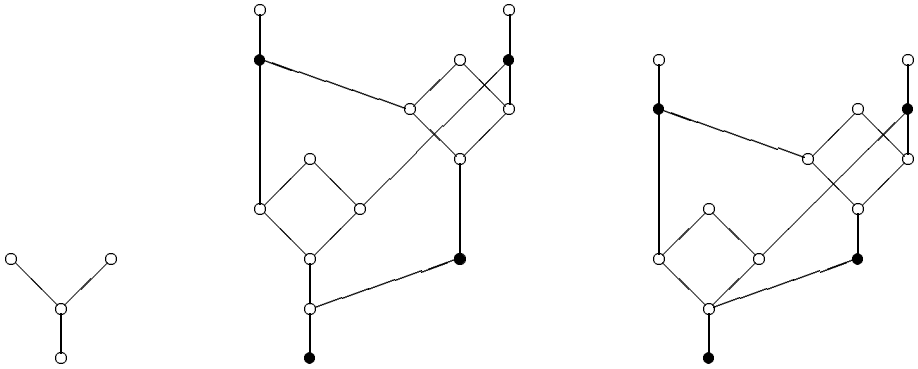
1. ω-$\mathbf{B}^{ep}$
2. ω-$\mathbf{BL}^{ep}$
3. ω-$\mathbf{S}^{ep}$

4. ω-$\mathbf{Lat}^{ep}$

It is interesting to note that Scott's universal domain for the consistently complete domains [Sco81b, Sco82a, Sco82b] is *not* saturated. To see this, it suffices to note that the meet of compact elements in the saturated consistently complete domain is not compact whereas the intersection of compact elements in Scott's universal domain *is* compact.



A A^+ in ω-$\mathbf{B}^{ep}$ A^+ in ω-$\mathbf{BL}^{ep}$



B B^+ in ω-$\mathbf{BL}^{ep}$ B^+ in ω-$\mathbf{S}^{ep}$

C C^+ in ω-$\mathbf{S}^{ep}$ C^+ in ω-$\mathbf{Lat}^{ep}$

Figure 3.7: Saturations in different categories.

Chapter 4

Types for database languages

In the previous chapter we saw how difficult it is to give a good mathematical account of the computational phenomena of sequentiality and polymorphism. In this chapter the situation is exactly the reverse, we address the problem of incorporating the mathematical notion of a *set* into functional languages. In a nutshell, the difficulty arises from the fact that while we have ways to *construct* and manipulate sets there is no deterministic way to *destruct* them, that is, to retrieve their elements.

Relational database theory rests firmly on the concept of a relation, that is, a set of tuples. The problem just mentioned does not appear in database query languages because all one can do is writing queries the result of which is a print-out of tuples. The order, in which these tuples appear on the screen, plays no role. More precisely, all programs in a query language have type $\mathsf{query} \times \mathsf{relation} \to \mathsf{relation}$, there is no connection with other data types nor are there higher types.

This problem has long been in the focus of Peter Buneman's research interests. Around 1986 he turned to the semantic models for functional programming languages, that is to domains, for gaining some understanding of how a type system including sets might be designed. The results of this research where recorded in [BDW91] and in a precursor of the paper reprinted below. I became interested in this problem during a visit to the University of Pennsylvania in Summer 1988. On the invitation of Peter Buneman I returned to Philadelphia for six weeks in early Spring 1989. I studied his draft paper and found that from the domain theoretic side a lot more could be done. The result was a thorough reworking of Sections 4.1.1–4.1.4 by me. The main definitions, however, are due to Peter Buneman. The paper appeared as

[BJO91] in *Theoretical Computer Science*.

The paper achieves a rather neat abstract view of relations and it is shown that the usual database operations have conservative extensions to the abstract setting. However, the problem mentioned in the beginning is not solved, that is, a definition of how to order abstract relations, so that they form another domain, is still missing.

The smoothness with which we could extend database operations to domains in that paper raised the question in how far traditional database algebra had really been generalized. Put in domain theoretic terms, the question was whether certain domains could be represented as subsets of products of flat domains. The research on this was carried out during Summer 1989, in the early phases with the help of Hermann Puhlmann, then working towards his diploma at Technische Hochschule Darmstadt. The decomposition theory I developed is very similar to a technique in distributive lattices [BD74] but due to the fact that domains need not have top elements the decomposition is rather more interesting.

During the course of writing a paper on these results (which was my task in this case) it emerged that Leonid Libkin in Philadelphia had similar ideas. We agreed to to prepare a joint submission to the conference *Mathematical Foundations of Programming Semantics* to take place in Pittsburgh in 1991. It was accepted and the paper appeared as [JLP92]. However, it turned out that our approaches were quite orthogonal after all and so the paper consists of two independent parts. For the present purposes I have included only my part.

I would like to make a few comments about this paper from today's perspective. First of all, while Theorem 4.2.18 is true, I did not have a correct proof of it at the time the paper was published. This was noticed and a complete proof was provided by Christian Haack in [Haa93]. Secondly, my hope that the decomposition theory would lead to a positive solution of the Full Abstraction Theorem for PCF was too optimistic. This can be seen from the unsuccessful attempts made by Steve Brookes and Shai Geva [BG92a, BG92b, BG93] which are based on similar ideas cast in the form of generalized concrete data structures. Thirdly, regarding the problem of defining a sensible power type, whose elements stand for partial descriptions of sets, some progress has been made. There are quite a few papers by Leonid Libkin and Limsoon Wong concerned with this question [Lib91, LW92, LW93] and I should also mention Hermann Puhlmann's PhD work which concentrates on what he calls the

"snack powerdomain" [Puh93]. When looked at localically (in the spirit of [Abr91]) then this construction exhibits a very natural logical structure. On the other hand, the localic viewpoint also shows that Peter Buneman's concept of "semi-factor" must be refined.

4.1 Using Powerdomains to Generalize Relational Databases

Introduction

There are two motivations for this study. The first is to draw together a number of approaches to data models and to examine the extent to which they can be viewed as generalizations of the relational data model. The second is to try to draw out the connection between data models and data types, something that is crucial if we are to achieve a proper integration of databases [Atk78, AB87, Sch77] and programming languages.

The main focus of this paper is the first of these. There are a number of attempts to generalize the relational data model beyond first-normal-form relations [FT83, RKS85, ÖY85]; there are also numerous formulations of other data models [AB84, HM81, BK86, HY84] that at first sight appear to have little to do with relations. We shall see that by exploiting the basic ideas of domain theory, well known in the study of semantics of programming languages, we can obtain generalizations of many of the basic results of relational databases in a way that has very little to do with the details of the data structures that are used to define them; and which allows the application of relational database principles to a much wider range of data models. Although some observations have been made [Ris85, Car85] that suggest a connection between database and programming language semantics, there appears to have been no attempt directly to characterize relational databases in the appropriate semantic domains.

To the hardened first-normal-form relational database theorist this paper offers little more than alternative, and perhaps simpler, derivations of some existing results. However, given the recent activity in the study of "higher order" relations, which attempts to apply the basic results of relational databases to other structures, it is interesting to ask how far this work can be pushed. What are the properties of the data model that allow us to define relational operators, functional dependencies etc.? In doing this, we shall find it useful to produce a simple denotational semantics for relations and other structures, which is an extension to the semantics for missing values proposed by Lipski [Lip79]. The idea is that these structures denote sets of val-

ues in some space which we may think of as the "real world". One of the advantages of our approach is that it allows us to provide a denotational semantics for structures such as sets of attribute names, which usually receive an operational treatment. Such a semantics will, we hope, ultimately be useful if we are ever to achieve our second goal of achieving a healthy marriage of databases and programming languages.

The organization of this paper is as follows. In Section 2 we describe the properties of the underlying domains that we shall need. Section 3 then shows how powerdomain orderings (orderings on sets of values) can be used to characterize the various joins that are discussed in relational algebra. In Section 4, in trying to characterize projection, we introduce the notion of *schemes*, which generalize relational schemes (sets of column names). Schemes enjoy some nice properties with respect to powerdomain orderings and allow us to characterize functional dependencies and universal relations, which is done in the following sections. Section 7 concludes by showing how these ideas can be applied to various extensions of relational databases including typed relations, relations with null values and various forms of higher order relations; it also suggests that there may be some limitations to what one can do with non first-normal-form relations. The reader who is more interested in data types and structures rather than some of the more esoteric areas of database theory may wish to skip much of Sections 5 and 6, and turn directly to Section 7.

4.1.1 Orderings and Domains

The idea that is fundamental in denotational semantics is that expressions denote values, and that the domain of values is partially ordered. In the same way we can think of database structures as descriptions and that these descriptions are partially ordered by how well they describe the real world. Without putting any particular structure on the real world, we can define the meaning $[\![d]\!]$ of a description d as the set of all real-world objects described by d. We can then say that a description d_1 is better than d_2, $d_1 \sqsupseteq d_2$, if d_1 describes a subset of the real-world objects described by d_2, i.e. $[\![d_1]\!] \subseteq [\![d_2]\!]$.

An example of such an ordering is to be found in flat record structures. A flat record is a partial function from a set $\mathcal{L}$ of labels to an unordered set $\mathcal{V}$ of values. If r_1 and r_2 are two such functions, then $r_1 \sqsupseteq r_2$ if the graph of r_1 contains the graph of r_2. For example,

$$\{Name{\Rightarrow}'J.\ Doe';Dept{\Rightarrow}'Sales';Office{\Rightarrow}33\}$$
$$\sqsupseteq \{Name{\Rightarrow}'J.\ Doe';Dept{\Rightarrow}'Sales'\}$$

Using the term "real world" to describe the semantics of such records is, of course, contentious. It is better to think of these records as partial descriptions (or approximations) to elements in some space or "universe" of total descriptions, in this case large – possibly infinite – record structures. Suppose that this universe were the function space $\mathcal{L} \to \mathcal{V}$ where $\mathcal{L} = \{Name, Dept, Office\}$, we would then have

$$[\![\{Name{\Rightarrow}'J.\ Doe';Dept{\Rightarrow}'Sales'\}]\!] =$$
$$\{\{Name{\Rightarrow}'J.\ Doe';Dept{\Rightarrow}'Sales';Office{\Rightarrow}\ v\} \mid v \in \mathcal{V}\}$$

Note that this formulation of the denotation of a record with incomplete information corresponds with that given in [Lip79], and as it will shortly appear, this space of flat records provides the basis for the relational model; however there are a number of other orderings that we shall examine later in this paper. These include Bancilhon's complex objects [BK86], orderings on tree structures that give rise to higher order relations [FT83, AB84, RKS85, RKS84, ÖY85], the feature structures in unification-based grammar formalisms (see [Shi85] for a survey), finite state automata [RK85], ψ-terms [AK86]. In this catalog we should also include Scott's aptly-named "information systems" – consistent, deductively closed sets of predicates [Sco82a]. In all of these it is possible to describe certain generalizations of relational operations.

We shall require somewhat more structure on our space D of partial descriptions than being partially ordered. The most important property is that it is *bounded complete*:

1. any non-empty subset S of D has a greatest lower bound $\sqcap S$,

In addition we shall also make two further assumptions that are common in denotational semantics [Sch86]:

2. any directed subset S of D has a least upper bound $\sqcup S$,

3. the set $\mathsf{K}(D)$ of *compact* elements in D forms a countable basis for D.

Partially ordered sets $(D, \sqsubseteq)$ with these properties are widely used in the semantics of programming languages, and are often called Scott domains [Sco82a].

Throughout this paper we shall refer to them as *domains*. We shall also use the notation $s_1 \sqcup s_2$ and $s_1 \sqcap s_2$ for $\sqcup\{s_1, s_2\}$ and $\sqcap\{s_1, s_2\}$ respectively.

It is an immediate consequence of the first condition that any subset of S of D that is bounded above has a least upper bound $\sqcup S$ and also that that D has a bottom element, $\bot_D$. The second condition, when taken with the axiom of choice, ensures that every member of D is bounded above by some member of D_{max}, the set of maximal elements of D. We shall therefore use D_{max} as the universe of complete descriptions; and the definition of $[\![d]\!]$ is then simply $\{x \in D_{max} \mid d \sqsubseteq x\} = \uparrow d \cap D_{max}$. Apart from some remarks at the end of the paper, we shall not make any use of the third condition; however we should note that in any practical database context this condition will surely be satisfied.

There is one extra condition which we shall need when we introduce *schemes* below:

4. A domain D is *distributive* if every principal ideal $\downarrow x$ is a distributive lattice.

Note that the space of flat record structures is a distributive domain. Even more is true of this domain: each principal ideal is a complete atomic boolean algebra, that is, a powerset. We shall not need to assume this in general, however.

We shall see that there are a number of ways to construct domains that represent the kinds of data structures we use in databases; particularly simple are the *flat* domains. Given a set of atomic values $\mathcal{V}$, a flat domain $\mathcal{V}_\bot$ of $\mathcal{V}$ is obtained by adding bottom element $\bot$ to $\mathcal{V}$ and ordering them as $x \sqsubseteq y$ if and only if $x = y$ or $x = \bot$. This domain is a domain of atomic descriptions; an element $v \in \mathcal{V}_\bot$ is either a complete description ($v \neq \bot$) with the meaning $\{v\}$ or the non-informative description $\bot$ with the meaning $\mathcal{V}$. The bottom element introduced in $\mathcal{V}_\bot$ can be interpreted as a *null value* representing "unknown values". There are number of other approaches to null values, some of them distinguish "inappropriate" and "unknown" values. Such an approach is entirely consistent with what we develop here and can be modeled by domains that are more complicated than $\mathcal{V}_\bot$. Later we shall comment more on null values.

We can now describe more precisely the domain of *labeled records* that we discussed in the introduction. Given a countable set of labels $\mathcal{L}$ and a domain D, a domain of labeled record $\mathcal{L} \to D$ over D is a set of total functions from $\mathcal{L}$ to D

with the ordering defined as $r_1 \sqsubseteq r_2$ if and only if for all $l \in \mathcal{L}$, $r_1(l) \sqsubseteq r_2(l)$. This can be thought of as a domain of descriptions by attributes. This ordering represents the fact that r_2 is a better description that r_1 if r_2 has better descriptions than r_1 in all attributes. The minimal element $\perp_{\mathcal{L}\to D}$ in $\mathcal{L} \to D$ is the constant function $\perp_D$ and if S is a set of functions, then $\sqcap S$ is the function r such that for all l, $r(l) = \sqcap\{s(l) \mid s \in S\}$, and $\sqcup S$ is the function r' such that for all $l, r'(l) = \sqcup\{s(l) \mid s \in S\}$ provided that all the least upper bounds exist.

The space of *flat records* is a special case of a domain of records where D is a flat domain $\mathcal{V}_\perp$. Indeed, the space of partial functions from $\mathcal{L}$ to $\mathcal{V}$ is isomorphic to $\mathcal{L} \to \mathcal{V}_\perp$. To make our notation for records precise, $\{l_1 \Rightarrow d_1; \cdots; l_n \Rightarrow d_n\}$ denotes an element r in $\mathcal{L} \to D$ such that $r(l_i) = d_i$ for $1 \leq i \leq n$ otherwise $r(l) = \perp_D$. For example, in $\mathcal{L} \to \mathcal{V}_\perp$, if

$$r_1 = \{Emp\#\Rightarrow 12345; Name\Rightarrow \textit{'J. Doe'}\}$$

and

$$r_2 = \{Emp\#\Rightarrow 12345; Sal\Rightarrow 20000\}$$

then

$$r_1 \sqcap r_2 = \{Emp\#\Rightarrow 12345\}$$

and

$$r_1 \sqcup r_2 = \{Emp\#\Rightarrow 12345; Name\Rightarrow \textit{'J. Doe'}; Sal\Rightarrow 20000\}.$$

However $\{Emp\#\Rightarrow 12345; Name\Rightarrow \textit{'J. Doe'}\} \sqcup \{Name\Rightarrow \textit{'K. Smith'}\}$ does not exist. An advantage of treating the space of flat records as $\mathcal{L} \to \mathcal{V}_\perp$ is that many results concerning flat records can be regarded as special cases of more general records and are readily applied to $\mathcal{L} \to D$ for a more complicated domain D.

As an example consider a database which lists the values of physical constants as they have been determined in particular experiments. Set up as a relational database, a typical entry might contain the following fields (among others): *author, publication, name of constant, lower bound, upper bound, dimension.* Being forced to express every record in first-normal-form has two obvious disadvantages. First, it does not reflect the property that the intervals [*lower bound, upper bound*] are partially ordered, smaller intervals being better approximations, and, second, there is no way how the obvious dependency (*name of constant* $\Longrightarrow$ [*lower bound, upper*

Name	Dept	Sal	Office
'K. Smith'	*'Mktg'*	30,000	275
'J. Doe'	*'Sales'*	20,000	147

$$\{\{Name{\Rightarrow}\textit{'J. Doe'};\ \ Dept{\Rightarrow}\textit{'Sales'};\ Sal{\Rightarrow}20{,}000;\ Office{\Rightarrow}147\},$$
$$\{Name{\Rightarrow}\textit{'K. Smith'};Dept{\Rightarrow}\textit{'Mktg'};\ Sal{\Rightarrow}30{,}000\ Office{\Rightarrow}275\}\}\ .$$

Figure 4.1: A relation and its representation as a set of records

bound]) could be expressed in ordinary relational algebra. Our formalism as developed below will allow to state such a dependency and will provide a simple formula for checking the consistency of the database. (An inconsistency is reached in our example if asserted intervals for the same constant do not overlap.)

4.1.2 Powerdomains and Relational Algebra

Databases usually contain sets of values which, from our foregoing discussion, we would expect to describe sets of objects in the real world. If we interpret database values as elements in a domain, then database sets, such as relations, must be interpreted as sets of elements in that domain. Indeed, we can interpret a *first-normal-form* relation r of a relational scheme (a set of attribute names) R in the relational model as a set S of elements in the domain of flat records $\mathcal{L} \to \mathcal{V}_\perp$ such that for any $d \in S, \{l \mid d(l) \neq \perp\} = R$. Later in this section, we shall see that this interpretation is faithful to various relational operations and that the domain of flat records, therefore, serves as a domain of the relational model. This is how relations are described in languages such as Pascal/R [Sch77], and extensions of this representation are to be found in Taxis [BMW80] and Galileo [ACO85]. Figure 4.1 shows a very simple relation and its representation as a set of flat records.

If we consider these sets of elements in a domain as sets of descriptions then we would like to order the sets themselves by how good they describe sets of real-world objects, but how? The study of the semantics of non-determinism, which attempts to describe the behavior of sets of processes, provides us with some answers. However,

A	B
a	b
a	$\perp$

(i)

A	B	C
a	b	$\perp$
a	$\perp$	c

(ii)

Figure 4.2: Some problematic relations

we must first decide whether we are prepared to work arbitrary sets, or whether some restrictions are needed.

Given a domain $(D, \sqsubseteq)$, a set $S \subset D$ is a *co-chain* if no member of S is greater than any other member of S, i.e. $\forall x, y \in S. x \sqsupseteq y$ implies $x = y$. If $S \subset D$ has the property that any two members of S are *inconsistent*, i.e. they do not have a defined join, then we shall call S *independent*. Note that an independent set is necessarily a co-chain.

First-normal-form relations are independent sets. If, however, we admit null values in relations by relaxing the condition $\{l \mid d(l) \neq \perp\} = R$ of first-normal-relation to $\{l \mid d(l) \neq \perp\} \subseteq R$, we have to decide whether structures such as (i) or (ii) of figure 4.2 are valid relations. (i) fails to be a co-chain because $\{A{\Rightarrow}a\} \sqsubseteq \{A{\Rightarrow}a;\ B{\Rightarrow}b\}$, and (ii) fails to be independent because $\{A{\Rightarrow}a;\ B{\Rightarrow}b\} \sqcup \{A{\Rightarrow}a;\ C{\Rightarrow}c\}$ is defined.

In what follows we shall assume that database sets are finite co-chains and we shall use the words *finite co-chain* and *relation* interchangeably. Using our simple notion of database semantics, we might justify this assumption by saying that if d_1 and d_2 are descriptions with d_2 a better description than d_1 then d_1 is redundant and can be eliminated from the database. This is equivalent to saying that for all pairs d_1, d_2 in S neither $[\![d_1]\!] \subseteq [\![d_2]\!]$ nor $[\![d_1]\!] \subseteq [\![d_2]\!]$. Whether or not this justification is reasonable depends on the intended semantics of the operations on co-chains which, in turn, depends on the circumstances in which they are used. See [Oho86] for a more detailed examination of the semantics of relational operations. Independence means that no two descriptions in S can describe the same real-world object, i.e. $[\![d_1]\!] \cap [\![d_2]\!] = \emptyset$. We shall need to discuss independent sets when we generalize the

R_1

Dept	Office
'Mktg'	275
'Sales'	147

R_2

Name	Dept	Office
'K. Smith'	*'Mktg'*	275
'L. Jones'	*'Mktg'*	275

R_3

Name	Dept	Office
'K. Smith'	*'Mktg'*	275
'L. Jones'	*'Mktg'*	275
'J. Doe'	*'Sales'*	147
'M. Blake'	*'Sales'*	147

$$R_1 \sqsubseteq^\sharp R_2 \quad R_2 \sqsubseteq^\flat R_3 \quad R_1 \sqsubseteq^\natural R_3$$

Figure 4.3: Examples of the three orderings

notion of schemes. We shall use $\mathcal{C}_D$ to refer to the set of finite co-chains in D and $\mathcal{I}_D$ for the set of finite independent sets.

To return to the problem of finding orderings on sets the study of the semantics of non-determinism provides us with three orderings[1]:

$$A \sqsubseteq^\flat B \quad \text{if} \quad \forall a \in A \exists b \in B . a \sqsubseteq b$$
$$A \sqsubseteq^\sharp B \quad \text{if} \quad \forall b \in B \exists a \in A . a \sqsubseteq b$$
$$A \sqsubseteq^\natural B \quad \text{if} \quad A \sqsubseteq^\flat B \text{ and } A \sqsubseteq^\sharp B$$

respectively called the Hoare, Smyth, and Egli-Milner ordering. Figure 4.3 shows examples of these orderings in first-normal-form relations.

For arbitrary sets, these are not orderings; they are pre-orderings and orderings are derived by taking equivalence classes. However, in each case there are canonical representatives for each equivalence class:

Lemma 4.1.1 *Let P be a partial order. Then the following is true for all subsets A and B of P.*

(i) $A =^\flat \downarrow A$.

[1]This melodious notation was suggested to us by Carl Gunter.

(ii) $A \sqsubseteq^\flat B \iff \downarrow A \subseteq \downarrow B$.

(iii) $A =^\sharp \uparrow A$.

(iv) $A \sqsubseteq^\sharp B \iff \downarrow A \subseteq \uparrow B$.

(v) $A =^\natural \uparrow A \cap \downarrow A$. ∎

So in reasoning about these orderings it is helpful to think in terms of lower sets, upper sets, and order-convex sets, respectively. We said before that we want to model database sets (or relations) as finite co-chains in our domains. Since databases tend to get bigger and bigger during their existence one might think that the Hoare ordering is the most natural for them. However, viewed as approximations of sets of real world objects it is the Smyth ordering which corresponds to this semantics. We regard it as a strength of our approach that it allows to formalize different intuitions about databases. The mathematics is nice in each case:

Lemma 4.1.2 *If D is a domain then $(\mathcal{C}_D, \sqsubseteq^\flat)$ and $(\mathcal{C}_D, \sqsubseteq^\sharp)$ are distributive lattices with bottom element. $(\mathcal{C}_D, \sqsubseteq^\sharp)$ also has a top element, namely, the empty co-chain.*

Proof. Given two finite co-chains S_1 and S_2, it is clear how the sup and inf are found for each of the two orderings:

$S_1 \sqcup^\flat S_2$ = the maximal elements of $\downarrow S_1 \cup \downarrow S_2 =$ the maximal elements of $S_1 \cup S_2$.

$S_1 \sqcap^\flat S_2$ = the maximal elements of $\downarrow S_1 \cap \downarrow S_2 \subseteq S_1 \sqcap S_2 = \{s_1 \sqcap s_2 \mid s_1 \in S_1, s_2 \in S_2\}$.

$S_1 \sqcup^\sharp S_2$ = the minimal elements of $\uparrow S_1 \cap \uparrow S_2 \subseteq S_1 \sqcap S_2$.

$S_1 \sqcap^\sharp S_2$ = the minimal elements of $\uparrow S_1 \cup \uparrow S_2 \subseteq S_1 \cup S_2$.

Distributivity follows because we can embed $\mathcal{C}_D$ in the distributive lattice of all lower (upper) sets in D. ∎

We wish to remark that these lattices are not complete: Neither $(\mathcal{C}_D, \sqsubseteq^\flat)$ nor $(\mathcal{C}_D, \sqsubseteq^\sharp)$ contain sups for directed subsets. If we want completeness then we have to take certain computability considerations into account which translate into topological restrictions on infinite subsets of a domain. We have no need to pursue this

theme further but note that sup and inf in both orderings are defined for any set of subsets of a domain. They may not be representable by their subset of minimal or maximal elements, however.

It the space of finite co-chains with the three orderings in which we represent various operations on database sets, some of which will emerge as generalizations of relational operations. We also mention that these ordered spaces are not the same as powerdomains in the programming language literature [Plo76, Smy78], where the ordered spaces of sets are constructed in such a way that they are themselves domains. True powerdomain constructions are not needed until we discuss higher-order relations, where a tuple can itself contain a set as an attribute value. We shall discuss how our presentation of database sets can also contain these higher-order values in Section 7, but for the time being we shall exploit the representation of database sets in the space of finite co-chains.

There is an immediate connection with relational algebra that indicates the importance of these orderings.

Theorem 4.1.3 *Interpreting relations as finite co-chains A, B in $\mathcal{L} \to \mathcal{V}_\perp$, $A \sqcup^\sharp B$ is the natural join of A and B. If a least upper bound for A, B exists in $\sqsubseteq^\sharp$ then it is a lossless join.*

This statement is actually more of a definition than a result. We can only prove it in the case of first-normal-form relations, for it is only then that we have accepted definitions for the various joins. Given relation schemes (sets of attribute names) $R_1, R_2 \subseteq \mathcal{L}$ and relation instances r_1, r_2, let r_1', r_2' and r_3' be the interpretations of r_1, r_2 and $r_1 \bowtie r_2$ in $\mathcal{L} \to \mathcal{V}_\perp$. Suppose $t \in r_3'$, then by the conventional definition of natural join, there are $t_1 \in r_1'$ such that $t(l) = t_1(l)$ for all $l \in R_1$ and $t_2 \in r_2'$ such that $t(l) = t_2(l)$ for all $l \in R_2$. By the definition of the interpretation, $t_1(l) \neq \perp$ iff $l \in R_1, t_2(l) \neq \perp$ iff $l \in R_2$. This implies $t_1 \sqsubseteq t$ and $t_2 \sqsubseteq t$ and clearly t is minimal with respect to this property. Therefore $t \in r_1' \sqcup^\sharp r_2'$. Conversely suppose $t \in r_1' \sqcup^\sharp r_2'$. There must exist $t_1 \in r_1, t_2 \in r_2$ such that $t_1 \sqsubseteq t$ and $t_2 \sqsubseteq t$. Since $\mathcal{V}_\perp$ is flat this implies that $t(l_1) = t_1(l_1)$ when $l_1 \in R_1$ and $t(l_2) = t_2(l_2)$ when $l_2 \in R_2$. By the minimality of t with respect to $\sqsubseteq$, $t(l) = \perp$ iff $l \notin R_1 \cup R_2$. Hence $t \in r_3'$. See [Oho86] for a discussion of the semantics of lossless join and the proof of the second part of this result. ∎

$$
\begin{aligned}
r_1 \quad &= \{\ \{Name \Rightarrow \text{'J. Doe'} \quad ; Status \Rightarrow \{\ Student\text{-}status \Rightarrow \text{'Graduate'}\}\} \\
&\qquad \{Name \Rightarrow \text{'M. Blake'}; Status \Rightarrow \{\ Student\text{-}status \Rightarrow \text{'Undergraduate'}\}\}\} \\
r_2 \quad &= \{\ \{Name \Rightarrow \text{'J. Doe'}; \quad Status \Rightarrow \{\ Employee\text{-}status \Rightarrow \text{'TA'}\}\} \\
&\qquad \{Name \Rightarrow \text{'L. Jones'}; \ Status \Rightarrow \{\ Employee\text{-}status \Rightarrow \text{'Faculty'}\}\}\} \\
r_1 \sqcup^\sharp r_2 = &\ \{\ \{Name \Rightarrow \text{'J. Doe'}; \\
&\qquad Status \Rightarrow \{\ Student\text{-}status \Rightarrow \text{'Graduate'}; Employee\text{-}status \Rightarrow \text{'TA'}\}\}\}
\end{aligned}
$$

Figure 4.4: Natural join in "nested" records

The importance of this result is that it provides a generalization of natural join to sets of values in arbitrary domains. Figure 4.4 shows an example of natural join in nested records.

A more intuitive way of thinking of these results is to view the natural join as the appropriate operation when two sets of database descriptions "over-approximate" some desired set in the real world. Suppose, for example, that we want to find the set of TEACHING-FELLOWS, but we only have available database sets describing EMPLOYEES and STUDENTS. Both of these over-approximate our desired set (any teaching fellow is both an employee and a student) and so the appropriate operation to achieve a better approximation to TEACHING FELLOWS is to take the natural join of EMPLOYEES and STUDENTS.

The partial ordering $\sqsubseteq^\natural$ does not give rise to least upper bounds when applied to co-chains. However, if two database sets have a least upper bound in $\sqsubseteq^\natural$, then any real world set that is "exactly" described by (i.e. above in $\sqsubseteq^\natural$) the two database sets is also "exactly" described by the least upper bound. Since a least upper bound in $\sqsubseteq^\natural$ is also a least upper bound in $\sqsubseteq^\sharp$, if $\sqcup^\natural$ exists then the natural join is the lossless join. Traditionally the lossless join condition is stated operationally, in terms of projections; from this we see that it has a simple denotational interpretation.

We might also ask whether $\sqcup^\flat$ corresponds to anything in the relational algebra. $S_1 \sqcup^\flat S_2$ is simply the set of maximal elements in $S_1 \cup S_2$ and is awkward to deal with in relational algebra as it generally requires the introduction of null values. However we shall make some use of this operator later. If we are prepared to introduce null values, then $\sqcup^\flat$ is what [RKS84] calls the "null union", and $S_1 \sqcup^\flat (S_1 \sqcup^\sharp S_2) \sqcup^\flat S_2$

is what is sometimes called the *outer join*. Merrett [Mer84] describes this operation and also the "left-wing" and "right-wing" operations, which are $S_1 \sqcup^\flat (S_1 \sqcup^\sharp S_2)$ and $(S_1 \sqcup^\sharp S_2) \sqcup^\flat S_2$ respectively.

In some cases these operations preserve independence:

Lemma 4.1.4 *If S_1 and S_2 are independent, so are $S_1 \sqcup^\sharp S_2$, $S_1 \sqcup^\flat (S_1 \sqcup^\sharp S_2) \sqcup^\flat S_2$, $S_1 \sqcup^\flat (S_1 \sqcup^\sharp S_2)$ and $(S_1 \sqcup^\sharp S_2) \sqcup^\flat S_2$.* ∎

However, the other operators ($\sqcap^\flat, \sqcap^\sharp$ and $\sqcup^\flat$) do not, in general, carry independent sets into independent sets.

We should also note that the co-chain $S_1 \sqcap^\sharp S_2$ is the set of minimal elements of $S_1 \cup S_2$. When $S_1 \cup S_2$ is a co-chain, $S_1 \sqcap^\sharp S_2 = S_1 \sqcup^\flat S_2$. The operator $\sqcap^\flat$ is, as we shall see in the next section, a general form of projection.

In order to conform to traditional notation, we shall generally replace the symbol $\sqcup^\sharp$ by what is conventionally used in databases, $\bowtie$.

4.1.3 Projection

The main point of the previous section is that we are able to define various joins without reference to the special structure of relations. In particular, we do not require any notion of sets of column names (or schemes as they are called in the relational database literature [Mai83, Ull82a]) in order to characterize natural join. Projection, however, makes explicit mention of a scheme. For example $\{Name, Office\}$ is a scheme and the projection $\pi_{\{\text{Name, Office}\}}(R)$ where R is the relation shown in Figure 4.1 is

$$\{\{Name \Rightarrow 'J.\ Doe'; Office \Rightarrow 147\},$$
$$\{Name \Rightarrow 'K.\ Smith'; Office \Rightarrow 275\}\}\ .$$

If, therefore, we are to carry further the idea of casting relational algebra in the theory of domains, we need to generalize the notion of relational schemes and projection.

We have essentially two options: the first is to look at what properties are desired of the projection function itself; the second is to identify schemes with some set of elements in the underlying domain D. The second approach is motivated by the idea that a set of column names gives rise to a smaller universe of descriptions. For example, we might say that the relational scheme $\{Name, Office\}$ denotes the set of

all descriptions (functions) of the form $\{\{Name \Rightarrow v,\ Office \Rightarrow w\} \mid v, w \in \mathcal{V}\}$. The course we shall follow is to look at both possibilities with the goal of finding some characterization that is natural in the sense that it admits some natural algebra over the set of schemes. This is essential if we are to generalize ideas about functional dependencies which are usually cast in the boolean algebra of sets. However, the authors should admit here that the generalization of schemes that we are going to provide, while it arises from extremely natural conditions and captures a number of relational database constructs, may require further refinement if it is to be used for all of relational database theory. We do not know, for example, whether we can represent multi-valued dependencies using our characterization.

We start from the observation that in relational databases we can say what projection means for a single tuple. It is simply the function that throws away certain fields from a tuple or record. More generally, we can think of projection as a function $p \in D \to D$ that is decreasing, idempotent and monotone, i.e. for all $x, y \in D$, $p(x) \sqsubseteq x$, $p(p(x)) = p(x)$, and $p(x) \sqsubseteq p(y)$ whenever $x \sqsubseteq y$. Computability of a projection is reflected in the property of preserving directed sups: $p(\bigsqcup_{i \in I} x_i) = \bigsqcup_{i \in I} p(x_i)$. Such functions are also known as projections in domain theory, and it is clear that a (relational) projection onto a set of column names satisfies these conditions.

Projections are completely determined by their image:

Lemma 4.1.5 *Let D be a domain and p, p' be projections on D.*

(i) $p(x) = \bigsqcup\{y \mid p(y) = y \sqsubseteq x\}$.

(ii) $p \sqsubseteq p' \iff \mathsf{im}(p) \subseteq \mathsf{im}(p') \iff p \circ p' = p' \circ p = p$.

(iii) *p preserves inf's of nonempty sets.*

(iv) $\mathsf{im}(p)$ *is closed under existing sup's.* ∎

We feel that arbitrary projections as defined above do have a significance in modeling databases domain theoretically. In this paper, however, we shall concentrate on a more restricted notion of projection which we shall develop in two steps.

In the case of a relational domain $\mathcal{L} \to \mathcal{V}_\perp$, restricting the set of labels to some subset L of $\mathcal{L}$ gives rise to a downward closed subset of $\mathcal{L} \to \mathcal{V}_\perp$, namely the set of all functions s for which $s(l) = \perp$ if $l \notin L$.

Definition. Let D be a domain. A *strong ideal* in D is a downward closed subset A of D which is closed under existing joins. By p_A we denote the unique projection on D with image A.

Projections onto strong ideals enjoy several desirable properties:

Lemma 4.1.6 *(i) Let A be a strong ideal in a domain D and let y be an element of A above $p_A(x)$ for some $x \in D$. Then $p_A(x) = x \sqcap y$.*

(ii) If D is distributive then p_A preserves all existing sup's.

Proof. (i) By definition we have $p_A(x) \sqsubseteq x$ and $p_A(x) \sqsubseteq y$ so $p_A(x) \sqsubseteq x \sqcap y$. On the other hand, $x \sqcap y$ is an element of A below x and $p_A(x)$ is the sup of all those elements, hence $x \sqcap y \sqsubseteq p_A(x)$.

(ii) Suppose $x \sqcup y$ exists. Then $p_A(x) \sqcup p_A(y) = (x \sqcap p_A(x \sqcup y)) \sqcup (y \sqcap p_A(x \sqcup y))$ (by (i)) $= p_A(x \sqcup y) \sqcap (x \sqcup y)$ (distributivity) $= p_A(x \sqcup y)$. The sup of any set is equal to the directed sup of its finite subsets. Our projection preserves both kinds of sups, hence arbitrary sups. ∎

The intersection of an arbitrary set of strong ideals is again a strong ideal. This immediately gives us the following

Theorem 4.1.7 *The set $(\mathcal{SI}_D, \subseteq)$ of strong ideals on a (distributive) domain D is a (distributive) algebraic lattice.* ∎

The second condition on projections we want to consider here is also easily motivated by the example of flat record structures $\mathcal{L} \to \mathcal{V}_\perp$. Suppose we project onto records with labels from some subset L of $\mathcal{L}$ and we find that a record s is projected onto $p_L(s)$ below some $s' \in L \to \mathcal{V}_\perp$. This means that $p_L(s)$ contains null values for some labels from L and can be updated using the corresponding entries of s'. It is clear, then, that s itself can be updated, resulting in the record $s \sqcup s'$. We incorporate this property in our model as follows.

Definition. A strong ideal A in a domain D satisfies the *slide condition* if $\forall x \in D. \forall y \in A. (p_A(x) \sqsubseteq y \Longrightarrow x \sqcup y$ exists). A co-chain S in D is a *scheme* if $\downarrow S$ is a strong ideal which satisfies the slide condition. The corresponding projection we denote by p_S (instead of $p_{\downarrow S}$).

We first note that projections defined by schemes fit in with our proposed semantics:

Theorem 4.1.8 *A strong ideal A on a domain D is generated by a scheme if and only if $\forall x \in D.\ p_A[\![x]\!]_D = [\![p_A(x)]\!]_A$.*

Proof. "$\Longrightarrow$" Let x be maximal in D and suppose $p_A(x)$ is not maximal in A, that is, $p_A(x) \sqsubset y \in A$. By the slide condition, $x \sqcup y$ exists and since x is maximal, $x \sqcup y = x$ and $p_A(x) \sqsupseteq y$. Contradiction. Given any $x \in D$ and any $y \in [\![p_A(x)]\!]_A$, the sup of x and y exists and is below some maximal element z of D. Clearly, $p_A(z) = y$ by the maximality of y.

"$\Longleftarrow$" Given $x \in D$ and $y \in A$, $y \sqsupseteq p_A(x)$, let y' be an element of A_{max} above y. This element must be in the image of $p_A[\![x]\!]_D$, that is, there exists an element z of $D_{max} \cap {\uparrow}x$ which is mapped onto y'. Therefore x and y are bounded and $x \sqcup y$ exists. ∎

In Section 7.1 we shall further substantiate our claim that schemes properly generalize the notion of schemes in relational database theory by showing that schemes in the domain $\mathcal{L} \to \mathcal{V}_\perp$ of flat record structures correspond exactly to the subsets of $\mathcal{L}$.

Lemma 4.1.9 *Let D be distributive domain.*

(i) If A and B are strong ideals generated by schemes then so is $A \sqcup B = \{a \sqcup b \mid a \in A, b \in B\}$.

(ii) If A and B are strong ideals generated by schemes then so is $A \cap B$.

(iii) If $(A_i)_{i \in I}$ is any set of strong ideals generated by schemes then so is $\bigsqcup_{i \in I} A_i = \{\bigsqcup_{i \in I} a_i \mid a_i \in A_i\}$.

(iv) If A, B are schemes then so is $A \sqcup B = \{a \sqcup b \mid a \in A, b \in B\}$.

(v) If $(A_I)_{i \in I}$ is any set of schemes then so is $\bigsqcup_{i \in I} A_i = \{\bigsqcup_{i \in I} a_i \mid a_i \in A_i\}$.

Proof. (i) $A \sqcup B$ is downward closed: $x \sqsubseteq a \sqcup b$ implies $x = x \sqcap (a \sqcup b) = (x \sqcap a) \sqcup (x \sqcap b)$ and $x \sqcap a$ is in A and $x \sqcap b$ is in B. If M is a bounded subset of $A \sqcup B$ then $M_A = \{a \in A \mid \exists b \in B. a \sqcup b \in M\}$ and $M_B = \{b \in B \mid \exists a \in A. a \sqcup b \in M\}$ are bounded and $\bigsqcup M_A = m_A \in A$ and $\bigsqcup M_B = m_B \in B$. Hence $\bigsqcup M = m_A \sqcup m_B$ is in

$A \sqcup B$. As for the slide condition, assume that x is an element of D and that $p_{A \sqcup B}(x)$ is below some $a \sqcup b$. Because $p_A(a \sqcup b) \sqsupseteq a$ and $p_B(a \sqcup b) \sqsupseteq b$ we may assume that $a = p_A(a \sqcup b)$ and $b = p_B(a \sqcup b)$. We can then calculate: $p_A(x) = p_A(p_{A \sqcup B}(x)) \sqsubseteq p_A(a \sqcup b) = a$ and similarly $p_B(x) \sqsubseteq b$. Since A satisfies the slide condition, the sup of a and x exists and by Lemma 4.1.6 $p_B(a \sqcup x) = p_B(a) \sqcup p_B(x) \sqsubseteq p_B(a \sqcup b) \sqcup b = b$. Using the slide condition for B we find that the sup of $a \sqcup x$ and b must exist. This proves the slide condition for $A \sqcup B$.

(ii) $A \sqcap B$ is clearly a strong ideal. The slide condition is seen to hold by the following argument. If $p_{A \sqcap B}(x)$ is below $y \in A \sqcap B$ then because of $p_{A \sqcap B} = p_A \circ p_B$, p_A maps $p_B(x)$ below y. Hence $y \sqcup p_B(x)$ exists and is an element of B. Using the slide condition for B we see that $y \sqcup x$ exists.

(iii) If I is empty then $\bigsqcup_{i \in I} A_i$ equals $\{\bot\}$ which is a scheme. If I is infinite then we may think of I as the directed union of its finite subsets. From part (i) we already know how to construct the sup of a finite set of strong ideals, so it remains to consider directed collections. Assume, therefore, that I is directed and that $A_i \subseteq A_j$ whenever $i \leq j$. Given an element x of $A = \bigsqcup_{i \in I} A_i$ first note that $x = \bigsqcup_{i \in I} p_{A_i}(x)$ and that this join is directed. It is clear that A is a strong ideal. Let X be any element of D and let $y \in A$ be above $p_A(x)$. Then for each $i \in I$, $p_{A_i}(y)$ is above $p_{A_i}(p_A(x))$. So the sup $z_i = p_{A_i}(y) \sqcup x$ exists and the directed sup of all z_i gives us the sup of y and x. Therefore A satisfies the slide condition.

(iv) By (i) it remains to show that $A \sqcup B$ is an independent set. Indeed, if x is above $a_1 \sqcup b_1$ and $a_2 \sqcup b_2$ it follows that $a_1 = a_2$ and $b_1 = b_2$ because both A and B are independent sets.

(v) Same proof as for (iv). ∎

Distributivity is essential for Part (i) of this lemma, as the example shown in Figure 4.5 demonstrates. There the pointwise sup of the schemes $\{a_1, a_2, \bot\}$ and $\{b_1, b_2, \bot\}$ does not satisfy the slide condition.

We plan to present a deeper investigation into the mathematics of schemes in a later paper, but mention that ideals generated by schemes form a complete lattice:

Theorem 4.1.10 *If D is a distributive domain then $(\mathcal{S}_D, \sqsubseteq^\flat)$ is a distributive complete lattice.* ∎

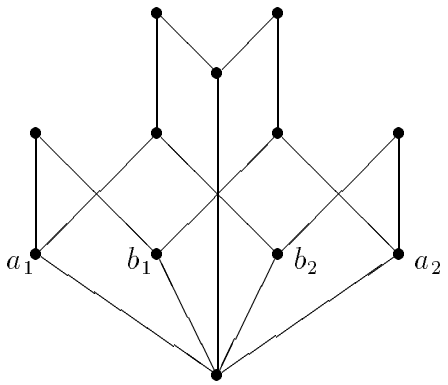


Figure 4.5: A domain where the join of two schemes cannot be calculated pointwise.

In the remainder of this section we shall work the generating co-chain of an ideal, that is, with schemes. It turns out that the ordering $\subseteq$ on strong ideals is replaced by the Egli-Milner ordering $\sqsubseteq^\sharp$ on schemes.

Theorem 4.1.11 *If A and B are schemes on a domain D then $A \sqsubseteq^\flat B$ if and only if $A \sqsubseteq^\sharp B$.*

Proof. "$\Longrightarrow$" Let x be an element of B and let y be maximal above x in D. Then $p_A(y) = p_A(p_B(y)) = p_A(x)$ and therefore $p_A(x)$ is maximal in $\downarrow A$ which means that it is contained in A.

"$\Longleftarrow$" If x is an element of A, let y be maximal in D above x. Since $x = p_A(y) \sqsupseteq p_A(p_B(y))$, the sup of x and $p_B(y)$ exists. $p_B(y)$ is maximal in $\downarrow B$ and because of $A \sqsubseteq^\sharp B$ it is above some $x' \in A$. The set $\{x, x'\}$ is bounded by $x \sqcup p_B(y)$ which is only possible if $x = x'$. Hence $x \sqsubseteq p_B(y)$ and $A \sqsubseteq^\flat B$. ∎

So far we have discussed the projection of individual elements ("records") into strong ideals. We shall now proceed to discuss the projection of relations, that is, finite co-chains. The obvious choice, namely, to apply the projection pointwise, has its particular applications. However, we might not get a co-chain as the image. Throwing away redundant information means in our case to keep only the maximal elements of the image.

Definition. Let D be a domain and A be a scheme in D.

(i) The function $\pi_A{:}\mathcal{C}_D \to \mathcal{C}_D$ is defined by $\pi_A(R) = \{x \in p_A(R) \mid x \text{ maximal in } p_A(R)\}$.

(ii) If $R \in \mathcal{C}_D$ is a subset of A, we shall call R an *instance* of A.

(iii) If $R \in \mathcal{C}_D$ is a subset of $\downarrow A$, we shall call it a *subinstance* of A.

Theorem 4.1.12 *Let A, B be schemes in a distributive domain D.*

(i) If R is an instance of A then $R \sqsubseteq^\flat A$ and $A \sqsubseteq^\sharp R$.

(ii) If R is an (sub-)instance of A and S is an (sub-)instance of B then $R \bowtie S$ and $R \sqcup^\sharp S$ (if it exists) are (sub-)instances of $A \sqcup B$ and $R \sqcup^\flat S$ is a subinstance of $A \sqcup B$.

(iii) If R is an instance of A then $\pi_B(R)$ is a subinstance of B.

(iv) If R is a finite co-chain in D then $p_A(R) \bowtie R = R$.

(v) If R is a finite co-chain in D then $\pi_A(R) \bowtie R \sqsupseteq^\sharp R$.

(vi) If R is a finite independent set in D then $\pi_A(R) \bowtie R \sqsupseteq^\natural R$.

Proof. Of these only (vi) is nontrivial. One half of the Egli-Milner ordering follows from (v). As for the "Hoare"-part we can copy the corresponding proof of Theorem 4.1.11. ▌

Let us recapitulate the development of our theory so far. We have exhibited a general structure which may take the place of attribute value sets in relational database, namely distributive Scott-domains. We proposed to model relations as finite co-chains in these domains. In Lemma 4.1.2 we have shown that relations form a distributive lattice under two natural orderings which correspond to the two intuitions one might have about a relation: One being that a relation gives information about a part of a set of real world objects, the other being that a relation approximates every element of a set of real world objects. We then proceeded to model the notions of scheme and projection and found (Theorem 4.1.10) that schemes form a distributive complete lattice. This says that the set of schemes is nearly a powerset and allows to

interpret intuitionistic logic in it. Along the way we have indicated the possibilities for fine tuning in this model: Using independent sets instead of co-chains or generalizing schemes to strong ideals. We shall now go on to test our theory in two fields, that of functional dependencies and that of universal relations.

4.1.4 Functional Dependencies

We start again with the familiar example of a relational database. Given some set of functional dependencies and given a set A of attribute names one can use Armstrong's Axioms in order to produce a set $A' \supseteq A$ which contains all attribute names depending on A. In our domain theoretic setting we may view this process as a function on the lattice of schemes, which is monotone, idempotent and increasing. These functions are the exact counterpart of projections as discussed in the previous section.

Definition. A *closure* on a domain D is a monotone function $f: D \to D$, such that $f \circ f = f \sqsupseteq id_D$.

Lemma 4.1.13 *Let D be a domain and f, f' be closures on D.*

(i) $f(x) = \sqcap\{y \mid f(y) = y \sqsupseteq x\}$.

(ii) $f \sqsupseteq f' \iff \mathsf{im}(f) \subseteq \mathsf{im}(f') \iff f \circ f' = f' \circ f = f$.

(iii) *f preserves all existing sup's.*

(iv) $\mathsf{im}(f)$ *is closed under nonempty inf's.* ∎

This is, of course, the exact dual of Lemma 4.1.5. Note that because of part (iii), closures are always continuous.

Given a function $f : D \to D$, we can define a relation $\widetilde{f} \subseteq D \times D$ by $\widetilde{f} = \{(x, y) \mid y \sqsubseteq f(x)\}$ and obtain an immediate connection with Armstrong's Axioms.

Theorem 4.1.14 *If f is a closure in $D \to D$, $\widetilde{f}$ satisfies*

(a) $\forall x, y \in D$ *if* $x \sqsupseteq y$ *then* $(x, y) \in \widetilde{f}$,

(b) *if $S \subseteq D$ is such that $\forall y \in S.(x, y) \in \widetilde{f}$ then $\sqcup S$ exists and $(x, \sqcup S) \in \widetilde{f}$, and*

(c) $\forall x, y, z \in D.(x, y) \in \widetilde{f}$ *and* $(y, z) \in \widetilde{f} \Longrightarrow (x, z) \in \widetilde{f}$.

When D *is finite* (b) *may be replaced by*

(b′) *for* $x, y, w \in D$ *if* $(x, y) \in \widetilde{f}$ *and* $x \sqcup w$ *exists then* $w \sqcup y$ *exists, and* $(w \sqcup x, w \sqcup y) \in \widetilde{f}$.

Conversely, suppose $\widetilde{f} \subseteq D \times D$ *satisfies* (a), (b) *or* (b') *as appropriate, and* (c) *above, and define* $f : D \to D$ *by* $f(x) = \sqcup\{y \mid (x, y) \in \widetilde{f}\}$. *Then* f *is a closure.*

From which (a), (b′) and (c) are immediately seen to be generalizations of Armstrong's Axioms. Before discussing the connection, we should prove this result. (a) follows immediately from the definition of a closure since if $y \sqsubseteq x$, then $y \sqsubseteq f(x)$ and $(x, y) \in \widetilde{f}$. (b) is also immediate because $f(x)$ must be a bound for S, therefore $\sqcup S$ exists and $\sqcup S \sqsubseteq f(x)$. To show (c), if $(x, y) \in \widetilde{f}$ then $y \sqsubseteq f(x)$ and by monotonicity and idempotence $f(y) \sqsubseteq f(x)$. The conditions also imply $z \sqsubseteq f(y)$. Combining these last two inequalities we have $z \sqsubseteq f(x)$, i.e. $(x, z) \in \widetilde{f}$. Conversely, we first note that condition (b) implies that $\sqcup\{y \mid (x, y) \in \widetilde{f}\}$ exists and f is well defined. If $x_1 \sqsubseteq x_2$ and $(x_1, y) \in \widetilde{f}$ then $(x_2, y) \in \widetilde{f}$ by (a) and (c) so that $\{y \mid (x_1, y) \in \widetilde{f}\} \subseteq \{y \mid (x_2, y) \in \widetilde{f}\}$, and hence $f(x_1) \sqsubseteq f(x_2)$ guaranteeing monotonicity. By (a) $(x, x) \in \widetilde{f}$, so $f(x) \sqsupseteq x$. Finally, by (b) $(x, \sqcup\{y \mid (x, y) \in \widetilde{f}\}) \in \widetilde{f}$, and so $(x, f(x)) \in \widetilde{f}$; similarly $(f(x), f(f(x))) \in f$. Using (c), $(x, f(f(x)) \in \widetilde{f}$ and so $f(f(x)) \sqsubseteq f(x)$. But we have just shown that f is increasing. Hence $f(f(x)) = f(x)$.

Suppose (a), (b), (c) hold and that $(x, y) \in \widetilde{f}$. For any $w \in D$, $(w \sqcup x, w) \in \widetilde{f}$ and $(w \sqcup x, x) \in \widetilde{f}$ by (a), and by (c) $(w \sqcup x, y) \in \widetilde{f}$. Therefore, by (b) $(w \sqcup x, w \sqcup y) \in \widetilde{f}$. Conversely, assume D finite. First note that, by putting $w = x$ in (b′) we have $x \sqcup y$ exists. Suppose $\forall y \in S.(x, y) \in \widetilde{f}$. If S has just two members, y_1, y_2 then $(x, x \sqcup y_1) \in \widetilde{f}$ by (b′) and $(x, x \sqcup y_1 \sqcup y_2) \in \widetilde{f}$ by (c), therefore $y_1 \sqcup y_2$ exists. Using (c) and (a) we get $(x, y_1 \sqcup y_2) \in \widetilde{f}$, i.e. $(x, \sqcup S) \in \widetilde{f}$. By induction, we can repeat this argument to derive (b) for any finite S. ∎

Armstrong's Axioms are precisely (a), (b′), (c) when applied to the lattice of subsets of the set of attribute names. Related characterizations of Armstrong's Axioms in a lattice-theoretic setting have been given by [JS87]. It is also interesting that in Scott's information systems [Sco82a] functions on domains are defined by a similar device of taking approximating relations.

We now connect this abstract notion of a functional dependency with our earlier semantics in which sets of attribute names are represented by schemes. A relation satisfies a functional dependency $A \to B$ if any two tuples that agree on the attribute names A agree on the attribute names B. Another way of stating this is to follow [CKS86] and say that a relation r satisfies $A \to B$ if the partition on r induced by A (i.e. the equivalence relation induced on the tuples by equality on A) is finer than the partition induced by B. In the standard theory there are no null values allowed in places corresponding to attributes from $A \cup B$. We keep this strong interpretation of satisfaction.

Definition. Let A, B be schemes in a domain D. A relation $R \in \mathcal{C}_D$ *satisfies* the functional dependency $A \to B$ if $R \sqsupseteq^\sharp A$ and $R \sqsupseteq^\sharp B$ and if $p_A(x) = p_A(y)$ implies $p_B(x) = p_B(y)$ for all $x, y \in R$.

Theorem 4.1.15 *For relations in distributive domains Armstrong's Axioms are consistent and complete.*

Proof. Given a relation R in a distributive domain D and given a scheme $A \sqsubseteq^\sharp R$ it is clear that R satisfies $A \to A$. If S is a collection of schemes and R satisfies $A \to B$ for all $B \in S$ and some scheme A, then S is bounded by R in the Smyth ordering. We claim that the sup of S is also below R: If x is an element of R and B is a scheme contained in S then x is above some element x_B of B. Therefore x bounds the set $X = \{x_B \mid B \in S\}$. The sup of X is an element of $\bigsqcup S$ by Lemma 4.1.9, (v) and is below x. This proves $\bigsqcup S \sqsubseteq^\sharp R$. Assume, then, that $p_A(x) = p_A(y)$. By assumption we know that $p_B(x) = p_B(y)$ for all $B \in S$. Hence we also have $p_{\sqcup S}(x) = \bigsqcup_{B \in S} p_B(x) = \bigsqcup_{B \in S} p_B(y) = p_{\sqcup S}(y)$, which proves $A \to \bigsqcup S$.

It is clear that transitivity holds. This proves that Armstrong's Axioms are correct with respect to our definition of satisfaction.

Completeness is trivial because we have more models available than in the relational case. ∎

It is an immediate consequence of the preceding theorem and Theorem 4.1.14 that a relation $R \in \mathcal{C}_D$ induces a closure f on the lattice of schemes with the property $f(A) \sqsupseteq B$ if and only if R satisfies $A \to B$.

Our definition of satisfaction of a dependency requires that the relation under consideration contains no partial information. If a relation does contain partial elements, a different concept is called for.

Definition. Let A, B be schemes in a domain D. A relation $R \in \mathcal{C}_D$ is *consistent* with the functional dependency $A \to B$ if there is a relation $R' \sqsupseteq^\natural R$ which satisfies $A \to B$.

This is natural enough. However, in a practical instance consistency may be hard to check. We therefore introduce a weaker notion of consistency with a more operational flavor. Given a scheme (or any independent set) A and a relation R then A induces a partial equivalence relation $\sim_A$ on R: $x \sim_A y$ if there is $a \in A$ such that $a \sqsubseteq x, y$. We may say that $\sim_A$ identifies those elements in R which contain the same total information in their A-part. By R/A we denote the set of equivalence classes of $\sim_A$.

Now assume that $A \to B$ is a dependency where $A \sqsubseteq^\natural B$ and that R is some relation. The result of restricting R to the 'columns' of B is expressed by $\pi_B(R)$. Wherever two elements of $\pi_B(R)$ contain the same total information in their A-part, consistency with $A \to B$ implies that their B-part can be updated to a common (total) value. This amounts to saying that each equivalence class in $\pi_B(R)/A$ has an upper bound in D. Let us denote the resulting set of suprema by $(\pi_B(R)/A)^\sqcup$. Formally we define

Definition. For $A \sqsubseteq^\natural B$ schemes and R a relation in a domain D, we say that R is *weakly consistent* with the dependency $A \to B$ if $(\pi_B(R)/A)^\sqcup$ exists.

Remember the example of physical constants, given in Section 2. Certainly we expect that the name of a constant will imply its value, although the exact numbers will never be known. To say that our database is weakly consistent with the implication *name of constant* $\to$ [*lower bound, upper bound*] amounts to the requirement that the entries for the same constant report intervals with at least one common point.

The reader will have noticed that weak consistency makes no requirement about those elements of the relation R which contain partial information in their A-part. The philosophy here is that any finite set of elements with partial information over some scheme A can be updated in such a way that its elements are pairwise different

in their A-part. We may call a domain in which this is always the case *rich* and obtain the following immediate characterization.

Lemma 4.1.16 *A domain D is rich if and only if for each $x \in D$ the denotation $[\![x]\!]$ of x cannot be covered by a finite set of denotations $[\![y_i]\!]$ with all $y_i \not\sqsubseteq x$.*

With this we can formulate

Theorem 4.1.17 *Let $A \sqsubseteq^\natural B$ be schemes in a domain D. Let R be a relation in D. If R is consistent with $A \to B$ then R is weakly consistent with $A \to B$. If $\downarrow A$ is rich and D distributive then the converse also holds. Moreover, if $R' \sqsupseteq^\natural R$ and R' satisfies $A \to B$ then $R' \sqsupseteq^\flat (\pi_B(R)/A)^\sqcup$.*

Proof. Suppose $R' \sqsupseteq^\natural R$ and R' satisfies (A, B), then $\pi_B(R') \sqsupseteq^\flat \pi_B(R)$ and the members of $(\pi_B(R')/A)$ are singleton sets. Thus any class in $(\pi_B(R)/A)$ is bounded above by one of these singletons, and $(\pi_B(R)/A)^\sqcup$ exists. This also establishes the second part of the theorem. Conversely, if $(\pi_B(R)/A)^\sqcup$ exists, we have, for each $a \in A$, the element $b_a = \sqcup\{r \mid r \in \pi_B(R) \text{ and } r \sqsupseteq a\} \in \downarrow B$. Now for each $r \in R \cap \uparrow A$ form the point $r' = b_A \sqcup r$ with $a \in A$ being the unique element of A below r. (This is where the slide condition comes in.) The set R' of these points certainly satisfies $A \to B$. But we also have to update the other elements of R which contain partial information in their A-part. We use the assumption that $\downarrow A$ is rich for this. The set $p_A(R \setminus \uparrow A)$ is a finite poset contained in $\downarrow A$. Because $\downarrow A$ is rich, we can find elements $\bar{r} \in [\![p]\!]_A(r)_A$ such that $r_1 \neq r_2$ implies $\bar{r_1} \neq \bar{r_2}$ and also $\bar{r} \neq p_A(r')$ for all $r' \in R'$. (Given $r \in p_A(R \setminus \uparrow A)$, choose $\bar{r} \in [\![r]\!]_A \setminus (\bigcup\{\uparrow s \mid s \not\sqsubseteq r, s \in p_A(R \setminus \uparrow A)\} \cup \bigcup\{\uparrow r' \mid r' \in R'\})$.) Finally let $\tilde{r}$ be the unique element of B above $\bar{r}$ for each $r \in p_A(R \setminus \uparrow A)$. The set $\tilde{R}$ of all these elements satisfies $A \to B$ and so does $R' \cup \tilde{R}$. ∎

Dependencies are often divided [Ull82a, Mai83] into two classes: those like functional dependencies that generate equality constraints, and those that generate new tuples. The "chase" is a procedure that performs all possible inferences on a set R to produce a new set R' where $R' \sqsupseteq^\natural R$. In fact, we can also use functional dependencies in the same way. The co-chain $(\pi_B(R)/A)^\sqcup$ describes the inferences that can be made, given that R is consistent with $A \to B$. In fact the co-chain

$$T = ((\pi_B(R)/A)^\sqcup \bowtie R) \sqcup^\flat R \tag{4.1}$$

is the least (in $\sqsubseteq^\flat$) set that contains all these inferences. Note that T is the outer join of $(\pi_B(R)/A)^\sqcup$ and R and that $R \sqsubseteq^\natural T$.

4.1.5 Universal Relations

Without involving ourselves in a discussion of the usefulness or practicality of the Universal Relation Assumption [Ull82b, Ull83, Ken83, AP82], we now investigate a general characterization of universal relations that shows how the general form of their implementation can be derived from their abstract properties. Behaviorally, a universal relation can be thought of as a simple query language, or transducer, in which the possible queries, or inputs, are sets of column names and the output from the input of a given set of column names is a relation defined on those names. More precisely, we can think of a universal relation as a function $\rho : \mathcal{S}_D \to \mathcal{C}_D$ with the property that $\rho(S)$ is an instance of S, i.e. $\rho(S) \subseteq S$.

In a survey [MRW86] of the various definitions of universal relations Maier *et al* give a condition, "containment", that all reasonable definitions satisfy. The condition, which is also noted in [Sci80], is that if A, B are schemes with $A \sqsubseteq^\natural B$, then $\pi_A(\rho(B)) \subseteq \rho(A)$. This is equivalent to requiring that ρ be monotone as a function from schemes under the natural ordering to the finite cochains $\mathcal{C}_D$ under the Smyth ordering, i.e. if $A \sqsubseteq^\natural B$ then $\rho(A) \sqsubseteq^\sharp \rho(B)$. There are various ways of obtaining such a function. A particularly simple method is given the *total projection* of an arbitrary subset T of D onto the schemes of D:

$$\rho(A) = \pi_A(T \cap \uparrow A) \tag{4.2}$$

(the expression $\pi_S(T \cap \uparrow A)$ is called the total projection of T onto the scheme A.) A more general method is obtained by projecting onto A those subsets T of some collection $\mathbf{T}$ of finite subsets of D that are contained in the upward closure of A:

$$\rho(A) = \cup\{\pi_A(T) \mid T \in \mathbf{T} \text{ and } T \subseteq \uparrow A\} \tag{4.3}$$

Most of the various definitions of universal relations given in [MRW86] appear to be expressible in this form. By using a result that is readily proved from theorem 4.1.12,

Lemma 4.1.18 *If A is a scheme, and S_1, S_2 are co-chains in D with $S_1 \sqsupseteq^\sharp A$ and $S_2 \sqsupseteq^\sharp A$ then $\pi_A(S_1 \sqcap^\sharp S_2) = \pi_A(S_1) \cup \pi_A(S_2)$*▮

we can write (4.3) as $\rho(A) = \pi_A(\sqcap^\sharp\{T \in \mathbf{T} \mid T \sqsubseteq^\sharp A\})$. We shall call a universal relation that can be described in this fashion a *closure universal relation* (because this last equation is closely related to the definition of a closure in $(\mathcal{C}_D, \sqsubseteq^\sharp)$. By taking $\mathbf{T}$ as a collection of singleton sets, equation (4.2) can be seen as a special case of (4.3). An example of a universal relation satisfying (4.2) is the *Universal Instance Assumption*, which says that $\rho(A) = \pi_A(I)$ where I is a subset of the maximal elements of D.

Theorem 4.1.19 *A universal relation defined by the universal instance assumption is a closure relation.*

The proof follows immediately from the observation that I, being a finite set of maximal elements, is contained in $\uparrow A$ for any scheme A ▮

Another reason for believing that closure universal relations are an appropriate class to consider is given by the following result.

Theorem 4.1.20 *In the relational domain $D = \mathcal{L} \to \mathcal{V}_\perp$, any universal relation satisfying the containment condition is a closure universal relation.*

Proof. If ρ is the given universal relation define

$$\rho'(A) = \pi_A(\sqcap^\sharp\{\rho(B) \mid B \in \mathcal{S}_D \text{ and } \rho(B) \subseteq \uparrow A\}).$$

ρ' is then a closure universal relation, and we need to show that, for any scheme A, $\rho(A) = \rho'(A)$. Because we are dealing with the relational domain, if B is a scheme such that $\phi \neq \rho(B) \subseteq \uparrow A$ then $B \sqsupseteq^\sharp A$. Using this fact and the containment condition, whenever $\rho(B) \subseteq \uparrow A$, we must have $\pi_A(\rho(B)) \subseteq \rho(A)$. Hence $\rho(A) = \rho'(A)$ for any scheme A. ▮

It is not true that any universal relation satisfying the containment condition can be cast in the form of a closure relation. Consider, for example, the domain in figure 4.6, in which the schemes are $A_1 = \{\perp\}$, $A_2 = \{a_1, a_2, d\}$, $A_3 = \{b_1, b_2, d\}$, $A_4 = \{a_1, a_2, e_1, e_2\}$, $A_5 = \{b_1, b_2, e_1, e_2\}$, and $A_6 = \{c_1, c_2, c_3, c_4, e_1, e_2\}$. Now consider a universal relation ρ such that $\rho(A_1) = \{\perp\}$, $\rho(A_2) = \rho(A_3) = \{d\}$, $\rho(A_4) = \{e_1, e_2\}$,

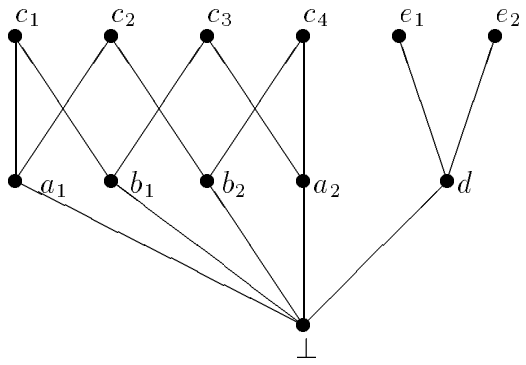


Figure 4.6: A universal relation not extending to a closure

$\rho(A_5) = \{e_1\}$, and $\rho(A_6) = \{e_1\}$, which satisfies the containment condition. If ρ is a closure universal relation, then $\mathbf{T}$ (as used in equation 4.3) must contain a set T which contains e_2 such that $T \subseteq \uparrow A_4$, but T cannot be contained in $\uparrow A_5$ because e_2 is not a member of $\rho(A_5)$. Therefore T must contain a_1 or a_2. But if this happens then $\rho(A_4)$ must also contain a_1 or a_2, which contradicts the definition of ρ.

A more sophisticated example of a universal relation definition arises from the *F-weak instance universal relation* [MRW86]. Suppose we are given a set of schemes $\{R_1, R_2, \ldots, R_n\}$ in D and instances $r_i \subseteq R_i$, $i \in 1 \ldots n$. Suppose we are also given a set F of functional dependencies and that $\sqcup^\flat\{r_i \mid i \in 1..n\}$ is consistent with F. Consider the universal relation defined by

$$\rho(C) = \cap\{\pi_C(S') \mid S' \sqsupseteq^\flat r_i, i \in 1 \ldots n, S' \in \mathcal{C}_D, \text{ and } S' \text{ satisfies } F\} \tag{4.4}$$

which, for each scheme C defines an instance of C. Let us assume, for simplicity, that $F+$ is generated by the single non-trivial dependency (A, B) where $B \sqsupseteq^\natural A$. From (4.1) of the previous section, we can write $\rho(C)$ as

$$\rho(C) = \tau_C(((\pi_B(S)/A)^\sqcup \bowtie S) \sqcup^\flat S) \tag{4.5}$$

where $S = \sqcup^\flat\{r_i \mid i \in 1..n\}$ and $\tau_C(T)$ is the total projection of a set T onto C, $\tau_C(T) = \pi_C(T \cap \uparrow C)$.

By manipulation of (4.5) we can now write it in a form consistent with the general form for closure universal relations given in (4.3). First observe that if S_1, S_2 are co-chains in D, then $\tau_C(S_1 \sqcup^\flat S_2) = \tau_C(S_1) \cup \tau_C(S_2)$. Therefore we can rewrite (4.5)

as

$$\rho(C) = \tau_C((\pi_B(S)/A)^{\sqcup} \bowtie S) \quad \cup \quad \cup\tau_C(\{r_i \mid R_i \sqsupseteq^{\natural} C\}) \tag{4.6}$$

Now consider the set $Q = (\pi_B(S)/A)^{\sqcup}$ which, by the definition of S is $(\pi_B(\sqcup^{\flat}\{r_i \mid i \in 1..n\})/A)^{\sqcup}$, By the distributivity of $(\mathcal{C}_D, \sqsubseteq^{\flat})$, $Q = (\sqcup^{\flat}\{\pi_B(r_i) \mid i \in 1..n\}/A)^{\sqcup}$. A point in Q is the least upper bound of some set of points, each chosen from some $\pi_B(r_i)$ where $R_i \sqsupseteq^{\natural} A$. Let I be the set of indices of all such schemes, $I = \{i \mid i \in 1..n \text{ and } R_i \sqsupseteq^{\natural} A\}$. We can then express Q as

$$Q = \sqcup^{\flat}\{\bowtie_{i\in I'} \pi_B(r_i) \mid I \subseteq I' \subseteq 1..n\} \tag{4.7}$$

The term $\tau_C((\pi_B(S)/A)^{\sqcup} \bowtie S)$, which is the left-hand component of (4.6) can therefore be written as the union of total projections of terms of the form

$$r_{i_0} \bowtie \pi_B(r_{i_1}) \bowtie \pi_B(r_{i_2}) \bowtie \cdots \bowtie \pi_B(r_{i_k}) \tag{4.8}$$

where $R_{i_j} \sqsupseteq^{\natural} A$ for $j \in 1..k$. The right hand component can, trivially, be written in this form too.

We have therefore succeeded in reducing the universal relation definition given in equation (4.5) to the projection of the union of a set of joins. More importantly, (4.5) is an example of an "FD-join" expression. A theorem of Maier *et. al.* and Chan [MRW86, Cha84] shows that the F-weak instance universal relation (4.5) can be computed as the union of FD-joins. Their proofs work by considering the properties of specific algorithms, whereas by considering the general properties of the spaces involved we have been able to produce a reasonably concise algebraic derivation. It should be noted that the proof outlined here is incomplete. We need to close this off under all functional dependencies; but this presents no difficulties.

4.1.6 Higher Order Relations and Other Models

One of the contentions of this paper is that much of our theory of relational databases is independent of the detailed structure of the relational model and depends only on some rather general properties of the spaces out of which we can construct such a model. It should be stressed that we have based the preceding analysis only on the assumption that the underlying space was a domain. Nowhere did we assume that we

were dealing with relations, although we frequently appealed to the first-normal-form relations for examples.

In trying to generalize various operations, we had no problem with the natural join, but in order to make projection generalize smoothly when dealing with functional dependencies and universal relations, we had to characterize first independent sets and then schemes. We shall therefore be particularly interested in identifying schemes in these other models. If we can do that, we can be sure that the basic ideas of functional dependencies, universal relations etc., generalize properly.

Typed first-normal-form relations

We have seen that the domain $\mathcal{L} \to \mathcal{V}_\perp$ of flat records is a domain of the relational model in the preceding sections, and it deserves little extra comment here. As we have noted earlier, this domain is a special case of a product domain.

Given a function F from a set of labels $\mathcal{L}$ to a set of sets $\mathbf{S}$, a *labeled product* $\prod_{l\in\mathcal{L}} F(l)$ is the set of functions $f : \mathcal{L} \to \bigcup \mathbf{S}$ such that for all $l \in \mathcal{L}$, $f(l) \in F(l)$. If $\mathbf{S}$ is a set of domains, then $\prod_{l\in\mathcal{L}} F(l)$ is also a domain, a *domain of labeled products*, under the componentwise ordering, i.e. $f_1 \sqsubseteq f_2$ iff $f_1(l) \sqsubseteq f_2(l)$ for all $l \in \mathcal{L}$. Furthermore, a scheme in a domain of labeled products is a product of schemes, i.e. it is easy to show that:

Lemma 4.1.21 *The set of schemes in* $\prod_{l\in\mathcal{L}} F(l)$ *is the set of labeled products of the form* $\prod_{l\in\mathcal{L}} \phi(l)$*, where* ϕ *is any function from* $\mathcal{L}$ *to* $\bigcup\{\mathcal{S}_S \mid S \in \mathbf{S}\}$ *such that* $\phi(l) \in \mathcal{S}_{F(l)}$ $\square$

Since the domain of flat records $\mathcal{L} \to \mathcal{V}_\perp$ is the domain of labeled product $\prod_{l\in\mathcal{L}} \mathcal{V}_\perp$, where we take $\mathcal{V}_\perp$ as the constant function on $\mathcal{L}$, the above result shows that a scheme in this domain is a product $\prod_{l\in\mathcal{L}} \phi(l)$ where ϕ is any function from $\mathcal{L}$ to $\mathcal{S}_{\mathcal{V}_\perp}$. Since $\mathcal{S}_{\mathcal{V}_\perp} = \{\{\perp\}, \mathcal{V}\}$, each such function $\phi : \mathcal{L} \to \mathcal{S}_{\mathcal{V}_\perp}$ is identified by the subset $L = \{l \mid \phi(l) \neq \{\perp\}\}$ of $\mathcal{L}$ and the corresponding scheme is isomorphic to the set of total functions from L to $\mathcal{V}$. Therefore the set of all schemes in this domain is isomorphic to the set of spaces of total functions $L \to \mathcal{V}, L \subseteq \mathcal{L}$ and is identified by the set of all subsets of $\mathcal{L}$. However, restrictions on these function spaces do not produce schemes, for example

$$\{\{\mathit{Name}{\Rightarrow}s;\ \mathit{Age}{\Rightarrow}i;\mathit{Shoe\text{-}size}{\Rightarrow}i\} \mid s, i \in \mathcal{V}\}$$

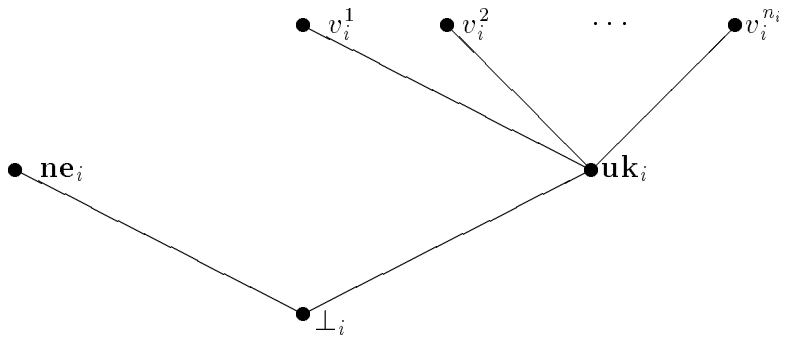


Figure 4.7: A domain V_i with null values

is not a scheme if $\mathcal{V}$ has more than one element.

If we require that the columns of a relation are "typed", we are given a set of flat domains $\mathbf{V}$ and an assignment of domains in $\mathbf{V}$ to labels in $\mathcal{L}$, i.e. a function $\Phi : \mathcal{L} \to \mathbf{V}$ ($\Phi(l)$ are called "domains" in database parlance). Then the domain of typed flat records D_Φ is the domain of labeled products $D_\Phi = \prod_{l \in \mathcal{L}} \Phi(l)$. A scheme in this domain is a product $\prod_{l \in \mathcal{L}} \phi(l)$ where ϕ is any function $\phi : \mathcal{L} \to \bigcup_{l \in \mathcal{L}} \mathcal{S}_{\Phi(l)}$ such that for all $l \in \mathcal{L}, \phi(l) \in \mathcal{S}_{\Phi(l)}$. Since each $\Phi(l)$ is a flat domain, $\mathcal{S}_{\Phi(l)}$ is either $\{\perp\}$ or the set of all maximal elements in $\Phi(l)$. Thus the set of all schemes in this domain is isomorphic to the set of all product domains of the form $\prod_{l \in L} \Phi(l), L \subseteq \mathcal{L}$. If each $\Phi(l)$ is represented by a type τ_l, then for a finite $L = \{l_1, \ldots, l_n\}$, a scheme $\prod_{l \in L} \Phi(l)$ is represented by the type $\{l_1 : \tau_{l_1}, \ldots, l_n : \tau_{l_n}\}$.

Null values

Our first "non-flat" example arises from the introduction of null values, which give rise to an ordering on tuples. The framework that we have developed here should allow us to ascribe semantics to the various kinds of null values and to investigate how the mathematical properties generalize.

Combining work in [Bis81, Lie82, Sci79] Zaniolo [Zan84] introduced an ordered space V_i with null values shown in figure 4.7.

$\perp_i$ is interpreted as *no information*; $\mathbf{ne}_i$ means *non-existent*, or *wrong*; $\mathbf{uk}_i$ means *unknown* – a value exists (other than $\mathbf{ne}_i$), but it is not yet known.

Tree-like domains such as this are domains with a particularly simple structure. In fact we can call a domain D a *tree* if, whenever $x, y \in D$ and $x \sqcup y$ exists then $x \sqsubseteq y$ or $y \sqsubseteq x$. A *section* of a tree D is a set S such that any path in D from the root ($\perp$) to a leaf contains exactly one member of S. The following results characterize independent sets and schemes in a tree.

Lemma 4.1.22 *D is a tree iff $\mathcal{C}_D = \mathcal{I}_D$ (i.e. the co-chains are the independent sets)*$\square$

Lemma 4.1.23 *S is a section of D iff it is a scheme for D*$\square$

For example, the schemes for V_i in figure 4.7 are $\{\perp_i\}$, $\{\mathbf{ne}_i, \mathbf{uk}_i\}$ and $\{\mathbf{ne}_i, v_i^1, \ldots, v_i^{n_i}\}$.

We can use this to define domains of typed records with null values by simply replacing flat domains with tree-like domains in the previous development. Given a set $\mathbf{T}$ of tree-like domains and a type assignment $\Phi : \mathcal{L} \to \mathbf{T}$, a domain of typed records D_Φ is the domain $D_\Phi = \prod_{l \in \mathcal{L}} \Phi(l)$ of labeled products. A scheme in this domain is a product $\prod_{l \in \mathcal{L}} \phi(l)$ where ϕ is a function $\phi : \mathcal{L} \to \bigcup_{l \in \mathcal{L}} \mathcal{S}_{\Phi(l)}$ such that for all $l \in \mathcal{L}, \phi(l) \in \mathcal{S}_{\Phi(l)}$. Unlike the case of typed flat records, $\mathcal{S}_{\Phi(l)}$ may contain schemes which are neither $\{\perp\}$ nor the set of maximal elements in $\Phi(l)$ and the set of schemes in this domain is no longer isomorphic to the set of products of the form $\prod_{l \in L} \Phi(l), L \in \mathcal{L}$. In order to represent schemes in this domain in a type system, we need to define "scheme-types" to represent schemes $\mathcal{S}_\mathcal{T}, \mathcal{T} \in \mathbf{T}$. We will show an example of such definition in the next section.

This allows us to establish that the whole apparatus of functional dependencies, universal relations, etc. works smoothly in the domain of relations with null values, i.e. relations defined over tree-like domains.

To take an example, in a payroll database, the values $\{v_i^1, \ldots, v_i^{n_i}\}$ could be the state tax rate with $\mathbf{ne}_i$ being used when such a tax was inappropriate, e.g.,. for overseas employees. There is then a functional dependency ADDRESS$\to$ $\{\mathbf{ne}_i, v_i^1, \ldots, v_i^{n_i}\}$ and an inferred dependency ADDRESS$\to \{\mathbf{ne}_i, \mathbf{uk}_i\}$. The investigation of such dependencies may be useful when attempting to do database design on databases with exceptional values such as those investigated in [Bor85].

Record structures

In programming languages such as Pascal, record types are constructed both by giving a labelled set of fields and by giving a **case** statement or discriminated union. Moreover record types can be components of other record types, and we can carry this construction to any depth. The domains of such records allows us a further generalization of the domains we have just considered. These domains can be also regarded as the domain of feature structures which are used to represent linguistic information [Shi85].

In the previous sections, we have constructed domains and their schemes of first-normal form relations with null values by using labeled product constructors. By simply iterating this construction process, we can construct domains and schemes of general record structures without discriminated union. Domains corresponding to discriminated union can be constructed by labeled sum constructors.

Given a function F from a set of labels $\mathcal{L}$ to a set of sets $\mathbf{S}$, a *labeled sum* $\sum_{l\in\mathcal{L}} F(l)$ is the set of pairs $\{< l, v > |\ v \in F(l)\}$. If $\mathbf{S}$ is a set of domains, we define the *domain of labeled sums* $\sum^{\perp}_{l\in L} F(l)$ to be the set $\{< l, v > |\ v \in F(l)\} \cup \{\perp\}$. This is indeed a domain under the ordering defined as $x \sqsubseteq y$ if and only if either $x = \perp$ or $x =< l, v >$ and $y =< l, v' >$ and $v \sqsubseteq v'$. Corresponding to the result for labelled products (lemma 4.1.21), a scheme in a domain of labeled sums is a labeled sum of schemes, i.e. it is easy to show that:

Lemma 4.1.24 *A scheme in $\sum^{\perp}_{l\in\mathcal{L}} F(l)$ is either the singleton set $\{\perp\}$ or a labeled sum $\sum_{l\in\mathcal{L}} S(l)$, where S is any function from $\mathcal{L}$ to $\bigcup\{\mathcal{S}_S \mid S \in \mathbf{S}\}$ such that $S(l) \in \mathcal{S}_{F(l)}$.* □

Starting with given primitive domains such as the flat domain of integers, we can now construct domains of record structures by applying product and sum constructions. We can then identify the set of schemes in those domains. Suppose we are given primitive domains $\mathcal{B}_1, \ldots, \mathcal{B}_n$ with corresponding sets of schemes $\mathcal{S}_{\mathcal{B}_1}, \ldots, \mathcal{S}_{\mathcal{B}_n}$. Then we can define the family **Dom** of domains with associated sets of schemes generated by $\mathcal{B}_i$'s as:

(1) $\mathcal{B}_i \in \mathbf{Dom}$. The associated set of scheme is $\mathcal{S}_{\mathcal{B}_i}$,

(2) If $\mathbf{D} \subseteq \mathbf{Dom}$ with associated sets of schemes $\mathcal{S}_D, D \in \mathbf{D}$, then for any function

$\Phi : \mathcal{L} \to \mathbf{D}$, $D_\Phi = \prod_{l \in \mathcal{L}} \Phi(l) \in \mathbf{Dom}$ with the set of schemes
$\mathcal{S}_{D_\Phi} = \{\prod_{l \in \mathcal{L}} \phi(l) \mid \exists \phi : \mathcal{L} \to \bigcup_{D \in \mathbf{D}} \mathcal{S}_D.\ \forall l \in \mathcal{L}.\ \phi(l) \in \mathcal{S}_{\Phi(l)}\}$.

(3) If $\mathbf{D} \subseteq \mathbf{Dom}$ with corresponding sets of schemes $\mathcal{S}_D, D \in \mathbf{D}$, then for any function $\Theta : \mathcal{L} \to \mathbf{D}$, $D_\Theta = \sum^{\perp}_{l \in \mathcal{L}} \Theta(l) \in \mathbf{Dom}$ with the set of schemes
$\mathcal{S}_{D_\Theta} = \{\sum_{l \in \mathcal{L}} \theta(l) \mid \exists \theta : \mathcal{L} \to \bigcup_{D \in \mathbf{D}} \mathcal{S}_D.\ \forall l \in \mathcal{L}.\ \theta(l) \in \mathcal{S}_{\Theta(l)}\}$.

Dom corresponds to domains of record structures generated from primitive values in $\mathcal{B}_1, \ldots, \mathcal{B}_n$.

We give an example of concrete representation of domains of record structures. By the analogy of a type system of a programming language, we call expression representing domains *types* and define the membership relation between records and domains as typing rules. We will comment more on the relationship between domains and types in a programming language later. We start with types. A *type expression* is one that can be constructed by the following rules:

(1) $B_1, \ldots, B_n$, the (names of) base types such as *int, bool, string* etc. are type expressions.

(2) If $\tau_1, \tau_2, \ldots, \tau_n$ are type expressions then $\{l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n\}$ is a type expression.

(3) If $\tau_1, \tau_2, \ldots, \tau_n$ are type expressions then $[l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n]$ is a type expression.

The notation $[l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n]$ indicates a discriminated union. An example of such a type expression is

$$\tau_1 = \{\mathit{Name}{:}\mathit{string};\ \mathit{Age}{:}\mathit{int};\ \mathit{Status}{:}[\mathit{Employee}{:}\{\mathit{Office}{:}\mathit{string};\ \mathit{Extension}{:}\mathit{int}\};\ \mathit{Consultant}{:}\{\mathit{Address}{:}\mathit{string};\ \mathit{Telephone}{:}\mathit{int}\}]\}$$

The syntax for records is similarly defined:

(1) For each base type B, we assume that we are given the corresponding primitive domain $\mathcal{B}$ such as the flat domain $N_\perp$ of integers. Then elements in $\mathcal{B}$ are records. $\perp_{\mathcal{B}}$ represents a null value in $\mathcal{B}$.

(2) If $r_1, r_2, \ldots, r_n$ are records then $\{l_1 \Longrightarrow r_1; l_2 \Longrightarrow r_2; \ldots; l_n \Longrightarrow r_n\}$ is a record.

(3) If r is a record, $[l \Longrightarrow r]$ is a record.

(4) If τ is a discriminated union type then $\perp_\tau$ is a record.

The following is an example of record:

$r_1 = \{Name \Rightarrow 'J.\ Doe'; \ Age \Rightarrow 21; \ Status \Rightarrow [Employee \Rightarrow \{Office \Rightarrow G7; \ Extension \Rightarrow 5556\}]\}$

Moreover, we regard the record r_1 having the type τ_1. Formally, a record r has type τ if one of the following conditions hold:

(1) $r \in \mathcal{B}$ and τ is the base type B corresponding to $\mathcal{B}$.

(2) $r = \{l_1 \Longrightarrow r_1; l_2 \Longrightarrow r_2; \ldots; l_n \Longrightarrow r_n\}$, $\tau = \{l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n\}$, and r_i has type τ_i for $1 \leq i \leq n$.

(3) $r = [l_i \Longrightarrow r_i]$, $\tau = [l_1 : \tau_1; l_2 : \tau_2; \ldots; l_m : \tau_m]$, $i \leq m$, and r_i has type τ_i.

(4) $r = \perp_\tau, \tau = [l_1 : \tau_1, \ldots, l_n : \tau_n]$

Records are ordered by the following rules:

(1) $v \sqsubseteq v'$ if $v, v' \in \mathcal{B}$ and v, v' are ordered in $\mathcal{B}$.

(2) $\{l_1 \Longrightarrow r_1; l_2 \Longrightarrow r_2; \ldots; l_n \Longrightarrow r_n\} \sqsubseteq \{l_1 \Longrightarrow r'_1; l_2 \Longrightarrow r'_2; \ldots; l_n \Longrightarrow r'_n\}$ if $r_i \sqsubseteq r'_i$ for all $1 \leq i \leq n$.

(3) $[l \Longrightarrow r] \sqsubseteq [l \Longrightarrow r']$ if $r \sqsubseteq r'$.

(4) $\perp_\tau \sqsubseteq \perp_\tau$ for any discriminated union type τ.

(5) $\perp_{[l_1:\tau_1,\ldots,l_n:\tau_n]} \sqsubseteq [l_i \Longrightarrow r]$ if $1 \leq i \leq n$ and r has the type τ_i.

Informally, one record is better than another if it has better values in same fields. For example, if

$r_2 = \{Name \Rightarrow 'J.\ Doe'; \ Age \Rightarrow \perp_{int}; \ Status \Rightarrow [Employee \Rightarrow \{Office \Rightarrow G7; \ Extension \Rightarrow \perp_{int}]\}\}$

then $r_2 \sqsubseteq r_1$. From these definitions we can immediately see that the set of all records of a type τ is a domain D_τ belonging to the family of domains **Dom** constructed from the set of primitive domains $\mathcal{B}_1, \ldots, \mathcal{B}_n$ and the ordering relation on records corresponds exactly to the orderings on domains in **Dom**.

We next define the syntax of *scheme-types* for a type τ. τ' is a scheme-type for τ if:

(1) τ is a base type and $\tau' = \tau$ or $\tau' = unit_\tau$. $unit_\tau$ denotes the trivial scheme $\{\perp_{\mathcal{B}}\}$ in $\mathcal{B}$.

(2) $\tau' = \{l_1 : \tau'_1; l_2 : \tau'_2; \ldots; l_n : \tau'_n\}$, $\tau = \{l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n\}$ and τ'_i is a scheme-type for τ_i, for $1 \leq i \leq n$.

(3) $\tau' = [l_1 : \tau'_1; l_2 : \tau'_2; \ldots; l_n : \tau'_n]$, $\tau = [l_1 : \tau_1; l_2 : \tau_2; \ldots; l_n : \tau_n]$, and τ'_i is a scheme-type for τ_i, for $1 \leq i \leq n$.

(4) $\tau' = unit_\tau$ and τ is any discriminated union type. $unit_\tau$ denotes the trivial scheme $\{\perp_\tau\}$ in D_τ.

The following is a scheme-type of the type τ_1 defined in our example of a record type above:

$$\tau_2 = \{Name{:}string;\ Status{:}[Employee{:}\{Office{:}string;\ Extension{:}unit_{int}\};\ Consultant{:}\{Address{:}string;\ Telephone{:}unit_{int}\}]\}$$

Moreover, we regard the record r_2 having the above scheme-type τ_2. Formally, a record r has a scheme type τ if:

(1) $r \in \mathcal{B}_i$ and $\tau = B_i$.

(2) $r = \perp_{\mathcal{B}_i}$, $\tau = unit_{B_i}$.

(3) $r = \{l_1 \Longrightarrow r_1; \ldots, l_n \Longrightarrow r_n\}$, $\tau = \{l_1 : \tau_1, \ldots, l_n : \tau_n\}$ and r_i has the scheme-type τ_i for $1 \leq i \leq n$.

(4) $r = [l_i \Longrightarrow r_i]$,$\tau = [l_1 : \tau_1, \ldots, l_n : \tau_n]$, $i \leq n$ and r_i has scheme-type τ_i.

(5) $\perp_{\tau'}$ and τ is any scheme-type of any discriminated union type τ'.

Then by the definition of the scheme-types, we can also see that the set of all records of the scheme-type τ' for the type τ is a scheme in D_τ.

Sets of records belonging to a given type therefore form an interesting generalization of first-normal-form relations for which we can define relational operations, functional dependencies etc.

A	B	
	C	D
a_1	c_1	d_1
	c_2	d_2
a_2	c_4	d_1
	c_3	d_3
	c_2	d_1

(a)

A	B	
	C	D
a_1	c_1	d_1
a_1	c_2	d_2
a_2	c_4	d_1
a_2	c_3	d_3
a_2	c_2	d_1

(b)

Figure 4.8: Restricted higher-order relation and equivalent relation

Structures that contain sets

An extension to the relational model that has recently enjoyed some popularity is the study of higher-order relations [FT83, AB84, RKS84, ÖY85]. In these model a value in a tuple can itself be a set of values, i.e. another relation. In order to obtain a class of higher-order relations that behave well under relational operations, [RKS84] describes *partition normal form* relations. In such relations the attributes with simple (atomic) values functionally determine the attributes with higher-order values, which must also be in partition normal form. However, because of this severe restriction, sets are not treated as first-class values in this model. Indeed, it is not hard to show that partition-normal form relations are isomorphic to relations over record structures (without labeled sums) defined in the previous section. For example, the relation (a) in figure 4.8 is equivalent to the relation (b).

In order to obtain a data model in which sets are treated as first-class values, we need to construct a space of sets as a domain. Since, in defining various database operations, we have only assumed that the underlying space is a domain, once we have done this then sets can be also treated as regular values. In order to construct a domain of sets, we need to define an ordering on sets as database values. One obvious possibility is to treat the space of sets as a flat domain so that two sets are comparable iff there are equal. However, as we have seen, a flat domain has only two schemes, the set of all maximal elements and the trivial scheme containing only

bottom element, and does not yield interesting structures.

A second possibility is to regard sets as ordered by $\sqsubseteq^\flat$, which is what Bancilhon used in his complex object model [BK86]. Given a domain D, it can be shown [Sco82a] that we can construct a domain $\mathcal{P}^\flat(D)$ corresponding to the space of sets of elements in D ordered by $\sqsubseteq^\flat$ (the Hoare powerdomain of D). Since $\mathcal{P}^\flat(D)$ is a domain, the results of previous sections are readily applicable. However, it is probably rather difficult to find semantics of a natural join since a natural join is determined by the ordering $\sqsubseteq^\sharp$ and therefore database sets and sets appears as values in database objects are treated differently. We should also note that, since $\mathcal{P}^\flat(D)$ is a lattice,

Lemma 4.1.25 *For a domain D, the schemes in $\mathcal{P}^\flat(D)$ are the singleton sets $\{\{d\}\}$ where $d \in D$* $\square$

which means that functional dependencies in such a domain are rather trivial constraints.

Another possibility is to consider sets as values ordered by $\sqsubseteq^\sharp$, which is done in [Bun85, BO86, Oho90]. Smyth showed that [Smy78] for any domain D, a domain $\mathcal{P}^\sharp(D)$ corresponding to the space of of sets of elements in D ordered by $\sqsubseteq^\sharp$, called Smyth powerdomain of D, can be constructed. Under this approach, a natural join can be given coherent semantics. Again there are no non-trivial schemes in $\mathcal{P}^\sharp(D)$. However, if we relax our definition of a scheme, we can make some progress. Recall that a scheme A is an independent set in a domain D satisfying

$$p_A(D) = \underline{A}$$

and

$$\forall x \in D.\ p_A^*[\![x]\!]_D = [\![p_A(x)]\!]_{\underline{A}}$$

One way to generalize this is to specify directly a subset of D that is not necessarily downward closed. We say that a subset S of D is a *generalized scheme* in D if (1) S is closed under bounded join, (2) S has a minimal element and (3) the set of maximal elements $maxset(S)$ of S satisfies the second condition of schemes, i.e. $\forall x \in D.\ p_S^*[\![x]\!]_D = [\![p_S(x)]\!]_S$ where $p_S(x) = \sqcup\{s \mid s \in S, s \sqsubseteq x\}$. The original definition of schemes is a special case of generalized schemes. We can then find interesting schemes in $\mathcal{P}^\sharp(D)$.

$$
\begin{aligned}
r_1 \;=\; \{\{&Pname \Rightarrow \textit{'Nut'}; Supplier \Rightarrow \{\; \{Sname \Rightarrow \textit{'Smith'}; City \Rightarrow \textit{'London'}\};\\
&\qquad\qquad\qquad\qquad\qquad\qquad \{Sname \Rightarrow \textit{'Blake'}; City \Rightarrow \textit{'Paris'}\}\}\};\\
&\{Pname \Rightarrow \textit{'Bolt'}; Supplier \Rightarrow \{\{Sname \Rightarrow \textit{'Blake'}; City \Rightarrow \textit{'Paris'}\};\\
&\qquad\qquad\qquad\qquad\qquad\qquad \{Sname \Rightarrow \textit{'Adams'}; City \Rightarrow \textit{'Athens'}\}\}\}\}
\end{aligned}
$$

$$
\begin{aligned}
r_2 = \{\{&Pname \Rightarrow \textit{'Nut'}; Supplier \Rightarrow \{\{City \Rightarrow \textit{'Paris'}\}\}; Qty \Rightarrow 100\};\\
&\{Pname \Rightarrow \textit{'Bolt'}; Supplier \Rightarrow \{\{City \Rightarrow \textit{'Paris'}\}\}; Qty \Rightarrow 200\}\}
\end{aligned}
$$

$$
\begin{aligned}
r_1 \bowtie r_2 = \{\{&Pname \Rightarrow \textit{'Nut'}; Supplier \Rightarrow \{\{Sname \Rightarrow \textit{'Blake'}; City \Rightarrow \textit{'Paris'}\}\};\\
&\quad Qty \Rightarrow 100\};\\
&\{Pname \Rightarrow \textit{'Bolt'}; Supplier \Rightarrow \{\{Sname \Rightarrow \textit{'Blake'}; City \Rightarrow \textit{'Paris'}\}\};\\
&\quad Qty \Rightarrow 200\}\}
\end{aligned}
$$

Figure 4.9: Natural Join of Higher-order Relations

Lemma 4.1.26 *If S is a generalized scheme in a domain D then the set $\mathcal{P}^\sharp(S)$ is a generalized scheme in $\mathcal{P}^\sharp(D)$*

This suggests that if we regard sets as values ordered by $\sqsubseteq^\sharp$, then the previously described type systems can be extended to include a set type constructor by adding the following rules:

(1) If τ is a type then $\{\tau\}$ is a type.

(2) If τ' is a scheme type of τ then $\{\tau'\}$ is a scheme-type of $\{\tau\}$.

(3) If $v_1, \ldots, v_n$ are database objects of type τ then $minset(\{v_1, \ldots, v_n\})$ is a database object of type $\{\tau\}$.

In the third rule, a given set of database objects is coerced to a canonical representative of an element in $\mathcal{P}^\sharp(D)$ by taking its minimal elements. Natural join and projection work properly on the extended structures. Figure 4.9 shows an example of a natural join in the domain of records extended by these rules.

One restriction of the above approach is that we presuppose the meaning of sets of database objects by choosing the ordering $\sqsubseteq^\sharp$, i.e. sets are overdescribing some desired set of objects. This choice may not be appropriate for some applications. An

idea that merits further investigation is to look at partial descriptions that consist of pairs of sets: a complete and a consistent description of some target set. This may be particularly valuable in constructing a semantics for database merging [MB81] where the individual databases may not form a complete description of the real world.

Recursive structures

It is reasonable to suppose that we can also generalize database theory to work for recursive types, which can be used to give a type to unbounded structures such as lists. For example, given a domain D represented by a type τ, we can define a type for τ-lists as the type satisfying the following equation:

$$list(\tau) = [null : \{\}; nonnull : \{first : \tau; rest : list(\tau)\}]$$

This is the type of all lists of elements in D. Then for any scheme-type τ' for τ, $list(\tau')$ is a scheme-type for $list(\tau)$. There are also other scheme-type for $list(\tau)$ than in the above form. For example, the following is also a scheme-type for $list(\tau)$ that corresponds to the set of all lists of length less than or equal to one.

$$onelist = [null : \{\}; nonnull : \{first : \tau; rest : unit_{list(\tau)}\}]$$

where $unit_{list(\tau)}$ is the scheme-type $list(unit_\tau)$ for $list(\tau)$.

The domain $D_{list(\tau)}$ corresponding to $list(\tau)$ can be defined as the domain equation:

$$D_{list(\tau)} = Null + (D \times D_{list(\tau)})$$

where $Null$ is the trivial one element domain. Let S be the scheme in D corresponding to the scheme-type τ'. Then the scheme corresponding to the scheme-type $list(\tau')$ is the set of maximal elements in the domain defined by the equation:

$$D_{list(\tau')} = Null + (\underline{S} \times D_{list(\tau')})$$

The scheme corresponding to the scheme-type $onelist$ can be also defined.

The general form of schemes in recursive types such as these requires further investigation.

4.1.7 Conclusion and Further Investigation

We have tried to show that the application of domain theory allows us to provide a clean semantics for relational databases and provides a generalization of many of the ideas in relational database theory – especially those concerned with database design – into a large class of higher-order and recursive structures.

One major limitation of our work is that our characterization of the relational databases is restricted to a single domain. Operations and notions such as join and functional dependency are defined only within a given domain. It is however desirable to allow databases to contain values of different domains. This becomes essential if we want to treat values in a database as typed data structures and to integrate them into a type system of a programming language. In previous section we have constructed a collection of domains of records. As we suggested, each domain corresponds to a type in a type system of a programming language. In such a type system, it is natural to represent a database as a collection of relations of different types. Our formalism cannot be directly applied to such a database. One way to overcome this limitation would be to develop a theory of the relationship between various domains and to extend our characterization of the relational databases to a family of domains. [Oho90] proposed one such theory for join and projection and showed that a family of database domains can be integrated in an ML style type system. In [OB88] we have also shown that ML type inference method can be generalized to such an integrated type system. We further hope that the theory of functional dependencies and universal relations we have developed in this paper can be also generalize to families of domains.

Finally we should note that in database programming languages [BMW80, ACO85, SFL83], in knowledge bases [BS85] and in Aït-Kaci's [AK86] calculus for type subsumption the ordering is not completely derived from the structure of the objects themselves. There is also an imposed lattice or partial order of "entities", "concept-s", or "head-terms". The possibility of generalizing relational database notions into these systems may require these imposed orderings to have certain properties.

4.2 Decomposition of Domains

Introduction

This work was initiated by Peter Buneman's interest in generalizing relational databases, see [BJO91]. He — quite radically — dismissed the idea that a database should be forced into the format of an n-ary relation. Instead he allowed it to be an arbitrary anti-chain in a Scott-domain. The reason for this was that advanced concepts in database theory, such as 'null values', 'nested relations', and 'complex objects' force one to augment relations and values with a notion of information order. Following Buneman's general approach, the question arises how to define basic database theoretic concepts such as 'functional dependency' for anti-chains in Scott-domains. For this one needs a way to speak about 'relational schemes' which are nothing but factors of the product of which the relation is a subset. Buneman successfully defined a notion of 'scheme' for Scott-domains and it is that definition which at the heart of this work. We show that his generalized 'schemes' behave almost like factors of a product decomposition. (Consequently, we choose the word *semi-factor* for them.) In the light of our results, Peter Buneman's theory of generalized databases becomes less miraculous: a large class of domains can be understood as sets of tuples.

Buneman's definition of scheme was discussed in [Lib91] and an alternative definition was proposed. The idea of both definitions is that the elements of a domain are treated as objects, and projecting an element into a scheme corresponds to losing some information about this object. The definition of [Lib91] is based on the assumption that the same piece of information is lost for every object. For example, if objects are records, it means that we lose information about some attributes' values. The idea of [BJO91] is that every scheme has a sort of complement, and if we project one object to a scheme and the other to its complement, then there exists a join of two projections, i.e. every object consists of two independent "pieces of information". Intuitively it means that the domain itself could be decomposed into two corresponding domains.

The definition of [BJO91] is stronger than the definition of [Lib91]. It is the first definition that is used in our decomposition theory while the second definition serves as a tool to describe direct product decompositions of domains. Combining the decomposition theorems, we will prove a formal statement that clarifies the informal

reasonings from the previous paragraph.

There is also a more philosophical or pedagogical motivation for this work. A feature that novices to domain theory frequently find unsettling is the profusion of different definitions it offers. Often these definitions are laid out at the beginning and the relation to the semantics of programming languages is established only later. In particular, useful closure properties of the respective categories are derived. In his 'Pisa Lecture Notes' [Plo81], Gordon Plotkin chose a rather more gentle approach. The 'domains' he considers are very primitive at the beginning, just sets, and step by step new constructs and properties are added to them: a bottom element transforms sets into flat domains, and thus the information order is introduced; next come slightly more complicated orders created by forming finite products of flat domains; function spaces call for the definition of dcpo and Scott-continuous function and, via bilimits and powerdomains, he finally arrives at bifinite domains. Furthermore, along the way he develops a syntax which allows to denote (most of) the elements of the domains, making them available for computation: the product appears as a set of arrays, the function space as a set of λ-terms, etc. (This aspect is also described elegantly and comprehensively in [Abr91].) In this way, Plotkin creates the impression that all (bifinite) domains are built up from flat domains using various domain constructors. This may be reassuring for the novice but of course it is not explicitly confirmed in the text. Plotkin is just very carefully expanding his definitions and motivating each new concept. But we may still ask to what extent this first impression could be transformed into a theorem. To be more precise, we may ask: "Is it true, that every bifinite domain can be derived from flat domains using only lifting, product, coalesced sum, function space and convex powerdomain as constructors?" (A similar question was in fact asked — and found difficult — by Carl Gunter for the universal bifinite domain.)

How would one attack such a problem? We think the natural way to do it is to work backwards and to try to *decompose* domains into pieces that decompose no further. If we can show that the only irreducible domains are the flat domains then we are done.

At this point the informed reader may already have become nervous because he may know small finite counterexamples to the above question. But there are many variations of it which are equally interesting. We can restrict (or augment)

the number of allowed constructions, we can change the class of domains we want to analyze, we can allow more (or fewer) primitive (i.e. irreducible) building blocks. The choice we have made for this paper is to consider Scott and dI-domains (cf. [Ber78, Ber79]) and a single, albeit rather general, constructor, and instead of prescribing the irreducible factors, we are curious what they will turn out to be. The advantage of a decomposition theorem of this kind is apparent: instead of proving a property for general domains we can prove that it holds for the irreducible factors and that it is preserved under the constructions. We allow ourselves to compare this endeavor with the similar (and only recently completed) project of decomposing finite groups into finite simple groups, although the comparison is somewhat flattering: we cannot expect to find so much mathematically intriguing structure in domains.

What are the practical implications of our decomposition theorem? Well, in our particular setting we derive a very concrete representation of dI-domains as a set of 'tuples' which should simplify the implementation of dI-domains as abstract data types. Of course, there is a well-developed theory of effective representations (see [Smy77, Kan79, WD80, LW84]), where one enumerates the set of compact elements and represents (a subset of) the infinite elements by recursively enumerable sets of compact approximations. However, this is more theoretical work and no one expects that we really ever use domains as data types represented this way. Instead, our representation is much more concrete. To give an example, consider a domain which is the product of two flat domains. The traditional effective domain theory simply enumerates all elements, and, if enumerations of the elements of the two factors are already given, then these are combined with the help of pairing functions. We work rather in the opposite direction. For a given domain we seek to decompose it as far as possible and we will only enumerate the bases of the (irreducible) factors in the traditional way. The representation of the original domain is then put together as a set of 'tuples'.

The paper is organized as follows. In the next section we shall quickly review some basic definitions from domain theory, mostly to fix notation and to remind the reader of a few less common concepts. In Section 4.2.2 we introduce semi-factors and prove basic properties of them. We apply these ideas and get a first decomposition theorem. This representation still contains a lot of redundancy and in Section 4.2.3 we show how to 'factor away' this redundancy. The resulting decomposition theorem

yields a representation of dI-domains which is very tight. (These sections report work by the first and the third author.)

A direct product decomposition is a particular and interesting instance of our general goal and deserves more detailed study. In Sections 5 and 6 (which were written by the second author) this is done by establishing a relationship between these decompositions and particular instances of congruence relations and neutral ideals. The idea to describe direct product decompositions via neutral ideals is borrowed from lattice theory where neutral ideals describe decompositions of bounded lattices. For domains we will obtain a more general kind of decomposition including direct product and coalesced sum as limit cases. These decompositions are given by families of subsets of a domain such that every element of the domain has a unique representation as the join of suitably chosen representatives of these sets. Pairs of permutable complemented congruences also describe direct product decompositions as well as they describe decompositions of algebras. Having proved characterizations of decompositions, we establish the result showing the relationship between the two notions of scheme.

4.2.1 Definitions

We are using the standard definitions such as they can be found in [Jun89] and in [Abr91]. In particular, *dcpo's* are *directed-complete partial orders* and they have suprema for all directed sets. Most of the time they have a least element, which we denote by $\perp$. *Compact elements* in a domain are such that they cannot be below a supremum of a directed set without being below some element of that set already, and if there are enough compact elements such that every element is the supremum of a directed collection of them, we call the dcpo *algebraic*. More suprema than just those of directed sets can exist: if every bounded set has a join then we call the dcpo *bounded-complete*; if every set has a join then we have a *complete lattice*. In case a bounded-complete dcpo is also algebraic we call it a *Scott-domain*. The expression 'algebraic complete lattice' is shortened to *algebraic lattice*. We will mostly study distributive Scott-domains, for which it is sufficient to require the distributive law to hold in the principal ideals. (The standard textbook on distributive lattices is [BD74]). Even more restrictive is the definition of *dI-domains* (cf. [Ber78, Ber79]): they are distributive Scott-domains in which every principal ideal generated by a

compact element is finite. Because of this strong finiteness property we can usually derive theorems about dI-domains very quickly from the same theorems stated for finite distributive Scott-domains.

All our functions are *Scott-continuous*, which means they carry the supremum of a directed set to the supremum of the image of the set. We do not make much use of them in this generality but mostly consider *projections*, which are in addition idempotent and below the identity. Recall that projections always preserve existing infima and are completely determined by their image. Even the order between projections can be read off their image: it is simply inclusion. For more detailed information we refer to [GHK$^+$80].

An element x in a lattice is *join– (meet–) irreducible* if from the equation $y \vee z = x$ ($y \wedge z = x$) we can deduce that x equals y or z. (In the presence of distributivity this is equivalent to the stronger property of *join– (meet–) primeness*, but we will not make much use of this.)

Domain theory also includes the concept of *ideal* which is a directed and downward closed subset. This is a generalization of 'ideal' as it is known in lattice theory, where these are sets which are downward closed and closed under finite suprema. We need a generalization which goes in a different direction:

Definition 4.2.1 *A* stable subdomain *in a Scott-domain D is a downward closed subset which is closed under all existing joins.*

The same concept is defined in [BJO91] and in [Cur86] where such subsets of Scott-domains are called *strong ideal* and *complete ideal*, respectively. We find either expression rather misleading as we are not dealing with a special kind of domain theoretic ideal but with a completely different concept. Instead we take the viewpoint that such subsets are special *substructures*, i.e. special subdomains. As it happens, they correspond one-to-one to images of projections p for which $y \leq x$ implies $p(y) = y \wedge p(x)$. (An even stronger property holds, see Proposition 4.2.4 (ii) below.) In domain theory such functions are known as *stable projections*, hence our terminology.

Factors of products of dcpo's with bottom have the property that there is always a canonical projection onto them. This is also true for stable subdomains in Scott-domains:

Lemma 4.2.2 *Let A be a stable subdomain of the Scott-domain D. Then $p_A: D \to D$, defined by*

$$p_A(x) = \bigvee(\downarrow x \cap A)$$

is a projection on D with image A.

Our first decomposition has the form of a general categorical *limit*. A concrete description is given in terms of certain elements of the product of the dcpo's involved.

Definition 4.2.3 *Let $\mathcal{D}$ be a set of dcpo's and let $\mathcal{F}$ be a set of Scott-continuous functions between elements of $\mathcal{D}$ (in the language of category theory: a diagram in* **DCPO***). Furthermore, let $\bar{x} = (x_D)_{D \in \mathcal{D}}$ be an element (a* tuple*) of the cartesian product of all elements of $\mathcal{D}$. We say that $\bar{x}$ is* commuting *if the equation $x_E = f(x_D)$ holds for all functions $f: D \to E$, $f \in \mathcal{F}$, and all elements D, E in $\mathcal{D}$. Similarly, it is called* hyper-commuting *if the inequality $x_E \geq f(x_D)$ holds.*

The set of all commuting tuples forms the categorical limit of the diagram $(\mathcal{D}, \mathcal{F})$ and we denote it by $\mathsf{lim}_{\mathcal{F}}\mathcal{D}$. The set of hyper-commuting tuples we call the *hyper-limit* and we reserve the notation $\mathsf{hyperlim}_{\mathcal{F}}\,\mathcal{D}$ for it. The latter construction is a special case of a more general concept developed in the theory of 2-categories, namely, lax limits. It is easy to see that **DCPO** is closed under limits and this kind of lax limit. Whether any of the other properties generally associated with domains is preserved depends on the structure of the diagram. For more detailed information consult [Tay87a].

4.2.2 Stable subdomains, semi-factors, and the First Decomposition Theorem

We begin by recalling from [BJO91] and [Puh90] some of the properties of stable subdomains.

Proposition 4.2.4 *Let D be a Scott-domain. Then the following hold:*

(i) $\{\perp_D\}$ *and D are stable subdomains of D.*

(ii) If x is an element of a stable subdomain A of D and if $p_A(y)$ is less than x then $p_A(y) = x \wedge y$.

(iii) *If D is distributive then p_A preserves existing suprema.*

(iv) *The set Q_D of all stable subdomains of D ordered by inclusion is an algebraic lattice.*

(v) *If D is distributive then Q_D is distributive.*

(vi) *In Q_D, the finite meet of stable subdomains is given by their intersection and $p_{A\cap B} = p_A \circ p_B = p_B \circ p_A$.*

(vii) *If D is distributive then (arbitrary) suprema in Q_D can be calculated pointwise, and for $\mathcal{A} \subseteq Q_D, x \in D : p_{\bigvee \mathcal{A}}(x) = \bigvee_{A\in\mathcal{A}} p_A(x)$.*

(Proofs can be found in [BJO91].)

The concept of 'stable subdomain' is still too general to serve as a definition of 'distinguished piece of a domain'. For example, every element x of a domain generates a stable subdomain $\downarrow x$, but in general such a principal ideal cannot be hoped to lead to a sensible decomposition. In [BJO91] a more restrictive definition is introduced, that of a *scheme*, and it is motivated by the database applications we had in mind there. Here we can give a new motivation based on the desired decomposition result. Consider the following theorem:

Theorem 4.2.5 *Let D be a finite distributive Scott-domain and let $\mathcal{A}$ be a set of stable subdomains the supremum of which equals $D = \top_{Q_D}$. Let $\mathcal{F}$ be the set of projections $p_A|_B : B \to A$ where $A \subseteq B$ are two elements of $\mathcal{A}$. Furthermore, let $\hat{D}$ consist of those commuting tuples $\bar{x} = (x_A)_{A\in\mathcal{A}}$ for which the set $\{x_A \mid A \in \mathcal{A}\}$ is bounded in D. Then $\hat{D}$ is isomorphic to D with the isomorphisms*

$$\begin{aligned} \Psi: D \to \hat{D}, \Psi(x) &= (p_A(x))_{A\in\mathcal{A}} \\ \Phi: \hat{D} \to D, \Phi(\bar{x}) &= \bigvee\{x_A \mid A \in \mathcal{A}\}. \end{aligned}$$

The proof of this theorem is straightforward, one only has to bear in mind that suprema in Q_D are calculated pointwise. The theorem is unsatisfying, however, because in order to represent D through a set of stable subdomains, we need to include information that can only be gained by looking at D itself: the boundedness of the coordinates of $\bar{x}$. We shall now give a definition of a *semi-factor*, such that boundedness comes for free if only the tuple commutes.

Definition 4.2.6 *A stable subdomain A of a Scott-domain D is called* semi-factor *if $p_A(x) \leq a$ implies that x and a are bounded, for all $x \in D$ and $a \in A$.*

In [BJO91] and in [Puh90] it is shown that this definition works well in the test case of direct product decompositions: the semi-factors of a direct product $D \times E$ are in 1–1 correspondence with products of semi-factors of D and E. In particular, $D \times \{\bot_E\}$ and $\{\bot_D\} \times E$ are semi-factors in $D \times E$.

We collect the basic properties of semi-factors in a fashion similar to that for stable subdomains:

Proposition 4.2.7 *Let D be a distributive Scott-domain. Then the following hold:*

(i) *$\{\bot_D\}$ and D are semi-factors of D.*

(ii) *The set S_D of all semi-factors of D, ordered by inclusion, is a distributive, complete lattice.*

(iii) *If S and T are semi-factors of D, then so are $S \cap T$ and $S \vee T$, where again the join is taken pointwise. (The latter also holds for arbitrary joins.)*

(iv) *S_D is a sublattice of Q_D.*

(For the proofs see [BJO91].)

The following lemma states that our definition yields the desired extension property:

Lemma 4.2.8 *Let $\mathcal{S}$ be a family of semi-factors of a finite distributive Scott-domain D and let $\mathcal{F}$ consist of all connecting projections as in Theorem 4.2.5 above. Let $\mathcal{S}$ be such that with $S, T \in \mathcal{S}$ we also have $S \cap T \in \mathcal{S}$. If $\bar{x} = (x_S)_{S \in \mathcal{S}}$ is a commuting tuple, then the set $\{x_S \mid S \in \mathcal{S}\}$ is bounded in D.*

Proof. We first show this for the case in which $\mathcal{S}$ consists of just three semi-factors, S, T and $S \cap T$. Let $\bar{x}$ be a commuting tuple in $S \times T \times S \cap T$.

$$\begin{aligned}
x_T &\geq p_{S\cap T}(x_T) && (p_{S\cap T} \leq id_D)\\
&= x_{S\cap T} && (\bar{x} \text{ is commuting})\\
&= p_{S\cap T}(x_S) && (\text{ditto})\\
&= p_T \circ p_S(x_S) && (4.2.4 - vi \;\&\; 4.2.7 - iii)\\
&= p_T(x_S) && (p_S \mid_T = id_T)
\end{aligned}$$

By the defining property of semi-factors, $\{x_S, x_T\}$ is bounded in D.

The general proof is by induction. Set $S = \bigvee_{i=1}^{n} S_i$ and $T = S_{n+1}$. By the induction hypothesis the join of $\{x_{S_1}, \ldots, x_{S_n}\}$ exists and we may set $x_S = \bigvee_{i=1}^{n} x_{S_i}$. The tuple $(x_S, x_T, p_{S\cap T}(x_S))$ is commuting for the three semi-factors S, T and $S \cap T$, because projections preserve suprema by 4.2.4-(iii): $p_{S\cap T}(x_S) = p_T \circ p_S(x_S) = p_T(x_S) = p_T(\bigvee_{i=1}^{n} x_{S_i}) = \bigvee_{i=1}^{n} p_T(x_{S_i}) = \bigvee_{i=1}^{n} x_{S_i\cap T} = \bigvee_{i=1}^{n} p_{S_i}(x_T) = p_S(x_T) = p_S \circ p_T(x_T) = p_{S\cap T}(x_T)$. So we can apply the result for the three element case for the induction step. ▮

In our decomposition theorem we want to use as few semi-factors as possible, which in turn should be as primitive as possible. As a first approximation we choose the set $J(S_D)$ of semi-factors which are join-irreducible in Q_D. This set has two properties which make it attractive: every semi-factor is a join of irreducibles (in the finite case, but it will generalize to dI-domains) and a join-irreducible cannot be reached by a join of strictly smaller semi-factors, so it is in a sense unavoidable. But in order to apply the previous lemma we need a set closed under finite intersections, and in general $J(Q_D)$ will not do us this favor. We need another preparatory lemma:

Lemma 4.2.9 *Let D be a finite distributive Scott-domain and let $J(S_D)$ be the set of join-irreducible semi-factors of D. Let $\bar{x} = (x_S)_{S\in J(S_D)}$ be a commuting tuple for $J(S_D)$ and the connecting projections $\mathcal{F}$. Let $\mathcal{F}'$ be the appropriately extended set of connecting projections for all of S_D. Then $\bar{x}$ can be extended uniquely to a commuting element $\bar{x}'$ for $S_D, \mathcal{F}'$.*

Proof. We first show that for two join-irreducible semi-factors U and V we have the following commutation rule: $p_U(x_V) = p_V(x_U)$. Indeed, if $U \cap V$ is the join of the join-irreducible semi-factors $U_1, \ldots, U_n$ then we can calculate: $p_U(x_V) = p_U \circ p_V(x_V) = p_{U\cap V}(x_V) = \bigvee_{i=1}^{n} p_{U_i}(x_V) = \bigvee_{i=1}^{n} x_{U_i} = \bigvee_{i=1}^{n} p_{U_i}(x_U) = p_{U\cap V}(x_U) = p_V \circ p_U(x_U) = p_V(x_U)$.

We extend the tuple $\bar{x}$ to all of S_D by setting

$$x_S = \bigvee\{x_U \mid S \supseteq U \in J(S_D)\}.$$

We have to show that $\bar{x}' = (x_S)_{S\in S_D}$ is commuting, so let $S \subseteq T$ be two semi-factors of D. Then we have

$$p_S(x_T) \;=\; p_S(\bigvee\{x_U \mid T \supseteq U \in J(S_D)\}) \quad \text{(by def.)}$$

$$
\begin{aligned}
&= \bigvee\{p_S(x_U) \mid T \supseteq U \in J(S_D)\} \quad (4.2.4 - iii)\\
&= \bigvee\{p_V(x_U) \mid T \supseteq U \in J(S_D), S \supseteq V \in J(S_D)\} \quad (4.2.4 - vii)\\
&= \bigvee\{p_U(x_V) \mid T \supseteq U \in J(S_D), S \supseteq V \in J(S_D)\} \quad (\text{as shown before})\\
&= \bigvee\{p_T(x_U) \mid S \supseteq V \in J(S_D)\} \quad (4.2.4 - vii)\\
&= \bigvee\{x_V \mid S \supseteq V \in J(S_D)\} \quad (V \supseteq S \supseteq T)\\
&= x_S \quad (\text{by def.}) \blacksquare
\end{aligned}
$$

We can now state

Theorem 4.2.10 ***(The First Decomposition Theorem)*** *Let $J(S_D)$ be the set of all join-irreducible semi-factors of the finite distributive Scott-domain D and let $\mathcal{F}$ be the set of connecting projections. Then D is isomorphic to the limit of $J(S_D)$ over $\mathcal{F}$. The isomorphisms are given by*

$$
\begin{aligned}
\Psi: D &\to \mathsf{lim}_{\mathcal{F}} J(S_D)\\
x &\mapsto (p_S(x))_{S \in J(S_D)}
\end{aligned}
$$

and

$$
\begin{aligned}
\Phi: \mathsf{lim}_{\mathcal{F}} J(S_D) &\to D\\
(x_S)_{S \in J(S_D)} &\mapsto \bigvee_{S \in J(S_D)} x_S.
\end{aligned}
$$

The proof of this theorem is contained completely in the previous lemma, where we showed how to extend a commuting tuple to all of S_D, in particular to $D \in S_D$ itself. ∎

We illustrate the First Decomposition Theorem for three finite domains.

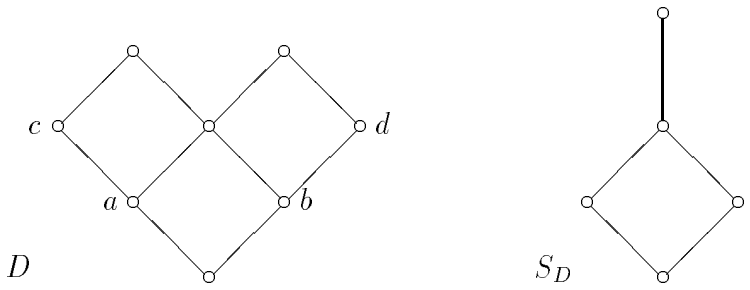
Example 1: $D = M_\perp$, M a finite set, i.e. D is a flat domain. We find that D possesses only the trivial semifactors $\{\perp\}$ and D, the latter being join-irreducible in S_D. Hence we conclude:

Observation 1: Flat domains are indecomposable.

Example 2: $D = 2 \times 2$, the four-element Boolean algebra. Since D is a lattice, it is isomorphic to its lattice of semi-factors. The join-irreducibles are $(\top, \perp)$ and $(\perp, \top)$ and the decomposition yields $D \simeq \downarrow(\top, \perp) \times \downarrow(\perp, \top)$, where $\downarrow(\top, \perp) \simeq \downarrow(\perp, \top) \simeq 2$. This is not a coincidence:

Observation 2: Direct product structure is recognized.

Example 3:



We find that D is join-irreducible in S_D and hence must be contained in any decomposition based on the First Decomposition Theorem. This is obviously not satisfactory and we shall derive a better decomposition theory below. Before doing so, let us study the situation for infinite domains. Here we have to deal with the following complication: the intersection of an infinite family of semi-factors is not necessarily a semi-factor again. We therefore do not know whether S_D is algebraic in general. We view this as the major open problem in our decomposition theory. In the case of dI-domains we are fine:

Proposition 4.2.11 *Let D be a dI-domain. Then S_D is algebraic and co-algebraic (i.e. S_D^{op} is algebraic).*

Proof. We only give an outline because we don't have the space to introduce the details of the theory of approximation via compact elements in domains in general and in our decomposition theory in particular.

One first observes that stable subdomains and semi-factors are completely determined by the set of compact elements they contain. Also, the canonical projection onto a stable subdomain can be seen as mapping each element onto the supremum of those compact elements of the subdomain which are below it: $p_A(x) = \bigvee \downarrow x \cap K(D) \cap A$. Furthermore, the canonical projection, as a Scott-continuous map, is completely determined by its behavior on compact elements. Since it is also sufficient to state the extension property of semi-factors for compact elements only, we have reduced the whole theory to $K(D)$, the set of compact elements in D. With this in mind, it is now easy to see that the arbitrary intersection of semi-factors is again

a semi-factor: below a compact element in a dI-domain there are only finitely many elements at all and so for a particular compact element the intersection behaves as if it were over a finite index set.

Similarly, it is easy to see that the directed union of semi-factors yields a semi-factor again. Together this shows that the set S_D of semi-factors forms an inductive hull system on D, which implies algebraicity. A semi-factor is compact in S_D if and only if it is generated by a finite set of compact elements of D.

The co-compact elements are found as follows: suppose a semi-factor S does not contain a certain element x of D. By algebraicity of D it follows that there is a compact element c of D which S does not contain. Furthermore, because $\downarrow c$ is a finite distributive lattice, there is a join-irreducible k below c which again does not belong to S. On the other hand, if k is join-irreducible (hence: prime) in $K(D)$, then the join of all semi-factors which do not contain k, will again not contain this element. From this it follows along standard lines that any finite set of join-irreducible elements of $K(D)$ defines a co-compact semi-factor and that there are enough co-compact semi-factors to generate the whole lattice S_D. So it is co-algebraic as well. ▌

From [GHK+80] we recall that algebraic lattices have an inf-basis of meet-irreducible elements, and so for a dI-domain D the distributive lattice S_D has both a sup-basis of join-irreducibles and an inf-basis of meet-irreducibles. We can therefore state:

Corollary 4.2.12 *The First Decomposition Theorem holds for dI-domains.*

4.2.3 Factoring by stable subdomains and the Second Decomposition Theorem

In group theory and in ring theory we are familiar with the following technique. For a given strong substructure (normal subgroup, ideal, respectively) one studies the equivalence relation which identifies those elements which differ only by an amount contained in the substructure. A similar notion works for ideals in distributive lattices: If A is an ideal in L then we can set $x \sim y$ if there is an $a \in A$ such that $x \vee a = y \vee a$. (for details see [BD74].) Since domains lack arbitrary suprema we have to rework this definition a little bit:

Definition 4.2.13 *Let A be a stable subdomain in a distributive Scott-domain D. On D define a binary relation θ_A by setting $x\ \theta_A\ y$ if there is $a \in A$ such that $y = x \vee a$. Let Θ_A be the symmetric and transitive hull of θ that is the smallest equivalence relation containing θ_A. (Θ_A can be described concretely as $\bigcup_{n\in\mathbb{N}}(\theta_A^{-1} \circ \theta_A)^n$.)*

This definition proves to be extremely fruitful. We list the following properties:

Proposition 4.2.14 *Let D be a finite distributive Scott-domain and let A be a stable subdomain in D. Then the following hold:*

(i) $x\ \theta_A\ y \Longrightarrow x \leq y$.

(ii) $x\ \theta_A\ y \Longrightarrow y = x \vee p_A(y)$, *and for all* $a \in A$, *if* $y = x \vee a$, *then* $a \leq p_A(y)$.

(iii) $\theta_A \circ \theta_A = \theta_A$.

(iv) $x\ \theta_A\ y, z \in D \Longrightarrow z \wedge x\ \theta_A\ z \wedge y$ *and* $z \vee x\ \theta_A\ z \vee y$. *(Provided the suprema exist.)*

(v) Θ_A *is a congruence relation on D with respect to finite infima and existing suprema.*

(vi) Each equivalence class of Θ_A *contains a least element.*

(vii) $\theta_A = \Theta_A \cap \leq$.

(viii) Each equivalence class of Θ_A *is order convex.*

(ix) $\Theta_A = \theta_A^{-1} \circ \theta_A$.

(x) p_A *is injective on every equivalence class of* Θ_A.

We denote the function which maps each element onto the smallest element in its equivalence class by q_A. With this notation we can add the following clauses:

(xi) q_A *is a projection on D.*

(xii) q_A *preserves existing suprema.*

Proof. (i) is trivial, for (ii) recall that p_A is join-preserving by 4.2.4-(iii).

(iii) $x\ \theta_A\ y\ \theta_A\ z \Longrightarrow y = x \vee a_1$ and $z = y \vee a_2 = x \vee a_1 \vee a_2$, and with a_1 and a_2 elements of A, their join is again in A.

(iv) $x\ \theta_A\ y \Longrightarrow y = x \vee a \Longrightarrow z \wedge y = z \wedge (x \vee a) = (z \wedge x) \vee (z \wedge a)$, and with $a \in A$, the element $z \wedge a$ is again in A. For suprema: $x\ \theta_A\ y \Longrightarrow y = x \vee a \Longrightarrow z \vee y = (z \vee x) \vee a$.

(v) It is immediate from the definition of Θ_A as a union of products of θ_A and θ_A^{-1} that (iv) also holds for Θ_A. Now, if $x\ \Theta_A\ y$ and $x'\ \Theta_A\ y'$ then $x \wedge x'\ \Theta_A\ y \wedge x'\ \Theta_A\ y \wedge y'$, and analogously for suprema.

(vi) follows from (v) by taking the infimum of the equivalence class.

(vii) Suppose $x \leq y$ and $x\ \Theta_A\ y$. Then by definition there is a chain $x_1, x_2, \ldots, x_n$ of elements such that $x = x_1\ \theta_A^{-1}\ x_2\ \theta_A\ x_3\ \theta_A^{-1}\ x_4 \ldots x_{n-1}\ \theta_A\ x_x = y$. By taking the supremum of each element of this sequence with x and then the infimum with y we derive a new sequence which is completely contained in the interval $[x, y]$. x_2 is then necessarily equal to x. We further shorten the sequence as follows: $x = x_2 = x_2 \wedge x_4\ \theta_A\ x_3 \wedge x_4\ \theta_A^{-1}\ x_4 \wedge x_4 = x_4\ \theta_A\ x_5 \ldots$ Since $x_3 \wedge x_4$ is below x_4 and in relation θ_A^{-1} it is actually equal to x_4, so the sequence now reduces to $x\ \theta_A\ x_4\ \theta_A\ x_5 \ldots$ Applying (iii) we find that x is in θ_A-relation to x_5 already. Continuing in this fashion will reduce the sequence eventually to $x\ \theta_A\ y$ which is what we want.

(viii) Assume $x\ \Theta_A\ y$ and $x \leq z \leq y$. By (vii) we have $x\ \theta_A\ y$ which implies $y = x \vee a$ for some $a \in A$. But then $z = z \wedge y = (x \vee z) \wedge (x \vee a) = x \vee (z \wedge a)$ which gives us $x\ \theta_A\ z$. The relation $z\ \theta_A\ y$ follows directly from $y = x \vee a$.

(ix) Combining (vi) and (vii) we find that the least element of an equivalence class is in θ_A-relation to each member.

(x) A projection always preserves infima and so if p_A maps two elements x and y to the same image a, it will map $x \wedge y$ to a as well, and, if $x\ \Theta_A\ y$ then $x \wedge y\ \Theta_A\ y$ and by (vii) $x \wedge y\ \theta_A\ y$. So consider w.l.o.g. $x\ \theta_A\ y$ and $p_A(x) = p_A(y)$. We directly get $y = x \vee p_A(y) = x \vee p_A(x) = x$.

(xi) We only have to show that q_A is monotone. So suppose $x \leq y$. By (vii) we have $q_A(y)\ \theta_A\ y$ which yields with (iv): $x \wedge q_A(y)\ \theta_A\ x \wedge y = x$. But $q_A(x)$ is the smallest element in the equivalence class of x. Hence $q_A(x) \leq x \wedge q_A(y) \leq q_A(y)$ follows.

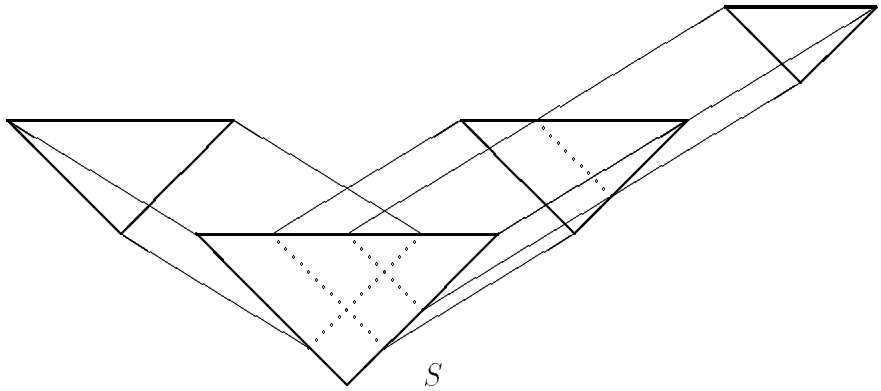


Figure 4.10: Decomposition of a domain by a semi-factor S.

(xii) From $z = x \vee y, q_A(x)\ \theta_A\ x, q_A(y)\ \theta_A\ y$ we conclude by (iv) that $q_A(x) \vee q_A(y)\ \theta_A\ x \vee y = z$. Since q_A is monotone it follows that $q_A(x) \vee q_A(y)$ must be equal to $q_A(z)$. ∎

Given a representation of a poset P as the cartesian product of two posets R and S we can understand P as follows: it consists of $|R|$ many copies S_x of S, and if $x \leq y$ in R then each element of S_x is below the corresponding element of S_y. A semi-factor S in a finite domain leads to a similar representation: for each element x in the image R of q_S we take the principal filter $F_x = \uparrow p_S(x)$ in S (instead of the whole semi-factor). These filters are connected as before, that is, if $x \leq y$ in R, then each element of F_x is below the corresponding element of F_y. However, there may be elements of F_x for which there is no corresponding element in F_y. This is the content of the following proposition. A picture illustrating this representation is given in Figure 4.10.

Knowing $S, \mathsf{im}(q_S)$ and the action of p_S on the image of q_S we can reconstruct the domain:

Proposition 4.2.15 *Let D be a finite distributive domain and let S be a semi-factor in D.*

(i) The image of an equivalence class of Θ_S under p_S is upward closed in S.

(ii) *D is isomorphic to the set $\hat{D} = \{(x,s) \in \mathsf{im}(q_S) \times S \mid p_S(x) \leq s\}$ ordered pointwise. The isomorphism is given by $q_S \times p_S : D \to \hat{D}$ and by the supremum function for the other direction.*

(iii) *If $\forall x \in D : p_S \circ q_S(x) = \bot_D$ then S is a direct factor of D.*

Proof. (i) If s is above $p_S(x)$ in S then by the extension property of semi-factors $s \vee x$ exists and is in θ_S^{-1}-relation to x. Also, $p_S(s \vee x) = p_S(s) \vee p_S(x) = s \vee p_S(x) = s$.

(ii) For $x \in D$ we have $q_S(x) \leq x$ and therefore $p_S(q_S(x)) \leq p_S(x)$. So the pair $(q_S(x), p_S(x))$ belongs to $\hat{D}$. The mapping $q_S \times p_S$ is injective by Proposition 4.2.14-(x). We claim that the inverse is given by the supremum function. First of all, the supremum exists for the pairs in $\hat{D}$ because S is a semi-factor. It is clearly monotonic and it inverts $q_S \times p_S$ because $q_S \times p_S(x \vee s) = (q_S(x \vee s), p_S(x \vee s)) = (q_S(x) \vee q_S(s), p_S(x) \vee p_S(s)) = (x \vee \bot, p_S(x) \vee s) = (x, s)$ and for the other composition: $q_S(x) \vee p_S(x) = x$ because $q_S(x)\, \theta_S\, x$ by definition.

(iii) This follows because for $p_S \circ q_S = \bot_{[D \longrightarrow D]}$ the condition in the definition of $\hat{D}$ is always satisfied. ∎

This proposition works with elements of the domain. But there is also a way of looking at this situation using congruence relations. Recall that every homomorphism $f : D \to E$ induces a canonical congruence relation on D, called the *kernel of f* ($\ker f$), which identifies exactly those elements of D which are mapped to the same element. Obviously, $\mathsf{ker} q_A = \Theta_A$. Let $\mathsf{Con}(D)$ be the complete lattice of all congruences (with respect to finite infima and existing suprema) on D.

Proposition 4.2.16 *Let D be a finite distributive Scott-domain and let A be a stable subdomain in D. Then the following is true:*

(i) *$\mathsf{ker} p_A$ is a congruence with respect to arbitrary infima and arbitrary (existing) suprema.*

(ii) *$\mathsf{ker} p_A \cap \Theta_A = \Delta_{D \times D} = 0_{\mathsf{Con}(D)}$.*

(iii) *$\mathsf{ker} p_A \vee \Theta_A = D \times D = 1_{\mathsf{Con}(D)}$.*

Proof. (i) holds because p_A is a projection on a distributive domain, (ii) restates 4.2.14-(x) and, finally, (iii) follows because every $x \in D$ is related to $\perp$ in the following way:

$x\ \Theta_A\ q_A(x)\ (\mathsf{ker} p_A)\ p_A(q_A(x))\ \Theta_A\ \perp$. ∎

The results of this section extend to dI-domains:

Proposition 4.2.17 *Proposition 4.2.14 and Proposition 4.2.16 hold for dI-domains, in particular, equivalence classes of Θ_A and $\mathsf{ker}\, p_A$ are closed under directed suprema and q_A is Scott-continuous.*

Proof. The main technical difficulty is to prove that equivalence classes of Θ_A have a least element. For details we refer the reader to [Puh90]. ∎

We use factorization to improve on our First Decomposition Theorem. We observed that it produces representations which are redundant, namely, if two comparable semi-factors $S \subset T$ are join-irreducible in S_D then both take part in the representation, T repeating the information given by S. We shall now factor out this repeated information. Given a collection $\mathcal{S}$ of semi-factors we define for each element $S \in \mathcal{S}$ its *lower $\mathcal{S}$-cover* S' by $S' = \bigvee\{T \in \mathcal{S} \mid T \subset S\}$. Also, if $S \subseteq T \in S_D$ let $S/_T$ stand for $\mathsf{im}(q_T|_S)$. With this notation we are now ready to formulate:

Theorem 4.2.18 ***(The Second Decomposition Theorem)*** *Let D be a finite distributive Scott-domain (a dI-domain) and let $J(S_D)$ be the set of all join-irreducible semi-factors of D. Define*

$$RJ(S_D) = \{S/_{S'} \mid S \in J(S_D)\}$$

and

$$\mathcal{F} = \{q_{S'} \circ p_S \big|_{T/_{T'}} \mid S \subseteq T \in J(S_D)\}.$$

Then D is isomorphic to the hyper-limit of $RJ(S_D)$ over $\mathcal{F}$ with the isomorphisms

$$\begin{array}{rcl} \Psi\colon D & \to & \mathsf{hyperlim}_{\mathcal{F}}\, RJ(S_D) \\ x & \mapsto & (q_{S'} \circ p_S(x))_{S \in J(S_D)} \end{array}$$

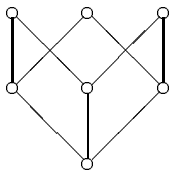


Figure 4.11: A non-flat indecomposable dI-domain.

and

$$\begin{aligned} \Phi\colon \operatorname*{hyperlim}_{\mathcal{F}} RJ(S_D) &\to D \\ (x_S)_{S\in J(S_D)} &\mapsto \bigvee_{S\in J(S_D)} x_S. \end{aligned}$$

(The proof of this should be clear from the First Decomposition Theorem and Proposition 4.2.15.)

We illustrate the representation of domains provided by the Second Decomposition Theorem with Example 3 from the last section. The three join-irreducible semi-factors are $\downarrow a, \downarrow b$, and D itself. By factoring D through the join of $\downarrow a$ and $\downarrow b$ we can replace it by the three element domain $\{\bot, c, d\}$.

Decomposition into flat domains is particularly satisfying and one may wonder whether it is achievable for all distributive Scott-domains or for all dI-domains. The answer is 'no'; a counterexample is given in Figure 4.11.

However, it turns out that the category **F** of those distributive Scott-domains which are representable as hyperlimits of flat domains, is cartesian closed and contains strictly all *concrete domains* (cf. [KP93, Win81]). Indeed, the connection to concrete domains seems to be very strong. Recent work by Geva and Brookes (see [BG92a]) suggests that every domain in **F** can be represented as a generalized concrete data structure.

Bibliography

[AB84] S. Abiteboul and N. Bidoit. Non first normal form relations to represent hierarchically organized data. In *Proc. 3rd ACM Symp. on Principles of Database Systems*, pages 191–200, 1984.

[Abr90] S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.

[Abr91] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[Abr93] S. Abramsky. Interaction categories. In G. Burn, S. Gay, and M. Ryan, editors, *Theory and Formal Methods 1993*, Workshops in Computing, pages 57–69. Springer Verlag, 1993.

[AJM] S. Abramsky, R. Jagadeesan, and P. Malacaria. Games and full abstraction for PCF, preliminary announcement. Note distributed August 1993.

[AJ94] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3. Oxford University Press, 1994.

[Acz88] P. Aczel. *Non-Well-Founded Sets*, volume 14 of *CSLI Lecture Notes*. Center for the Study of Language and Information, 1988.

[AK86] H. Aït-Kaci. An algebraic semantics approach to the effective resolution on type equations. *Theoretical Computer Science*, 45:239–351, 1986.

[ACO85] A. Albano, L. Cardelli, and R. Orsini. Galileo: A strongly typed, interactive conceptual language. *ACM Transactions on Database Systems*, 10:230–260, 1985.

[ABL86] R. Amadio, K. B. Bruce, and G. Longo. The finitary projection model for second order lambda calculus and solutions to higher order domain equations. In *Logic in Computer Science*, pages 122–130. IEEE Computer Society Press, 1986.

[Atk78] M. P. Atkinson. Programming languages and databases. In S. B. Yao, editor, *Proc. 4th International Conference on Very Large Data Bases*, pages 408–419, 1978.

[AB87] M. P. Atkinson and O. P. Buneman. Types and persistence in database programming languages. *ACM Computing Surveys*, 1987.

[AP82] P. Atzeni and D. S. Parker. Assumptions in relational database theory. In *Proc. 1st ACM Symp. on Principles of Database Systems*, pages 1–9, 1982.

[BD74] R. Balbes and P. Dwinger. *Distributive Lattices*. University of Missouri Press, Columbia, 1974.

[BH81] B. Banaschewski and R.-E. Hoffmann, editors. *Continuous Lattices*, volume 871 of *Lecture Notes in Mathematics*. Springer Verlag, 1981.

[BK86] F. Bancilhon and S. Khoshafin. A calculus for complex objects. In *Proc. 5th ACM Conference on Principles of Database Systems*, pages 53–59, 1986.

[Ber91] S. Berardi. Retractions on dI-domains as a model for type:type. *Information and Computation*, 94:204–231, 1991.

[BMW80] P. Bernstein, J. Mylopoulos, and H. K. T. Wong. A language facility for designing database intensive applications. *ACM Transactions on Database Systems*, 5, 1980.

[Ber78] G. Berry. Stable models of typed λ-calculi. In *Proceedings of the 5th International Colloquium on Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 72–89. Springer Verlag, 1978.

[Ber79] G. Berry. Modèles Complèment Adéquats et Stables des Lambda-calculs typés, 1979. Thèse de Doctorat d'Etat, Université Paris VII.

[BCL85] G. Berry, P.-L. Curien, and J.-J. Lévy. Full abstraction for sequential languages: The state of the art. In M. Nivat and J. Reynolds, editors, *Algebraic Semantics*, pages 89–132. Cambridge University Press, 1985.

[Bis81] J. Biskup. *Advances in Data Base Theory*, volume 1, chapter A Formal Approach to Null Values in Database Relations. Prenum Press, 1981.

[Blo90] B. Bloom. Can LCF be topped? Flat lattice models of typed λ-calculus. *Information and Computation*, 87:264–301, 1990.

[Bor85] A. Borgida. Thoughts on accomodating exceptions to (type) constraints in information systems. In *Proceedings of the Appin workshop on Persistence and Data Types*, volume 16 of *Persistent Programming Research Report*, pages 275–280, 1985.

[BS85] R. J. Brachman and J. G. Schmolze. An overview of the KL-one knowledge representation system. *Cognitive Science*, 9, 1985.

[BG92a] S. Brookes and S. Geva. Continuous functions and parallel algorithms on concrete data structures. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics*, volume 598 of *Lecture Notes in Computer Science*, pages 326–349. Springer Verlag, 1992.

[BG92b] S. Brookes and S. Geva. Stable and sequential functions on scott domains. Technical Report CMU-CS-92-121, Carnegie Mellon University, 1992.

[BG93] S. Brookes and S. Geva. Sequential functions on indexed domains and full abstraction for a sub-language of PCF. Technical Report CMU-CS-93-163, Carnegie Mellon University, 1993.

[Bun85] O. P. Buneman. Data types for data base programming. In *Proceedings of the Appin workshop on Persistence and Data Types*, volume 16 of *Persistent Programming Research Report*, 1985.

[BO86] P. Buneman and A. Ohori. A domain theoretic approach to higher-order relations. In *International Conference on Database Theory*, volume 243 of *Lecture Notes in Computer Science*. Springer Verlag, 1986.

[BJO91] P. Buneman, A. Jung, and A. Ohori. Using powerdomains to generalize relational databases. *Theoretical Computer Science*, 91:23–55, 1991.

[BDW91] P. Buneman, S. Davidson, and A. Watters. A semantics for complex objects and approximate answers. *Journal of Computer and System Sciences*, 43:170–218, 1991.

[Car84] L. Cardelli. A semantics of multiple inheritance. In G. Kahn, D. B. MacQueen, and G. D. Plotkin, editors, *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 51–67. Springer Verlag, 1984.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17:471–522, 1985.

[Car85] J. Cartmell. Formalising the network and hierarchical data models – an application of categorical logic. In *Category Theory and Computer Programming*, volume 240 of *Lecture Notes in Computer Science*, pages 466–492, 1985.

[CCF93] R. Cartwright, P.-L. Curien, and M. Felleisen. Fully abstract semantics for observably sequential functions. Technical Report COMP TR93-219, Rice University, 1993.

[Cha84] E. P. F. Chan. Optimal computation of total projections with unions of chase join expressions. In *Proc. ACM SIGMOD Conference*, pages 149–163, 1984.

[CK73] C. C. Chang and H. J. Keisler. *Model Theory*. North Holland, Amsterdam, 1973.

[Coq88] T. Coquand. Categories of embeddings. In *Logic in Computer Science*, pages 256–263. IEEE Computer Society Press, 1988.

[Coq89] T. Coquand. Categories of embeddings. *Theoretical Computer Science*, 68:221–238, 1989.

[CGW89] T. Coquand, C. Gunter, and G. Winskel. Domain theoretic models of polymorphism. *Information and Computation*, 81:123–167, 1989.

[CKS86] S. S. Cosmdakis, P. C. Kanellakis, and N. Spyratos. Partition semantics for relations. *Journal of Computer and System Sciences*, 32:280–293, 1986.

[Cur86] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Pitman, 1986.

[Cur93] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Progress in Theoretical Computer Science. Birkhäuser, second edition, 1993.

[DG90] M. Droste and R. Göbel. Universal domains in the theory of denotational semantics of programming languages. In *IEEE Symposium on Logic in Computer Science*, pages 19–34. IEEE Computer Society Press, 1990.

[DG91] M. Droste and R. Göbel. Universal information systems. *International Journal of Foundations of Computer Science*, 1:413–424, 1991.

[DG93] M. Droste and R. Göbel. Universal domains and the amalgamation property. *Mathematical Structures in Computer Science*, 3:137–159, 1993.

[Dro92] M. Droste. Finite axiomatizations for universal domains. *Journal of Logic and Computation*, 2:119–131, 1992.

[Ehr93] T. Ehrhard. Hypercoherences: A strongly stable model of linear logic. *Mathematical Structures in Computer Science*, 3:365–386, 1993.

[FT83] P. C. Fischer and S. J. Thomas. Operators for non-first-normal-form relations. In *Proc. IEEE COMPSAC*, 1983.

[Ghe90] G. Ghelli. *Proof theoretic studies about a minimal type system integrating inclusion and parametric polymorphism.* PhD thesis, Universitá di Pisa, 1990.

[GHK+80] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer Verlag, 1980.

[Gir71] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. North-Holland, 1971.

[Gir72] J.-Y. Girard. Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur, 1972. Thèse d'Etat, Université Paris VII.

[Gir86] J.-Y. Girard. The system f of variable types: Fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.

[Gra88] S. Graham. Closure properties of a probabilistic powerdomain construction. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Language Semantics*, volume 298 of *Lecture Notes in Computer Science*, pages 213–233. Springer Verlag, 1988.

[Gun85] C. Gunter. *Profinite Solutions for Recursive Domain Equations*. PhD thesis, University of Wisconsin, Madison, 1985.

[Gun87] C. Gunter. Universal profinite domains. *Information and Computation*, 72(1):1–30, 1987.

[Gun92] C. Gunter. *Semantics of Programming Languages. Structures and Techniques*. Foundations of Computing. MIT Press, 1992.

[GJ89] C. Gunter and A. Jung. Coherence and consistency in domains. *Journal of Pure and Applied Algebra*, 63:49–66, 1989.

[GS90] C. Gunter and D. S. Scott. 12: Semantic Domains. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 633–674. Elsevier Science Publishers, Amsterdam, 1990.

[Haa93] C. Haack. Die Hyperlimeskonstruktion, 1993. Diploma thesis, 86 pp.

[HM81] N. Hammer and D. McLeod. Database description with SMD: a semantic database model. *ACM Transactions on Database Systems*, 6:351–386, 1981.

[HH87] R.-E. Hoffmann and K. H. Hofmann, editors. *Continuous Lattices and their Applications*, volume 101 of *Lecture Notes in Pure and Applied Mathematics*. Marcel Dekker Inc., 1987.

[Hut90] M. Huth. *Projection-stable and zero dimensional domains.* PhD thesis, Tulane University, 1990.

[HY84] R. Hull and C.K. Yap. The format model: a theory of database organization. *Journal of the ACM*, 31:518–537, 1984.

[Hyl82] J. M. E. Hyland. The effective topos. In A. S. Troelstra and D. van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium*, pages 165–216. North-Holland, 1982.

[HO] M. Hyland and L. Ong. Dialogue games and innocent strategies: An approach to intensional full abstraction for PCF (preliminary announcement). Note distributed August 1993.

[Joh82] P. T. Johnstone. *Stone Spaces*, volume 3 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1982.

[Jon90] C. Jones. *Probabilistic Non-Determinism.* PhD thesis, University of Edinburgh, Edinburgh, 1990. Also published as Technical Report No. CST-63-90.

[JP89] C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Logic in Computer Science*, pages 186–195. IEEE Computer Society Press, 1989.

[Jun88] A. Jung. *Cartesian Closed Categories of Domains.* PhD thesis, Technische Hochschule Darmstadt, 1988.

[Jun89] A. Jung. *Cartesian Closed Categories of Domains*, volume 66 of *CWI Tracts*. Centrum voor Wiskunde en Informatica, Amsterdam, 1989.

[Jun90a] A. Jung. Cartesian closed categories of algebraic CPO's. *Theoretical Computer Science*, 70:233–250, 1990.

[Jun90b] A. Jung. The classification of continuous domains. In *Logic in Computer Science*, pages 35–40. IEEE Computer Society Press, 1990.

[Jun91] A. Jung. The dependent product construction in various categories of domains. *Theoretical Computer Science*, 79:359–364, 1991.

[JLP92] A. Jung, L. Libkin, and H. Puhlmann. Decomposition of domains. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics*, volume 598 of *Lecture Notes in Computer Science*, pages 235–258. Springer Verlag, 1992.

[JS93] A. Jung and A. Stoughton. Studying the fully abstract model of PCF within its continuous function model. In M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 230–244. Springer Verlag, 1993.

[JT93] A. Jung and J. Tiuryn. A new characterization of lambda definability. In M. Bezem J. F. Groote, editor, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 245–257. Springer Verlag, 1993.

[JS87] H. Jürgensen and D. Simovici. Towards an abstract theory of dependency constraints. *Information Systems*, 41, 1987.

[KP93] G. Kahn and G. Plotkin. Concrete domains. *Theoretical Computer Science*, 121:187–277, 1993. Translation of a technical report from 1978.

[KT84] T. Kamimura and A. Tang. Finitely continuous posets. Technical Report TR-84-1, University of Kansas, 1984.

[Kan79] A. Kanda. Fully effective solutions of recursive domain equations. In J. Beçvar, editor, *Mathematical Foundations of Computer Science*, volume 74. Springer Verlag, 1979. Lecture Notes in Computer Science.

[Ken83] W. Kent. Consequences of assuming a universal relation. *ACM Transactions on Database Systems*, 6:331–347, 1983.

[Kir93] O. Kirch. Bereiche und Bewertungen, 1993. Diploma thesis, 77 pp.

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge Studies in Advanced Mathematics Vol. 7. Cambridge University Press, 1986.

[LW84] K. G. Larsen and G. Winskel. Using information systems to solve recursive domain equations effectively. In G. Kahn, D. B. MacQueen, and G. Plotkin, editors, *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 109–130. Springer Verlag, 1984.

[Läu70] H. Läuchli. An abstract notion of realizability for which intuitionistic predicate calculus is complete. In A. Kino, J. Myhill, and R. E. Vesley, editors, *Intuitionism and Proof Theory, Proc. summer conference at Buffalo N.Y., 1968*, pages 227–234. North-Holland, 1970.

[Law88] J. Lawson. The versatile continuous order. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Language Semantics*, volume 298 of *Lecture Notes in Computer Science*, pages 134–160. Springer Verlag, 1988.

[Lib91] L. Libkin. A relational algebra for complex objects based on partial information. In J. Demetrovics and B. Thalheim, editors, *Mathematical Fundamentals of Database Systems–91*, volume 495 of *Lecture Notes in Computer Science*, pages 36–41. Springer-Verlag, 1991.

[LW92] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. Technical Report MS-CIS-93-88, University of Pennsylvania, 1992.

[LW93] L. Libkin and L. Wong. Query languages for bags. Technical Report MS-CIS-93-36, University of Pennsylvania, 1993.

[Lie82] Y. Lien. On the equivalence of database models. *Journal of the ACM*, 29:333–362, 1982.

[Lip79] W. Lipski. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, 4:262–296, 1979.

[Loa93] R. Loader. The undecidability of λ-definability. Notes, Oxford University, June 1993.

[LM92] S. Mac Lane and I. Moerdijk. *Sheaves in Geometry and Logic.* Springer Verlag, 1992.

[MPS86] D. MacQueen, G. D. Plotkin, and R. Sethi. An ideal model for recursive polymorphic types. *Information and Control*, 71:95–130, 1986.

[Mai83] D. Maier. *The Theory of Relational Databases.* Computer Science Press, 1983.

[MRW86] D. Maier, D. Rozenshtein, and D. S. Warren. Window functions. In P. Kanellakis and F. P. Preparata, editors, *The Theory of Databases*, volume 3 of *Advances in Computing Research*, pages 213–246. JAI Press, 1986.

[ML84] P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, Naples, 1984.

[McC79] N. McCracken. *An investigation of a programming language with a polymorphic type structure.* PhD thesis, Syracuse University, 1979.

[Mer84] T. H. Merrett. *Relational Information Systems.* Reston Publishing Co, 1984.

[Mil77] R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[Mit90] J. C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 365–458. North Holland, 1990.

[MM91] J. C. Mitchell and E. Moggi. Kripke-style models for typed lambda calculus. *Annals of Pure and Applied Logic*, 51:99–124, 1991. Preliminary version in *Proc. IEEE Symp. on Logic in Computer Science,* 1987, pages 303–314.

[MB81] A. Motro and O. P. Buneman. Constructing superviews. *SIGMOD*, 1981.

[Mul87] K. Mulmuley. *Full Abstraction and Semantic Equivalence.* The MIT Press, 1987.

[Oho86] A. Ohori. Denotational semantics of relational databases. Master's thesis, University of Pennsylvania, 1986.

[Oho90] A. Ohori. Semantics of types for database objects. *Theoretical Computer Science*, 76:53–91, 1990.

[OB88] A. Ohori and P. Buneman. Type inference in a database programming language. In *Proc. ACM Conference on LISP and Functional Programming*, pages 174–183, 1988.

[ÖY85] Z. Özsoyoğlu and L. Yuan. A normal form for nested relations. In *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pages 251–260, 1985.

[Pfe93] F. Pfenning. On the undecidability of partial polymorphic type reconstruction. *Fundamenta Informaticae*, 19:185–199, 1993.

[Pit87] A. M. Pitts. Polymorphism is set-theoretic, constructively. In D. H. Pitt, A. Poigné, and D. E. Rydeheard, editors, *Category Theory and Computer Science*, volume 283 of *Lecture Notes in Computer Science*, pages 12–39. Springer Verlag, 1987.

[Plo73] G. D. Plotkin. Lambda-definability and logical relations. Memorandum SAI-RM-4, University of Edinburgh, October 1973.

[Plo76] G. D. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5:452–487, 1976.

[Plo77] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[Plo78] G. D. Plotkin. $\mathbf{T}^{\omega}$ as a universal domain. *J. Computer and System Sciences*, 17:209–236, 1978.

[Plo80] G. D. Plotkin. Lambda-definability in the full type hierarchy. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–373. Academic Press, London, 1980.

[Plo81] G. D. Plotkin. Post-graduate lecture notes in advanced domain theory (incorporating the "Pisa Notes"). Dept. of Computer Science, Univ. of Edinburgh, 1981.

[Puh90] H. Puhlmann. Verallgemeinerung relationaler Schemata in Datenbanken mit Informationsordnung, 1990. Diploma thesis, 78 pp.

[Puh93] H. Puhlmann. The snack powerdomain for database semantics. In A. M. Borzyszkowski and S. Sokołowski, editors, *Mathematical Foundations of Computer Science*, volume 711 of *Lecture Notes in Computer Science*, pages 650–659, 1993.

[Rey74] J. C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer Verlag, 1974.

[Rey84] J. C. Reynolds. Polymorphism is not set-theoretic. In G. Kahn, D. B. MacQueen, and G. D. Plotkin, editors, *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 145–156. Springer Verlag, 1984.

[Ris85] N. Rishe. On denotational semantics of data bases. In A. Melton, editor, *Mathematical Foundations of Programming Semantics*, volume 239 of *Lecture Notes in Computer Science*, pages 249–274. Springer Verlag, 1985.

[RKS84] A. M. Roth, H. F. Korth, and A. Silberschatz. Extended algebra and calculus for ¬1nf relational databases. Technical Report TR-84-36, The University of Texas at Austin, 1984. revised 1985.

[RKS85] A. M. Roth, H. F. Korth, and A. Silberschatz. Null values in ¬1nf relational databases. Technical Report TR-85-32, The University of Texas at Austin, 1985.

[Rot91] M. Rothe. Retraktionen auf stetigen Bereichen, 1991. Diploma thesis, 34pp.

[RK85] W. C. Rounds and R. Kasper. A complete logical calculus for record structures representing linguistic information. Technical report, University of Michigan, 1985.

[SD80] N. Saheb-Djahromi. CPO's of measures for nondeterminism. *Theoretical Computer Science*, 12:19–37, 1980.

[Sch86] D. A. Schmidt. *Denotational Semantics.* Allyn and Bacon, 1986.

[Sch77] J. W. Schmidt. Some high level language constructs for data of type relation. *ACM Transactions on Database Systems*, 5, 1977.

[Sci79] E. Sciore. Null values, updates, and incomplete information. Technical report, Princeton University, 1979.

[Sci80] E. Sciore. *The universal instance and database design.* PhD thesis, Princeton University, 1980.

[Sco72] D. S. Scott. Continuous lattices. In E. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer Verlag, 1972.

[Sco76] D. S. Scott. Data types as lattices. *SIAM J. Computing*, 5:522–587, 1976.

[Sco81a] D. S. Scott. Lectures on a mathematical theory of computation. Monograph PRG-19, Oxford University Computing Laboratory, Oxford, 1981.

[Sco81b] D. S. Scott. Some ordered sets in computer science. In I. Rival, editor, *Ordered Sets*, pages 677–718, Dordrecht, 1981. Reidel.

[Sco82a] D. S. Scott. Domains for denotational semantics. In M. Nielson and E. M. Schmidt, editors, *International Colloquium on Automata, Languages and Programs*, volume 140 of *Lecture Notes in Computer Science*, pages 577–613. Springer Verlag, 1982.

[Sco82b] D. S. Scott. Lectures on a mathematical theory of computation. In M. Broy and G. Schmidt, editors, *Theoretical Foundations of Programming Methodology*, NATO Advanced Study Institutes Series, pages 145–292. Reidel, 1982.

[Sco93] D. S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121:411–440, 1993. Reprint of a manuscript written in 1969.

[SFL83] J. M. Smith, S. Fox, and T. Landers. *ADAPLEX: Rationale and Reference Manual*. Computer Corporation of America, second edition, 1983.

[Shi85] S. M. Shieber. An introduction to unification-based approaches to grammar. In *Proc. 23rd Annual Meeting of the Association for Computational Linguistics*, 1985.

[Sie92] K. Sieber. Reasoning about sequential functions via logical relations. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Proc. LMS Symposium on Applications of Categories in Computer Science, Durham 1991*, volume 177 of *LMS Lecture Note Series*, pages 258–269. Cambridge University Press, 1992.

[Smy77] M. B. Smyth. Effectively given domains. *Theoretical Computer Science*, 5:257–274, 1977.

[Smy78] M. B. Smyth. Powerdomains. *Journal of Computer and Systems Sciences*, 16:23–36, 1978.

[Smy83] M. B. Smyth. The largest cartesian closed category of domains. *Theoretical Computer Science*, 27:109–119, 1983.

[Sta82a] R. Statman. Completeness, invariance and λ-definability. *Journal of Symbolic Logic*, 47:17–26, 1982.

[Sta82b] R. Statman. Embeddings, homomorphisms and λ-definability. Manuscript, Rutgers University, 1982.

[Sta83] R. Statman. λ-definable functionals and $\beta\eta$ conversion. *Arch. Math. Logik*, 23:21–26, 1983.

[Sta85] R. Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65:85–97, 1985.

[SD92] R. Statman and G. Dowek. On statman's finite completeness theorem. Technical Report CMU-CS-92-152, Carnegie Mellon University, 1992.

[Sto88] A. Stoughton. *Fully Abstract Models of Programming Languages.* Research Notes in Theoretical Computer Science. Pitman/Wiley, 1988.

[Sto90] A. Stoughton. Equationally fully abstract models of PCF. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics*, volume 442 of *Lecture Notes in Computer Science*, pages 271–283. Springer Verlag, 1990.

[Tay87a] P. Taylor. Homomorphisms, bilimits and saturated domains. Some very basic domain theory. Draft, Imperial College London, 15 pp., 1987.

[Tay87b] P. Taylor. *Recursive Domains, Indexed category Theory and Polymorphism.* PhD thesis, Cambridge University, 1987.

[Ull82a] J. D. Ullman. *Principle of Database Systems.* Pitman, second edition, 1982.

[Ull82b] J. D. Ullman. The U.R. strikes back. In *Proc 1st ACM Symp on Principles of Database Systems*, pages 10–22, 1982.

[Ull83] J. D. Ullman. Universal relation interfaces for database systems. In *Proc. IFIP*, 1983.

[Vau61] R. Vaught. *Infinitistic Methods*, chapter Denumerable models of complete theories, pages 303–321. Pergamon Press, 1961.

[WD80] K. Weihrauch and T. Deil. Berechenbarkeit auf cpo's. Technical Report 63, Rheinisch-Westfälische Technische Hochschule Aachen, 1980.

[Win81] G. Winskel. *Events in Computation*. PhD thesis, University of Edinburgh, 1981.

[Zan84] C. Zaniolo. Database relation with null values. *Journal of Computer and System Sciences*, 28:142–166, 1984.